Imperial College London

Worst–Case Resource–Usage Analysis of Java Card Classic Editions Application Bytecode

Luke Dominic Mark Urquhart

Supervised by Professor Chris Hankin

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in the Department of Computing of Imperial College London.
Except for those parts in which it is explicitly stated to the contrary,
this thesis is the original work of Luke Dominic Mark Urquhart
and has not been submitted for any degree at this or
any other academic or professional institution.

April 2016

©The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work

Abstract

Java Card is the dominant smartcard technology in use today, with over 12 billion *Java Card* smartcards having shipped globally in the last 15 years. Almost exclusively, the deployed *Java Card* smartcards are instances of a Classic edition for which garbage collection is an optional component in even the most recent Classic edition. Poorly written or malicious *Java Card* applications may drain the available memory of a *Java Card* Virtual Machine to the point the card becomes unusable, and undisciplined use of the transaction mechanism may exhaust the available transaction buffers, resulting in programmatic abort by the *Java Card* Runtime Environment and so limit the range of services a *Java Card* application may successfully be able to offer. Given the size and global nature of the user base, and the commercial importance of *Java Card*, there is a stunning lack of tools supporting analysis or certification of the memory, transactional or CPU usage of *Java Card* applications.

In this thesis we present a worst-case resource-usage analysis tool for *Java Card* which is capable of producing worst-case memory usage and worst-case execution-time estimates for *Java Card* applications (also known as applets). Our main theoretical contribution is a static analysis for *Java Card* applets at the bytecode level which conservatively approximates properties of interest affecting memory usage, input-output/APDU usage and transaction usage.

Our static analysis provides the high-level information for subsequent worst-case resource-usage analysis in our tool which exploits well-known results and techniques from hard real-time systems. We generate a resource usage graph per registered applet lifecycle method entry point as the start node and the control-flow returning to the *Java Card* Runtime Environment as the final node. We use the Implicit Path Enumeration Technique to generate and solve Integer Linear Programming problems representing the worst-case memory-usage and worst-case execution-time.

Acknowledgements

I would like to thank my PhD supervisor for his endless patience, unfailing encouragement, good humour and sage advice.

I would also like to thank my viva examiners for the time and effort they spent reading this thesis, for their constructive and probing questions, and for the handful of minor corrections they requested, each of which has served to strengthen this thesis. Special thanks go to René Hansen for travelling from Denmark to London for the sole purpose of being the external examiner on my viva.

Dedication

This thesis is dedicated jointly:

- to my late mum, who was quite simply the best, kindest, most loving, selfless and *good* person I've ever known. Thank you for consciously allowing your children to make their own choices in life, decide for themselves the answers to life's big questions, for loving, encouraging and supporting each of your children unconditionally, and for your sole ambition for each of your children be that they are happy. Your family and friends love and miss you. May you rest in peace mum.
- to the Lord God Almighty. As a committed Christian, in this, as in all my endeavours, I give thanks to the Ancient of Days. Thank you for your common grace to the whole of mankind. Your Name be praised! Amen.

Contents

1 Introduction **13**

1.1 Thesis statement and high-level supporting arguments . . . 13

1.2 Structure of this Thesis . . . 16

1.3 Scope and Assumptions . . . 17

1.4 Individual contribution/work . . . 19

2 Background Material on Smartcards and *Java Card* Classic Edition 3 **23**

2.1 Overview . . . 23

2.2 Smartcards . . . 23

2.3 *Java Card* Platform . . . 27

3 The Carmel programming language **42**

3.1 Introduction . . . 42

3.2 The Carmel Syntax . . . 45

3.3 Program Structures . . . 48

3.4 Values and Runtime Data Types . . . 49

3.5 Run-Time Structures . . . 52

3.6 Exceptions . . . 59

3.7 The Applet Firewall . . . 61

3.8 Firewall Predicates . . . 67

3.9 Carmel Operational Semantics . . . 71

4 Base Control-Flow Analysis **96**

4.1 Introduction . . . 96

4.2 Abstract Domains including Analysis Domains . . . 96

4.3 Analysis Specification. . . . 116

4.4 Theoretical Properties . . . 126

4.5 Specification . . . 130

4.6 Whole Program Analysis . . . 167
4.7 Constructing a worklist solver . . . 169

5 Extended Control-Flow Analysis 179
5.1 Introduction . . . 179
5.2 Deficiency in base control-flow analysis handling of loops . . . 179
5.3 Extended control-flow analysis remedies deficiency in base control-flow analysis handling of loops . 182
5.4 Modifying the base control-flow analysis worklist solver . . . 197

6 Worst–Case Resource–Usage Analysis 198
6.1 Introduction . . . 198
6.2 Identifying natural loops in Carmel . . . 199
6.3 Capture local/loop variable information from each natural loop . . . 202
6.4 Validate conditions for being able to determine loop-bounds . . . 205
6.5 Calculate loop-bounds . . . 206
6.6 Generating an integer linear programming problem per registered applet lifecycle method . . . 207
6.7 Evaluation of the Precision of the Worst-Case Resource-Usage Analysis . . . 224

7 Related Work 234
7.1 Introduction . . . 234
7.2 *Java Card* . . . 236
7.3 Embedded systems . . . 238
7.4 General programming languages . . . 240
7.5 Evaluation of the loop-bound precision of our tool . . . 242

8 Conclusions and further work 245
8.1 Introduction . . . 245
8.2 Main Achievements of this Thesis . . . 245
8.3 Future Work . . . 247
8.4 Thesis statement evaluation . . . 248

Bibliography 249

A API methods 259
A.1 Introduction . . . 259
A.2 Transaction Methods . . . 260
A.3 APDU methods . . . 284

B Base Control-Flow Analysis **285**

B.1 Introduction . . . 285

B.2 Abstract Domains including Analysis Domains . . . 285

B.3 Analysis Specification. . . . 309

B.4 Theoretical Properties . . . 318

B.5 Auxiliary Functions and Helper Proofs . . . 402

C A Reaching Definitions for Local Variables Analysis **436**

C.1 Introduction . . . 436

C.2 Scope and Definition of Analysis . . . 437

D Initial States for Object Heap and Static Heap **440**

D.1 Introduction . . . 440

D.2 JCRE–owned exceptions . . . 441

D.3 JCRE–owned global arrays . . . 447

D.4 Miscellaneous JCRE-owned objects . . . 449

D.5 Initial State for Object Heap - H_{Init} . . . 449

D.6 Initial State for Static Heap - K_{Init} . . . 450

D.7 Abstract Initial State for Object Heap - $\widehat{H_{Init}}$ . . . 453

D.8 Abstract Initial State for Object Heap - $\widehat{K_{Init}}$ . . . 455

E Design and implementation of the Fulgurite iterative worklist solver **456**

E.1 Overview . . . 456

E.2 Implementation of Abstract Values in Fulgurite . . . 457

E.3 Fulgurite worklist algorithm . . . 462

List of Figures

2.1 Typical appearance of the contact-strip on a smartcard 24

3.1 Subtype Relation Rules 48

6.1 Control-flow graph of a Carmel program, with three embedded loops 201
6.2 Control-flow graph with resource costs of an overridden method 214
6.3 Control-flow graph with maximum resource costs of an overridden method 218
6.4 Control-flow graph of a Carmel program showing a triple-nested triangular loop. 221
6.5 Control-flow graph of a Carmel program showing a triple-nested triangular loop, with frequencies . . 223

E.1 UML diagram showing the implementation hierarchy of abstract values in Fulgurite 464
E.2 UML diagram showing the high-level interfaces and base abstract class in Fulgurite 465
E.3 UML diagram showing the abstract number values in Fulgurite 466
E.4 UML diagram showing the abstract object reference and abstract return address values in Fulgurite 467
E.5 Implementation of `numop` t $binop$ 471
E.6 Implementation of subtraction method for abstract short numbers 472
E.7 Implementation of add method for abstract integer numbers 473

List of Tables

2.1 Java Features supported in *Java Card* 3 Classic Edition . . . 31
2.2 Java Keywords supported in *Java Card* 3 Classic Edition . . . 32
2.3 Java Keywords NOT supported in *Java Card* 3 Classic Edition . . . 32
2.4 Java bytecode instructions supported in *Java Card* 3 Classic Edition . . . 33

3.1 Valid Initial Configurations in Carmel – Table 1 . . . 62
3.2 Valid Initial Configurations in Carmel – Table 2 . . . 63
3.3 Valid Initial Configurations in Carmel – Table 3 . . . 63
3.4 Valid Initial Configurations in Carmel – Table 4 . . . 64
3.5 Valid Initial Configurations in Carmel – Table 5 . . . 65
3.6 Exception Handling in Carmel . . . 66

4.1 Binary least upper-bounds operator $\bigsqcup_{Val}$ over the complete lattice $\langle \widehat{\mathsf{Val}},\ \sqsubseteq_{Val} \rangle$ . . . 103
4.2 Abstract context representation . . . 107
4.3 Explanation of flow-logic rule for the `nop` bytecode instruction . . . 121
4.4 Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 1 . . . 122
4.5 Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 2 . . . 123
4.6 Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 1 . . . 124
4.7 Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 2 . . . 125
4.8 Fundamental Constraint Satisfaction/Update Functions . . . 177
4.9 Higher-Order Constraint Satisfaction/Update Functions . . . 178

5.1 Simple bounded loop, in Java and Carmel . . . 180
5.2 Base CFA specification of `if` t $cmpop$ `goto` *addr* statement . . . 181
5.3 Desired least solution to the base control-flow analysis constraints for sample Carmel program . . . 186
5.4 Actual least solution to the base control-flow analysis constraints for sample Carmel program . . . 187
5.5 Reaching definition and loaded from cross analysis of cases of interest . . . 188

6.1 Algorithm to identify natural loops in a control-flow graph . . . 202

6.2 Carmel program demonstrating overriding, part 1 . . . 212
6.3 Carmel program demonstrating overriding, part 2 . . . 213
6.4 ILP generated from Carmel program demonstrating overriding, Part 1 . . . 215
6.5 ILP generated from Carmel program demonstrating overriding, Part 2 . . . 216
6.6 Solution to ILP generated from Carmel program demonstrating overriding . . . 217
6.7 Carmel program demonstrating triangular triple-nested loop . . . 220
6.8 Calculation of loop bounds for Carmel program demonstrating triangular triple-nested loop . . . 221
6.9 ILP generated from Carmel program demonstrating a triangular triple-nested loop . . . 222
6.10 Solution to ILP generated from Carmel program demonstrating triangular loops . . . 224
6.11 Illustration on how the use of reaching definitions improve the prevision of loop bounds . . . 228
6.12 An algorithm for determining whether a given positive integer is prime . . . 229
6.13 Illustration on how Fulgurite is able to calculate loop bounds for a class of simple looping algorithms 230
6.14 Exploratory Fulgurite code handling primes scenario, part 1 . . . 231
6.15 Exploratory Fulgurite code handling primes scenario, part 2 . . . 232
6.16 Cross-comparison of our tool with the Mälardalen worst-case execution-time benchmarks [GBEL10] 233

7.1 Reproduction of [Listing 6.1: The example used for WCET analysis] from [Sch09] . . . 244

B.1 Binary least upper-bounds operator $\bigsqcup_{Val}$ over the complete lattice $\langle \widehat{\mathsf{Val}},\ \sqsubseteq_{Val} \rangle$ . . . 292
B.2 Abstract context representation . . . 297
B.3 Explanation of flow-logic rule for the `nop` bytecode instruction . . . 313
B.4 Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 1 . . . 314
B.5 Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 2 . . . 315
B.6 Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 1 . . . 316
B.7 Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 2 . . . 317
B.8 arraystore t first two cases . . . 366
B.9 arraystore t third until sixth cases . . . 367
B.10 arraystore t seventh until tenth cases . . . 390

C.1 Monotone framework definition for Reaching Definitions for Local Variables Analysis . . . 438
C.2 Kill- and gen- function definition for Reaching Definitions for Local Variables Analysis . . . 438
C.3 Flow relation $flow$ for Reaching Definitions for Local Variables Analysis . . . 438
C.4 Algorithm for solving dataflow equations arising from analysis . . . 439

D.1 A Succinct Notation for Defining Objects in the domain of object heap *H* . . . 441

E.1 Binary least upper-bounds operator $\bigsqcup_{Val}$ over the complete lattice $\langle \widehat{\mathsf{Val}},\ \sqsubseteq_{Val} \rangle$ . . . 459
E.2 Excerpt from the core of the Fulgurite worklist algorithm, Part 1 . . . 468

E.3 Excerpt from the core of the Fulgurite worklist algorithm, Part 2 469
E.4 Excerpt from the core of the Fulgurite worklist algorithm, Part 3 470

Chapter 1

Introduction

In this Chapter:

- the central proposition of this thesis is stated and initial high–level arguments advanced to support the claim;
- the structure of this thesis is outlined;
- the scope and assumptions of this thesis are specified;
- a clear account is given of my individual contributions/work. This is particularly important in this thesis as it builds on top of the substantial and successful SecSafe project [HSN+03, Pro02] and related work [HS05, NN02, Han05].

1.1 Thesis statement and high-level supporting arguments

Formally speaking, our thesis statement is:

"The *Java Card* Classic edition platform is worst-case resource-usage analysis-*friendly*. In particular, the *Java Card* platform is *highly amenable* to worst-case execution-time analysis and worst-case dynamic memory allocation analysis through a combination of techniques from the fields of static analysis and integer linear programming".

1.1.1 Definition of a worst-case resource-usage analysis

The seminal papers for determining conservative worst-case execution-time estimates for programming languages recast the classical search problem of *explicitly* finding the worst-case execution path through a program to one of

generating and solving an integer linear programming problem *implicitly* representing all possible paths of execution through the program [LM95, PS97] and essentially reduces the problem to one of:

- decomposing the program into basic blocks and determining control-flow between basic blocks
- identifying the resource cost (execution time) of each basic block
- identifying loops in the program under consideration in terms of basic blocks (loop-header detection)
- finding maximum number of iterations for each loop in the program under consideration (loop-bounds)
- ensuring isolated entry and exit points exist in the control-flow of the program under consideration
- ensuring no recursive calls occur in any basic block reachable from the isolated entry point

of which determining maximum loop-bounds for each loop remains a "hard" problem and an area of active research. Indeed in his excellent survey paper [WEE+08], the author identified the need for novel and more powerful static program analyses capable of capturing loop-bound information as the most pressing and significant outstanding issue in worst-case execution-time analysis. This is where we make our most significant contribution: our program analysis is the most precise static analysis for *Java Card* of which we are aware, and has been designed to capture sufficient information to facilitate loop-bounds calculation.

Determining the maximum of the objective function for the integer linear programming yields both the worst-case execution-time of the program under consideration and sufficient information to identify the set of worst-case execution paths. This is known as the Implicit Path Enumeration Technique (IPET) and is the dominant approach employed in industry to compute the worst-case execution-time in hard real-time systems [WEE+08].

By noting it is only the association of execution time as the cost of enacting a basic block which links this approach to determining worst-case execution-time, we generalise the approach to any property of interest that may be expressed as a resource cost function on each basic block. This is how we define a worst-case resource-usage analysis: the combination of our static analysis and standard Implicit Path Enumeration Techniques with a resource cost function defined on basic blocks. Of particular importance in this thesis are the worst-case dynamic memory allocation analysis, where the resource cost function expresses the memory cost of dynamic memory allocation of enacting a basic block, and the worst-case execution-time analysis, where the resource cost function expresses either the number of instructions executed in a block or the number of CPU cycles of enacting a basic block; space and time being the classical axes of interest wrt resource usage. *Java Card* applications follow a lifecycle model and so a worst-case resource-usage analysis of a set of *Java Card* applications generates a family of integer linear

programming problems and corresponding worst-case resource-usage metrics, one per applet lifecycle method, per applet in the program under consideration.

1.1.2 High-level supporting arguments

We deem the high-level arguments for believing the *Java Card* Classic edition platform to be worst-case resource-usage analysis-friendly to be comprehensible without prior knowledge of the *Java Card* platform[1]. Many of these directly or indirectly suggest the determination of loop-bounds in *Java Card* applications is inherently simpler than in other languages. Since as noted above, the calculation of loop-bounds is both a necessary condition and the hardest subtask of formulating a worst-case resource-usage analysis, this augurs well for the precision of the worst-case resource-usage estimates we produce – always a safe/conservative estimate of the actual worst-case resource-usage (i.e. the worst-case resource estimate we generate is always equal to or greater than the actual worst-case) but we should prefer our estimates to be as close as possible to the actual worst-case resource-usage as possible whilst still being safe.

1.1.2.1 Architecture-based

The *Java Card* Classic edition platform [Ora11a, Ora11b, Ora11c]:

- is a request-reply architecture with ISO well-defined input- and output-formats and sizes, and each *Java Card* application invocation is expected to terminate in a finite number of steps and return control to the *Java Card* Runtime Environment. A subtle corollary of this is that it is straightforward in our analysis to describe the set of *all* possible input values – which necessarily includes the particular values that lead to the "worst-case" resource usage[2];
- supports a minimal set of primitive types and minimal API: only integral data types are supported and standard Java container/Collection types such as List, Set and Collections which use memory dynamically are not built into the *Java Card* platform;
- has a purely-interpreted, single-threaded execution model[3];

[1] In Chapter 2, we present an introduction to smartcards and the *Java Card* Classic Edition 3, and in Chapter 3 we detail a structured operational semantics for the language Carmel which is a rationalisation of the *Java Card* Virtual Machine Language and its packaged-for-deployment-on-card format.

[2] This is in stark contrast to the general case. Per [WEE+08]: "A reliable guarantee based on the worst-case execution-time of a task could easily be given if the worst-case input for the task were known. Unfortunately, in general the worst-case input is not known and hard to derive".

[3] In contrast to the Common Intermediate Language (CIL) into which languages such as C# and F# are compiled for execution in a Common Language Infrastructure of .NET/Mono runtime engines, or Dalvik bytecode for execution on a Dalvik or Android Runtime Environment. In the former case, CIL is always compiled into native code either at runtime or ahead-of-time. In the latter case, Dalvik bytecode may be executed in one of three modes: purely interpreted, partly interpreted and partly native (via just-in-time compilation of frequently executed sequences of bytecode), or ahead-of-time.

1.1.2.2 Programming conventions-based

- *Java Card* application developers are expected to always be mindful of the extremely resource–constrained nature of the smartcards on which their applications typically run and to follow certain coding conventions when writing applications. These coding conventions reflect a need for developers to write Java code in a way which minimises dynamic resource usage, particularly dynamic memory allocations and call-stack depth (for this reason, the use of recursion is very strongly discouraged [Che00]);

- *Java Card* applications are expected to be smaller and simpler in nature than standard or Enterprise Java applications and loop conditions are expected to be simpler and more likely to be tied to array lengths (due to the lack of the Collections framework of discrete mathematical structures available, and widely used, in standard Java) whose lengths are more likely to be statically known due to the coding convention of applications reserving/allocating their normal life-long memory requirements upfront at applet creation and registration/install time[4];

- The general conditions to make the worst-case execution-time decidable (and so make the worst-case resource-usage analysis decidable) for Java applications are likely to be satisfied by most *Java Card* applications [Sch09]:

  1. Programs must not contain any recursion (our analysis will determine)
  2. Dynamic class loading is forbidden (also a very natural *Java Card* platform restriction)
  3. The upper bound of each loop has to be known (our analysis will determine)

1.2 Structure of this Thesis

So as not to bog down the reader with excessive technical detail on a first reading, this thesis has been written and structured to communicate the high–level ideas and include sufficient technical content to engender confidence in the correctness of the material presented in the main chapters while providing full technical details in the appendices.

- Chapter 1: Introduction – this chapter;

- Chapter 2: Introduction to smartcards and the *Java Card* Classic Edition 3;

[4] This is recommended in [Che00] as – separate from programmatic transactions using the *Java Card API* – the applet `install` method is wrapped in a card-specific/native transaction so it can claw back any allocated memory should any unhandled exceptions occur during that method.

- Chapter 3: Introduction to the *Java Card* virtual machine language equivalent language Carmel and specification of the *Java Card Virtual Machine* bytecode instructions, including initial configurations and exception handling. For brevity, the specification of operational semantics for *Java Card* platform API methods is given in Appendix A;
- Chapter 4: Base control-flow analysis and proofs for a handful of key cases, including presentation of a worklist algorithm capable of computing the least solution to the constraints generated by the control-flow analysis. The development of the worklist algorithm follows naturally from the program analysis specification as a compositional verbose flow-logic analysis [NN02] without stratified constraints. For brevity, the full correctness proofs are given in Appendix B;
- Chapter 5: Evaluation of the ability of the base control-flow analysis to determine loop-bounds for simple loops. Extended control-flow analysis with the ability to bound simple numeric loops, based on a reaching definitions analysis for local variables, and refinement of the worklist algorithm presented in Chapter 4. Evaluation of the ability of the extended control-flow analysis to determine loop-bounds for simple loops. The Reaching Definitions for Local Variable analysis is given in Appendix C;
- Chapter 6: Worst-case resource-usage analysis, including dominators approach, bounds analysis computed from the extended control-flow analysis of Chapter 5, and generation of a max integer linear problem whose solution yields the worst-case demands, generated per applet lifecycle method. Utilisation of exception behaviour, and the ability to handle triangular loops makes the worst-case resource analysis superior to other known WCET tools e.g. JOP and SWEET;
- Chapter 7: Related work;
- Chapter 8: Conclusions and further work;

1.3 Scope and Assumptions

To manage the scale and complexity of providing a worst-case resource-usage analysis for *Java Card* Classic editions, we limit the scope of this work to *Java Card* Classic 3.0.1 applications which do not use standard *Java Card* cryptography or the optional *Java Card* packages from the *Java Card* Platform 3.0.1 API. In particular our program analysis specifies operational semantics and abstract counterparts for:

- All the atomic bytecodes supported in the *Java Card* Virtual Machine 3.0.1 Specification;
- The following packages of the *Java Card* API 3.0.1 Specification:

- java.io
- java.lang
- java.rmi
- javacard.framework
- javacard.framework.service

and is capable of providing conservative worst-case resource-usage estimates for *Java Card* platform 3.0.1 applications whose bytecode meet these restrictions.

It is further assumed that all *Java Card* applications to be analysed:

- Have successfully passed bytecode-verification by a correct bytecode verifier to satisfy fundamental Java structural and type constraints;
- Have flow-reducible control-flow graphs (or been preprocessed to be flow-reducible via node-splitting techniques[5]). Flow-reducibility is a requirement in the algorithm we use for finding natural loops [LT79];
- Supply full bytecode of any third-party libraries used by the application, and that bytecode has successfully passed bytecode-verification by a correct bytecode-verifier;
- Are binary-compatible i.e. the JCRE would allow these versions of the applications to be co-loaded on the card based on their export files.

Finally it is assumed users are only interested in semantics-based worst-case resource-usage estimates i.e. resource estimates on a *Java Card* smartcard which correctly implements the *Java Card* platform specification [Ora11a, Ora11b, Ora11c] and whose hardware has not been tampered with. It has been shown in [LBL+15, LBR+13, BG14, LBL+13, BLLL15, BTL13a, SFL13, BTL13b, BICL11] that it is possible to induce illegal control-flow and non-deterministic behaviour on particular *Java Card* smartcards through poor implementation of API methods or the transaction mechanism or via hardware attacks – we do not model and cannot provide resource-usage guarantees for such cards.

[5] It is always possible to generate a flow-reducible graph/program from a flow-irreducible graph/program e.g.[JC97]. The approach in the cited paper has the additional advantage of trying to minimise the code size increase implicit in the node splitting/copying techniques, a particular advantage when dealing with such resource-constrained devices as *Java Card* smartcards.

1.4 Individual contribution/work

1.4.1 Operational semantics

Independently verified the correctness of the operational semantics of the atomic bytecode instructions for Carmel at the end of SecSafe (consistent with *Java Card* Classic edition 2.1) and updated them to *Java Card* Classic edition 3 (including logical channel related changes to firewall predicates for proper handling of `MultiSelectable` and non-`MultiSelectable` applets). In doing so I:

- Corrected a handful of minor mistakes e.g. bytecode instructions:
  - `dup` *n d* the pattern-matching on the size of $n = nbWords(S_1)$ and $d = nbWords(S_2 : S_1)$ ensures $d \geq n$ but the most common use of the instruction is `dup` *1 0* which copies/duplicates a freshly created object reference on the top of the operand stack as a prelude to invoking an object initialiser on one of the two identical object references.
  - `putstatic` *f* requires a firewall access check as per Section 6.2.8.1 of [Ora11b]. This firewall check was missing in SecSafe;
  - `push` *t c* requires a byte value to be sign-extended to a short before being pushed on top of the operand stack.
- Integrated transaction semantics (including semantic components *JH* for transaction heap and I for invalidated references) into the operational semantics for bytecode instructions that either persist values to the heap or consult values from the transaction heap/commit buffer e.g.
  - `new` τ
  - `new` τ`[ ]`
  - `putfield` *f*
  - `arraystore` *t*

  Indeed, anywhere `getJHorH()` occurs in an operational semantic rule. the global heap or the transaction heap may be referenced.
- The invalidated components I allows a record to be kept of the locations created on the heap typically for new objects or API object creation methods inside a transaction that was subsequently aborted. References in I are to be treated as `null` pointers. Also incorporated initial method transaction depth and current method transaction depth into each stack frame from [HS05]. Knowing the current method transaction depth

is necessary as nested transactions are not possible in Classic editions and we need to know the current transaction depth (i.e. whether a transaction is currently in progress) for when a Carmel program attempts to start, commit or abort a transaction to behave in compliance with the JCRE and JCVM specifications.

- Integrated explicit semantic component *R* for applet registry into the operational semantics and added explicit reference io_n into each stack frame to keep track of the current Input-Output (IO) state of the smartcard. We do so because, in the former case, only registered applets may have lifecycle method invoked on them and because e.g. an applet identifier (AID) may be registered only once, as the AID is the unique key used to tell the JCRE to select an applet and make it ready to receive commands. In the latter case, the changes in the IO state of the card follow a strict ordering: in particular, that the communications state should be set to read prior to any attempt to read the incoming buffer, then communications should be set to write to send information back to the Card Acceptance Device or terminal (CAD). Once the smartcard is set to send information back to the CAD, attempting to read the incoming data from the smartcard is forbidden and throws an exception. So we need to include this information as part of the machine configuration so that we can comply with the JCRE and JCVM specifications.

- Introduced an additional component to each runtime value: a runtime address at which the value was created or last modified. This runtime address serves no function in the operational semantics, but its inclusion in the operational semantics is required so our standard proof techniques of representation functions and structural induction preserve the runtime label in the abstract runtime value, facilitating in the extended control-flow analysis the use of novel analogues of classical loop-induction analyses and reaching definitions analyses.

- Clearly defined initial configurations including initial states of the persistent global object heap and static heap.

- Extended exception handler to communicate current IO state and current method transaction depth to the handler stack frame and address.

- Independently developed operational semantics for the *Java Card* API methods in Carmel consistent with *Java Card* Classic Edition 3 and with the operational semantics for the atomic bytecode instructions. While some effort had been made to develop operational semantics for some API methods in SecSafe, it was simpler to start from scratch using the same framework as for the atomic bytecodes due to its integration with the transaction and IO semantics.

1.4.2 Overall strategy for determining worst-case resource-usage

Determined overall strategy for calculating worst-case resource-usage. Having analysed the *Java Card* platform specification [Ora11c, Ora11b, Ora11a] and the Sun official *Java Card* book [Che00], and surveyed sample applets from various *Java Card* development kits, I came to the conclusion outlined in Section 1.1.2 for believing that the *Java Card* Classic platform would be highly amenable to worst-case resource-usage analysis when used in the way it is expected and instructed to be used, since *Java Card* Classic is essentially an array-based language, where recursion is very strongly discouraged, with iteration likely to be tied to array lengths. Due to these properties, of the various approaches to determining maximum resource-usage bounds researched, the Implicit Path Enumeration Technique and integer-linear programming stood out as the perfect companion technique, since its principal requirements are to ensure no recursion and the availability of maximum iteration/loop bounds for each reachable loop in the control-flow graph of the program under consideration.

1.4.3 Program analyses

Having established that our program analysis/analyses must support the determination of maximum loop bounds for each loop and the detection of recursion to utilise the IPET technique we have selected for use, I reviewed the program analyses from SecSafe and related works [HSN+03, Pro02, HS05, NN02, Han05]. None of the existing program analyses were able to bound loops even for simple loops or detect recursion. Further, the existing program analyses:

- only captured aspects of the operational semantics, the control-flow analyses were very approximate and in particular were weak around the handling of loops/numbers, whereas bounding loops requires as strong and accurate modelling and handling of loops/numbers as possible
- the choice of contexts was minimal e.g. the transaction depth for the transaction flow analysis.

I resolved to produce as precise a program analysis as possible to maximise:

- the number and types of Carmel programs for which we would be able to provide worst case resource usage;
- the precision of the maximum resource bounds we calculate. To this end, abstract values (numbers, object references and objects including arrays and class instance) are modelled in greater detail and generally much more closely to the operational semantics than in SecSafe. Similarly, I chose the context to include sufficient machine state information to accurately answer questions around:
  - Firewall predicate access checks
  - Transactional and IO states

To improve precision of control-flow (including exceptional behaviour), and data-flow.

I produced the extended control-flow program analysis covering both atomic bytecode instructions and the API methods and proved them correct wrt to the operational semantics. To improve the precision of loop bounds calculations, I developed and employed novel analogues of classical loop-induction analyses and reaching definitions analyses.

1.4.4 Implementation of a Worst-Case Resource-Usage Tool including a Worklist Solver

Implemented a worst-case resource-usage tool named Fulgurite comprising:

- a worklist solver for the extended control-flow analysis of Chapter 4;
- automatic generation of an integer-linear programming problem for each applet lifecycle for each registered applet in the program under consideration. Currently the integer-linear programming problems are written out in the format required by the open source mixed-integer linear-programming solver LPSolve[6];
- automatic generation of .DOT graphs for all analysed applets showing their control-flow.

Due to the the nature of the *Java Card* virtual machine, most of the code has been written from scratch. For example, in a normal Java virtual machine, values are sign-extended to integer before being pushed on the stack, whereas in a *Java Card* virtual machine, values are sign-extended to short. The security framework particular to *Java Card* – the applet firewall – has to be integrated with the bytecode instructions. Hence it was not possible to use existing Java bytecode program analysis tools like Soot[7] or the program analysis part of TJ Watson libraries[8] for static analysis, or at least, not as-is. We still utilise the latter tool for its graphing abilities and ability to calculate dominators in the determination of natural loops.

[6] http://lpsolve.sourceforge.net/5.5/
[7] https://sable.github.io/soot/
[8] http://wala.sourceforge.net/wiki/index.php/Main$_{P}age$

Chapter 2

Background Material on Smartcards and *Java Card* Classic Edition 3

2.1 Overview

The technologies under consideration in this thesis are non-standard and specialised (namely, smartcards and the *Java Card* Classic Edition 3) and so a general introduction to these technologies is included for completeness. For simplicity and brevity, the choice of the method of presentation for this general introduction may be said to be question-and-answer or frequently-asked-questions. The material in this chapter is based principally on [Ora11b, Ora11a, Ora11c, Che00].

2.2 Smartcards

2.2.1 What is a smartcard? In what areas are they used?

Usually the size of a credit card, and typically exhibiting a metal contact strip similar to that shown in Figure 2.1, a smartcard has either or both on-card processing power and memory, and is capable of participating in input/output (I/O) behaviour.

In particular, simple magnetic stripe cards – which do not feature a metal contact strip – are *not* smartcards, since these require access to a remote information system *at the time* a transaction is attempted.

Smartcards are predominantly used in the following areas:

- banking;
- wireless telecommunications (including mobile 'phone SIM cards);
- access control (both building access control and software access control via smartcard authentication);
- public transport tokens

and any other applications where data security/privacy is of paramount importance.

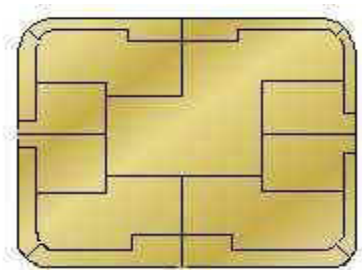


Figure 2.1: Typical appearance of the contact-strip on a smartcard

2.2.2 What are the governing standards regarding smartcards?

From the point of view of this thesis, the most important smartcard standards is the family of international standards ISO-7816 "Identification cards – Integrated circuit cards with contacts", which details the standards for smartcards with metal contact strips. ISO-7816 has 7 parts:

1. Physical characteristics;
2. Dimensions and locations of the contacts;
3. Electronic signals and transmission protocols;
4. Inter-industry commands for exchange;
5. Application identifiers;
6. Inter-industry data elements;
7. Inter-industry command for Structured Card Query Language (SCQL);

Other important smartcard standards can be found in Section 2.7.1 of [Che00].

2.2.3 What is the protocol for communicating with smartcards?

The normal mode of communication with smartcards is as follows. A smartcard is inserted into a card-acceptance device (CAD), usually connected to a computer, that is capable of exchanging data packets with the smartcard. The terminal, connected to the card-acceptance device, issues a command to the smartcard and the smartcard processes the command and then responds with a reply.

The format of the terminal request and smartcard reply is known as APDU (application protocol data unit) and is a protocol standard, like TCP/IP. We omit the technical details here due to space considerations. As a standard, information on the format of the APDU command and reply messages is well-published, and may be found e.g. in Section 2.4.1 of [Che00]. Other than the existence of the standard format for message exchange, communication between a smartcard and a terminal has two noteworthy features: firstly, communication is half-duplexed i.e. only one side at a time can send a message; secondly, **every** terminal request is responded to and a reply returned by the smartcard.

The APDU standard for message exchange between smartcard and terminal is the same regardless of whether the smartcard is a contact card or a contactless card (see Section 2.2.5 for further details).

2.2.4 What are the benefits of smartcards?

Summarising Section 1.1.2 of [Che00], smartcards provide four major benefits:

1. *Processing power*. Depending on the particular kind of smartcard under consideration, a smartcard may be capable of only very simple behaviours (such as reducing the stored value on a pre-paid 'phonecard) or have a more general-purpose processor capable of executing more sophisticated applications;

2. *Security*. Physical possession of the smartcard, a knowledge of specialised technologies, and access to specialised hardware is necessary to mount an attack on a smartcard. Additionally, that a smartcard's processor, memory and I/O facilities are tightly packaged into a single integrated circuit makes it even less vulnerable to attack;

3. *Portability*. Smartcards can be carried on the person of the card holder, and access to the services of, and data on, the smartcard can be accessed when needed. This is of particular importance for identity cards, smartcards with medical data, and cards holding cryptographic keys (typically used to verify an individual's digital identity);

4. *Ease of use*. When access to a card's data or services is required, the smartcard is inserted into a card-acceptance device, and is removed from the card-acceptance device when transactions with the smartcard have been completed.

2.2.5 What different kinds of smartcards are there? What are their typical applications?

Following Section 2.2 of [Che00], smartcards can be distinguished on two bases.

Firstly, whether they are memory cards or microprocessor cards:

- *Memory cards*. Memory cards do not have a processor. They have either memory or memory and non-programmable logic. Typical applications are prepayment cards e.g. for use of the public payphone network. Memory cards can only execute preprogrammed operations, such as debiting the stored value on the card. More sophisticated memory cards can additionally detect and reject attempts to compromise the card e.g. by preventing the stored value in a card from being increased;
- *Microprocessor cards*. As indicated by the name, microprocessor smartcards have at least one processor on-card. By virtue of the microprocessor, the card is capable of providing general functionality (including security functions) and serving multiple applications; indeed, the computational capabilities of the smartcard are limited only by the processor and available memory resources. Plus the microprocessor separates applications from data and memory and so are well-suited for applications that require data security.

Secondly, whether the card is a contact card or a contactless card:

- *Contact cards*. Contact cards must be placed– or inserted– into a card-acceptance device. Contact cards exhibit a metal contact strip and communicate with card-acceptance devices using its serial connection. Typical applications include banking, telecommunications and viewer subscription cards for satellite television channels. Note that the SIM card inside (GSM) mobile 'phones is usually a microprocessor contact card, whereas payphone cards are memory contact cards;
- *Contactless cards*. Contactless cards use electromagnetic fields to communicate with card-acceptance devices. Typical applications include public transport systems and access control to buildings.

To help avoid unnecessary and potentially tedious and repetitious distinctions in this document, we assume smartcards are contact cards and talk of inserting them into card-acceptance devices. Wherever such a statement occurs, it may equally well be said that if the smartcard is a contactless card, that it is brought within range of an appropriate proximity-coupling device.

2.2.6 What is the typical memory model of smartcards?

Per Section 2.3.4 of [Che00]. smartcards usually have three different types of memory on-card:

- Persistent *immutable* memory – typically implemented in Read-Only Memory (ROM) – housing the smartcard operating system, utilities and other programs or data that are permanent and not capable of being upgraded. This type of memory does not need electricity to retain data across card sessions;
- Persistent *mutable* memory – typically implemented in electrical erasable programmable read-only memory (EEPROM) – housing the applications data that is meant to persist across card sessions and which applications are capable of, and expect to, change over time. This type of memory does not need electricity to retain data across card sessions. Reading from EEPROM is as fast as reading from ROM, but writing to it is approximately a thousand times slower, and there is an upper limit to the number of times EEPROM may reliably be written to;
- Non-persistent working memory – typically implemented in Random Access Memory (RAM) – used as scratch space for computation and temporary working space for storing and modifying data. RAM is non-persistent memory: information is not retained across card sessions and is lost when the power to the smartcard ends. There are no limits to the number of times RAM may be written to;

2.3 *Java Card* Platform

2.3.1 What is a *Java Card* smartcard? What different kinds of smartcards are there? What are their typical applications?

A *Java Card* smartcard is a microprocessor smartcard including an implementation of the *Java Card* platform (see Section 2.3.3). In the remainder of this chapter, general references to smartcards are always references to *Java Card* smartcards.

Java Card smartcards may be contacted (in which case a card session may be initiated when the card is inserted into a card-acceptance device (CAD)) or contactless (in which case a card session may be initiated when the card comes within range of a proximity coupling device (PCD)). In the interest of brevity, whether the particular card is contacted or contactless, we shall refer to card acceptance devices (CADs), and we shall refer to “card tear” to describe when a contacted card is removed abruptly during a card session or when a contactless card moves out of range of the proximity coupling device during a card session (or the wireless connection is otherwise dropped

during a contactless card session).

As explained in Section 2.2.5, as a microprocessor smartcard, typical applications (called applets) that might be registered with *Java Card* smartcards include:

- banking;
- wireless telecommunications (including mobile 'phone SIM cards);
- access control (both building access control and software access control via smartcard authentication);
- public transport tokens

and any other applications where data security/privacy is of paramount importance.

One of the most attractive qualities of the *Java Card* platform is the potential for dynamic download, deletion and upgrade of registered applets on *Java Card* smartcards. Although not common, it is possible for a *Java Card* smartcard not to allow update to the smartcard, and so its functions and value can be fixed.

Applets follow a lifecycle as explained in Section 2.3.3.7 and each applet is uniquely identified on a smartcard by its application identifier. An Application IDentifier (AID) is defined in ISO 7816-5 to be a sequence of bytes between 5 and 16 bytes in length, of which the first five bytes are known as the Resource IDentifier (RID) and is used to identify the applet vendor; the remaining bytes are known as the Proprietary Identifier eXtension (PIX).

Applets registered on a smartcard remain in a suspended state until selected by the CAD. At most one applet at a time on a smartcard can be currently selected and capable of exchanging messages with the CAD. Section 2.3.3.6 explains more.

As well as being used to uniquely identify an applet on a smartcard, AIDs play a central role in *Java Card* security as explained in Section 2.3.3.11.

2.3.2 What is the protocol for communicating with *Java Card* smartcards? What is *Java Card* Remote Method Invocation?

The canonical protocol for communicating with *Java Card* smartcards is the canonical smartcard communication protocol Application Protocol Data Unit (APDU) outlined in Section 2.2.3. Support for this protocol is enshrined in

the standard *Java Card* developer libraries (the *Java Card API*[1]) e.g. in the `APDU` class which provides access to the components of the APDU commands and replies, as defined in ISO-7816. Note that regardless of whether the smartcard is a contacted card or contactless card, APDU is the canonical protocol for communicating with a *Java Card* smartcard.

Since *Java Card* platform 2.2 an alternative, object-centric, protocol for communicating between *Java Card* smartcards and card-acceptance-device-side Java applications based on Java remote method invocation, known as *Java Card* Remote Method Invocation, is available. This is discussed in Chapter 8 of [Ora11b] and not any further here, because whether it is APDUs or *Java Card* RMI, the *same* applet lifecycle methods are invoked on the smartcard.

2.3.3 What is the *Java Card* platform? What are the governing standards regarding *Java Card* smartcards?

Roughly speaking, the *Java Card* platform is a scaled-down version of the Java Standard Edition for smartcards and other highly resource-constrained devices, whose feature set has been carefully chosen to be useful for writing typical applications for such devices.

Properly speaking, the current *Java Card* Platform 3 Classic Edition is formally defined by three natural language specifications: the *Java Card* Runtime Environment (JCRE) specification [Ora11b]; the *Java Card* API (JCAPI) specification [Ora11a]; and the *Java Card* Virtual Machine (JCVM) specification [Ora11c].

As an indication of the success or rate of adoption of *Java Card* by the smartcard community, Oracle announced in 2015 that over 12 billion *Java Card* smartcards have been deployed in the last fifteen years. Hence there is potentially great commercial advantage in developing software to analyse interesting properties and behaviours of *Java Card* applets. Security properties and resource-consumption demands are likely to be of particular interest. The most appropriate basis for validating/verifying such software is formal methods. Prominent on the list of the reasons we prefer and advocate program analysis for the choice of formal method is the high degree of automation typical of software tools implementing static analyses.

The intention of this section is two-fold: firstly, to present a broad overview of the *Java Card* platform; and secondly, to include sufficient detail in this broad overview to make it unnecessary to look elsewhere for a basic understanding

[1] See Section 2.3.3.2

of the *Java Card* platform. Resources [Ora11b, Ora11a, Ora11c, Che00] are prime starting points for further information.

2.3.3.1 The *Java Card* Runtime Environment (JCRE)

Defined by the *Java Card* Runtime Environment specification [Ora11b], the JCRE is in essence the operating system of the *Java Card* smartcard. Key responsibilities of the JCRE include: resource-management; provision of entry-point methods[2] to allow unprivileged (user) applets to invoke (privileged) system calls; receiving and reacting to all commands received by the *Java Card* smartcard (typically by invoking the JCVM).

2.3.3.2 The *Java Card* API (JCAPI)

Defined by the *Java Card* API (JCAPI) specification [Ora11a], the JCAPI specification details the standard user libraries available to *Java Card* applet developers, providing the usual Java abstraction away from the particular underlying hardware (here smartcard technologies).

2.3.3.3 The *Java Card* Virtual Machine (JCVM)

Defined by the *Java Card* Virtual Machine specification [Ora11c], the JCVM defines the instruction set (bytecode instructions) of the *Java Card* virtual machine. Further details on the features supported by the current JCVM specification are given below.A fundamental and striking difference between JCVMs and standard Java virtual machines is that the JCVM appears to run forever. When power is restored to a *Java Card* smartcard, the JCVM recovers from its persistent memory the persistent object heap – the JCVM is (essentially) suspended between card sessions (i.e. sessions where APDUs are exchanged between the smartcard and the terminal). (See Section 2.3.3.8 for further details on this point).

2.3.3.4 Java Features supported in *Java Card*

Section 2.2 of [Ora11c] defines the subset of Java supported by the *Java Card* platform. Key details are given below.

Based on Section 2.2 of [Ora11c], we consolidate in Table 2.1 the Java features that are and are not supported by *Java Card* platform 3 Classic Edition.

[2] See Sections 2.3.3.11 to 2.3.3.13 for further details.

Package access control in *Java Card* is essentially the same as in Java. There are seven specific package access patterns that are valid in Java that are not supported in *Java Card*. These are given in Section 2.2.1.1 of [Ora11c].

To compensate for the lack of security managers, *Java Card* has its own security architecture known as the applet firewall. Details of the applet firewall are given below in Section 2.3.3.11.

Supported Java Features	**Unsupported Java Features**
Small primitive data types: `boolean, byte, short`	Other primitive data types: `char, long, double, float`
One-dimensional arrays	Multi-dimensional arrays
Java packages, classes, interfaces and exceptions	Garbage collection and finalization
Java object-oriented features: inheritance, virtual methods, overloading and dynamic object creation, access scope and binding rules.	Characters and strings
The `int` keyword and 32-bit integer data type support are optional	Security managers
Generics	Dynamic class loading
Annotations	Security managers
Static import	Threads
	Finalization
	Typesafe Enums
	Enhanced For loop
	Varags
	Assertions
	Object cloning

Table 2.1: Java Features supported in *Java Card* 3 Classic Edition

Java Card Keywords

Tables 2.2 and 2.3 reproduce from Section 2.2.2.2 of [Ora11c] the Java keywords that are, and are not, respectively, supported by the *Java Card* platform. For those Java keywords that are supported in *Java Card* keywords: "Their use is the same as in the Java programming language".

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	else	import	public	throws
byte	extends	instanceof	return	try
case	final	int	short	void
catch	finally	interface	static	while
class	for	new	super	
continue	goto	package	switch	

Table 2.2: Java Keywords supported in *Java Card* 3 Classic Edition

native	synchronized	transient	assert
volatile	strictfp	enum	

Table 2.3: Java Keywords NOT supported in *Java Card* 3 Classic Edition

Java Card JCVM instruction set (bytecode instructions)

Table 2.4 lists the Java bytecode instructions supported by the *Java Card* virtual machine.

2.3.3.5 Split Architecture

The computing power and resources on the current generation of smartcards falls far short of that required to perform the functions typical of standard Java development and deployment activities. For example, *standard* bytecode verification is too resource-intensive for a typical smartcard.

The solution to this problem is a split-architecture where more computationally intensive activities take place off-card and demands on the smartcard are kept to a minimum. Of course, the on-card JCVM includes a *Java Card* bytecode interpreter that can interpret the bytecode instructions of Table 2.4. One of the key responsibilities of the JCVM bytecode interpreter is to enforce the applet firewall (see below). As per Section 6.1 of [Ora11b]: "Applet firewalls are always enforced in the *Java Card* VM. They allow the VM to automatically perform additional security checks at runtime".

nop	aconst_null	iconst_<i>	bipush
sipush	ldc	ldc_w	iload
aload	iload_<n>	aload_<n>	iaload
aaload	baload	saload	istore
astore	istore_<n>	astore_<n>	iastore
aastore	bastore	sastore	pop
pop2	dup	dup_x1	dup_x2
dup2	dup2_x1	dup2_x2	swap
iadd	isub	imul	idiv
irem	ineg	ior	ishl
ishr	iushr	iand	ixor
iinc	i2b	i2s	if<cond>
ificmp_<cond>	ifacmp_<cond>	goto	jsr
ret	tableswitch	lookupswitch	ireturn
areturn	return	getstatic	putstatic
getfield	putfield	invokevirtual	invokespecial
invokestatic	invokeinterface	new	newarray
anewarray	arraylength	athrow	checkcast
instanceof	wide	ifnull	ifnonnull

Table 2.4: Java bytecode instructions supported in *Java Card* 3 Classic Edition

2.3.3.6 Logical Channels

Java Card platform 3 provides support for logical channels as defined in *ISO 7816-4:2013 Specification*. This allows up to twenty sessions to be opened up into the smart card per (contacted and contactless) I/O interface and have an applet selected and ready to receive commands on each channel.

Java Card classic applets written to take advantage of logical channels[3] can be selected concurrently on different logical channels, or selected alongside other applets on different logical channels. The JCRE will ensure applets written prior to logical channels support will be handled correctly i.e. there must be at most one applet from a package active on at most one logical channel.

Note that only at most one applet at a time is designated as the currently active applet and only the currently active applet may receive–, process- and reply to– non-SELECT FILE and non-MANAGE CHANNEL APDU commands from the card terminal.

To be absolutely clear, under all Classic editions of *Java Card* platforms, *Java Card* Classic is a single-threaded environment.

[3] This is signalled by implementing the methods of the `javacard.framework.MultiSelectable` interface.

2.3.3.7 Applet Lifecycle

In a similar way to, for example, Java web applets and Enterprise Java Beans, card applets extend a base abstract class `Applet`, which defines a set of lifecycle methods which the JCRE invokes appropriately in response to command APDUs received from the terminal connected to a card-acceptance device. These include `select` methods to inform an applet that a request has been made to make it the currently selected applet on a particular logical channel, a `process` method which is used to process each command APDU received and returns to the JCRE the result of the operation (i.e. the reason code – like HTTP servers, smartcards return numeric status words to indicate the success of an operation, along with any data to return to the terminal), and `deselect` methods to inform the currently selected applet on a particular logical channel that a card-acceptance device session is complete, or that another applet is to be selected to run on that logical channel.

2.3.3.8 What is the typical memory model of *Java Card* smartcards?

Specialising Section 2.2.6 in the light of Section 4.1 of [Che00]: *Java Card* smartcards usually have three different types of memory on-card:

- Persistent *immutable* memory – typically implemented in Read-Only Memory (ROM) – housing the smartcard operating system **and the *Java Card* platform including JCRE, JCVM and JCAPI, including any fixed applets**, utilities and other programs or data that are permanent and not capable of being upgraded. This type of memory does not need electricity to retain data across card sessions;
- Persistent *mutable* memory – typically implemented in electrical erasable programmable read-only memory (EEPROM) – housing the applications data **and *Java Card* applets' data and any downloaded *Java Card* applets, and any objects allocated when an object is created using the `new` command** that is meant to persist across card sessions and which applications are capable of, and expect to, change over time. This type of memory does not need electricity to retain data across card sessions. Reading from EEPROM is as fast as reading from ROM, but writing to it is approximately a thousand times slower, and there is an upper limit to the number of times EEPROM may reliably be written to;
- Non-persistent working memory – typically implemented in Random Access Memory (RAM) – used as scratch space for computation and temporary working space for storing and modifying data, **including run-time *Java Card* operand stack, local variables and results from card specific native cryptographic functions**. RAM is non-persistent memory: information is not retained across card sessions and is lost when the power to the smartcard ends. There are no limits to the number of time RAM may be written to;

In *Java Card*, all class instances are *persistent* and arrays may be *persistent* or *transient*. The former type of memory, *persistent* memory, typically using EEPROM, is allocated when an object is created using the `new` command. This enables persistent objects to hold state across card-acceptance-device sessions. The latter type, *transient* memory has nothing to do with the keyword `transient` in standard Java (and which is not a valid keyword in *Java Card*). Here transient memory means that the value of each array element is cleared to its default values (of `null` for object types and zero for primitive numeric types) depending on what card event occurs:

- the card is reset (known as `CLEAR_ON_RESET`);
- the currently selected applet is deselected or an applet from another group context than the currently selected applet has been selected to be the currently selected applet (known as `CLEAR_ON_DESELECT`)

Additionally, if a card suffers card tear, this causes elements of transient arrays to be cleared to their default value.

The APDU buffer and the byte array input parameter to the Applet install method are both *transient* objects. A user applet may call the `JCSystem.makeTransientXXXArray` methods to create a dynamic transient array.

2.3.3.9 Transaction Model

The *Java Card* platform provides a transactional model to enable groups of operations to *persistent*[4] object fields to be treated as an atomic operation. If an error occurs during a transaction (whether internal or external by e.g. a card tear), the next time the smartcard is inserted into a card-acceptance device, the state of persistent objects is restored to its previous state prior to the smartcard being ready to receive APDUs.

The JCRE maintains a commit buffer in which is recorded provisional state changes to *persistent* objects since the current transaction was successfully started. Reads from fields in the commit buffer return the provisionally updated values. When the current transaction is successfully committed, state changes in the commit buffer are persisted. The JCAPI `JCSystem` provides the following static methods which behave in the obvious way: `abortTransaction()`, `beginTransaction()`, `commitTransaction()`, `getMaxCommitCapacity()`, `getTransactionDepth()`, `getUnusedCommitCapacity()`.

Due to resource limitations, nested transactions are not possible and an attempt to start a fresh transaction when one is already underway causes an exception to be thrown. Similarly, attempts to commit a transaction when none is underway causes an exception to be thrown. Also once an applet's lifecycle method has terminated (whether normally or by generating an exception), and control is returned to the JCRE, if a transaction is currently underway,

[4] See Section 2.3.3.8 for an explanation of persistent and transient objects.

the JCRE aborts it automatically.

The transaction mechanism in *Java Card* is deceptively simple. While it is true to say the transaction mechanism of the Classic editions allows only a single level of transaction (no nested transactions), it is *not* the case that within an active transaction all changes to *all* object fields are made to a commit buffer and then persisted only when the transaction is committed as might be expected. Similarly, if an active transaction is aborted, it is not the case that changes made to the persistent memory (not the commit buffer) during the span of the transaction will be rolled back, as might be expected. As per Sections 7.6.3 and 7.7 of [Ora11b]:

> "Object instances created during the transaction that is being aborted can be deleted only if references to these deleted objects can no longer be used to access these objects. The *Java Card* RE shall ensure that a reference to an object created during the aborted transaction is equivalent to a `null` reference. Alternatively, programmatic abortion after creating objects within the transaction can be deemed to be a programming error. When this occurs, the *Java Card* RE may, to ensure the security of the card and to avoid heap space loss, lock up the card session to force tear or reset processing".

> "Only updates to persistent objects participate in the transaction. Updates to transient objects and global arrays are never undone, regardless of whether or not they were "inside a transaction.""

We adopt the former approach of invalidating/treating as `null` object references created within a transaction that is then aborted. The semantic transitions (and thus the program analyses based on them) will then be a superset of the semantics transitions possible by locking up the card and forcing tear or reset processing.

The rationale for transient[5] objects not participating in transaction appears to be for security: consider a banking applet which uses a Personal Identification Number (PIN) which gives the user a maximum number of tries to enter the correct PIN before locking the applet or card. Keeping track of the number of failed PIN entries must not be subject to transaction semantics, or an attacker could brute-force attack the application until access is gained by first initiating a transaction, then trying an all-zeroes PIN, aborting the transaction if the PIN authentication failed, then retrying with the next number in sequence, and repeating this process until access is gained. It is reported the reference implementation of the Sun *Java Card* Development Kit 2.0 contained such a defect for `OwnerPIN`, which was corrected in the reference implementation of the Sun *Java Card* Development Kit 2.1.1 [HMP06].

[5] See Section 2.3.3.8 for differences between permanent and transient memory.

2.3.3.10 Applet Development Cycle and the CAP File Format

The *Java Card* application (i.e. applet) development cycle is different from the normal Java development cycle in two regards: firstly, the keywords of the Java source files must come from Table 2.2; secondly, the unit of installation on Java smartcards is the package (in the normal Java sense) in a particular format, the compressed applet (CAP) format. The exact format of the CAP file can be found in Chapter 6 of [Ora11c]. We omit the details here.

In particular, the normal applet development cycle is to develop Java applications using the keywords of Table 2.2 in a traditional computing environment (e.g. a desktop PC) where all the source (non-JCAPI, non-library packages *known* to be present on the target smartcard(s)) Java files are defined in the same package. At that point, the normal development cycle is followed e.g. design, implementation and testing. From the source Java files, normal Java classes are output. Once the developer is satisfied with the behaviour of the applet(s) and supporting classes, a piece of off-card software called the *converter* transforms the classes into a CAP file in the prescribed format. Another special piece of off-card software called the *installer* is responsible for transmitting and registering the CAP file with the JCRE on the smartcard. [The installation process is card– and card-vendor– specific and this process is not specified by the *Java Card* platform].

2.3.3.11 Applet Firewall and Applet Selection

In the absence of security managers and class loaders, and to preclude the possibility that malicious applets may be able to exploit unintentionally lax access modifier attributes in other applets, the *Java Card* platform defines its own security architecture.

As noted above, the unit of installation on the *Java Card* smartcard is the package. For compatibility with ISO-7816, the familiar Java package naming convention is mapped to (numeric) application identifiers (AIDs). An Application IDentifier (AID) is defined in ISO 7816-5 to be a sequence of bytes between 5 and 16 bytes in length, of which the first five bytes are known as the Resource IDentifier (RID) and is used to identify the applet vendor; the remaining bytes are known as the Proprietary Identifier eXtension (PIX). [AIDs and contexts are used synonymously].

When CAP files (i.e. JCVM bytecode in a particular format) are installed on the *Java Card* smartcard, each applet in the CAP file is assigned a unique AID and a shared (group) AID, both of which must not be currently registered on the particular smartcard prior to installation. When the client application wants to select a particular applet on the *Java Card* as the currently active applet, it sends a SELECT FILE APDU or MANAGE CHANNEL OPEN command to the smartcard with the AID of the desired applet and assuming the smartcard accepts the selection

request, the applet identified by that AID is then set to receive subsequent applet-specific (non-SELECT FILE, non-MANAGE CHANNEL APDU) commands, until some other applet is selected or the card is removed from the card-acceptance device.

Broadly speaking, the applet firewall operates on the basis of group AIDs/contexts. When a request is made to invoke a method on, or otherwise access, an object, a runtime check is made to ensure that the currently active applet has the same group context as the object being accessed or invoked, or that the object being accessed is JCRE-owned and the currently active applet is invoking or accessing a permitted object or method. If the group contexts do not match, or if an improper attempt is made to access a JCRE method or object, then the JCVM throws an exception. Section 3.8 codifies the applet firewall rules via a set of security predicates which apply depending on the bytecode instruction being executed.

Every object o in the *Java Card* object system has an owning context comprised of two AIDs – for user applets, the AID of the applet app that was active at the point of o's creation, and the AID of the group context that was active at the point of o's creation i.e. app's group context. For resource conservation reasons, and to control user applet access to system resources, some objects are owned by the JCRE and different firewall rules apply in these scenarios – these are discussed in Sections 2.3.3.12 to 2.3.3.15.

2.3.3.12 Object access across contexts – JCRE Entry Points Objects (temporary)

The APDU object and all JCRE owned exception objects are examples of temporary JCRE Entry Point objects, and its methods may be invoked from any context.

The JCRE (more specifically the JCVM) throws a `SecurityException` if a user applet attempts to store a reference to a temporary JCRE Entry Point object in instance variables, or array elements, or class variables.

2.3.3.13 Object access across contexts – JCRE Entry Points (permanent)

When an applet is installed on a *Java Card*, and more particularly when it is registered on a *Java Card*, it does so with a particular AID. If the registration is successful, the JCRE creates a JCRE-owned instance of the corresponding `AID` class, and adds them to the applet registry. These JCRE owned `AID` instances are examples of permanent JCRE Entry Point objects and its methods may be invoked from any context, typically to identify a particular applet on the card from which to request a `Shareable` service as explained in Section 2.3.3.15 below.

A user applet is allowed to store and reuse references to permanent JCRE Entry Point objects in instance variables, or array elements, or class variables.

2.3.3.14 Object access across contexts – Global Arrays

The APDU buffer and the byte array input parameter to the Applet install method are both examples of *global arrays* and are accessible from any context. A user applet may call the `JCSystem.makeGlobalArray()` methods to create a dynamic global array.

Section 6.2.2 of [Ora11b] states that global arrays are temporary JCRE entry points. The JCRE (more specifically the JCVM) throws a `SecurityException` if a user applet attempts to store a reference to a temporary JCRE Entry Point object in instance variables, or array elements, or class variables.

2.3.3.15 Object access across contexts – `Shareable` Objects

The applet firewall allows no communication between applets in different packages (i.e. applets with different group contexts). This may be considered too restrictive for classes of useful applets e.g. loyalty-card style applets. The *Java Card* platform defines a tagging interface `Shareable` to allow controlled access to an applet's methods (but never directly its fields). In a similar way to standard Java remote method invocation, an applet that wishes to make services available to other clients defines a public interface extending `Shareable`, say `S1`, and then declares that it implements the particular interface `S1`. It then publishes the `Shareable` reference to other applets by over-riding the empty method `getShareableInterfaceObject(AID, byte)` defined in the `Applet` class, the base class which all applets must extend. The parameters to this method allows an applet to determine what – if any – `Shareable` references it wishes to return to the applet requesting the `Shareable` reference. [The JCRE automatically maps the request for a `Shareable` reference, which must specify the AID of the applet providing the desired services in the invoking applet, to the AID of the requesting applet AIDs in the invoked applet. This allows an applet providing shareable services to control to which applets it is willing to offer shareable object references, and allows different shareable services to be offered to different requesting applets. This automatic mapping is part of the applet (individual– and group– AID) context switch that occurs when the JCRE manages the request for shareable objects, since only the JCRE can bypass the applet firewall and access any field or object it wishes].

Note that this mechanism for returning `Shareable` references is not sufficiently strong in itself to prevent unauthorised access to `Shareable` services. Suppose an applet A returns to applet B a shareable reference it would not return to applet C. Nothing stops applet B from publishing to applet C the shareable object that A returned to

it. Hence the JCRE specification advises, (in Chapter 6 of [Ora11b]), that at *each* attempt to access a shareable method it authenticates the invoking client to guard against improper service access. At the very least, the applet providing shareable services can use a JCAPI method to determine the AID of the invoking applet. Of course, much more sophisticated authentication strategies are possible and appropriate for applets managing more sensitive data, such as applets implementing an electronic purse or wallet.

2.3.4 What are the benefits of *Java Card* smartcards?

The *Java Card* platform brings to the smartcard arena many of the benefits enjoyed in more traditional computing environments by Java. Reproducing from Section 1.3.1 of [Che00], the benefits of *Java Card* technology include:

1. *Ease of application development* – The Java language brings smart card programming into the mainstream of software development, relieving developers from going through the swamps of microprocessor programming, such as programming in 6805 and 8051 assembly languages. . . . The (*Java Card*) platform encapsulates the underlying complexity and details of the smartcard system. Applet developers work with the high-level programming interfaces. They can concentrate most of their effort on the details of the application and leverage extensions and libraries that others have written;

2. *Security* – Security is always of paramount concern when working with smartcards. Java's built-in security features fit in well with the smartcard environment. For example, the level of access to all methods and variables is strictly controlled and there is no way to forge pointers to enable malicious programs to snoop around inside memory. In addition, applets on the *Java Card* platform are separated by the applet firewall. This way the system can safeguard against an application's attempts to damage other parts of the system;

3. *Hardware independence* – *Java Card* technology is independent of the type of hardware used. It can run on any smart card platform. Applets are written on top of the *Java Card* platform and hence are smartcard hardware independent. Ready-to-use applets can be loaded into any *Java Card* smartcard without recompilation;

4. *Ability to store and manage multiple applications* – A Java smartcard can host multiple applets, such as an electronic purse, authentication, loyalty and health care program, from different service providers. Because of the the *Java Card* applet firewall, applets are not allowed to access each other unless explicitly permitted to do so. . . More applets can be downloaded to the card. A Java smartcard's functionality can be continually upgraded with new or updated applets, without the need for issuing a new or different card;

5. *Compatibility with existing smartcard standards* – *Java Card* technology is based on the smartcard international standard ISO-7816, so it can easily support smartcard systems and applications that are generally compatible with ISO 7816.

Chapter 3

The Carmel programming language

3.1 Introduction

This thesis is firmly rooted in the SecSafe project[1] [HSN+03, Pro02], a successfully completed EC-funded investigation whose main focus was on the development of static techniques to analyse security properties of realistic languages. Such analyses may be used as the formal basis for obtaining security certification at the higher levels of the (US, Canadian and European) Common Criteria (for evaluation of computer security) [The06]. A substantial proportion of the workload of, and effort expended during, the SecSafe project, relates to the *Java Card* platform. In particular, this thesis is firmly rooted in the *Java Card* aspects of the SecSafe project. For the reader unfamiliar with smartcards and/or the *Java Card* Classic Edition 3, Chapter 2 presents an introduction to these technologies.

In the context of this thesis, our principal interest in revisiting the SecSafe material is to evaluate its fitness for facilitating loop-bounds calculations and detecting recursion[2]. We also take the opportunity to update the operational semantics and program analyses to *Java Card* Classic Edition 3; the version considered by SecSafe is *Java Card* 2.1. We have identified the following ways in which we will extend the SecSafe material for our purposes:

- Enable the identification of local variables being used as loop variables in `if` bytecode instructions to allow simple loops to be bounded by our program analyses. The technical means by which we achieve this is to extend the runtime value of Carmel values to include the address at which a value was generated or most recently saved in a manner consistent with a *reaching definitions for local variables analysis*[3] developed from

[1] Project IST-1999-29075, *Secure and Safe Systems based on Static Analysis*, funded by the European Community under the "Information Society Technologies" Programme (1998-2002)

[2] As outlined in Section 1.1, and as explained more fully in Chapter 6, the hardest part of determining worst-case resource-usage is determining loop-bounds, and we must ensure there are no recursive calls reachable from any of the applet lifecycle methods.

[3] See Appendix C for details of the reaching definitions for local variables analysis and classical reaching definitions analysis.

the classical *reaching definitions analysis*. The correctness of this approach is shown in two phases:

  - firstly, we prove using structural induction and representation functions that the base control-flow analysis of Chapter 4 correctly captures (i.e. over-approximates the set of) the runtime values of the operational semantics (including the address at which each value was generated or most recently saved) given in this Chapter;
  - secondly, we prove using the reaching definitions for local variables analysis and operational semantics that the base control-flow analysis clauses for the `if` bytecode instructions may be split into cases in the extended control-flow analysis of Chapter 5 and so we propagate only (a possible over-approximation of the set of) semantically possible values of loop variables to the appropriate branches of the `if` statements;

- In any program analysis, record at each program point the *set* of all method names in the call-stack and propagate these to all successors of the program point – this will make detecting recursive calls straightforward at method invocation bytecode instructions;
- Integrate the full semantics of the transaction mechanism in the atomic bytecode instructions;
- Parameterise the applet firewall security predicates for use in the operational semantics and the abstract program analyses;
- Restructure the operational semantics to be much closer in form to the program analyses and so
  - simplify the proofs;
  - amplify the correspondence of the operational semantics and the program analyses;
- all runtime values have explicit types and we support the optional 32-bit `int`eger in our program analyses as well as our operational semantics;
- specify the operational semantics for the *Java Card API* and corresponding program analysis clauses.

In this chapter, we present an operational semantics for the programming language Carmel, which models the *Java Card Virtual Machine* language, based principally on [Ora11a, Ora11b, Ora11c, HS05, Siv04, SH01, Che00] and explicitly represent the state of the transaction mechanism and the state of the I/O APDU buffer[4] as part of the JCVM machine state. More specifically, in this chapter we present an operational semantics for the Carmel form

[4] In *Java Card* Classic Edition 3, an applet must order its input/output behaviour with the card-acceptance device (CAD) via the APDU buffer by first (optionally) signalling it wants to read from the APDU buffer and then reading and then (optionally) signalling it wants to return data to the CAD and then writing to the APDU buffer. If the APDU buffer is set for writing, any attempt to read from it will cause an exception to be raised, as will signalling the intention to read from the APDU buffer. The states of the APDU buffer, and their valid transitions, are well-specified in [Ora11b].

of the bytecode instructions in the *Java Card Virtual Machine* specification [Ora11c] and define in Appendix A the operational semantics for the *Java Card API* methods specified in [Ora11a]. We use the *instruction set* and extend the *program structures* from [SH01] and define a small-step relation between program configurations, including rules for exception handling and subroutines. We also include the structures needed to model object ownership, the Java Card firewall and transactional state.

3.1.1 Notation

Format of the operational semantics The format of the operational semantics presented in this thesis is non-standard. Firstly, we use whitespace and indentation to indicate scope and to avoid excessive bracketing and conjunctions and so to improve clarity. Our semantics have the form:

$$\frac{\begin{array}{l} P_1 \\ \vdots \\ P_n \end{array}}{\begin{array}{l} {\scriptstyle P}\big| \; Q_1 \;\Rightarrow \\ \quad\; Q_2 \end{array}}$$

which may be read as meaning in a Carmel Program P, given a machine state of Q_1, then when $P_1 \ldots P_n$ are true, the machine state transitions to Q_2. Notationally, this is equivalent to:

$$\frac{P_1 \;\wedge \ldots \wedge P_n}{Q_1 \Rightarrow Q_2}$$

which is the more familiar form for describing how and when state transitions occur in operational semantics. Note that some of the P_i may be conditional statements of the form:

$$\begin{array}{lll} P_i & \Rightarrow & P_f \\ & & \vdots \\ & & P_w \end{array}$$

which is equivalent to the more familiar $\big(P_i \;\Rightarrow\; (P_f \;\wedge \ldots \wedge P_w)\big)$. This is merely a notational convenience.

Records. Some new domains are defined as records. Records are defined using the following notation:

$$\text{Dom} = (f_1 : \text{Dom}_1) \times ... \times (f_n : \text{Dom}_n)$$

Access to field f_i in $e \in \text{Dom}$ is written as $e.f_i$. Field update is written as $e.[f_i \mapsto f]$, where $f \in \text{Dom}_i$.

Conjunctions and Disjunctions. Conjunctions and disjunctions are grouped using parentheses and curly braces respectively. In particular, we write:

$$x = \begin{cases} v_1 & \text{if } \text{cond}_1 \\ ... & \\ v_n & \text{if } \text{cond}_n \end{cases}$$

instead of

$$\left\{ \begin{array}{ll} (x = v_1) \wedge \text{cond}_1 & \vee \\ \ldots & \vee \\ (x = v_i) \wedge \neg\text{cond}_1 \wedge \ldots \wedge \neg\text{cond}_{i-1} \wedge \text{cond}_i & \vee \\ \ldots & \vee \\ (x = v_n) \wedge \neg\text{cond}_1 \wedge \ldots \wedge \neg\text{cond}_{n-1} \wedge \text{cond}_n & \end{array} \right\}$$

3.2 The Carmel Syntax

3.2.1 The Carmel Instruction Set

As per [SH01], the *Java Card Virtual Machine* Bytecode Language Instruction set is partitioned into six groups:

$$\begin{aligned} I \in \text{Instruction} \quad ::= \quad & \text{CoreInstruction} \mid \text{ObjectInstruction} \mid \text{MethodInstruction} \mid \text{ArrayInstruction} \\ \mid \quad & \text{ExceptionInstruction} \mid \text{SubroutineInstruction} \end{aligned}$$

$$\begin{aligned} \text{CoreInstruction} \quad ::= \quad & \texttt{nop} \mid \texttt{push}\ \text{OpType}\ \text{Constant} \mid \texttt{pop}\ \text{integer} \mid \\ & \texttt{dup}\ \text{NbWords}\ \text{NbWords} \mid \texttt{swap}\ \text{NbWords}\ \text{NbWords} \mid \\ & \texttt{numop}\ \text{OpType}\ \text{NumericOperator}\ \text{OpType}_{opt} \mid \\ & \texttt{load}\ \text{OpType}\ \text{Index} \mid \texttt{store}\ \text{OpType}\ \text{Index} \mid \\ & \texttt{inc}\ \text{OpType}\ \text{Index}\ \text{integer} \mid \texttt{goto}\ \text{Address} \mid \\ & \texttt{if}\ \text{ComparisonOperator}\ \texttt{null}_{opt}\ \texttt{goto}\ \text{Address} \mid \\ & \texttt{lookupswitch}\ \text{OpType}\ (\text{integer}\ \texttt{=>}\ \text{Address})^*\ ,\ \texttt{default}\ \texttt{=>}\ \text{Address} \mid \\ & \texttt{tableswitch}\ \text{OpType}\ \text{integer}\ \texttt{=>}\ \text{Address}^*\ ,\ \texttt{default}\ \texttt{=>}\ \text{Address} \end{aligned}$$

ObjectInstruction ::= `new` ReferenceType |
`checkcast` ReferenceType | `instanceof` ReferenceType |
`getstatic` Field | `putstatic` Field |
`getfield` `this`$_{opt}$ Field | `putfield` `this`$_{opt}$ Field

MethodInstruction ::= `invokedefinite` Method | `invokevirtual` Method |
`invokeinterface` Method | `return` OpType$_{opt}$

ArrayInstruction ::= `arraylength` | `arrayload` OpType | `arraystore` OpType
ExceptionInstruction ::= `throw`
SubroutineInstruction ::= `jsr` Address | `ret` Index

where

Constant ::= integer | `null`
NbWords ::= integer
op ∈ NumericOperator ::= UnaryNumericOperator | BinaryNumericOperator
UnaryNumericOperator ::= `neg` | `to`
BinaryNumericOperator ::= `add` | `sub` | `mul` | `div` | `rem` | `cmp`
| `and` | `or` | `xor` | `shl` | `shr` | `ushr`
ComparisonOperator ::= `eq` | `ne` | `ge` | `gt` | `le` | `lt`
Index ::= integer

An operand type,

$t \in$ OpType ::= `b` | `s` | `i` | `r`,

indicates if the instruction is going to work with elements of type byte, short, integer or reference respectively. We reuse this notation in section 3.4 when we use run-time types. Similarly we reuse ReferenceType, defined in 3.2.2.

3.2.2 Types

A Carmel program deals with the following types:

$$
\begin{aligned}
\tau \in \text{Type} &::= \text{ReferenceType} \mid \text{PrimitiveType} \\
\text{PrimitiveType} &::= \text{BooleanType} \mid \text{NumericType} \mid \text{ReturnAddressType} \\
\text{NumericType} &::= \text{IntegralType} \\
\text{IntegralType} &::= \texttt{byte} \mid \texttt{short} \mid \texttt{int} \\
\text{BooleanType} &::= \texttt{boolean} \\
\text{ReturnAddressType} &::= \texttt{ret} \\
\tau_{\mathrm{r}} \in \text{ReferenceType} &::= \text{ArrayType} \mid \text{ClassType} \mid \text{InterfaceType} \\
\text{ClassType} &::= \text{ClassName} \\
\text{InterfaceType} &::= \text{InterfaceName} \\
\text{ComponentType} &::= \text{BooleanType} \mid \text{NumericType} \mid \text{ClassType} \mid \text{InterfaceType} \\
\text{ArrayType} &::= \text{ComponentType}\texttt{[ ]}
\end{aligned}
$$

A special type MethodType is defined:

$$
\begin{aligned}
\tau_{\mathrm{m}} \in \text{MethodType} &= (\textit{parameterTypes} : [\text{Type}]) \times (\textit{resultType} : \text{ResultType}) \\
\tau_{\mathrm{res}} \in \text{ResultType} &= \text{Type} \cup \text{VoidType}
\end{aligned}
$$

but will usually write[5]:

$$
\begin{aligned}
\text{MethodType} &::= (\tau_i)_1^n \to \tau_{\mathrm{res}} \\
\text{ResultType} &::= \text{Type} \mid \text{VoidType} \\
\text{VoidType} &::= \texttt{void}
\end{aligned}
$$

The subtype relation $\tau \preceq \tau'$ is defined following the "can be assigned to" criteria as indicated by the description of `checkcast` and `instanceof` in [Ora11b]. The following functions will be useful:

$$
\begin{aligned}
\textit{superClasses}(\bot) &= \{\,\} \\
\textit{superClasses}(\textit{cl}) &= \{\textit{cl}.\textit{superClass}\} \cup \textit{superClasses}(\textit{cl}.\textit{superClass})
\end{aligned}
\tag{3.1}
$$

$$
\begin{aligned}
\textit{superInterfaces}^*(\textit{iface}) = \bigcup\{\textit{superInterfaces}^*(\textit{iface}_i) \mid \textit{iface}_i \in \textit{superInterfaces}(\textit{iface})\} \cup \\
\textit{superInterfaces}(\textit{iface})
\end{aligned}
\tag{3.2}
$$

[5] Empty arrays can be represented with $(\tau_i)_1^n$ and $n = 0$

$$\frac{\tau \in \text{ComponentType}}{\tau \preceq \tau} \qquad \frac{cl' \in \mathit{superClasses}(cl)}{cl \preceq cl'} \qquad \frac{\mathit{iface} \in \mathit{implements}(cl)}{cl \preceq \mathit{iface}}$$

$$\frac{}{\mathit{iface} \preceq \texttt{Object}} \qquad \frac{\mathit{iface}' \in \mathit{superInterfaces}^*(\mathit{iface})}{\mathit{iface} \preceq \mathit{iface}'}$$

$$\frac{}{\tau\texttt{[ ]} \preceq \texttt{Object}} \qquad \frac{\tau \preceq \tau'}{\tau\texttt{[ ]} \preceq \tau'\texttt{[ ]}} \qquad \frac{\mathit{iface} \in \mathit{implements}(\texttt{Array})}{\tau\texttt{[ ]} \preceq \mathit{iface}}$$

Figure 3.1: Subtype Relation Rules

$$\begin{aligned} \mathit{implements}(\bot) &= \{\,\} \\ \mathit{implements}(cl) &= \bigcup\{\mathit{superInterfaces}^*(\mathit{iface}_i) \mid \mathit{iface}_i \in \mathit{implementedInterfaces}(cl)\} \cup \\ &\quad \mathit{implementedInterfaces}(cl) \cup \mathit{implements}(\mathit{superClass}(cl)) \end{aligned} \tag{3.3}$$

We write $cl \in \mathit{superClasses}(cl')$ instead of "cl' is a subclass of cl". Similarly, we write $\mathit{iface} \in \mathit{superInterfaces}^*(\mathit{interface}')$ and $cl \in \mathit{implements}(\mathit{iface})$ if iface' is a subinterface of iface and if cl implements interface iface, respectively.

The subtype relation $\tau \preceq \tau'$ is defined by the rules in Figure 3.1.

3.3 Program Structures

3.3.1 Programs and Packages

The information contained in the JCVM CAP files is abstracted by the framework into a *program structure*. A program P contains a set of packages and each package a set of classes. Classes and packages can be extracted from a program using the interface defined in [Mar01]. We define the following domains:

$$P \in \text{Program} \quad p \in \text{Package} \quad cl \in \text{Class} \quad \mathit{iface} \in \text{Interface}$$

A package is uniquely identified by an AID (application identifier):

$$\mathit{aid} \in \text{PackageAID}$$

Notation: Names are dropped from CAP files. In order to make the presentation more readable, names - concrete

syntax - are introduced when necessary, as in the case of types and exception names.

3.3.2 Classes, Interfaces, methods and fields

The class, interface, method and field structures can be accessed using the interfaces defined in [Mar01].

3.3.2.1 Method Lookup

All methods sharing the same signature are assigned the same method id. Given a method id and a class *cl*, the function *methodLookup*(id, cl) returns the first method structure with the same method id found in the superclass hierarchy of class *cl*:

$$methodLookup : \text{MethodID} \times \text{Class}_{\perp} \rightarrow \text{Method}_{\perp}$$

where

$$\begin{aligned}
&methodLookup(id, cl) = m && \Leftrightarrow \quad m \in cl.methods \wedge m.id = id \\
&methodLookup(id, cl) = methodLookup(id, cl.superClass) && \Leftrightarrow \quad \forall m \in cl.methods,\ m.id \neq id \\
&methodLookup(id, \perp) = \perp
\end{aligned}$$

3.4 Values and Runtime Data Types

Carmel supports two kind of values: *primitive values* and *reference values*. Primitive values consist of numeric values (`byte`, `short` and `int`[6]) and return addresses. Reference values consist of references to class instances and arrays (locations, section 3.5.2) and the special constant `null`. Java Card virtual machine values can be:

$$\begin{aligned}
v \in \text{Value} &= \text{OpType} \times (\text{PrimitiveValue} \cup \text{ReferenceValue}) \times \text{Address} \\
\text{PrimitiveValue} &= \text{NumericValue} \cup \text{ReturnAddressValue} \\
\text{NumericValue} &= \text{ByteValue} \cup \text{ShortValue} \cup \text{IntegerValue}
\end{aligned}$$

where

$$\text{ReferenceValue} = \text{Location} \cup \{\texttt{null}\}$$

and ReturnAddressValue values are runtime representations of Address elements. The Java Card virtual machine runtime types associated to the values defined above are:

$$\text{JCVMType} ::= \texttt{b} \mid \texttt{s} \mid \texttt{i} \mid \texttt{r} \mid \texttt{ra}$$

[6] Not all JCVM implementations support integers.

for byte, short, integer, reference and return address values respectively. Note that we are re-using the OpType set: JCVMType $=$ OpType $\cup$ {`ra`}.

We use the notation $(t, v, (m_n, pc_n))$ to express the runtime value that represents the value v with type *t* and which we've associated with address (m_n, pc_n). Note that the address component is never used in the operational semantics to transition between configurations.

Storage is managed by the JCVM in terms of an abstract storage unit called *word*. A word is large enough to hold a value of type `b`, `s`, `r` and `ra`. Two words are large enough to hold a value of type `i`. The function *nbWords* returns the number of words of a value or a sequence of values.

$$
\begin{array}{lcll}
\textit{nbWords} & : & \text{Value}^* & \rightarrow \quad \mathbb{N}_0 \\
\textit{nbWords} & : & \epsilon & = \quad 0 \\
\textit{nbWords} & : & (t, Y, (m_n, pc_n)) & = \quad \begin{cases} 2, & t = \texttt{i} \\ 1, & otherwise \end{cases} \\
\textit{nbWords} & : & (t_1, Y_1, (m_1, pc_1)) :: \ldots :: (t_p, Y_p, (m_p, pc_p)) & = \quad \sum_{i=1}^{p} \textit{nbWords}\big(\ (t_i, Y_i, (m_i, pc_i))\ \big)
\end{array}
$$

3.4.1 Numeric Values

The Java types `byte` and `short` are supported by the JCVM. These types correspond to the `b` and `s` JCVM types. Some JCVM implementations may also support the `int` type. The numeric values supported by the JCVM consist of:

- Type `b` (`byte`), represented as 8-bit signed two's complement integers.
- Type `s` (`short`), represented as 16-bit signed two's complement integers.
- Type `i` (`int`), represented as 32-bit signed two's complement integers.

Values of the Java type `boolean` are implemented as byte values, where 1 represents `true` and 0 `false`.

Operations between numeric values are performed using the functions *applyUnary* and *applyBinary*:

$$
\begin{aligned}
\textit{applyUnary} &: \text{UnaryNumericOperator} \times \text{Value} \times \text{Address} \\
\textit{applyUnary} &: \text{UnaryNumericOperator} \times \text{JCVMType} \times \text{Value} \times \text{Address} \\
\textit{applyBinary} &: \text{BinaryNumericOperator} \times \text{Value} \times \text{Value} \times \text{Address}
\end{aligned}
$$

As mentioned in section 3.5.3.1, `boolean` and `byte` values have to be sign-extended to a `short` before they are pushed to the operand stack. Similarly, `short` values placed on the stack have to be truncated before they are stored in to a field or array element of type `boolean` or `byte`. The *toShort* and *fromShort* functions are defined such that:

$$
\begin{aligned}
\textit{toShort}((t, \textit{c}, (\textit{m}_n, pc_n))) &= \textit{applyUnary}(\texttt{to}, \texttt{s}, (\textit{t}, \textit{c}, (\textit{m}_n, pc_n)), (\textit{m}_n.pc_n)) \\
\textit{fromShort}((\texttt{s}, \textit{c}, (\textit{m}_n, pc_n)), \textit{t}) &= \textit{applyUnary}(\texttt{to}, \textit{t}, (\texttt{s}, \textit{c}, (\textit{m}_n, pc_n)), (\textit{m}_n.pc_n))
\end{aligned}
$$

3.5 Run-Time Structures

3.5.1 Object definitions

Carmel allows two object types, array objects (arrays) and non-array objects (class instances). For simplicity and uniformity, we specify both object types in terms of 11 common attributes:

$$
\begin{aligned}
o \in \text{Object} \quad = \quad & (\textit{type} : \text{Type}) \\
\times \quad & (\textit{refType} : \text{ReferenceType}) \\
\times \quad & (\textit{isArray} : \text{boolean}) \\
\times \quad & (\textit{owner} : \text{Owner}) \\
\times \quad & (\textit{entryPoint} : \text{EntryPoint}) \\
\times \quad & (\textit{isGlobal} : \text{boolean}) \\
\times \quad & (\textit{transient} : \text{Transient}) \\
\times \quad & (\textit{creationPoint} : \text{Address}) \\
\times \quad & (\textit{values} : (\text{FieldID} \cup \text{integer}) \rightarrow \text{Value}) \\
\times \quad & (\textit{length} : \text{integer}) \\
\times \quad & (\textit{heapID} : \text{integer})
\end{aligned}
$$

and distinguish between object types based on the value of the *isArray* attribute i.e. an array would have the *isArray* attribute set to true and a class instance has the *isArray* attribute set to false i.e.

$$
\begin{aligned}
o \in \text{ClassInstance} \quad &\Longleftrightarrow \quad o.\textit{isArray} = \texttt{false} \\
o \in \text{ArrayObject} \quad &\Longleftrightarrow \quad o.\textit{isArray} = \texttt{true}
\end{aligned}
$$

Considering each attribute in turn:

- An object's *type* is its class name if it's a class instance, or the array element type if it's an array object;
- An object's *refType* is its reference type, if it's a class instance, then $\textit{refType} = \textit{type}$, if it's an array object, the $\textit{refType} = \textit{type}[\,]$ i.e. the reference type is an array whose elements are of type *type*;
- An object's *isArray* attribute specifies whether the object is an array instance (in which case this attribute is `true`) or a class instance (in which case the attribute is `false`);
- An object's *owner* attribute specifies the object's owner AID/context and object's group/package AID/context

– see Section 2.3.3.11 for more information. This information is kept by a special structure:

$$\text{Owner} = (\textit{context} : \text{Context}) \times (\textit{aid} : \text{AID})$$

where contexts are defined by:

$$\textit{ctxt} \in \text{Context} = \text{PackageAID} \cup \{\texttt{JCRE}\};$$

- An object's *entryPoint* attribute indicates whether the object is a permanent or temporary `JCRE` entry point object or not a temporary entry point object – see Sections 2.3.3.12 and 2.3.3.13 for further information. We define:

$$\text{EntryPoint} = \{\texttt{permanent}, \texttt{temporary}, \texttt{no}\};$$

- An object's *isGlobal* attribute is a boolean attribute specifying whether the object is a global array instance – see Section 2.3.3.14 for further details;

- An object's *transient* attribute specifies whether the object is a transient object – see Section 2.3.3.8 for further details. In all the *Java Card* Classic Editions so far, only arrays can be designated as transient objects. We define:

$$\text{Transient} = \{\texttt{CLEAR_ON_RESET}, \texttt{CLEAR_ON_DESELECT}, \texttt{NOT_TRANSIENT}\};$$

- An object's *creationPoint* is the address at which the object was instantiated;

- An object's *values* is a map from field ids to runtime values for class instances and a map from non-negative array indexes to runtime values for array objects;

- An object's *length* is the length of the array if the object is an array object, otherwise it is zero;

- An object's *heapID* is a unique integer across all objects to allow testing for object reference/pointer equality.

We use the *instanceFields* function to determine the set of non-static fields of a class, including the fields defined in its superclasses. *instanceFields* is defined inductively:

$$\textit{instanceFields}(\bot) = \{\ \}$$
$$\textit{instanceFields}(\textit{cl}) = \{\textit{f} \mid \textit{f} \in \textit{cl.fields} \wedge \neg \textit{f.isStatic}\} \cup \textit{instanceFields}(\textit{cl.superClass})$$

The function *def* maps types to their default initial values. Numeric types map to 0 and reference types to `null`.

$$\mathit{def}(\tau) = \begin{cases} 0 & \text{if } \tau \in \text{NumericType} \\ \texttt{null} & \text{if } \tau \in \text{ReferenceType} \end{cases}$$

3.5.2 Objects and the Heap

Objects in Carmel are stored in one or both of two heaps depending on the transactional state of the JCVM:

- The persistent non-transactional object heap *H*. When:
  - no transaction is in progress and an update to an object's field is requested, or
  - regardless of whether a transaction is in progress and an update to a transient object is requested (i.e. the object's transient status is one of $\{\texttt{CLEAR_ON_RESET}, \texttt{CLEAR_ON_DESELECT}\}$), or
  - regardless of whether a transaction is in progress and an update to a global object is requested (i.e. the object's attribute *isGlobal* $= \texttt{true}$)

  changes are made to the persistent heap *H* and subsequent reads of the object field returns the updated value. Should the card suffer card-tear, these updated values will be restored at next card session.
- The temporary transactional object heap *JH*. When:
  - a transaction is in progress and an update to an object's field is requested, and both of the following are true:
    - the object's field transient status is `NOT_TRANSIENT`
    - the object's attribute *isGlobal* $= \texttt{false}$

    changes are made to the temporary transactional heap *JH* and subsequent reads of the object field return the updated value in the *JH*. If the transaction is committed, all the changes in the temporary heap *JH* are copied to the permanent heap *H* and the *JH* is then cleared. Should the card suffer card-tear before an open transaction is committed, the updated values held in the temporary transactional object heap *JH* are lost and at the next card session the temporary heap is initialised to empty.

We define the helper function $\texttt{getJHorH}()$ and its extensions to aid the reading of the appropriate values from the two object heaps:

$$\texttt{getJHorH}(\mathit{loc}) = \begin{cases} \mathit{JH}(\mathit{loc}) & \mathit{loc} \in \mathit{dom}(\mathit{JH}) \\ \mathit{H}(\mathit{loc}) & \text{otherwise} \end{cases}$$

$$\texttt{getJHorH}(\mathit{loc}.\mathit{values}(\mathit{f}.\mathit{id})) = \begin{cases} \mathit{JH}(\mathit{loc}).\mathit{values}(\mathit{f}.\mathit{id}) & \mathit{loc} \in \mathit{dom}(\mathit{JH}) \wedge \mathit{f} \in \mathit{dom}(\mathit{JH}(\mathit{loc}.\mathit{values})) \\ \mathit{H}(\mathit{loc}).\mathit{values}(\mathit{f}.\mathit{id}) & \text{otherwise} \end{cases}$$

The heap is defined as a mapping from locations to objects, which can be class instances or arrays.

$$\mathit{H}, \mathit{JH} \in \text{Heap} \quad = \quad \text{Location} \rightarrow \text{Object}$$

3.5.3 Frames

A frame contains information about the state of execution of a method. It is defined as follows:

$\mathit{F} \in \text{Frame} = \text{Location} \times \text{TransactionDepth} \times \text{TransactionDepth} \times \text{IOState} \times \text{Address} \times \text{LocalVar} \times \text{OperandStack}$
$\mathit{F} ::= \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n \rangle$

A frame contains the following components:

- The location loc_n of the object on which the method in the Address component is currently invoked;
- The *initial* transaction depth itd_n in which this frame was called at the *firstAddress* address of the Address component;
- The *current* transaction depth ctd_n on which the method in the Address component is currently invoked;
- The current state of the APDU input/output buffer io_n;
- The current program point (combination of $\mathit{m}_n \in$ Method and $pc_n \in$ Address);
- The array of local variables L_n;
- The operand stack S_n.

3.5.3.1 Operand Stack

Carmel instructions take their dynamic operands from a special structure, the operand stack[7]. The operand stack is defined as a sequence of values:

$$\mathit{S} = \mathit{v}_1 :: \ldots :: \mathit{v}_n \in \text{OperandStack} = \text{StackValue}^*$$

[7] Some instructions also take dynamic operands from the local variables.

where values on top of the stack appear on the right-hand side of the sequence i.e. v_n is on top of the stack above. The empty stack is represented with the symbol ϵ. An element can be added to a stack with the **::** operator (S::v) and two stacks can be concatenated with the **:** operator (S:S'). For clarity we will write v_1::v_2 instead of ϵ::v_1::v_2. The operand stack stores values of all types except those of runtime type `b`(`byte` or `boolean` types). Instead, the Java Card virtual machine converts - sign-extends - `byte` and `boolean` values to `short` values when they are pushed to the operand stack. The domain of stack values is defined as:

$$\text{StackValue} = \text{Value} - \text{ByteValue} \qquad \text{StackType} = \text{JCVMType} - \{\texttt{b}\}$$

3.5.3.2 Local Variables

A local variable is not referenced by name but by the position it occupies in the array of local variables, defined by:

$$L \in \text{LocalVar} = [\text{StackValue}_{\perp}]$$

The array of local variables can store values of any of the stack types. Local variable slots are not statically typed so values of different types can be stored in the same slot at different times during the execution of a method. Special care must be taken when using multiword values. An integer value stored in $L[i]$ takes two slots and access to $L[i+1]$ is prohibited. We can model this by making $L[i+1] = \perp$ while position i holds the integer value.

Notation: We write:

$$L = v_0 :: \ldots :: v_n$$

when local variable array L is initialized with the sequence of values (usually from the operand stack) $v_0 :: \ldots :: v_n$. Note that $L[0] = v_0$ but v_i does not necessarily correspond to $L[i]$. Instead we have:

$$L[0] = v_0 \ \wedge \ L[\mathit{nbWords}(v_0 :: \ldots :: v_{i-1})] = v_i \quad i \in \{1, \ldots, n\}$$

3.5.4 Configurations

A runtime configuration describes the state of execution of the JCVM. We define three kinds of configurations:

$$C \in \mathit{Config} \ = \ \mathit{RConfig} \ \cup \ \mathit{EConfig} \ \cup \ \mathit{HConfig}$$

A running configuration - $RConfig$ - keeps track of the chain of invoked methods by using a stack of call frames.

An exception configuration - $EConfig$- represents the state of uncaught exceptions or machine errors.

The halt configuration is the result of returning from the method that started the execution of the current applet.

A **running configuration** $C = \langle R, K, H, I, HID, JH, CHN, SF \rangle$ contains the following components:

- Applet registry *R*, a map from AIDs to Location of registered applets;
- The persistent static fields memory *K*, a mapping FieldID $\rightarrow$ Value;
- The persistent object heap *H*, a mapping from locations to runtime objects (class and array instances);
- A set of invalidated object locations I, of objects created inside a transaction that was subsequently aborted which are to be treated as `null`;
- A monotonically increasing sequence number (integer) HID, used whenever a new object is created, whether by atomic bytecode instructions or *Java Card API* calls that allocate memory dynamically. This sequence number is used to check for object reference/pointer equality;
- The journalling/transactional object heap *JH*, a mapping from locations to runtime objects (class and array instances) of conditionally updated object fields made inside a transaction;
- Logical channels component CHN, a map from a channel number between 0 and 39 to Location of a registered applet (or `null`);
- The call-stack SF of frames. See Section 3.5.3 for a description of the structure of frames.

An **exception configuration** represents a terminal configuration where an uncaught exception reaches the `JCRE`:

$$C = \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon \rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, cmtd, io, (dispatch(), 10), [\,], (\texttt{r}, \mathit{loc}_{exc}, (\mathit{m}, apc)) \rangle \rangle$$

where $(\texttt{r}, \mathit{loc}_{exc}, (\mathit{m}, apc))$ is the location of the exception object that was thrown and not caught, and $cmtd$ was the current transaction depth of the current frame at the point the JCVM threw the uncaught exception, and io was the state of the APDU buffer at the point the JCVM threw the uncaught exception.

An **halting configuration** represents a terminal configuration where an applet is returning normally (i.e. not via exception) to the JCRE:

$$C = \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\ ], \epsilon \rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, cmtd, io, (dispatch(), 20), [\ ], S \rangle \rangle$$

where $S = \epsilon$ or $S = (\texttt{s}, c, (\mathit{m}, apc)), c \in \{0, 1\}$ depending on the return type of the applet lifecycle method invoked, and $cmtd$ was the current transaction depth of the current frame at the point the applet returned control to the JCRE, and io was the state of the APDU buffer at the point the applet returned control to the JCRE.

A running configuration in a Carmel program P is either an initial configuration as per Section 3.7.1 or is reachable from an initial configuration via repeated application of the configuration transitions of Section 3.9 $P \Big| \begin{matrix} Config \ \Rightarrow \\ Config \end{matrix}$ wherever the call-stack of the resulting configuration is not one of the following forms:

- $\langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\ ], \epsilon \rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, cmtd, io, (dispatch(), 10), [\ ], (\texttt{r}, \mathit{loc}_{exc}, (\mathit{m}, apc)) \rangle$
- $\langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\ ], \epsilon \rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, cmtd, io, (dispatch(), 20), [\ ], S \rangle$

An exception configuration in a Carmel program P is the result of repeated application of the configuration transitions of Section 3.9 $P \Big| \begin{matrix} Config \ \Rightarrow \\ Config \end{matrix}$ from an initial configuration as per Section 3.7.1 and the call-stack of the resulting configuration is of the following form:

$$\langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\ ], \epsilon \rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, cmtd, io, (dispatch(), 10), [\ ], (\texttt{r}, \mathit{loc}_{exc}, (\mathit{m}, apc)) \rangle$$

A halting configuration in a Carmel program P is the result of repeated application of the configuration transitions of Section 3.9 $P \Big| \begin{matrix} Config \ \Rightarrow \\ Config \end{matrix}$ from an initial configuration as per Section 3.7.1 and the call-stack of the resulting configuration is of the following form:

$$\langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\ ], \epsilon \rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, cmtd, io, (dispatch(), 20), [\ ], S \rangle$$

3.6 Exceptions

An **exception** is thrown whenever a program violates the semantic constraints of *Java Card* Classic Edition 3 or any other constraint specified by the programmer. Exceptions can be thrown explicitly using the `throw` instruction.

An exception is said to be caught if an appropriate **exception handler** is found in the current stack of call frames. If an exception is caught, control is transferred to the start address indicated by the exception handler and execution is resumed at that point. Note that we require there always to be at least one stack frame on the JCVM: the bottom stack frame is always the JCRE, a looping process which has the responsibility to listen for– and reply to– commands from the CAD client, and this bottom frame is owned by the JCRE itself. In our operational semantics, an uncaught exception reaching the JCRE level i.e. an exception configuration from the previous section, will return the location of the exception or JCVM error object to the JCRE and it will communicate the error back to the CAD.

The process of throwing and catching an exception is defined by the *catchException* function, defined in Table 3.6, which searches the stack of call frames according to the following criteria:

- It first inspects the top call frame to see if the exception of class τ and referenced by *loc* can be handled. If no appropriate exception handler is found in the top frame, the frame is popped and the search continues from the invoker's frame ("... frame is popped, the frame of its invoker is reinstated, and *loc* is rethrown"[Ora11c, Chapter 7, page 133]).

- If the top frame contains an appropriate handler, the pc is reset to the value indicated by the handler, the operand stack of the frame is cleared and *loc* is pushed back onto the operand stack and the new stack of frames is returned.

- If no user frame is found that can handle the exception - if the stack of frames is the empty stack - the function returns the reference to the uncaught exception.

The *findHandler*(m, pc, τ) function searches for the **first** exception handler in method *m* that can handle an exception of class τ at address pc. It returns the address of the code intended to handle the exception.The semantics of *findHandler* is described below.

Let $E = m.\mathit{exceptionHandlers}$:

$$\mathit{findHandler}(m, pc, \tau) = \begin{cases} E[i].\mathit{handlerAddress} & \exists i \in \{0, \ldots, E.\mathit{length} - 1\}. \\ & \quad \mathit{isCorrectHandler}(E, i, pc, \tau) \wedge \\ & \quad \forall j, 0 \leq j < i : \neg \mathit{isCorrectHandler}(E, j, pc, \tau) \\ \bot & \text{otherwise} \end{cases}$$

The predicate $\mathit{isCorrectHandler}(E, i, pc, \tau)$ determines if exception handler $E[i]$ can handle an exception τ at address pc:

$$\mathit{isCorrectHandler}(E, i, pc, \tau) = \begin{cases} \texttt{true} & (E[i].\mathit{startAddress} \leq pc \leq E[i].\mathit{endAddress}) \wedge \\ & (\tau \preceq E[i].\mathit{catchType} \vee E[i].\mathit{catchType} = \bot) \\ \texttt{false} & \text{otherwise} \end{cases}$$

It is the job of the compiler to order the table in such a way that, given a set of nested exceptions, the innermost exceptions appear first.

3.6.1 Runtime Exceptions

There are eight runtime exceptions which may be raised by the JCVM during the execution of a program. These are `JCRE` owned, pre-allocated exceptions objects designated as temporary `JCRE` Entry Point Objects. In our operational semantics, we refer to them using the symbolic constant of column two in the following table, column one of the same row gives their class name:

Exception class	Literal Location constant appearing in operational semantics
`java.lang.ArithmeticException`	$\mathit{loc}_{\texttt{ArithmeticException}}$
`java.lang.ArrayIndexOutOfBoundsException`	$\mathit{loc}_{\texttt{ArrayIndexOutOfBoundsException}}$
`java.lang.ArrayStoreException`	$\mathit{loc}_{\texttt{ArrayStoreException}}$
`java.lang.ClassCastException`	$\mathit{loc}_{\texttt{ClassCastException}}$
`java.lang.IndexOutOfBoundsException`	$\mathit{loc}_{\texttt{IndexOutOfBoundsException}}$
`java.lang.NegativeArraySizeException`	$\mathit{loc}_{\texttt{NegativeArraySizeException}}$
`java.lang.NullPointerException`	$\mathit{loc}_{\texttt{NullPointerException}}$
`java.lang.SecurityException`	$\mathit{loc}_{\texttt{SecurityException}}$

3.7 The Applet Firewall

As explained in Sections 2.3.3.11 – 2.3.3.15, arrays and class instances are owned by the applet that created them. The context (or package) where the applet is defined determines the owning context. The applet firewall provides a security mechanism that prevents an object from being accessed by code running in a different context (different from the owning context).

The JCRE provides several mechanisms to allow object access across contexts. JCRE entry point objects - temporary and permanent - and global arrays are objects owned by the JCRE context that can be accessed from any context. Another mechanism that allows object access among contexts is the use of *Shareable* interfaces. Methods of shareable interfaces can be invoked from one context even if the object implementing them is owned by an applet in another context. A context switch occurs when a method of an object owned by a different context is invoked i.e. two consecutive frames have different contexts.

Prior to performing an operation on an object, the JCVM performs an access check. The access checks depend on the type and owner of the referenced object, the instruction, and the currently active context. Chapter 6 of [Ora11b] details the applet firewall and the access checks comprehensively. Section 3.8 codifies the access checks of the applet firewall into a set of security predicates.

3.7.1 Initial Configurations

Let P be a Carmel program. Then

$\langle [\,], K_{init}, H, 0, [\,], [\,], \texttt{CHN}, \langle loc_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon\rangle :: \langle loc_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\,], \epsilon\rangle :: \langle loc_{\texttt{packageaid}}, 0, 0, \texttt{INITIAL}, (m_{\texttt{install}}, m_{\texttt{install}}.\mathit{firstAddress}), L_{\texttt{install}}, \epsilon\rangle\rangle$
is a valid initial configuration in P whence:

$applet_{class} \in P.\mathit{classes}$. $applet_{class} \prec \texttt{javacard.framework.Applet}$:

$m_{install} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.install(byte[ ], short, byte)}.\mathit{id}, applet_{class})$
$0 \leq length \leq 127$
$0 \leq offset \leq 127$
$L_{install} = [0 \mapsto (\texttt{r}, loc_{\texttt{AppletInstallBuffer}}, (1,1)), 1 \mapsto (\texttt{s}, \texttt{length}, (m_{\texttt{install}}, m_{\texttt{install}}.\mathit{firstAddress})), 2 \mapsto (\texttt{b}, \texttt{offset}, (m_{\texttt{install}}, m_{\texttt{install}}.\mathit{firstAddress}))]$
$loc_{packageaid} \notin dom(H_{Init})$

$$\left(\begin{array}{c}(o \in \text{Object}) \wedge (o.\mathit{type} = applet_{class}) \wedge (o.\mathit{refType} = applet_{class}) \wedge (o.\mathit{isArray} = \texttt{false}) \wedge (o.\mathit{owner} = (applet_{class}.packageAID, \texttt{null})) \wedge \\ (o.\mathit{entryPoint} = \texttt{no}) \wedge (o.\mathit{isGlobal} = \texttt{false}) \wedge (o.\mathit{transient} = \texttt{NOT_TRANSIENT}) \wedge (o.\mathit{creationPoint} = (1,1)) \wedge (o.\mathit{values} = \texttt{[ ]}) \wedge (o.\mathit{length} = 0) \wedge \\ (o.\mathit{heapID} = -50)\end{array}\right)$$

$H = H_{Init}[loc_{packageaid} \mapsto o]$

and $loc_{\texttt{AppletInstallBuffer}}, loc_{\texttt{JCRE}}, H_{Init}$ and K_{init} are as defined in Appendix D

Table 3.1: Valid Initial Configurations in Carmel – Table 1

Let P be a Carmel program. Then

$\langle R, K, H_2, HID, [\,], [\,], \texttt{CHN}, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{packageaid}}, 0, 0, \texttt{INITIAL}, (m_{\texttt{install}}, m_{\texttt{install}}.\mathit{firstAddress}), L_{\texttt{install}}, \epsilon\rangle\rangle$
is a valid initial configuration in P whence:

$\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, imtd_1, cmtd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, imtd_2, cmtd_2, io_2, (m_2, pc_2), L_2, S_2\rangle\rangle \in (EConfig \cup HConfig)$

$applet_{class} \in P.\mathit{classes}\ .\ applet_{class} \prec \texttt{javacard.framework.Applet}$:

$m_{install} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.install(byte[ ], short, byte)}.\mathit{id}, applet_{class})$
$0 \leq length \leq 127$
$0 \leq offset \leq 127$
$L_{install} = [0 \mapsto (\texttt{r}, \mathit{loc}_{\texttt{AppletInstallBuffer}}, (1,1)), 1 \mapsto (\texttt{s}, \texttt{length}, (m_{\texttt{install}}, m_{\texttt{install}}.\mathit{firstAddress})), 2 \mapsto (\texttt{b}, \texttt{offset}, (m_{\texttt{install}}, m_{\texttt{install}}.\mathit{firstAddress}))]$
$\mathit{loc}_{packageaid} \notin dom(H)$

$$\left(\begin{array}{c}(o \in \text{Object}) \ \wedge (o.\mathit{type} = applet_{class}) \ \wedge (o.\mathit{refType} = applet_{class}) \ \wedge (o.\mathit{isArray} = \texttt{false}) \ \wedge (o.\mathit{owner} = (applet_{class}.packageAID, \texttt{null})) \wedge \\ (o.\mathit{entryPoint} = \texttt{no}) \ \wedge (o.\mathit{isGlobal} = \texttt{false}) \ \wedge (o.\mathit{transient} = \texttt{NOT_TRANSIENT}) \ \wedge \ (o.\mathit{creationPoint} = (1,1)) \ \wedge (o.\mathit{values} = \texttt{[ ]}) \ \wedge (o.\mathit{length} = 0) \wedge \\ (o.\mathit{heapID} = -50)\end{array}\right)$$

$H_2 = H[\mathit{loc}_{packageaid} \mapsto o]$

and $\mathit{loc}_{\texttt{AppletInstallBuffer}}, \mathit{loc}_{\texttt{JCRE}}$ is as defined in Appendix D

Table 3.2: Valid Initial Configurations in Carmel – Table 2

The following configurations are all valid initial configurations:

$\langle R, K, H, I, HID, [\,], [\,], \texttt{CHN}, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{applet}}, 0, 0, \texttt{INITIAL}, (m_{\texttt{process}}, m_{\texttt{process}}.\mathit{firstAddress}), L_{\texttt{process}}, \epsilon\rangle\rangle$

whence :

$\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, imtd_1, cmtd_1, io_1, m_1, pc_1, L_1, S_1\rangle :: \langle \mathit{loc}_2, imtd_2, cmtd_2, io_2, m_2, pc_2, L_2, S_2\rangle\rangle \in (EConfig \cup HConfig)$

$aid \in dom(R)$:

$R(aid) = (\mathit{loc}_{AID}, \mathit{loc}_{applet})$

$m_{process} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.process(APDU)}.\mathit{id}, H(\mathit{loc}_{applet}).\mathit{refType})$

$L_{process} = [0 \mapsto (\texttt{r}, \mathit{loc}_{\texttt{applet}}, H(\mathit{loc}_{\texttt{applet}}).\mathit{creationPoint}), 1 \mapsto (\texttt{r}, \mathit{loc}_{\texttt{APDUBuffer}}, (1,1))]$

and $\mathit{loc}_{\texttt{JCRE}}$ is as defined in Appendix D

Table 3.3: Valid Initial Configurations in Carmel – Table 3

The following configurations are all valid initial configurations:

$\langle R, K, H, I, HID, [\,], [\,], \texttt{CHN}, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{applet}}, 0, 0, \texttt{INITIAL}, (m_{\texttt{select}}, m_{\texttt{select}}.\mathit{firstAddress}), L_{\texttt{select}}, \epsilon\rangle\rangle$

$\langle R, K, H, I, HID, [\,], [\,], \texttt{CHN}, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{applet}}, 0, 0, \texttt{INITIAL}, (m_{\texttt{deselect}}, m_{\texttt{deselect}}.\mathit{firstAddress}), L_{\texttt{deselect}}, \epsilon$

whence :

$\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, imtd_1, cmtd_1, io_1, m_1, pc_1, L_1, S_1\rangle :: \langle \mathit{loc}_2, imtd_2, cmtd_2, io_2, m_2, pc_2, L_2, S_2\rangle\rangle \in (\mathit{EConfig} \cup \mathit{HConfig})$

$aid \in dom(R)$:

$R(aid) = (\mathit{loc}_{AID}, \mathit{loc}_{applet})$

$|\mathit{loc}_{AID} \in \texttt{range}(CHN)| = 0$

$m_{select} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.select}().\mathit{id}, H(\mathit{loc}_{applet}).\mathit{refType})$

$m_{deselect} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.deselect}().\mathit{id}, H(\mathit{loc}_{applet}).\mathit{refType})$

$L_{select} = [0 \mapsto (\texttt{r}, \mathit{loc}_{\texttt{applet}}, H(\mathit{loc}_{\texttt{applet}}).\mathit{creationPoint})]$

$L_{deselect} = [0 \mapsto (\texttt{r}, \mathit{loc}_{\texttt{applet}}, H(\mathit{loc}_{\texttt{applet}}).\mathit{creationPoint})]$

and $\mathit{loc}_{\texttt{JCRE}}$ is as defined in Appendix D

Table 3.4: Valid Initial Configurations in Carmel – Table 4

The following configurations are all valid initial configurations:

$\langle R, K, H, I, HID, [\,], [\,], \texttt{CHN}, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{applet}}, 0, 0, \texttt{INITIAL}, (m_{\texttt{select}}, m_{\texttt{select}}.\mathit{firstAddress}), L_{\texttt{select}}, \epsilon\rangle\rangle$

$\langle R, K, H, I, HID, [\,], [\,], \texttt{CHN}, \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\,], \epsilon\rangle :: \langle \mathit{loc}_{\texttt{applet}}, 0, 0, \texttt{INITIAL}, (m_{\texttt{deselect}}, m_{\texttt{deselect}}.\mathit{firstAddress}), L_{\texttt{deselect}}, \epsilon$

whence :

$\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, imtd_1, cmtd_1, io_1, m_1, pc_1, L_1, S_1\rangle :: \langle \mathit{loc}_2, imtd_2, cmtd_2, io_2, m_2, pc_2, L_2, S_2\rangle\rangle \in (\mathit{EConfig} \cup \mathit{HConfig})$

$aid \in dom(R)$:

$R(aid) = (\mathit{loc}_{AID}, \mathit{loc}_{applet})$

$|\mathit{loc}_{AID} \in \texttt{range}(CHN)| \geq 1$

$H(\mathit{loc}_{applet}).\mathit{refType} \preceq \texttt{javacard.framework.MultiSelectable}$

$m_{select} = \mathit{methodLookup}(\texttt{javacard.framework.MultiSelectable.select}().\mathit{id}, H(\mathit{loc}_{applet}).\mathit{refType})$

$m_{deselect} = \mathit{methodLookup}(\texttt{javacard.framework.MultiSelectable.deselect}().\mathit{id}, H(\mathit{loc}_{applet}).\mathit{refType})$

$appInstStillActive \in \{0, 1\}$

$L_{select} = [0 \mapsto (\texttt{r}, \mathit{loc}_{\texttt{applet}}, H(\mathit{loc}_{\texttt{applet}}).\mathit{creationPoint}, 1 \mapsto (\texttt{s}, \texttt{appInstStillActive}, (m_{\texttt{install}}, m_{\texttt{select}}.\mathit{firstAddress})))]$

$L_{deselect} = [0 \mapsto (\texttt{r}, \mathit{loc}_{\texttt{applet}}, H(\mathit{loc}_{\texttt{applet}}).\mathit{creationPoint}), 1 \mapsto (\texttt{s}, \texttt{appInstStillActive}, (m_{\texttt{install}}, m_{\texttt{deselect}}.\mathit{firstAddress}))]$

and $\mathit{loc}_{\texttt{JCRE}}$ is as defined in Appendix D

Table 3.5: Valid Initial Configurations in Carmel – Table 5

Let $loc \in dom(H) \cup dom(JH) \wedge \texttt{getJHorH}(loc).refType = \tau \wedge \tau \preceq \texttt{java.lang.Throwable}$:

Then define the *catchException* function:

$$catchException : \text{Frame}^* \times \text{ReferenceValue} \times \text{Type} \times \text{TransactionDepth} \times \text{IOState} \rightarrow \text{Frame}^*$$

$$catchException(\langle loc_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\ ], \epsilon\rangle :: \langle loc_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{dispatch}(), 0), [\ ], \epsilon\rangle :: \langle loc_3, imtd_3, cmtd_3, io_3, (m_3, pc_3), L_3, S_3\rangle, (\texttt{r}, loc_{exc}, (m, apc)), class_{exception}, cmtd, io)$$

$$\langle loc_{\texttt{JCRE}}, 0, 0, \texttt{INITIAL}, (\texttt{loop}(), 0), [\ ], \epsilon\rangle :: \langle loc_{\texttt{JCRE}}, 0, cmtd, io, (dispatch(), 10), [\ ], (\texttt{r}, loc_{exc}, (m, apc))\rangle$$

if $findHandler(m_3, pc_3, class_{exception}) = \bot$

$$catchException(\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, s_n\rangle, (\texttt{r}, loc_{exc}, (m, apc)), class_{exception}, cmtd, io) =$$

$$\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle loc_n, itd_n, cmtd, io, (m_n, pc'), L_n, (\texttt{r}, loc_{exc}, (m, apc))\rangle$$

if $findHandler(m_n, pc_n, class_{exception}) = pc' \neq \bot$

$$catchException(\ \langle loc_1, imtd_1, cmtd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, imtd_2, cmtd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \langle loc_3, imtd_3, cmtd_3, io_3, (m_3, pc_3), L_3, S_3\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, s_n\rangle,$$
$$(\texttt{r}, loc_{exc}, (m, apc)), class_{exception}, cmtd, io\ ) =$$

$$catchException(\ \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle loc_{(n-1)}, itd_{(n-1)}, ctd_{(n-1)}, io_{(n-1)}, (m_{(n-1)}, pc_{(n-1)}), L_{(n-1)}, S_{(n-1)}\rangle,$$
$$(\texttt{r}, loc_{exc}, (m, apc)), class_{exception}, cmtd, io\ )$$

if $findHandler(m_n, pc_n, class_{exception}) = \bot$

Table 3.6: Exception Handling in Carmel

3.8 Firewall Predicates

checkPutObjectStatic	:	Owner × boolean × boolean × EntryPoint → boolean
checkGetField	:	Owner × Owner → boolean
checkPutField	:	Owner × Owner × boolean × boolean × EntryPoint × boolean → boolean
checkThrow	:	Owner × Owner × Owner × boolean × boolean × EntryPoint × Transient → boolean
checkInvokeDefinite	:	Owner × Owner × Owner × boolean × boolean × EntryPoint × Transient → boolean
checkInvokeVirtual	:	Owner × Owner × Owner × boolean × boolean × EntryPoint × Transient → boolean
checkInvokeInterface	:	Owner × Owner × Owner × boolean × boolean × EntryPoint × Transient × Class × Interface → boolean
checkCast	:	Owner × Owner × Owner × boolean × boolean × EntryPoint × Transient × ReferenceType × ReferenceType → boolean
checkArrayLoad	:	Owner × Owner × Owner × boolean × Transient → boolean
checkArrayStore	:	Owner × Owner × Owner × boolean × Transient × boolean → boolean

checkPutObjectStatic : Owner × boolean × boolean × EntryPoint → boolean

checkPutObjectStatic$(own_{current}, isArray_{object}, isGlobal_{object}, entryPoint_{object}) =$

$$\begin{aligned}
&\big(own_{current} = (\texttt{JCRE}, M)\big) \vee \\
&\big(isArray_{object} \wedge \neg isGlobal_{object}\big) \vee \\
&\big(\neg isArray_{object} \wedge entryPoint_{object} \neq \texttt{temporary}\big)
\end{aligned}$$

checkGetField : Owner × Owner → boolean

checkGetField$(own_{current}, own_{object}) =$

$$\begin{aligned}
&\big(own_{current} = (\texttt{JCRE}, M)\big) \vee \\
&\big(own_{current} = (L, M) \wedge own_{object} = (L, N)\big)
\end{aligned}$$

checkPutField : Owner × Owner × Value × boolean → boolean

checkPutField$(own_{current}, own_{object}, (t, v, (m, pc)), isObject_v) =$

$$\begin{aligned}
&\big(own_{current} = (\texttt{JCRE}, M)\big) \vee \left(\begin{array}{l}
\big(own_{current} = (L, M) \wedge own_{object} = (L, N)\big) \\
\wedge \\
\left(\begin{array}{l}
(isObject_v) \Rightarrow \\
\\
\quad \big(\texttt{getJHorH}(v).\textit{isArray} \wedge \neg \texttt{getJHorH}(v).\textit{isGlobal}\big) \vee \\
\quad \big(\neg \texttt{getJHorH}(v).\textit{isArray} \wedge \texttt{getJHorH}(v).\textit{entryPoint} \neq \texttt{temporary}\big)
\end{array} \right)
\end{array} \right)
\end{aligned}$$

checkThrow, *checkInvokeDefinite*, *checkInvokeVirtual* : Owner × Owner × Owner × boolean × boolean × EntryPoint × Transient → boolean

checkThrow$(own_{current}, own_{applet}, own_{object}, isArray_{object}, isGlobal_{object}, entryPoint_{object}, transient_{object}) =$

checkInvokeDefinite$(own_{current}, own_{applet}, own_{object}, isArray_{object}, isGlobal_{object}, entryPoint_{object}, transient_{object}) =$

checkInvokeVirtual$(own_{current}, own_{applet}, own_{object}, isArray_{object}, isGlobal_{object}, entryPoint_{object}, transient_{object}) =$

$$
\begin{array}{l}
\big(own_{current} = (\texttt{JCRE}, M)\big) \ \vee \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\big(own_{current} = (L, M) \wedge \ own_{object} = (L, N)\big) \vee \\
\big(isArray_{object} \wedge isGlobal_{object}\big) \ \vee \\
\big(\neg isArray_{object} \wedge entryPoint_{object} \neq \texttt{no}\big)
\end{array} \right) \\
\qquad\qquad \wedge \\
\left(\begin{array}{l}
isArray_{object} \ \Rightarrow \\
\quad \big(transient_{object} = \texttt{CLEAR_ON_DESELECT}\big) \ \Rightarrow \\
\qquad \big(own_{current} = (L, M) \wedge own_{applet} = (L, N)\big)
\end{array} \right)
\end{array} \right)
\end{array}
$$

checkInvokeInterface : Owner × Owner × Owner × boolean × boolean × EntryPoint × Transient × Class × Interface × boolean × Class → boolean

checkInvokeInterface$(own_{current}, own_{applet}, own_{obj}, isArray_{obj}, isGlobal_{obj}, entryPoint_{obj}, transient_{obj}, class_{obj}, iface, active, obj_{applet}) =$

$$
\begin{array}{l}
\big(own_{current} = (\texttt{JCRE}, M)\big) \ \vee \\
\left(\begin{array}{l}
\left(\begin{array}{l}
\big(own_{current} = (L, M) \wedge \ own_{object} = (L, N)\big) \vee \\
\big(isArray_{object} \wedge isGlobal_{object}\big) \ \vee \\
\big(\neg isArray_{object} \wedge entryPoint_{object} \neq \texttt{no}\big) \ \vee \\
\left(\begin{array}{l}
class_{object} \preceq iface \ \wedge \\
\\
iface \preceq \texttt{java.lang.Shareable}
\end{array}\right)
\end{array}\right) \\
\\
\wedge \\
\\
\left(\begin{array}{l}
isArray_{object} \ \Rightarrow \\
\\
\big(transient_{object} = \texttt{CLEAR_ON_DESELECT}\big) \ \Rightarrow \\
\\
\big(own_{current} = (L, M) \wedge own_{applet} = (L, N)\big)
\end{array}\right) \\
\\
\wedge \\
\\
\neg \left(\begin{array}{l}
iface \preceq \texttt{javacard.framework.MultiSelectable} \ \wedge \\
\\
obj_{applet} \not\preceq \texttt{javacard.framework.MultiSelectable} \ \wedge \\
own_{current} = (L, M) \wedge \ own_{object} = (N, O) \ \wedge \\
L \neq N \ \wedge \neg active
\end{array}\right)
\end{array}\right)
\end{array}
$$

checkCast : Owner $\times$ Owner $\times$ Owner $\times$ boolean $\times$ boolean $\times$ EntryPoint $\times$ Transient $\times$ ReferenceType $\times$ ReferenceType $\rightarrow$ boolean

checkCast$(own_{current}, own_{applet}, own_{object}, isArray_{object}, isGlobal_{object}, entryPoint_{object}, transient_{object}, refType_{object}, \tau) =$

$$\begin{array}{l} \big(own_{current} = (\texttt{JCRE}, M)\big) \ \vee \\ \left(\begin{array}{l} \left(\begin{array}{l} \big(own_{current} = (L,M) \wedge \ own_{object} = (L,N)\big) \vee \\ \big(isArray_{object} \wedge isGlobal_{object}\big) \ \vee \\ \big(\neg isArray_{object} \wedge entryPoint_{object} \neq \texttt{no}\big) \ \vee \\ \left(\begin{array}{l} \tau \in \text{InterfaceType} \wedge \\ \\ refType_{object} \preceq \tau \wedge \\ \\ \tau \preceq \texttt{java.lang.Shareable} \end{array} \right) \end{array} \right) \\ \\ \wedge \\ \\ \left(\begin{array}{l} isArray_{object} \ \Rightarrow \\ \\ \big(transient_{object} = \texttt{CLEAR_ON_DESELECT}\big) \ \Rightarrow \\ \\ \big(own_{current} = (L,M) \wedge own_{applet} = (L,N)\big) \end{array} \right) \end{array} \right) \end{array}$$

checkArrayLoad : Owner $\times$ Owner $\times$ Owner $\times$ ReferenceValue $\rightarrow$ boolean

checkArrayLoad$(own_{current}, own_{applet}, own_{object}, (\texttt{r}, \mathit{loc}_{array}, (\mathit{m}_q, pc_q))) =$

$$\begin{array}{l} \big(own_{current} = (\texttt{JCRE}, M)\big) \ \vee \\ \left(\begin{array}{l} \left(\big(own_{current} = (L,M) \wedge \ own_{object} = (L,N)\big) \vee \texttt{getJHorH}(\mathit{loc}_{array}).\mathit{isGlobal} \right) \\ \\ \wedge \\ \\ \left(\begin{array}{l} \big(\texttt{getJHorH}(\mathit{loc}_{array}).\mathit{transient} = \texttt{CLEAR_ON_DESELECT}\big) \ \Rightarrow \\ \\ \big(own_{current} = (L,M) \wedge own_{applet} = (L,N)\big) \end{array} \right) \end{array} \right) \end{array}$$

checkArrayStore : Owner × Owner × Owner × ReferenceValue × Value × boolean → boolean

checkArrayStore$(own_{current}, own_{applet}, own_{object}, (\mathtt{r}, \mathit{loc}_{array}, (\mathit{m}_q, pc_q)), (t, v, (\mathit{m}, pc)), isObject_v) =$

$$\begin{array}{l} \big(own_{current} = (\mathtt{JCRE}, M)\big) \ \vee \\ \left(\begin{array}{l} \left(\begin{array}{l} \big(own_{current} = (L, M) \wedge \ own_{object} = (L, N)\big) \vee \mathtt{getJHorH}(\mathit{loc}_{array}).\mathit{isGlobal} \end{array} \right) \\ \qquad \wedge \\ \qquad \left(\begin{array}{l} \big(\mathtt{getJHorH}(\mathit{loc}_{array}).\mathit{transient} = \mathtt{CLEAR_ON_DESELECT}\big) \ \Rightarrow \\ \\ \big(own_{current} = (L, M) \wedge own_{applet} = (L, N)\big) \end{array} \right) \\ \qquad \wedge \\ \quad \left(\begin{array}{l} (isObject_v) \ \Rightarrow \\ \\ \big(\mathtt{getJHorH}(v).\mathit{isArray} \ \wedge \neg \mathtt{getJHorH}(v).\mathit{isGlobal}\big) \ \vee \\ \big(\neg \mathtt{getJHorH}(v).\mathit{isArray} \ \wedge \mathtt{getJHorH}(v).\mathit{entryPoint} \neq \mathtt{temporary}\big) \end{array} \right) \end{array} \right) \end{array}$$

$checkMakeTransient$: Owner × Owner × Transient → boolean

$checkMakeTransient(own_{current}, own_{applet}, transient_{object}) =$

$$\big(own_{current} = (\mathtt{JCRE}, M)\big) \ \vee \left(\begin{array}{l} \big(transient_{object} = \mathtt{CLEAR_ON_DESELECT}\big) \ \Rightarrow \\ \\ \big(own_{current} = (L, M) \wedge own_{applet} = (L, N)\big) \end{array} \right)$$

3.9 Carmel Operational Semantics

The small step semantics is the smallest relation $P \Big| \begin{array}{c} Config \\ Config \end{array} \Rightarrow$ defined by the rules below.

$P \Big| \begin{array}{c} C_1 \\ C_2 \end{array} \Rightarrow$ may be read as "when C_1 is a valid configuration at a program point in P, then configuration C_1 transitions to configuration C_2".

3.9.1 The Core Language [C1]

3.9.1.1 `nop` instruction

$$\frac{m_n.\mathit{instructionAt}(pc_n) = \texttt{nop}}{\begin{array}{l}{}_P\big| \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \ \Rightarrow \\ \quad \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n\rangle\rangle\end{array}}$$

3.9.1.2 `push` *t c* instruction

$$\frac{\begin{array}{lcl} m_n.\mathit{instructionAt}(pc_n) & = & \texttt{push } t \ c \\ t \in \{\texttt{b}, \texttt{s}, \texttt{i}, \texttt{r}\} & & \\ t \in \{\texttt{b}, \texttt{s}\} & \Rightarrow & t_2 = \texttt{s} \\ t = \texttt{i} & \Rightarrow & t_2 = \texttt{i} \\ t = \texttt{r} & \Rightarrow & t_2 = \texttt{r} \end{array}}{\begin{array}{l}{}_P\big| \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \ \Rightarrow \\ \quad \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n :: (t_2, c, (m_n, pc_n))\rangle\rangle\end{array}}$$

3.9.1.3 `pop` *p* instruction

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{pop } p \\ p \in \mathbb{N}_0 \\ S_n = T_2 : T_1 \\ p = \mathit{nbWords}(T_1) \end{array}}{\begin{array}{l}{}_P\big| \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \ \Rightarrow \\ \quad \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, T_2\rangle\rangle\end{array}}$$

3.9.1.4 `dup` *p d* instruction

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{dup } p \ q \\ p, q \in \mathbb{N}_0 \\ S_n = T_2 : T_1 \\ S_n = T_4 : T_3 \\ \mathit{nbWords}(T_1) = p \\ \mathit{nbWords}(T_3) = q \\ S' = T_4 : T_1 : T_3 \end{array}}{\begin{array}{l}{}_P\big| \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \ \Rightarrow \\ \quad \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle\rangle\end{array}}$$

3.9.1.5 `swap` p_1 p_2 instruction

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{swap } p_1 \ p_2 \\ p_1, p_2 \in \mathbb{N}_0 \\ S_n = T_3 : T_2 : T_1 \\ p_1 = \mathit{nbWords}(T_1) \\ p_2 = \mathit{nbWords}(T_2) \\ S' = T_3 : T_1 : T_2 \end{array}}{\begin{array}{l}{}_P\big| \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \ \Rightarrow \\ \quad \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle\rangle\end{array}}$$

3.9.1.6 `numop` *t op* t_{opt} instruction

- Binary Operations:

$$\frac{\begin{array}{l}\big((m_n.\mathit{instructionAt}(pc_n) = \texttt{numop}\ \ t\ \ binop) \vee (m_n.\mathit{instructionAt}(pc_n) = \texttt{numop}\ \ t\ \ binop\ \ t_{opt})\big)\\ t \in \{\texttt{s},\texttt{i}\}\\ S_n = S{::}(t, c_1, (m_p, pc_p)){::}(t, c_2, (m_q, pc_q))\\ \neg\big(binop \in \{\texttt{div},\texttt{rem}\} \wedge c_2 = 0\big)\ \Rightarrow\\ \quad (t, r, (m_n, pc_n)) = \mathit{applyBinary}(binop, (t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q)), (m_n, pc_n))\\ \quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle\!\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S{::}(t, r, (m_n, pc_n))\rangle\!\rangle\\ \big(binop \in \{\texttt{div},\texttt{rem}\} \wedge c_2 = 0\big)\ \Rightarrow\\ \quad SF' = \mathit{catchException}\left(\begin{array}{l}\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle,\\ \\ (\mathtt{r}, \mathit{loc}_{\texttt{ArithmeticException}}, (1,1)), \texttt{java.lang.ArithmeticException}, ctd_n, io_n\end{array}\right)\end{array}}{P\ \big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\rangle\ \Rightarrow\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle}$$

- Unary Operations:

$$\frac{\begin{array}{l}m_n.\mathit{instructionAt}(pc_n) = \texttt{numop}\ \ t\ \ \texttt{neg}\\ t \in \{\texttt{s},\texttt{i}\}\\ S_n = S{::}(t, c, (m, apc))\\ (t, v, (m_n, pc_n)) = \mathit{applyUnary}(\texttt{neg}, (t, c, (m, apc)), (m_n, pc_n))\end{array}}{P\ \big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\rangle\ \Rightarrow\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S{::}(t, v, (m_n, pc_n))\rangle\!\rangle\rangle}$$

$$\frac{\begin{array}{l}m_n.\mathit{instructionAt}(pc_n) = \texttt{numop}\ \ t\ \ \texttt{to}\ \ t_{opt}\\ t \in \{\texttt{s},\texttt{i}\}\\ S_n = S{::}(t, c, (m, apc))\\ (t_{opt}, v, (m_n, pc_n)) = \mathit{applyUnary}(\texttt{to}, t_{opt}, (t, c, (m, apc)), (m_n, pc_n))\end{array}}{P\ \big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\rangle\ \Rightarrow\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S{::}(t_{opt}, v, (m_n, pc_n))\rangle\!\rangle\rangle}$$

3.9.1.7 `load` *t i* instruction

$$\frac{\begin{array}{l}m_n.\mathit{instructionAt}(pc_n) = \texttt{load}\ \ t\ \ j\\ j \in \mathbb{N}_0\\ t \in \{\texttt{s},\texttt{i},\texttt{r}\}\end{array}}{P\ \big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\rangle\ \Rightarrow\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::_{\ldots} ::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n{::}L_n(j)\rangle\!\rangle\rangle}$$

3.9.1.8 `store` *t i* instruction

$$
\begin{array}{lcl}
m_n.\mathit{instructionAt}(pc_n) & = & \texttt{store}\ t\ j \\
t \in \{\texttt{s}, \texttt{i}, \texttt{r}\} & & \\
j \in \mathbb{N}_0 & & \\
S_n = S{::}(t, v, (m, apc)) & & \\
(t \in \{\texttt{s}, \texttt{i}\} & \Rightarrow & (m_q = m_n \wedge pc_q = pc_n)) \\
(t = \texttt{r} & \Rightarrow & (m_q = m \wedge pc_q = apc))
\end{array}
$$

$$
{}_P\big|\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle\!\rangle \Rightarrow
\langle\!\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc)), L_n[j \mapsto (t, v, (m_q, pc_q))], S\rangle\!\rangle
$$

3.9.1.9 `inc` *t j c* instruction

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{inc}\ t\ j\ c \\
t \in \{\texttt{s}, \texttt{i}\} \\
j \in \mathbb{N}_0 \\
c \in \mathbb{Z} \\
(t, a, (m, apc)) = L_n(j) \\
(t, r, (m_n, pc_n)) = \mathit{applyBinary}(\texttt{add}, (t, c, (m_n, pc_n)), (t, a, (m, apc)), (m_n, pc_n))
\end{array}
$$

$$
{}_P\big|\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle\!\rangle \Rightarrow
\langle\!\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc)), L_n[j \mapsto (t, r, (m_n, pc_n))], S_n\rangle\rangle
$$

3.9.1.10 `goto` *addr* instruction

$$
m_n.\mathit{instructionAt}(pc_n) = \texttt{goto}\ addr
$$

$$
{}_P\big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \Rightarrow
\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle loc_n, itd_n, ctd_n, io_n, (m_n, addr), L_n, S_n\rangle\rangle
$$

3.9.1.11 `if` *t op* $\texttt{null}_{opt}$ `goto` *addr* instruction

Then there are two cases, depending on whether the optional keyword is present:

1. $m_n.\mathit{instructionAt}(pc_n) = \texttt{if}\ t\ op\ \texttt{goto}\ addr$. Then $t \in \{\texttt{r}, \texttt{s}\}$ and *op* ∈ ComparisonOperator. If $t = \texttt{r}$ then $op \in \{\texttt{eq}, \texttt{ne}\}$.

$$
\begin{array}{l}
m.\mathit{instructionAt}(pc_n) = \texttt{if}\ t\ op\ \texttt{goto}\ addr \\
t \in \{\texttt{r}, \texttt{s}\} \\
S_n = S'{::}(t, c_2, (m_p, pc_p)){::}(t, c_1, (m_q, pc_q)) \\
\mathit{applyBinary}(op, (t, c_1, (m_q, pc_q)), (t, c_2, (m_p, pc_p)), (m_n, pc_n)) \Rightarrow \\
\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, addr), L_n, S'\rangle \\
\\
\neg \mathit{applyBinary}(op, (t, c_1, (m_q, pc_q)), (t, c_2, (m_p, pc_p)), (m_n, pc_n)) \Rightarrow \\
\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle
\end{array}
$$

$$
{}_P\big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle {::}\ldots{::}\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \Rightarrow
\langle R, K, H, I, HID, JH, CHN, SF'\rangle
$$

2. m_n.*instructionAt*$(pc_n) =$ `if` *t op* `null goto` *addr* . Then *t* = `r` and *op* $\in \{$`eq`, `ne`$\}$ *or* *t* = `s` and comparison is made with the `short` value 0.

$$
\frac{\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{if}\ t\ op\ \texttt{null goto}\ addr \\
S_n = S'::(t, v, (m, apc)) \\
t \in \{\texttt{r}, \texttt{s}\} \\
(t = \texttt{r})\ \Rightarrow \\
\quad \mathit{applyBinary}(op, (t, v, (m, apc)), (\texttt{r}, \texttt{null}, (m_n, pc_n)), (m_n, pc_n))\ \Rightarrow \\
\quad\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, \mathit{addr}), L_n, S'\rangle \\
\\
\quad \neg\mathit{applyBinary}(op, (t, v, (m, apc)), (\texttt{r}, \texttt{null}, (m_n, pc_n)), (m_n, pc_n))\ \Rightarrow \\
\quad\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\\
(t = \texttt{s})\ \Rightarrow \\
\quad \mathit{applyBinary}(op, (t, v, (m, apc)), (\texttt{s}, 0, (m_n, pc_n)), (m_n, pc_n))\ \Rightarrow \\
\quad\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, \mathit{addr}), L_n, S'\rangle \\
\\
\quad \neg\mathit{applyBinary}(op, (t, v, (m, apc)), (\texttt{s}, 0, (m_n, pc_n)), (m_n, pc_n))\ \Rightarrow \\
\quad\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle
\end{array}}{P \mid \langle R, K, H, I, HID, JH, CHN, \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle\ \Rightarrow \langle R, K, H, I, HID, JH, CHN, SF'\rangle}
$$

3.9.1.12 `lookupswitch` *t* $(k_i$ `=>` $apc_i)_1^r$, `default =>` $apc_{default}$ instruction

$$
\frac{\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{lookupswitch}\ t\ (k_i\ \texttt{=>}\ apc_i)_1^r\ ,\ \texttt{default =>}\ apc_{default} \\
t \in \{\texttt{s}, \texttt{i}\} \\
k_1, \ldots, k_r \in \mathbb{N}_0 \\
S_n = S'::(t, key, (m, apc)) \\
(\exists i \in \{1, \ldots, r\}\ .\ key = k_i)\ \Rightarrow \\
\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, apc_i), L_n, S'\rangle \\
\\
\neg(\exists i \in \{1, \ldots, r\}\ .\ key = k_i)\ \Rightarrow \\
\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, apc_{default}), L_n, S'\rangle
\end{array}}{P \mid \langle R, K, H, I, HID, JH, CHN, \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle\ \Rightarrow \langle R, K, H, I, HID, JH, CHN, SF'\rangle}
$$

3.9.1.13 `tableswitch` *t low* `=>` $(apc_i)_1^r$, `default =>` $apc_{default}$ instruction

$$
\frac{\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{tableswitch}\ t\ low\ \texttt{=>}\ (apc_i)_1^r\ ,\ \texttt{default =>}\ apc_{default} \\
t \in \{\texttt{s}, \texttt{i}\} \\
low \in \mathbb{N}_0 \\
S_n = S'::(t, key, (m, apc)) \\
(low \leq key < (low + r))\ \Rightarrow \\
\quad i = ((key - low) + 1) \\
\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, apc_i), L_n, S'\rangle \\
\\
((key < low) \vee (key \geq (low + r)))\ \Rightarrow \\
\quad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle loc_n, itd_n, ctd_n, io_n, (m_n, apc_{default}), L_n, S'\rangle
\end{array}}{P \mid \langle R, K, H, I, HID, JH, CHN, \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::_{\ldots} ::\langle\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle\ \Rightarrow \langle R, K, H, I, HID, JH, CHN, SF'\rangle}
$$

3.9.2 The Object Language [O2]

3.9.2.1 `new` τ instruction

Then there are two cases: array objects and non-array objects.

1. Array objects

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{new } \tau\texttt{[ ]} \\
S_n = S' :: (\texttt{s}, \ell, (m, apc)) \\
(\ell \geq 0) \Rightarrow \\
\quad \mathit{loc} \notin \mathit{dom}(H) \\
\quad \mathit{loc} \notin \mathit{dom}(I) \\
\quad \mathit{loc} \notin \mathit{dom}(JH) \\
\quad \mathit{loc} \neq \texttt{null} \\
\quad a \in \text{Object} \\
\quad a.\mathit{type} = \tau \\
\quad a.\mathit{refType} = \tau\texttt{[ ]} \\
\quad a.\mathit{isArray} = \texttt{true} \\
\quad a.\mathit{owner} = \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner} \\
\quad a.\mathit{entryPoint} = \texttt{no} \\
\quad a.\mathit{isGlobal} = \texttt{false} \\
\quad a.\mathit{transient} = \texttt{NOT_TRANSIENT} \\
\quad a.\mathit{creationPoint} = (m_n, pc_n) \\
\quad a.\mathit{length} = \ell \\
\quad a.\mathit{heapID} = HID \\
\quad dom(a.\mathit{values}) = \{i \in \mathbb{N}_0 \mid 0 \leq \imath < \ell\} \\
\quad \forall\, i \in \mathbb{N}_0 : \;\; 0 \leq \imath < \ell : a.\mathit{values}(i) = \begin{cases} (\texttt{r}, \texttt{null}, (m_n, pc_n)), & \tau \in \text{ReferenceType} \\ (\texttt{b}, 0, (m_n, pc_n)), & \tau = \texttt{byte} \\ (\texttt{s}, 0, (m_n, pc_n)), & \tau = \texttt{short} \\ (\texttt{i}, 0, (m_n, pc_n)), & \tau = \texttt{int} \end{cases} \\
\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' :: (\texttt{r}, \mathit{loc}, (m_n, pc_n)) \rangle \\
\quad HID' = (HID + 1) \\
\quad (ctd_n = 0) \Rightarrow \\
\quad\quad H' = H[\mathit{loc} \mapsto a] \\
\quad\quad JH' = JH \\
\quad (ctd_n = 1) \Rightarrow \\
\quad\quad H' = H \\
\quad\quad JH' = JH[\mathit{loc} \mapsto a] \\
(\ell < 0) \Rightarrow \\
\quad H' = H \\
\quad JH' = JH \\
\quad HID' = HID \\
\quad SF' = \mathit{catchException} \left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{NegativeArraySizeException}}, (m, \texttt{java.lang.NegativeArraySizeException})), \texttt{java.lang.NegativeArraySizeException}, ctd_n, io_n \end{array} \right) \\
\hline
P \mid \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle \rangle \Rightarrow \\
\quad \langle R, K, H', I, HID', JH', CHN, SF' \rangle
\end{array}
$$

2. For class instances:

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{new}\ \ \tau \\
\tau \in \text{Class} \\
\mathit{loc} \notin \mathit{dom}(H) \\
\mathit{loc} \notin \mathit{dom}(I) \\
\mathit{loc} \notin \mathit{dom}(JH) \\
\mathit{loc} \neq \texttt{null} \\
a \in \text{Object} \\
a.\mathit{type} = \tau \\
a.\mathit{refType} = \tau \\
a.\mathit{isArray} = \texttt{false} \\
a.\mathit{owner} = \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner} \\
a.\mathit{entryPoint} = \texttt{no} \\
a.\mathit{isGlobal} = \texttt{false} \\
a.\mathit{transient} = \texttt{NOT_TRANSIENT} \\
a.\mathit{creationPoint} = (m_n, pc_n) \\
a.\mathit{length} = 0 \\
a.\mathit{heapID} = HID \\
\\
dom(a.\mathit{values}) = \{f.\mathit{id} \mid f \in \mathit{instanceFields}(\tau)\} \\
\\
\forall\, f \in \mathit{instanceFields}(\tau) : a.\mathit{values}(f.\mathit{id}) = \begin{cases}
(\texttt{r}, \texttt{null}, (m_n, pc_n)), & f.\mathit{type} \in \text{ReferenceType} \\
(\texttt{b}, d, (m_n, pc_n)), & f.\mathit{type} = \texttt{byte}, f.\texttt{hasInitValue} = \texttt{true} \wedge f.\texttt{initValue} = d \\
(\texttt{b}, 0, (m_n, pc_n)), & f.\mathit{type} = \texttt{byte}, f.\texttt{hasInitValue} = \texttt{false} \\
(\texttt{s}, d, (m_n, pc_n)), & f.\mathit{type} = \texttt{short}, f.\texttt{hasInitValue} = \texttt{true} \wedge f.\texttt{initValue} = d \\
(\texttt{s}, 0, (m_n, pc_n)), & f.\mathit{type} = \texttt{short}, f.\texttt{hasInitValue} = \texttt{false} \\
(\texttt{i}, d, (m_n, pc_n)), & f.\mathit{type} = \texttt{int}, f.\texttt{hasInitValue} = \texttt{true} \wedge f.\texttt{initValue} = d \\
(\texttt{i}, 0, (m_n, pc_n)), & f.\mathit{type} = \texttt{int}, f.\texttt{hasInitValue} = \texttt{false}
\end{cases} \\
\\
HID' = (HID + 1) \\
(ctd_n = 0)\ \Rightarrow \\
\quad H' = H[\mathit{loc} \mapsto a] \\
\quad JH' = JH \\
(ctd_n = 1)\ \Rightarrow \\
\quad H' = H \\
\quad JH' = JH[\mathit{loc} \mapsto a] \\
\hline
{}_P\vdash\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle\ \Rightarrow \\
\quad \langle R, K, H', I, HID', JH', CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::: \ldots ::: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n :: (\texttt{r}, \mathit{loc}, (m_n, pc_n))\rangle\!\rangle\rangle
\end{array}
$$

3.9.2.2 `getstatic` *f* instruction

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{getstatic}\ \ f \\
f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}\ \Rightarrow \\
\quad (\texttt{b}, c, (m, pc)) = K(f.\mathit{id}) \\
\quad (\texttt{s}, v, (m, pc)) = \mathit{toShort}((\texttt{b}, c, (m, pc)) \\
\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n :: (\texttt{s}, v, (m, pc))\rangle \\
\\
f.\mathit{type} \notin \{\texttt{boolean}, \texttt{byte}\}\ \Rightarrow \\
\quad (t, v_q, (m_q, pc_q)) = K(f.\mathit{id}) \\
\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n :: (t, v_q, (m_q, pc_q))\rangle \\
\hline
{}_P\vdash\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\ \Rightarrow \\
\quad \langle R, K, H, I, HID, JH, CHN, SF'\rangle
\end{array}
$$

3.9.2.3 `putstatic` *f* instruction

$$
\frac{
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{putstatic}\ f \\
S_n = S' :: (t, v, (m, apc)) \\
f.\mathit{type} \in \mathsf{PrimitiveType}\ \Rightarrow \\
\quad f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}\ \Rightarrow \\
\qquad (\texttt{b}, w, (m, apc)) = \mathit{fromShort}((t, v, (m, apc)), \texttt{b}) \\
\qquad K' = K[f.\mathit{id}\ \mapsto\ (\texttt{b}, w, (m_n, pc_n))] \\
\qquad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\\
\quad f.\mathit{type} \notin \{\texttt{boolean}, \texttt{byte}\}\ \Rightarrow \\
\qquad K' = K[f.\mathit{id}\ \mapsto\ (t, v, (m_n, pc_n))] \\
\qquad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\\
f.\mathit{type} \in \mathsf{ReferenceType}\ \Rightarrow \\
\quad \mathit{checkPutObjectStatic}(\texttt{getJHorH}(loc_n).\mathit{owner}, \texttt{getJHorH}(v).\mathit{isArray}, \texttt{getJHorH}(v).\mathit{isGlobal}, \texttt{getJHorH}(v).\mathit{entryPoint})\ \Rightarrow \\
\qquad K' = K[f.\mathit{id}\ \mapsto\ (t, v, (m, apc))] \\
\qquad SF' = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\\
\quad \neg \mathit{checkPutObjectStatic}(\texttt{getJHorH}(loc_n).\mathit{owner}, \texttt{getJHorH}(v).\mathit{isArray}, \texttt{getJHorH}(v).\mathit{isGlobal}, \texttt{getJHorH}(v).\mathit{entryPoint})\ \Rightarrow \\
\qquad K' = K \\
\qquad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ \\ (\texttt{r}, loc_{\texttt{SecurityException}}, (1, 1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array} \right)
\end{array}
}{
{}_P\vdash \langle R, K, H, I, HID, JH, CHN, \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle\ \Rightarrow\ \langle R, K', H, I, HID, JH, CHN, SF' \rangle
}
$$

3.9.2.4 `getfield this`$_{opt}$ *f* instruction

$$
\frac{
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{getfield } f \\
S_n = S' :: (\texttt{r}, \mathit{loc}, (m, apc)) \\
\big((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false})\big) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \;\Rightarrow \\
\quad f.\mathit{id} \in \mathit{dom}(\texttt{getJHorH}(\mathit{loc}).\mathit{values}) \\
\quad \mathit{checkGetField}(\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}) \;\Rightarrow \\
\quad\quad \big(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}\big) \;\Rightarrow \\
\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \neg(\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values})))) \;\Rightarrow \\
\quad\quad\quad\quad (\texttt{s}, v, (m_q, pc_q)) = \mathit{toShort}(H(\mathit{loc}).\mathit{values}(f.\mathit{id})) \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' :: (\texttt{s}, v, (m_q, pc_q)) \rangle\!\rangle \\
\quad\quad\quad (ctd_n = 1 \wedge (\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values}))) \;\Rightarrow \\
\quad\quad\quad\quad (\texttt{s}, v, (m_q, pc_q)) = \mathit{toShort}(JH(\mathit{loc}).\mathit{values}(f.\mathit{id})) \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' :: (\texttt{s}, v, (m_q, pc_q)) \rangle\!\rangle \\
\quad\quad \neg\big(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}\big) \;\Rightarrow \\
\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \neg(\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values})))) \;\Rightarrow \\
\quad\quad\quad\quad (t, v, (m_q, pc_q)) = H(\mathit{loc}).\mathit{values}(f.\mathit{id}) \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' :: (t, v, (m_q, pc_q)) \rangle \\
\quad\quad\quad (ctd_n = 1 \wedge (\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values}))) \;\Rightarrow \\
\quad\quad\quad\quad (t, v, (m_q, pc_q)) = JH(\mathit{loc}).\mathit{values}(f.\mathit{id}) \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' :: (t, v, (m_q, pc_q)) \rangle \\
\quad \neg \mathit{checkGetField}(\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}) \;\Rightarrow \\
\quad\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array}\right) \\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \;\Rightarrow \\
\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array}\right)
\end{array}
}{
{}_P| \;\; \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle \;\Rightarrow\; \langle R, K, H, I, HID, JH, CHN, SF' \rangle
}
$$

3.9.2.5 `getfield this` *f* instruction

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{getfield this } f \\
(\texttt{r}, \mathit{loc}, (m, apc)) = L_n(0) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \;\Rightarrow \\
\quad f.\mathit{id} \in \mathit{dom}(\texttt{getJHorH}(\mathit{loc}).\mathit{values}) \\
\quad \mathit{checkGetField}(\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}) \;\Rightarrow \\
\quad\quad (f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \;\Rightarrow \\
\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \neg(\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values})))) \;\Rightarrow \\
\quad\quad\quad\quad (\texttt{s}, v, (m_q, pc_q)) = \mathit{toShort}(H(\mathit{loc}).\mathit{values}(f.\mathit{id})) \\
\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n ::: (\texttt{s}, v, (m_q, pc_q)) \rangle \\
\quad\quad\quad (ctd_n = 1 \wedge (\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values})) \;\Rightarrow \\
\quad\quad\quad\quad (\texttt{s}, v, (m_q, pc_q)) = \mathit{toShort}(JH(\mathit{loc}).\mathit{values}(f.\mathit{id})) \\
\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n ::: (\texttt{s}, v, (m_q, pc_q)) \rangle \\
\quad\quad \neg(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \;\Rightarrow \\
\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \neg(\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values})))) \;\Rightarrow \\
\quad\quad\quad\quad (t, v, (m_q, pc_q)) = H(\mathit{loc}).\mathit{values}(f.\mathit{id}) \\
\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n ::: (t, v, (m_q, pc_q)) \rangle \\
\quad\quad\quad (ctd_n = 1 \wedge (\mathit{loc} \in dom(JH) \wedge f.\mathit{id} \in dom(JH(\mathit{loc}).\mathit{values})) \;\Rightarrow \\
\quad\quad\quad\quad (t, v, (m_q, pc_q)) = JH(\mathit{loc}).\mathit{values}(f.\mathit{id}) \\
\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S_n ::: (t, v, (m_q, pc_q)) \rangle \\
\quad \neg \mathit{checkGetField}(\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}) \;\Rightarrow \\
\quad\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array} \right) \\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \;\Rightarrow \\
\quad\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array} \right) \\
\hline
{}_P| \;\; \langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle \;\Rightarrow \\
\quad\quad \langle R, K, H, I, HID, JH, CHN, SF' \rangle
\end{array}
$$

3.9.2.6 `putfield` *f* instruction

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{putfield}\ \ f \\
S_n = S'::(\mathtt{r}, \mathit{loc}, (m_q, pc_q))::(t, v, (m, apc_2)) \\
((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false})) \\
\\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \ \Rightarrow \\
\quad f.\mathit{id} \in \mathit{dom}(\texttt{getJHorH}(\mathit{loc}).\mathit{values}) \\
\quad \mathit{checkPutField}\,(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (t, v, (m, apc_2)), (f.\mathit{type} \in \text{ReferenceType})\ ) \ \Rightarrow \\
\\
\quad\quad f.\mathit{type} \in \text{PrimitiveType} \ \Rightarrow \\
\quad\quad\quad (f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \ \Rightarrow \\
\quad\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad (\mathtt{b}, w, (m, apc_2)) = \mathit{fromShort}((t, v, (m, apc_2)), \mathtt{b}) \\
\quad\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \ \mapsto\ (\mathtt{b}, w, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\\
\quad\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(v).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad (\mathtt{b}, w, (m, apc_2)) = \mathit{fromShort}((t, v, (m, apc_2)), \mathtt{b}) \\
\quad\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(f.\mathit{id}) \ \mapsto\ (\mathtt{b}, w, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\\
\quad\quad\quad \neg(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \ \Rightarrow \\
\quad\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \ \mapsto\ (t, v, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\\
\quad\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(v).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(f.\mathit{id}) \ \mapsto\ (t, v, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\\
\quad\quad f.\mathit{type} \in \text{ReferenceType} \ \Rightarrow \\
\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \ \mapsto\ (t, v, (m, apc_2))] \\
\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle\!\rangle \\
\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(v).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(f.\mathit{id}) \ \mapsto\ (t, v, (m, apc_2))] \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle\!\rangle \\
\neg\mathit{checkPutField}\,(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (t, v, (m, apc_2)), (f.\mathit{type} \in \text{ReferenceType})\ ) \ \Rightarrow \\
\\
\quad H' = H, JH' = JH \\
\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ \\ (\mathtt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array}\right) \\
\\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \ \Rightarrow \\
\quad H' = H, JH' = JH \\
\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ \\ (\mathtt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array}\right) \\
\\
\hline
P\Big| \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle \ \Rightarrow \\
\quad \langle R, K, H', I, HID, JH', CHN, SF'\rangle\!\rangle
\end{array}
$$

3.9.2.7 `putfield this` *f* instruction

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{putfield this } f \\
(\mathtt{r}, \mathit{loc}, (m_q, pc_q)) = L_n(0) \\
S_n = S' :: (t, v, (m, apc_2)) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \;\Rightarrow \\
\quad f.\mathit{id} \in \mathit{dom}(\texttt{getJHorH}(\mathit{loc}).\mathit{values}) \\
\quad \mathit{checkPutField}\,(\;\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (t, v, (m, apc_2)), (f.\mathit{type} \in \text{ReferenceType})\;) \;\Rightarrow \\
\quad\quad f.\mathit{type} \in \text{PrimitiveType} \;\Rightarrow \\
\quad\quad\quad (f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \;\Rightarrow \\
\quad\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{isGlobal})) \;\Rightarrow \\
\quad\quad\quad\quad\quad (\mathtt{b}, w, (m, apc_2)) = \mathit{fromShort}((t, v, (m, apc_2)), \mathtt{b}) \\
\quad\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \;\mapsto\; (\mathtt{b}, w, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\quad\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(v).\mathit{isGlobal})) \;\Rightarrow \\
\quad\quad\quad\quad\quad (\mathtt{b}, w, (m, apc_2)) = \mathit{fromShort}((t, v, (m, apc_2)), \mathtt{b}) \\
\quad\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(f.\mathit{id}) \;\mapsto\; (\mathtt{b}, w, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\quad\quad\quad \neg(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \;\Rightarrow \\
\quad\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{isGlobal})) \;\Rightarrow \\
\quad\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \;\mapsto\; (t, v, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\quad\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(v).\mathit{isGlobal})) \;\Rightarrow \\
\quad\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(f.\mathit{id}) \;\mapsto\; (t, v, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle \\
\quad\quad f.\mathit{type} \in \text{ReferenceType} \;\Rightarrow \\
\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{isGlobal})) \;\Rightarrow \\
\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \;\mapsto\; (t, v, (m, apc_2))] \\
\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle\!\rangle \\
\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(v).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(v).\mathit{isGlobal})) \;\Rightarrow \\
\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(f.\mathit{id}) \;\mapsto\; (t, v, (m, apc_2))] \\
\quad\quad\quad\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'\rangle\!\rangle \\
\quad \neg\mathit{checkPutField}\,(\;\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (t, v, (m, apc_2)), (f.\mathit{type} \in \text{ReferenceType})\;) \;\Rightarrow \\
\quad\quad H' = H, JH' = JH \\
\quad\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ \\ (\mathtt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array}\right) \\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \;\Rightarrow \\
\quad H' = H, JH' = JH \\
\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ \\ (\mathtt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array}\right) \\
\hline
P \;\; \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle \;\Rightarrow \\
\quad\; \langle R, K, H', I, HID, JH', CHN, SF'\rangle\!\rangle
\end{array}
$$

3.9.2.8 `checkcast` τ instruction

$$
\begin{array}{l}
\mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{checkcast}\ \ \tau \\
\mathit{S}_n = \mathit{S}'::(\texttt{r}, \mathit{loc}, (\mathit{m}, apc)) \\
\left(\begin{array}{c} (\mathit{loc} = \texttt{null}) \vee \\ (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(\mathit{H}) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false}) \vee \\ (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(\mathit{H}) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true}) \end{array} \right) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \ \Rightarrow \\
\quad \mathit{checkCast} \left(\begin{array}{l} \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}, \tau \end{array} \right) \Rightarrow \\
\quad \big(\texttt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \tau\big) \ \Rightarrow \\
\quad\quad \mathit{SF}' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), \mathit{L}_n, \mathit{S}_n \rangle \\
\quad \neg\big(\texttt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \tau\big) \ \Rightarrow \\
\quad\quad \mathit{SF}' = \mathit{catchException} \left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n \rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{ClassCastException}}, (1, 1)), \\ \texttt{java.lang.ClassCastException}, ctd_n, io_n \end{array} \right) \\
\neg \mathit{checkCast} \left(\begin{array}{l} \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}, \tau \end{array} \right) \Rightarrow \\
\quad\quad \mathit{SF}' = \mathit{catchException} \left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n \rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1, 1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array} \right) \\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \ \Rightarrow \\
\quad\quad \mathit{SF}' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), \mathit{L}_n, \mathit{S}_n \rangle \\
\hline
{}_P\vdash \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n \rangle \rangle \ \Rightarrow \\
\quad \langle R, K, H, I, HID, JH, CHN, \mathit{SF}' \rangle
\end{array}
$$

3.9.2.9 `instanceof` τ instruction

$$m_n.\mathit{instructionAt}(pc_n) = \texttt{instanceof}\ \tau$$

$$S_n = S'::(\mathtt{r}, \mathit{loc}, (m, apc))$$

$$\left(\begin{array}{c} (\mathit{loc} = \texttt{null}) \vee \\ (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false}) \vee \\ (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true}) \end{array}\right)$$

$$(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \Rightarrow$$

$$\mathit{checkCast}\left(\begin{array}{l} \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}, \tau \end{array}\right) \Rightarrow$$

$$\big(\texttt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \tau\big) \Rightarrow$$

$$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::(\mathtt{s}, 1, (m_n, pc_n)) \rangle\!\rangle$$

$$\neg\big(\texttt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \tau\big) \Rightarrow$$

$$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::(\mathtt{s}, 0, (m_n, pc_n)) \rangle\!\rangle$$

$$\neg\mathit{checkCast}\left(\begin{array}{l} \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}, \tau \end{array}\right) \Rightarrow$$

$$SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\mathtt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array}\right)$$

$$(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \Rightarrow$$

$$\frac{SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::(\mathtt{s}, 0, (m_n, pc_n)) \rangle\!\rangle}{{}_P\big| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF' \rangle \end{array}$$

3.9.3 Method Support [Mx]

3.9.3.1 `invokedefinite` p instruction

The `invokedefinite` instruction contains a resolved method as argument. No method search through the class hierarchy is neccesary, even if the original call was made to an inherited method. The method is resolved statically i.e. before execution. The information on the original (subclass) class where the method was invoked is not present.

– Static Methods

$m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p$

$p.\mathit{isStatic}$

$p \notin \texttt{SEPARATELY_HANDLED_API_METHODS}$

$(\tau_i)_0^q \to \tau_r = p.\mathit{type}$

$S_n = S' :: (t_0, v_0, (m_{r_0}, pc_{r_0})) :: \ldots :: (t_q, v_q, (m_{r_q}, pc_{r_q}))$

$dom(L') = \{0, \ldots, q\}$

$$L'(i) = \begin{cases} (t_i, v_i, (m_{r_i}, pc_{r_i})), & \text{if } t_i = \texttt{r} \\ (t_i, v_i, (p, p.\mathit{firstAddress})), & \text{otherwise} \end{cases}$$

$F = \langle loc_n, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle$

$_P|$ $\langle R, K, H, I, HID, JH, CHN, \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle \Rightarrow$
$\langle R, K, H, I, HID, JH, CHN, \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle :: F \rangle$

– Non-static methods:

$m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p$
$\neg p.\mathit{isStatic}$
$p \notin \texttt{SEPARATELY_HANDLED_API_METHODS}$
$S_n = S' ::(\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0}))::(t_1, v_1, (m_{r_1}, pc_{r_1}))::\ldots::(t_q, v_q, (m_{r_q}, pc_{r_q}))$
$\big((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false})\big)$
$(\tau_i)_1^q \rightarrow \tau_r = p.\mathit{type}$
$(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I))\ \Rightarrow$

$\mathit{checkInvokeDefinite}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}\ )\ \Rightarrow$

$dom(L') = \{0, \ldots, q\}$
$L'(0) = (\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0}))$

$$L'(i) = \begin{cases} (t_i, v_i, (m_{r_i}, pc_{r_i})), & \text{if } t_i = \mathtt{r} \\ (t_i, v_i, (p, p.\mathit{firstAddress})), & \text{otherwise} \end{cases}$$

$F = \langle \mathit{loc}, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle$
$SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle ::F$

$\neg\mathit{checkInvokeDefinite}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}\ )\ \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle, \\ \\ (\mathtt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array}\right)$$

$(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I))\ \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle, \\ \\ (\mathtt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array}\right)$$

$$_P| \quad \langle\!\langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle \rangle\!\rangle \ \Rightarrow$$
$$\langle\!\langle R, K, H, I, HID, JH, CHN, SF' \rangle\!\rangle$$

3.9.3.2 `invokevirtual` w instruction

$$
\frac{
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{invokevirtual}\ \ w \\
(\tau_i)_1^q \rightarrow \tau_r = w.\mathit{type} \\
S_n = S' ::(\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0})) ::(t_1, v_1, (m_{r_1}, pc_{r_1})) :: \ldots ::(t_q, v_q, (m_{r_q}, pc_{r_q})) \\
\big((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false})\big) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \ \Rightarrow \\
\quad p = \mathit{methodLookup}(w.\mathit{id}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}) \neq \bot \\
\quad p \notin \texttt{SEPARATELY_HANDLED_API_METHODS} \\
\quad \mathit{checkInvokeVirtual}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}\ ) \ \Rightarrow \\
\qquad dom(L') = \{0, \ldots, q\} \\
\qquad L'(0) = (\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0})) \\
\qquad L'(i) = \begin{cases} (t_i, v_i, (m_{r_i}, pc_{r_i})), & \text{if } t_i = \mathtt{r} \\ (t_i, v_i, (p, p.\mathit{firstAddress})), & \text{otherwise} \end{cases} \\
\qquad F = \langle\!\langle \mathit{loc}, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle\!\rangle \\
\qquad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle :: F \\
\quad \neg \mathit{checkInvokeVirtual}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}\ ) \ \Rightarrow \\
\qquad SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\mathtt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array} \right) \\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \ \Rightarrow \\
\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle, \\ (\mathtt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array} \right)
\end{array}
}{
{}^{P}\big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle \ \Rightarrow \\
\quad \langle R, K, H, I, HID, JH, CHN, SF' \rangle\!\rangle
}
$$

3.9.3.3 `invokeinterface` w instruction

$m_n.\mathit{instructionAt}(pc_n) = \texttt{invokeinterface}\ \ w$
$(\tau_i)_1^q \rightarrow \tau_r = w.\mathit{type}$
$S_n = S' :: (\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0})) :: (t_1, v_1, (m_{r_1}, pc_{r_1})) :: \ldots :: (t_q, v_q, (m_{r_q}, pc_{r_q}))$
$\big((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false})\big)$
$(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \ \Rightarrow$

$p = \mathit{methodLookup}(w.\mathit{id}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}) \neq \bot$
$p \notin \texttt{SEPARATELY_HANDLED_API_METHODS}$

$\texttt{getJHorH}(\mathit{loc}).\mathit{owner} = (N, O)$
$R(O) = (\mathit{loc}_{AID}, \mathit{loc}_{applet})$

$active = |\mathit{loc}_{AID} \in \texttt{range}(CHN)| \geq 1$

$$\mathit{checkInvokeInterface}\left(\begin{array}{l}\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal},\\ \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}, p.\mathit{classI}, active, \texttt{getJHorH}(\mathit{loc}_{applet}).\mathit{refType}\end{array}\right) \Rightarrow$$

$dom(L') = \{0, \ldots, q\}$
$L'(0) = (\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0}))$

$$L'(i) = \begin{cases}(t_i, v_i, (m_{r_i}, pc_{r_i})), & \text{if } t_i = \mathtt{r}\\ (t_i, v_i, (p, p.\mathit{firstAddress})), & \text{otherwise}\end{cases}$$
$F = \langle\!\langle \mathit{loc}, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle\!\rangle$
$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle :: F$

$$\neg\mathit{checkInvokeInterface}\left(\begin{array}{l}\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal},\\ \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}, \texttt{getJHorH}(\mathit{loc}).\mathit{refType}, p.\mathit{classI}, active, \texttt{getJHorH}(\mathit{loc}_{applet}).\mathit{refType}\end{array}\right) \Rightarrow$$

$$SF' = \mathit{catchException}\left(\begin{array}{l}\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle,\\ (\mathtt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n\end{array}\right)$$

$(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \ \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l}\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle,\\ (\mathtt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n\end{array}\right)$$

$$_P\big| \ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle \ \Rightarrow$$
$$\langle R, K, H, I, HID, JH, CHN, SF' \rangle$$

3.9.3.4 `return` t_{opt} instruction

$$
\frac{\begin{array}{l}
\mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{return} \\
(\tau_i)_1^q \rightarrow \tau_r = \mathit{m}_n.\mathit{type} \\
(n = 3) \Rightarrow \\
\quad \mathit{SF}' = \langle\!\langle \mathit{loc}_1, imtd_1, cmtd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, imtd_2, ctd_n, io_n, (m_2, 20), L_2, \epsilon \rangle \\
(n > 3) \Rightarrow \\
\quad \neg(\mathit{m}_n.\mathit{isStatic}) \Rightarrow \\
\quad\quad \mathit{S}_{(n-1)} = W_2 :: (\mathtt{r}, \mathit{loc}, (\mathit{m}_{r_0}, pc_{r_0})) :: (t_1, \mathit{v}_1, (\mathit{m}_{r_1}, pc_{r_1})) :: \ldots :: (t_q, \mathit{v}_q, (\mathit{m}_{r_q}, pc_{r_q})) \\
\quad\quad \mathit{SF}' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_{(n-1)}, itd_{(n-1)}, ctd_n, io_n, (m_{(n-1)}, m_{(n-1)}.\mathit{nextAddress}(pc_{(n-1)})), L_{(n-1)}, W_2 \rangle \\
\quad (\mathit{m}_n.\mathit{isStatic}) \Rightarrow \\
\quad\quad \mathit{S}_{(n-1)} = W_2 :: (t_1, \mathit{v}_1, (\mathit{m}_{r_1}, pc_{r_1})) :: \ldots :: (t_q, \mathit{v}_q, (\mathit{m}_{r_q}, pc_{r_q})) \\
\quad\quad \mathit{SF}' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_{(n-1)}, itd_{(n-1)}, ctd_n, io_n, (m_{(n-1)}, m_{(n-1)}.\mathit{nextAddress}(pc_{(n-1)})), L_{(n-1)}, W_2 \rangle
\end{array}}{{}_P\left|\begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle \rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, \mathit{SF}' \rangle \end{array}\right.}
$$

$$
\frac{\begin{array}{l}
\mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{return}\ \ t \\
\mathit{S}_n = W_1 :: (t, \mathit{v}, (\mathit{m}, apc)) \\
(\tau_i)_1^q \rightarrow \tau_r = \mathit{m}_n.\mathit{type} \\
(n = 3) \Rightarrow \\
\quad \mathit{SF}' = \langle\!\langle \mathit{loc}_1, imtd_1, cmtd_1, io_1, m_1, pc_1, L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, imtd_2, ctd_n, io_n, m_2, 20, L_2, (t, \mathit{v}, (\mathit{m}_2, 20)) \rangle \\
(n > 3) \Rightarrow \\
\quad \neg(\mathit{m}_n.\mathit{isStatic}) \Rightarrow \\
\quad\quad \mathit{S}_{(n-1)} = W_2 :: (\mathtt{r}, \mathit{loc}, (\mathit{m}_{r_0}, pc_{r_0})) :: (t_1, \mathit{v}_1, (\mathit{m}_{r_1}, pc_{r_1})) :: \ldots :: (t_q, \mathit{v}_q, (\mathit{m}_{r_q}, pc_{r_q})) \\
\quad\quad W_3 = \begin{cases} W_2 :: (t, \mathit{v}, (\mathit{m}, apc)), & \text{if } t = \mathtt{r} \\ W_2 :: (t, \mathit{v}, (\mathit{m}_{(n-1)}, pc_{(n-1)})), & \text{otherwise} \end{cases} \\
\quad\quad \mathit{SF}' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_{(n-1)}, itd_{(n-1)}, ctd_n, io_n, m_{(n-1)}, m_{(n-1)}.\mathit{nextAddress}(pc_{(n-1)}), L_{(n-1)}, W_3 \rangle \\
\quad (\mathit{m}_n.\mathit{isStatic}) \Rightarrow \\
\quad\quad \mathit{S}_{(n-1)} = W_2 :: (t_1, \mathit{v}_1, (\mathit{m}_{r_1}, pc_{r_1})) :: \ldots :: (t_q, \mathit{v}_q, (\mathit{m}_{r_q}, pc_{r_q})) \\
\quad\quad W_3 = \begin{cases} W_2 :: (t, \mathit{v}, (\mathit{m}, apc)), & \text{if } t = \mathtt{r} \\ W_2 :: (t, \mathit{v}, (\mathit{m}_{(n-1)}, pc_{(n-1)})), & \text{otherwise} \end{cases} \\
\quad\quad \mathit{SF}' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_{(n-1)}, itd_{(n-1)}, ctd_n, io_n, m_{(n-1)}, m_{(n-1)}.\mathit{nextAddress}(pc_{(n-1)}), L_{(n-1)}, W_3 \rangle
\end{array}}{{}_P\left|\begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle \rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, \mathit{SF}' \rangle \end{array}\right.}
$$

3.9.4 The Array Language [A1]

3.9.4.1 `arraylength` instruction

$$
\frac{
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{arraylength} \\
S_n = S'::(\texttt{r}, \mathit{loc}, (m, apc)) \\
\big((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true})\big) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \;\Rightarrow \\
\quad \mathit{checkArrayLoad}\,(\;\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m, apc))\;) \;\Rightarrow \\
\quad\quad v = \texttt{getJHorH}(\mathit{loc}).\mathit{length} \\
\quad\quad \mathit{SF}' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots:: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::(\texttt{s}, v, (m_n, pc_n))\rangle \\
\quad \neg\mathit{checkArrayLoad}\,(\;\texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m, apc))\;) \;\Rightarrow \\
\quad\quad \mathit{SF}' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots:: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array}\right) \\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \;\Rightarrow \\
\quad \mathit{SF}' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots:: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array}\right)
\end{array}
}{
{}_P\vdash \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots:: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\rangle \;\Rightarrow\; \langle R, K, H, I, HID, JH, CHN, \mathit{SF}'\rangle
}
$$

3.9.4.2 `arrayload` τ instruction

$m_n.\mathit{instructionAt}(pc_n) = \texttt{arrayload}\ \ t$

$S_n = S' ::(\texttt{r}, \mathit{loc}, (m_p, pc_p)) ::(\texttt{s}, i, (m_q, pc_q))$

$\big((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true})\big)$

$(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \ \Rightarrow$

$\mathit{checkArrayLoad}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p))\ \ ) \ \Rightarrow$

$(0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length}) \ \Rightarrow$

$(t = \texttt{b}) \ \Rightarrow$

$((ctd_n = 0) \vee (ctd_n = 1 \wedge \neg(\mathit{loc} \in dom(JH) \wedge i \in dom(JH(\mathit{loc}).\mathit{values})))) \ \Rightarrow$
$(\texttt{s}, v, (m, apc)) = \mathit{toShort}(H(\mathit{loc}).\mathit{values}(i))$
$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::(\texttt{s}, v, (m, apc))\rangle$

$(ctd_n = 1 \wedge (\mathit{loc} \in dom(JH) \wedge i \in dom(JH(\mathit{loc}).\mathit{values}))) \ \Rightarrow$
$(\texttt{s}, v, (m, apc)) = \mathit{toShort}(JH(\mathit{loc}).\mathit{values}(i))$
$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::(\texttt{s}, v, (m, apc))\rangle$

$\neg(t = \texttt{b}) \ \Rightarrow$

$((ctd_n = 0) \vee (ctd_n = 1 \wedge \neg(\mathit{loc} \in dom(JH) \wedge i \in dom(JH(\mathit{loc}).\mathit{values})))) \ \Rightarrow$
$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::H(\mathit{loc}).\mathit{values}(i)\rangle$

$(ctd_n = 1 \wedge (\mathit{loc} \in dom(JH) \wedge i \in dom(JH(\mathit{loc}).\mathit{values}))) \ \Rightarrow$
$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S'::JH(\mathit{loc}).\mathit{values}(i)\rangle$

$\neg(0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length}) \ \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l}\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{ArrayIndexOutOfBoundsException}}, (1,1)), \texttt{java.lang.ArrayIndexOutOfBoundsException}, ctd_n, io_n\end{array}\right)$$

$\neg\mathit{checkArrayLoad}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p))\ \ ) \ \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l}\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n\end{array}\right)$$

$(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \ \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l}\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n\end{array}\right)$$

$$_P\big|\ \langle R, K, H, I, HID, JH, CHN, \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle \ \Rightarrow$$
$$\langle R, K, H, I, HID, JH, CHN, SF'\rangle$$

3.9.4.3 `arraystore` *t* instruction

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{arraystore}\ \ t \\
S_n = S' :: (\texttt{r}, \mathit{loc}, (m_p, pc_p)) :: (\texttt{s}, i, (m_q, pc_q)) :: (t_1, v, (m, apc_3)) \\
((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true})) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \ \Rightarrow \\
\quad \mathit{checkArrayStore}(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ ) \ \Rightarrow \\
\quad\quad (0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length}) \ \Rightarrow \\
\quad\quad\quad (t = \texttt{b}) \ \Rightarrow \\
\quad\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad (\texttt{b}, w, (m, apc_3)) = \mathit{fromShort}((t_1, v, (m, apc_3)), \texttt{b}) \\
\quad\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(i) \ \mapsto \ (\texttt{b}, w, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\quad\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad (\texttt{b}, w, (m, apc_3)) = \mathit{fromShort}((t_1, v, (m, apc_3)), \texttt{b}) \\
\quad\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(i) \ \mapsto \ (\texttt{b}, w, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\quad\quad\quad (t \in \{\texttt{s}, \texttt{i}\}) \ \Rightarrow \\
\quad\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(i) \ \mapsto \ (t_1, v, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\quad\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(i) \ \mapsto \ (t_1, v, (m_n, pc_n))] \\
\quad\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\quad\quad\quad (t = \texttt{r}) \ \Rightarrow \\
\quad\quad\quad\quad \texttt{getJHorH}(v).\mathit{isArray} = \texttt{false} \\
\quad\quad\quad\quad (\texttt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \texttt{getJHorH}(v).\mathit{refType}) \ \Rightarrow \\
\quad\quad\quad\quad\quad ((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad\quad H' = H[\mathit{loc}.\mathit{values}(i) \ \mapsto \ (t_1, v, (m, apc_3))] \\
\quad\quad\quad\quad\quad\quad JH' = JH \\
\quad\quad\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\quad\quad\quad\quad\quad ((ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg\texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \ \Rightarrow \\
\quad\quad\quad\quad\quad\quad H' = H \\
\quad\quad\quad\quad\quad\quad JH' = JH[\mathit{loc}.\mathit{values}(i) \ \mapsto \ (t_1, v, (m, apc_3))] \\
\quad\quad\quad\quad\quad\quad SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\quad\quad\quad\quad \neg(\texttt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \texttt{getJHorH}(v).\mathit{refType}) \ \Rightarrow \\
\quad\quad\quad\quad\quad H' = H, JH' = JH \\
\quad\quad\quad\quad\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{ArrayStoreException}}, (1, 1)), \texttt{java.lang.ArrayStoreException}, ctd_n, io_n \end{array} \right) \\
\quad\quad\quad \neg(0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length}) \ \Rightarrow \\
\quad\quad\quad\quad H' = H, JH = JH \\
\quad\quad\quad\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{ArrayIndexOutOfBoundsException}}, (1, 1)), \texttt{java.lang.ArrayIndexOutOfBoundsException}, ctd_n, io_n \end{array} \right) \\
\quad \neg\mathit{checkArrayStore}(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ ) \ \Rightarrow \\
\quad\quad H' = H, JH' = JH \\
\quad\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1, 1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array} \right) \\
(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \ \Rightarrow \\
\quad H' = H, JH' = JH \\
\quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1, 1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array} \right) \\
\hline
P \mid \ \{R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle\} \ \Rightarrow \\
\quad \{R, K, H', I, HID, JH, CHN, SF'\}
\end{array}
$$

3.9.5 The Exception support [E3]

$m_n.\mathit{instructionAt}(pc_n) = \texttt{throw}$
$S_n = S' :: (\texttt{r}, \mathit{loc}, (m, apc))$
$\big((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{false})\big)$
$(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \Rightarrow$

$\mathit{checkThrow}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}\ ) \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}, (m, apc)), \texttt{getJHorH}(\mathit{loc}).\mathit{refType}, ctd_n, io_n \end{array}\right)$$

$\neg\mathit{checkThrow}\ (\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{isArray}, \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \texttt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \texttt{getJHorH}(\mathit{loc}).\mathit{transient}\ ) \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array}\right)$$

$(\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \Rightarrow$

$$SF' = \mathit{catchException}\left(\begin{array}{l} \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle, \\ (\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1,1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array}\right)$$

$$\frac{}{{}_P\big|\ \begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\!\rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, SF'\rangle\!\rangle \end{array}}$$

3.9.6 The Subroutine Support [S]

3.9.6.1 `jsr` *addr* instruction

$$\frac{m_n.\mathit{instructionAt}(pc_n) = \texttt{jsr}\ \ \mathit{addr}}{{}_P\big|\ \begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\!\rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, \mathit{addr}), L_n, S_n :: (\texttt{ra}, m_n.\mathit{nextAddress}(pc_n), (m_n, pc_n))\rangle\!\rangle\!\rangle \end{array}}$$

3.9.6.2 `ret` *i* instruction

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{ret}\ \ i \\ (\texttt{ra}, \mathit{addr}, (m_n, apc)) = L(i) \end{array}}{{}_P\big|\ \begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\!\rangle\!\rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, \mathit{addr}), L_n, S_n\rangle\!\rangle\!\rangle \end{array}}$$

3.9.7 The Integer and Multi-Word Operations [I]

Already considered by the less-specialised instruction set.

Chapter 4

Base Control-Flow Analysis

4.1 Introduction

In this Chapter we present our base control-flow analysis for Carmel and prove it correct with respect to the operational semantics of Chapter 3. The base control-flow analysis is specified using the constraint-based, specification-oriented and implementation-agnostic flow-logic framework of Nielson and Nielson [NN02]. Further, we show how to systematically construct from the flow-logic specification a worklist algorithm capable of generating the least solution to the constraints of the base control-flow analysis. In proving the correctness of the base control-flow analysis, we have been greatly aided by Chapter 3 of [Han05] with the approach and structure of this chapter very much based on that one.

4.2 Abstract Domains including Analysis Domains

Following the original SecSafe material and [Han05], in this thesis we use the notation of "overlined" domains, e.g. $\overline{\mathrm{RetAddr}}$ for abstract return addresses, to indicate abstract counterparts of concrete domains, and domains with a

hat, e.g. $\widehat{\mathsf{Val}}$, to indicate complete lattices over abstract domains.

It is customary to present the abstract domains first and then later the choice of representation functions. However, we shall be presenting each abstract domain and its representation function together as we believe it is helpful to understand immediately the nature of the correspondence between concrete and abstract entities.

4.2.1 Abstract Domains for Values

We define abstract domains and their corresponding representation functions for the values that can occur in a Carmel program: numbers, object references (for class instances and array objects), and return addresses (for subroutines).

$$\overline{\mathsf{Val}} \quad = \quad \overline{\mathsf{Num}} + \overline{\mathsf{Ref}} + \overline{\mathsf{RetAddr}}$$

The high-level representation function for values in given before; the representation functions on the right-hand-side are defined in the next few pages.

$$\beta^{R,H,JH}_{Val}(t, Y, (m_n, pc_n)) = \begin{cases} \beta^{R,H,JH}_{Ref}(t, Y, (m_n, pc_n)) & t = \mathtt{r} \\ \beta_{ReturnAddress}(t, Y, (m_n, pc_n)) & t = \mathtt{ra} \\ \beta_{Num}(t, Y, (m_n, pc_n)) & t \in \{\mathtt{b}, \mathtt{s}, \mathtt{i}\} \end{cases}$$

4.2.1.1 Return Addresses

We simply enclose the return address value in a set for its abstract representation. There is no need for approximation since the value can be determined statically from the `jsr` *addr* instruction, the only instruction that can produce such values.

$$\overline{\mathsf{RetAddr}} \quad : \quad \mathsf{OpType} \times \mathbb{N}_0 \times \mathsf{Address}$$

$$\overline{\mathsf{RetAddr}} \quad = \quad \left\{ \; (\mathtt{ra}, apc, (m, pc)) \quad | \quad (m, apc), (m, pc) \in \mathsf{Address} \; \right\}$$

$$\beta_{ReturnAddress}(\mathtt{ra}, addr, (m_n, pc_n)) \quad = \quad \{(\mathrm{ra}, \; addr, (m_n, pc_n))\}$$

4.2.1.2 Numbers

The abstract domain for numbers is motivated by the desire to be able to represent and track a runtime value number $(t, n, (m, pc))$ as an interval (inclusive each end) from lowest number l to highest number h, such that $l \leq n \leq h$, associated with (m, pc), and augmented by a *modification count*. The intention is that we will track changes to numbers represented as intervals up to a maximum number of changes, `MAX_MOD_COUNT`, parametric to the analysis, and then map the number to the well-defined minimum and maximum values the number may hold according to its type i.e. $(t, (\bot_t, \top_t, \texttt{MAX_MOD_COUNT}), (m, pc))$.

$$\overline{\mathsf{Num}} \quad : \quad \mathsf{OpType} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}_0 \times \mathsf{Address}$$

$$\overline{\mathsf{Num}} \quad = \quad \left\{ (\mathit{t}, (l, h, mod_{count}), (\mathit{m}, pc)) \;\middle|\; \begin{array}{c} \bot_{\mathit{t}} \leq l \leq h \leq \top_{\mathit{t}} \\ 0 \leq mod_{count} \leq \mathrm{MAX_MOD_COUNT} \\ \bot_{\mathit{t}}, l, h, \top_{\mathit{t}} \in \mathbb{Z} \\ mod_{count} \in \mathbb{N}_0 \\ (\mathit{m}, pc) \in \mathsf{Address} \\ \mathit{t} \in \{\mathtt{b}, \mathtt{s}, \mathtt{i}\} \\ \bot_{\mathtt{b}} = -2^7, \top_{\mathtt{b}} = 2^7 - 1 \\ \bot_{\mathtt{s}} = -2^{15}, \top_{\mathtt{s}} = 2^{15} - 1 \\ \bot_{\mathtt{i}} = -2^{31}, \top_{\mathtt{i}} = 2^{31} - 1 \end{array} \right\}$$

$$\beta_{Num}(t, \mathit{c}, (\mathit{m}_n, pc_n)) \quad = \quad \{(\mathit{t}, (c, c, 0), (\mathit{m}_n, pc_n))\}$$

4.2.1.3 Object references

Abstract object references include 9 of the 11 attributes of concrete objects defined in Section 3.5.1. Of the 9 included attributes, only *length* and *owner* attributes differ from their concrete counterparts:

- *length* is now an interval, in line with the abstract representation of numbers, since the program analysis clause for the `new` τ`[]` operator takes an abstract number off the stack;

- *owner* records the package AID as per the concrete context, for use in the security predicates, but does not take the applet AID, since the applet could be registered with a multitude of different AIDs. Instead the second component of the abstract context is the applet's fully qualified class name. The second component of the concrete context – the applet AID – is not used in any of the security predicates at present. So currently, it is more meaningful to record the class name of the applet – this can always be changed to the default AID that should be provided with each concrete applet in the Carmel program if the security predicates ever change to additionally use the applet AID.

This approach is similar in spirit to the *textual object graphs* of [VHU92]. There is sufficient information in abstract object references to be able to check firewall predicates and array lengths for the associated object.

$$
\begin{aligned}
\overline{\mathsf{Ref}} \quad = \quad & (\mathit{type} : \mathsf{Type}) \\
\times \quad & (\mathit{refType} : \mathsf{ReferenceType}) \\
\times \quad & (\mathit{isArray} : \mathsf{boolean}) \\
\times \quad & (\mathit{owner} : (\mathit{context} : \mathsf{Context}) \times (\mathit{type} : \mathsf{Type})) \\
\times \quad & (\mathit{entryPoint} : \mathsf{EntryPoint}) \\
\times \quad & (\mathit{isGlobal} : \mathsf{boolean}) \\
\times \quad & (\mathit{transient} : \mathsf{Transient}) \\
\times \quad & (\mathit{creationPoint} : \mathsf{Address}) \\
\times \quad & (\mathit{length} : (min : \mathbb{N}_0) \times (max : \mathbb{N}_0))
\end{aligned}
$$

$$\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (\mathit{m}_n, \mathit{pc}_n)) = \begin{cases} \{\widehat{\sigma}_{\mathtt{Null}}\} & \mathit{loc} = \mathtt{null} \\ \left\{ \left(\mathtt{r}, \left(\begin{array}{l} \mathit{type} : \tau_1, \\ \mathit{refType} : \tau_2, \\ \mathit{isArray} : is, \\ \mathit{owner} : own, \\ \mathit{entryPoint} : ep, \\ \mathit{isGlobal} : global, \\ \mathit{transient} : trans, \\ \mathit{creationPoint} : (\mathit{m}, pc), \\ \mathit{length} : (min, max) \end{array} \right), (\mathit{m}_n, \mathit{pc}_n) \right) \middle| \begin{array}{lcl} \tau_1 & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{type} \\ \tau_2 & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{refType} \\ is & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray} \\ own & = & \beta^{R,H,JH}_{Own}(\mathtt{getJHorH}(\mathit{loc}).\mathit{owner}) \\ ep & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{entryPoint} \\ global & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal} \\ trans & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} \\ (\mathit{m}, pc) & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{creationPoint} \\ min & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \\ max & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \end{array} \right\} & \mathit{loc} \neq \mathtt{null} \end{cases}$$

$$\beta^{R,H,JH}_{Own}(package_{aid}, applet_{aid}) = (package_{aid}, \mathtt{getJHorH}(\mathit{loc}_{applet}).\mathit{type}),\ \mathit{R}(applet_{aid}) = (\mathit{loc}_{aid}, \mathit{loc}_{applet})$$

4.2.2 Abstract Domains for Analysis

4.2.2.1 Fundamental Complete Lattice Underpinning Analysis Domains

Having defined the abstract domains for values:

$$\overline{\mathsf{Val}} \quad = \quad \overline{\mathsf{Num}} + \overline{\mathsf{Ref}} + \overline{\mathsf{RetAddr}}$$

corresponding closely to the concrete domains, we define the fundamental complete lattice underpinning our analysis and analysis domains:

$$\widehat{\mathsf{Val}} = \left\langle \mathcal{P}(\overline{\mathsf{Val}}), \ \sqsubseteq_{Val} \right\rangle$$

where partial order $\sqsubseteq_{Val}$ is defined for $v_1.v_2 \in \widehat{\mathsf{Val}}$:

$$
\begin{aligned}
& \forall\, (t, Y, (m, pc)) \in v_1 : \\
& \quad (t \in \{\mathtt{r}, \mathtt{ra}\}) \ \Rightarrow \\
& \qquad \{(t, Y, (m, pc))\} \subseteq v_2 \\
& \quad (t \in \{\mathtt{b}, \mathtt{s}, \mathtt{i}\}) \ \Rightarrow \\
& \qquad \exists\, \{(t, Y_2, (m, pc))\} \subseteq v_2 \ . \\
v_1 \sqsubseteq_{Val} v_2 \quad \Longleftrightarrow \quad & \qquad Y = (l, h, mod_{count1}) \\
& \qquad Y_2 = (l_2, h_2, mod_{count2}) \\
& \qquad \bot_t \leq l_2 \leq l \leq h \leq h_2 \leq \top_t \\
& \qquad 0 \leq mod_{count1} \leq mod_{count2} \leq \mathrm{MAX_MOD_COUNT} \\
& \qquad \bot_t, l, l_2, h, h_2, \top_t \in \mathbb{Z} \\
& \qquad mod_{count1}, mod_{count2} \in \mathbb{N}_0
\end{aligned}
$$

By Lemma A.2 of Appendix A of [NNH10], $\langle \widehat{\mathsf{Val}}, \sqsubseteq_{Val} \rangle$ is a complete lattice with bottom $\bot_{\mathsf{Val}} = \emptyset$ and $\top_{\mathsf{Val}} = \overline{\mathsf{Val}}$ and binary least-upper bounds $\sqcup_{Val}\{v_1, v_2\}$, written infix as $v_1 \sqcup_{Val} v_2$, defined in Table 4.1. In words, the least upper bound of two values in $\widehat{\mathsf{Val}}$ depends on whether both values are numbers or not:

- if both values are not numbers sharing the same type and associated address, the least upper bound is simply the union of the two values;
- if both values are numbers sharing the same type and associated address, the least upper bound is then:
  - if the modification count of one or both of the values is $\geq$ `MAX_MOD_COUNT`, then the numeric interval component of the resulting abstract number is widened to the minimum and maximum values supported by that type i.e. $(\bot_t, \top_t, \texttt{MAX_MOD_COUNT})$;
  - otherwise, the numeric interval component of the resulting abstract number is widened to:

$$\left(\begin{array}{ll} \texttt{minimum}(first\ value\ left\ interval\ number, & second\ value\ left\ interval\ number), \\ \texttt{maximum}(first\ value\ right\ interval\ number, & second\ value\ right\ interval\ number), \\ \texttt{maximum}(first\ value\ modification\ count, & second\ value\ modification\ count) \end{array}\right)$$

$$
(t_1, Y_1, (m_1, pc_1)) \sqcup_{Val} (t_2, Y_2, (m_2, pc_2)) \quad = \quad \begin{cases} \{(t_1, (min, max, mod), (m_1, pc_1))\} & t_1 = t_2 \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \\ & m_1 = m_2 \\ & pc_1 = pc_2 \\ & Y_1 = (l_1, h_1, mod_1) \\ & Y_2 = (l_2, h_2, mod_2) \\ & min = \texttt{minimum}(l_1, l_2) \\ & max = \texttt{maximum}(h_1, h_2) \\ & mod = \texttt{maximum}(mod_1, mod_2)) \\ & mod < \texttt{MAX_MOD_COUNT} \\ \{(t_1, (\bot_t, \top_t, \texttt{MAX_MOD_COUNT}), (m_1, pc_1))\} & t_1 = t_2 \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \\ & m_1 = m_2 \\ & pc_1 = pc_2 \\ & Y_1 = (l_1, h_1, mod_1) \\ & Y_2 = (l_2, h_2, mod_2) \\ & min = \texttt{minimum}(l_1, l_2) \\ & max = \texttt{maximum}(h_1, h_2) \\ & mod = \texttt{maximum}(mod_1, mod_2)) \\ & mod \geq \texttt{MAX_MOD_COUNT} \\ \{(t_1, Y_1, (m_1, pc_1)), (t_2, Y_2, (m_2, pc_2))\} & \text{otherwise} \end{cases}
$$

Table 4.1: Binary least upper-bounds operator $\bigsqcup_{Val}$ over the complete lattice $\langle \widehat{\mathsf{Val}}, \ \sqsubseteq_{Val} \rangle$

4.2.2.2 Object State - Class instances and Array objects

The fields of a class instance may be determined statically and in a finite program is a finite (and typically small) number of fields. Therefore a map from fields to elements of $\widehat{\text{Val}}$ is an appropriate representation of a class instance's object state.

In contrast, according to the semantics of arrays in Carmel, an array's length is allowed to be up to $\top_{\mathbf{s}} = 2^{15} - 1$. While such a length is unlikely in a Carmel program, due to resource limitations, it does highlight the question of how best to represent the object state of an array object. In line with the representation of abstract numbers in Carmel, and desiring a compact representation of array object state, a map from intervals of array index numbers to elements of $\widehat{\text{Val}}$ is an appropriate representation of array object state. In particular, the default value of zero or `null` for an abstract array would have the abstract representation of zero or `null` mapped from the interval $(0, \top_{\mathbf{s}})$. To ensure finiteness in the analysis results, and to allow a trade-off between memory costs and precision, parametric to the analysis is `MAX_DOM_DYN_ARRAY` – the maximum size of the domain of a dynamic array created via the `new` τ`[]`. When a value is loaded from an array object, the array index is passed as an abstract number and the interval (l, h) extracted, and each value i in (l, h) is checked against each of the array's domain of intervals $(dl_1, dh_1), (dl_2, dh_2), \ldots, (dl_n, dh_n)$, and where $dl_j \leq i \leq dh_j$ the mapped value is added to the set of values whose LUB is pushed onto the stack as the result of the array load instruction. When a value is to be stored to an array object, the array index is passed as an abstract number and the interval (l, h) extracted. If (l, h) matches exactly one of the array's domain of intervals $(dl_1, dh_1), (dl_2, dh_2), \ldots, (dl_n, dh_n)$, then the LUB of the value to be stored and the existing value mapped against (l, h) is stored against (l, h). Otherwise, if $n <$ `MAX_DOM_DYN_ARRAY`, (l, h) is added to the array's domain of intervals and the value to be stored is mapped against (l, h). When $n \geq$ `MAX_DOM_DYN_ARRAY`, the LUB of the value to be stored and the current value mapped against the interval $(0, \top_{\mathbf{s}})$ is stored against the interval $(0, \top_{\mathbf{s}})$.

We define across class instances and array objects:

$$\widehat{\text{Object}} = (\text{Field} \cup (\mathbb{N}_0 \times \mathbb{N}_0)) : \textit{values} \longrightarrow \widehat{\text{Val}}$$

and extend point-wise the $\sqsubseteq_{Val}$ ordering to objects $o_1, o_2 \in \widehat{\text{Object}}$:

$$o_1 \sqsubseteq_{\widehat{\text{Object}}} o_2 \iff \textit{dom}(o_1.\textit{values}) \subseteq \textit{dom}(o_2.\textit{values}) \wedge \forall f \in \textit{dom}(o_1.\textit{values}) : o_1.\textit{values}(f.\textit{id}) \sqsubseteq_{Val} o_2.\textit{values}(f.\textit{id})$$

Note that when $o_1$and o_2 are array objects, $f.\textit{id} = [f, f]$. Also note that in a Carmel program P, where an object or static field specifies an array's elements e.g.

```
private static byte[] bootstrapSENCKeyData = { 77, 118, 55, 118, -86, -94, -80, 91, -87, 91, 88, -82, 11, 76, 40, -93 };
```

since we can determine statically the length and elements of the array, regardless of the value of the `MAX_DOM_DYN_ARRAY` parameter, the length is set correctly and the array's *values* accordingly. For the example above *values* would be set to:

$$[(0,0) \mapsto (\mathtt{b}, (77,77,0), (m,pc)), \ldots, (15,15) \mapsto (\mathtt{b}, (-93,-93,0), (m,pc))]$$

For the avoidance of doubt, the parameter `MAX_DOM_DYN_ARRAY` is only ever consulted during an `arraystore` *t* instruction and both statically declared arrays such as `bootstrapSENCKeyData` above and any dynamically created instruction created by the `new` τ`[ ]` instruction will have new intervals added to the array's domain of intervals only if the current size of the array's domain of interval is less than `MAX_DOM_DYN_ARRAY`, otherwise the LUB of the value to be stored and the current value mapped against the interval $(0, \top_{\mathtt{S}})$ is stored against the interval $(0, \top_{\mathtt{S}})$, as described above.

NB it is worth reiterating the heightened importance of arrays in Java Card (and so in Carmel) as a result of the lack of the Collections framework of discrete mathematical structures available in standard Java. Consequently, we have taken particular care in our choice of representation of abstract array objects including state.

4.2.2.3 Context-Sensitive Domains

To facilitate loop-bounds calculations and to reduce false positives in the detection of potentially recursive method calls, we require our analysis to be precise as possible *at each Carmel address wrt the possible*:

- JCVM machine states including transaction mechanism state and IO/APDU state (for precision of control-flow and data-flow);
- callstacks (for checking possibly recursive calls);
- class instance and array object *values* in the transaction buffer (for precision of control-flow and data-flow);
- operand stack (for analysing loops);
- local variable array (for analysing loops);

To this end, our choice of abstract context, defined in Table 4.2, includes the machine state and sufficient information from the abstract callstack to be able to check the firewall security predicates. Since infinite callstacks are a possibility, due to recursion, to ensure finiteness in the analysis results, and to allow a trade-off between memory costs and precision, parametric to the analysis is $\mathtt{k}$ – the maximum number of stack frames from the concrete callstack to include in the abstract context. For the firewall security predicates, and for a minimal level of acceptable

precision, we require always $\mathtt{k} \geq 5$ and our worklist algorithm checks this parameter. In program analysis parlance, our base control-flow analysis is a k-CFA analysis, as is the extended control-flow analysis of Chapter 5.

Context information is recorded for each address in a context-cache: $\widehat{\mathsf{ContextCache}} = \mathsf{Address} \to \mathcal{P}(\overline{\mathsf{Context}})$ and we calculate, for each address (m, pc), and for each context associated with that address:

- a possible over-approximation of the set of all method names that have invoked the method m (for checking possibly recursive calls);
- a possible over-approximation of the set of all class instance and array object *values* in the transaction buffer (for precision of control-flow and data-flow);
- abstract operand stack (for analysing loops);
- abstract local variable array (for analysing loops);

Each of the context-sensitive domains is defined in the following way:

$$\widehat{\mathsf{AbstractDomain}} = \mathsf{Address} \to \overline{\mathsf{Context}} \to X$$

that is the abstract domain is defined as a map from address to contexts in which the address may be executed, to (the abstract version of the concrete domain). For example, the abstract local variable array is defined as a map from address to contexts in which the address may be executed, and from each context to a map from local variable array indices to abstract values:

$$\widehat{\mathsf{LocalVar}} = \mathsf{Address} \to \overline{\mathsf{Context}} \to \mathbb{N}_0 \to \widehat{\mathsf{Val}}$$

where $\mathbb{N}_0 \to \widehat{\mathsf{Val}}$ is the abstract version of the local variable array i.e. as a map from natural numbers to abstract values. The representation function for each of the context-sensitive domains then shows how to produce the abstract version X of the concrete domain.

$$\overline{\text{Context}}: \quad (\overline{\text{Ref}} \times \text{TransactionDepth} \times \text{TransactionDepth} \times \text{IOState} \times \text{Address})^*$$

$$\beta^{R,H,JH,k}_{Context}(\quad \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \ldots :: \langle \mathit{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle) =$$

$$\begin{cases} \left(\begin{array}{l} \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_1, \mathtt{getJHorH}(\mathit{loc}_1).\mathit{creationPoint}), imtd_1, cmtd_1, io_1, (m_1, pc_1))\right), \\ \ldots, \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_i, \mathtt{getJHorH}(\mathit{loc}_i).\mathit{creationPoint}), imtd_i, cmtd_i, io_i, (m_i, pc_i))\right) \end{array} \right), & i < k \wedge k \geq 5 \\ \\ \left(\begin{array}{l} \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_1, \mathtt{getJHorH}(\mathit{loc}_1).\mathit{creationPoint}), imtd_1, cmtd_1, io_1, (m_1, pc_1))\right), \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_2, \mathtt{getJHorH}(\mathit{loc}_2).\mathit{creationPoint}), imtd_2, cmtd_2, io_2, (m_2, pc_2))\right), \\ \ldots, \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_{(k-2)}, \mathtt{getJHorH}(\mathit{loc}_{(k-2)}).\mathit{creationPoint}), itd_{(k-2)}, ctd_{(k-2)}, io_{(k-2)}, (m_{(k-2)}, pc_{(k-2)}))\right), \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_{(i-1)}, \mathtt{getJHorH}(\mathit{loc}_{(i-1)}).\mathit{creationPoint}), itd_{(i-1)}, ctd_{(i-1)}, io_{(i-1)}, (m_{(i-1)}, pc_{(i-1)}))\right), \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_i, \mathtt{getJHorH}(\mathit{loc}_i).\mathit{creationPoint}), imtd_i, cmtd_i, io_i, (m_i, pc_i))\right) \end{array} \right), & i \geq k \wedge k \geq 5 \end{cases}$$

Table 4.2: Abstract context representation

4.2.2.3.1 Abstract Context Cache

The abstract context cache is modelled as a map from address to contexts in which the address may be executed:

$$\widehat{\mathsf{ContextCache}} = \mathsf{Address} \to \widehat{\mathsf{Context}}$$

and define the ordering $\sqsubseteq_C$ to abstract context caches $c_1, c_2 \in \widehat{\mathsf{ContextCache}}$:

$$c_1 \sqsubseteq_C c_2 \iff \forall\, addr \in \mathit{dom}(c_1) :\ c_1(addr) \subseteq c_2(addr)$$

4.2.2.3.2 Local Variable Array

Modelling of the abstract local variable array is as a map from address to contexts in which the address may be executed, and from each context to local variable array indices to abstract values.

$$\widehat{\mathsf{LocalVar}} = \mathsf{Address} \to \overline{\mathsf{Context}} \to \mathbb{N}_0 \to \widehat{\mathsf{Val}}$$

with representation function:

$$\begin{aligned} \beta^{R,H,JH}_{LocalVar}(L) \quad = \quad & M\ . \\ & \forall\, i \in \mathit{dom}(L) : \\ & \beta^{R,H,JH}_{Val}(L(i)) \sqsubseteq_{Val} M(i) \end{aligned}$$

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract local variable arrays $l_1, l_2 \in \widehat{\mathsf{LocalVar}}$:

$$\begin{aligned} l_1 \sqsubseteq_L l_2 \iff\ & \forall\, addr \in \mathit{dom}(l_1), \\ & \forall\, ctxt \in l_1(addr), \\ & \forall\, idx \in dom(l_1(addr)(ctxt)) : \\ & l_1(addr)(ctxt)(idx) \sqsubseteq_{Val} l_2(addr)(ctxt)(idx) \end{aligned}$$

4.2.2.3.3 Operand Stack

Modelling of the abstract operand stack is as a map from address to contexts in which the address may be executed, and from each context to a sequence of abstract values. Since each Carmel program under consideration has been bytecode verified, abstract operand stacks must be of finite length.

$$\widehat{\mathsf{Stack}} = \mathsf{Address} \rightarrow \overline{\mathsf{Context}} \rightarrow (\widehat{\mathsf{Val}})^*$$

with representation function:

$$\beta^{R,H,JH}_{Stack}\big((t_1, v_1, (m_1, pc_1))::\ldots::(t_o, v_o, (m_o, pc_o))\big) = \beta^{R,H,JH}_{Val}(t_1, v_1, (m_1, pc_1))::\ldots::\beta^{R,H,JH}_{Val}(t_o, v_o, (m_o, pc_o))$$

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract operand stacks $s_1, s_2 \in \widehat{\mathsf{Stack}}$:

$$\begin{aligned}
s_1 \sqsubseteq_S s_2 \quad \Longleftrightarrow \quad & \forall\, addr \in \mathit{dom}(s_1), \\
& \forall\, ctxt \in s_1(addr) : \\
& \quad s_1(addr)(ctxt) = A_1::A_2::\ldots::A_q \wedge \\
& \quad s_2(addr)(ctxt) = B_1::B_2::\ldots::B_r \wedge \\
& \quad r \geq q \wedge \\
& \quad \forall i \in \{1, \ldots, q\} : \\
& \qquad A_i \sqsubseteq_{Val} B_i
\end{aligned}$$

4.2.2.3.4 Transactional Heap

Modelling of the abstract transactional heap is as a map from address to contexts in which the address may be executed, from each context to a map from object references to object (both class instance and array object) *values* map from fields or numeric intervals to abstract values.

$$\begin{aligned}
\widehat{\mathsf{TransactionDynamicHeap}} &= \mathsf{Address} \rightarrow \overline{\mathsf{Context}} \rightarrow \overline{\mathsf{Ref}} \rightarrow (\mathsf{Field} \cup (\mathbb{N}_0 \times \mathbb{N}_0)) : \mathit{values} \longrightarrow \widehat{\mathsf{Val}} \\
&= \mathsf{Address} \rightarrow \overline{\mathsf{Context}} \rightarrow \overline{\mathsf{Ref}} \rightarrow \widehat{\mathsf{Object}}
\end{aligned}$$

with representation function[1]:

[1] Here HEAP is JH; we use the same representation function for both the object heap and the transactional object heap, and supply JH or H as a parameter as appropriate.

$$\beta^{H,JH}_{DynamicHeap}(HEAP) = M$$

$$\iff \quad \forall\ \textit{loc} \in \textit{dom}(HEAP) :$$

$$o = \beta^{R,H,JH}_{Ref}(\mathtt{r}, \textit{loc}, HEAP(\textit{loc}).\textit{creationPoint})$$

$$\wedge \qquad \forall\ f \in \textit{dom}(HEAP(\textit{loc}).\textit{values}) :$$

$$\beta^{R,H,JH}_{Val}(HEAP(\textit{loc}).\textit{values}(f.\textit{id})) \sqsubseteq_{Val} M(o).\textit{values}(f.\textit{id})$$

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract transactional heaps $t_1, t_2 \in \widehat{\text{TransactionDynamicHeap}}$:

$$t_1 \sqsubseteq_{JH} t_2 \quad \iff \quad \forall\ addr \in \textit{dom}(t_1),$$
$$\forall\ ctxt \in t_1(addr) :$$
$$\forall\ ref \in t_1(addr)(ctxt) :$$
$$o_1 = t_1(addr)(ctxt)(ref)\ \wedge$$
$$o_2 = t_2(addr)(ctxt)(ref)\ \wedge$$
$$dom(o_1.\textit{values}) \subseteq dom(o_2.\textit{values})\ \wedge$$
$$\forall f \in \textit{dom}(o_1.\textit{values}) :$$
$$o_1.\textit{values}(f) \sqsubseteq_{Val} o_2.\textit{values}(f)$$

consistent with the ordering on $\sqsubseteq_{\widehat{\text{Object}}}$.

4.2.2.3.5 Method Names Cache

A purely semantic component, to overcome potential loss of callstack information due to our choice of context (see Table 4.2), and so to ensure at each address (m, pc) we have a conservative (i.e. a possible over-approximation) of the set of methods that may have invoked m, we developed relation $\widehat{\text{MethodNamesCache}}$. Modelling of the method name cache $\widehat{\text{MethodNamesCache}}$ is as a map from address (m, pc) to contexts in which the address may be executed, and from each context to the set of Carmel methods that may have invoked m.

$$\widehat{\text{MethodNamesCache}} = \text{Address} \rightarrow \overline{\text{Context}} \rightarrow \mathcal{P}(\text{Method})$$

$\widehat{\text{MethodNamesCache}}$ admits the following ordering $\sqsubseteq_{\widehat{MNAMES}}$ to method name caches $mnames_1, mnames_2 \in \widehat{\text{MethodNamesCache}}$:

$$\begin{aligned} mnames_1 \sqsubseteq_{\widehat{MNAMES}} mnames_2 \quad \Longleftrightarrow \quad & \forall\, addr \in \mathit{dom}(mnames_1), \\ & \forall\, ctxt \in mnames_1(addr) : \\ & \quad mnames_1(addr)(ctxt) \subseteq mnames_2(addr)(ctxt) \end{aligned}$$

4.2.2.4 Context-Insensitive Domains

4.2.2.4.1 Applet Registry

The abstract applet registry records the set of abstract object references that are registered via one of the applet `register` methods:

$$\widehat{\text{Registry}} \quad = \quad \mathcal{P}(\overline{\text{Ref}})$$

with representation function:

$$\begin{aligned} \beta^{R,H,JH}_{Registry}(R) = \quad & M\ . \\ & \forall\, aid \in \mathit{dom}(R) : \\ & \quad R(aid) = (\mathit{loc}_{JCREOwnedAID}, \mathit{loc}_{applet}) \wedge \\ & \quad \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_{applet}, \mathtt{getJHorH}(\mathit{loc}_{applet}).\mathit{creationPoint}) \sqsubseteq_{Val} M \end{aligned}$$

and define the ordering on elements $r_1, r_2 \in \widehat{\text{Registry}}$:

$$r_1 \sqsubseteq_R r_2 \quad \Longleftrightarrow \quad r_1 \sqsubseteq_{Val} r_2$$

4.2.2.4.2 Static Heap

The abstract static heap simply maps static fields to abstract values:

$$\widehat{\mathsf{StaticHeap}} = \mathsf{Field} \to \widehat{\mathsf{Val}}$$

with representation function:

$$\begin{aligned}\beta^{H}_{StaticHeap}(STATIC_HEAP) = \quad & M. \\ & \forall\ f \in \mathit{dom}(STATIC_HEAP) : \\ & \quad \beta^{R,H,JH}_{Val}(STATIC_HEAP(f.\mathit{id})) \sqsubseteq_{Val} M(f.\mathit{id})\end{aligned}$$

and define the ordering on elements $k_1, k_2 \in \widehat{\mathsf{StaticHeap}}$:

$$k_1 \sqsubseteq_K k_2 \iff \mathit{dom}(k_1) \subseteq \mathit{dom}(k_2)\ \wedge \forall\ f \in \mathit{dom}(k_1) :\ k_1(f.\mathit{id}) \sqsubseteq_{Val} k_2(f.\mathit{id})$$

4.2.2.4.3 Object Heap

Modelling of the abstract object heap is as a map from object references to object (both class instance and array object) *values* map from fields or numeric intervals to abstract values.

$$\begin{aligned}\widehat{\mathsf{ObjectHeap}} &= \overline{\mathsf{Ref}} \to (\mathsf{Field} \cup (\mathbb{N}_0 \times \mathbb{N}_0)) : \mathit{values} \longrightarrow \widehat{\mathsf{Val}} \\ &= \overline{\mathsf{Ref}} \to \widehat{\mathsf{Object}}\end{aligned}$$

with representation function[2]:

$$\begin{aligned}& \beta^{H,JH}_{DynamicHeap}(HEAP) = M \\ \iff\ & \forall\ \mathit{loc} \in \mathit{dom}(HEAP) : \\ & \quad o = \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, HEAP(\mathit{loc}).\mathit{creationPoint}) \\ \wedge\ & \quad \forall\ f \in \mathit{dom}(HEAP(\mathit{loc}).\mathit{values}) : \\ & \quad\quad \beta^{R,H,JH}_{Val}(HEAP(\mathit{loc}).\mathit{values}(f.\mathit{id})) \sqsubseteq_{Val} M(o).\mathit{values}(f.\mathit{id})\end{aligned}$$

[2]Here HEAP is *H*; we use the same representation function for both the object heap and the transactional object heap, and supply JH or H as a parameter as appropriate.

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract object heaps $h_1, h_2 \in \widehat{\text{ObjectHeap}}$:

$$
\begin{aligned}
h_1 \sqsubseteq_H h_2 \quad \Longleftrightarrow \quad & \forall\, ref \in \textit{dom}(h_1) : \\
& o_1 = h_1(ref) \ \wedge \\
& o_2 = h_2(ref) \ \wedge \\
& dom(o_1.\textit{values}) \subseteq dom(o_2.\textit{values}) \ \wedge \\
& \forall f \in \textit{dom}(o_1.\textit{values}) : \\
& \quad o_1.\textit{values}(f.\textit{id}) \sqsubseteq_{Val} o_2.\textit{values}(f.\textit{id})
\end{aligned}
$$

consistent with the ordering on $\sqsubseteq_{\widehat{\text{Object}}}$.

4.2.2.4.4 Invalidated Objects Cache

The abstract invalidated objects cache records the set of abstract object references whose related object may have been created inside a transaction that was subsequently aborted and so members of this set should be treated – for safety – simultaneously as being both equal to a `null` object reference and as a regular non-`null` object reference[3].

$$
\widehat{\text{InvalidatedReferences}} \quad = \quad \mathcal{P}(\overline{\text{Ref}})
$$

with representation function:

$$
\begin{aligned}
\beta^{R,H,JH}_{Invalidated}(I) = \quad & M \ . \\
& \forall\ \textit{loc} \in \textit{dom}(I) : \\
& \quad \beta^{R,H,JH}_{Ref}(\mathtt{r}, \textit{loc}, \mathtt{getJHorH}(\textit{loc}).\textit{creationPoint}) \sqsubseteq_{Val} M
\end{aligned}
$$

and define the ordering on elements $i_1, i_2 \in \widehat{\text{InvalidatedReferences}}$:

$$
i_1 \sqsubseteq_I i_2 \quad \Longleftrightarrow \quad i_1 \sqsubseteq_{Val} i_2
$$

[3] See Section 2.3.3.9 for fuller information.

4.2.2.4.5 Exceptions Cache

The exceptions cache records the abstract reference of the exception object, the abstract context in which the exception was thrown, and the abstract context in which the exception was caught.

$$\widehat{\mathsf{ExceptionsCache}} \quad = \quad \mathcal{P}(\overline{\mathsf{Ref}} \times \overline{\mathsf{Context}} \times \overline{\mathsf{Context}})$$

Lemma B.5.1 (on page 407) proves that whenever the concrete semantics throws an exception, the program analysis ensures, among other actions, $\widehat{\mathsf{ExceptionsCache}}$ is updated appropriately. In particular, when:

$$SF' = \textit{catchException}(SF, (\mathtt{r}, \textit{loc}, (\textit{m}_r, pc_r)), \mathtt{getJHorH}((\mathtt{r}, \textit{loc}, (\textit{m}_r, pc_r)).\textit{refType}), ctd_n, io_n)$$

Then the program analysis ensures via the HANDLE predicate:

$$\begin{array}{rlll} & \{(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \textit{loc}, (\textit{m}_r, pc_r)), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF'))\} & \sqsubseteq_E & \widehat{\mathsf{ExceptionsCache}} \\ \iff & \{(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \textit{loc}, (\textit{m}_r, pc_r)), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF'))\} & \subseteq & \widehat{\mathsf{ExceptionsCache}} \end{array}$$

and define the ordering on elements $e_1, e_2 \in \widehat{\mathsf{ExceptionsCache}}$:

$$e_1 \sqsubseteq_E e_2 \quad \iff \quad e_1 \subseteq e_2$$

4.2.2.4.6 Recursive Method Calls Cache

When an attempt to invoke a potentially recursive method m is detected at an address $addr$, we store the address $addr$, method to invoke m and the abstract context in which $addr$ was attempted to invoke m.

$$\widehat{\mathsf{RecursiveMethodsCache}} = \mathcal{P}(\mathsf{Address} \times \mathsf{Method} \times \overline{\mathsf{Context}})$$

$\widehat{\mathsf{RecursiveMethodsCache}}$ admits the following ordering $\sqsubseteq_{REC}$ to recursive method name caches $rmc_1, rmc_2 \in \widehat{\mathsf{RecursiveMethodsCache}}$:

$$rmc_1 \sqsubseteq_{\widehat{REC}} rmc_2 \quad \iff \quad rmc_1 \subseteq rmc_2$$

4.2.2.4.7 Context Graph

Whenever a possible transition from one abstract context $current$ to another abstract context $next$ occurs in the program analysis clause, then $(current, next)$ is recorded in the context graph:

$$\widehat{\mathsf{ContextGraph}} = \mathcal{P}(\overline{\mathsf{Context}} \times \overline{\mathsf{Context}})$$

$\widehat{\mathsf{ContextGraph}}$ admits the following ordering $\sqsubseteq_{CG}$ to abstract context graphs $cg_1, cg_2 \in \widehat{\mathsf{ContextGraph}}$:

$$cg_1 \sqsubseteq_{CG} cg_2 \quad \Longleftrightarrow \quad cg_1 \subseteq cg_2$$

4.2.3 Abstract Analysis Domains Define Complete Lattices

Proposition 4.2.1 (Analysis domains define complete lattices)**.** The following domains are each complete lattices:

- $\widehat{\mathsf{Registry}} = \langle \widehat{\mathsf{Registry}}, \sqsubseteq_R \rangle$
- $\widehat{\mathsf{StaticHeap}} = \langle \widehat{\mathsf{StaticHeap}}, \sqsubseteq_S \rangle$
- $\widehat{\mathsf{ObjectHeap}} = \langle \widehat{\mathsf{ObjectHeap}}, \sqsubseteq_H \rangle$
- $\widehat{\mathsf{InvalidatedReferences}} = \langle \widehat{\mathsf{InvalidatedReferences}}, \sqsubseteq_I \rangle$
- $\widehat{\mathsf{TransactionDynamicHeap}} = \langle \widehat{\mathsf{TransactionDynamicHeap}}, \sqsubseteq_{JH} \rangle$
- $\widehat{\mathsf{Context}} = \langle \widehat{\mathsf{Context}}, \sqsubseteq_C \rangle$
- $\widehat{\mathsf{LocalVar}} = \langle \widehat{\mathsf{LocalVar}}, \sqsubseteq_L \rangle$
- $\widehat{\mathsf{Stack}} = \langle \widehat{\mathsf{Stack}}, \sqsubseteq_S \rangle$
- $\widehat{\mathsf{ExceptionsCache}} = \langle \widehat{\mathsf{ExceptionsCache}}, \sqsubseteq_E \rangle$
- $\widehat{\mathsf{MethodNamesCache}} = \langle \widehat{\mathsf{MethodNamesCache}}, \sqsubseteq_{MNAMES} \rangle$
- $\widehat{\mathsf{RecursiveMethodsCache}} = \langle \widehat{\mathsf{RecursiveMethodsCache}}, \sqsubseteq_{REC} \rangle$
- $\widehat{\mathsf{ContextGraph}} = \langle \widehat{\mathsf{ContextGraph}}, \sqsubseteq_{CG} \rangle$

Proof: $\widehat{\mathsf{Val}}$ is a complete lattice under ordering $\sqsubseteq_{Val}$ as shown in Section 4.2.2.1. The orderings on domains $\widehat{\mathsf{Registry}}$, $\widehat{\mathsf{StaticHeap}}$, $\widehat{\mathsf{ObjectHeap}}$, $\widehat{\mathsf{InvalidatedReferences}}$, $\widehat{\mathsf{TransactionDynamicHeap}}$, $\widehat{\mathsf{LocalVar}}$, $\widehat{\mathsf{Stack}}$ are pointwise extensions of the ordering on $\widehat{\mathsf{Val}}$ i.e. $\sqsubseteq_{Val}$ and the result follows from Paragraph 2.15 of [DP02]. The remaining domains are powersets of finite sets, ordered by $\subseteq$, which are complete lattices by Examples 2.6 (2) on [DP02]

4.2.4 Abstract Domain for High-Level Base CFA Analysis

Having defined the abstract value- and analysis- domains, we may now define the high-level domain for the base control-flow analysis analysis as the cross-product of the analysis domains:

$$\begin{aligned}\widehat{\text{Analysis}} \quad = \quad & \widehat{\text{Registry}} \times \widehat{\text{StaticHeap}} \times \widehat{\text{ObjectHeap}} \times \\ & \widehat{\text{InvalidatedReferences}} \times \widehat{\text{TransactionDynamicHeap}} \times \\ & \widehat{\text{Context}} \times \widehat{\text{LocalVar}} \times \widehat{\text{Stack}} \times \widehat{\text{ExceptionsCache}} \times \\ & \widehat{\text{MethodNamesCache}} \times \widehat{\text{RecursiveMethodsCache}} \times \widehat{\text{ContextGraph}}\end{aligned}$$

$\widehat{\text{Analysis}}$ admits the following ordering $\sqsubseteq_{Analysis}$ to abstract analysis results:
Let $(\widehat{R}_1, \widehat{K}_1, \widehat{H}_1, \widehat{I}_1, \widehat{JH}_1, \widehat{C}_1, \widehat{L}_1, \widehat{S}_1, \widehat{E}_1, \widehat{MNAMES}_1, \widehat{REC}_1, \widehat{CG}_1), (\widehat{R}_2, \widehat{K}_2, \widehat{H}_2, \widehat{I}_2, \widehat{JH}_2, \widehat{C}_2, \widehat{L}_2, \widehat{S}_2, \widehat{E}_2, \widehat{MNAMES}_2, \widehat{REC}_2, \widehat{CG}_2) \in \widehat{\text{Analysis}}$. Then:

$$\begin{aligned}& (\widehat{R}_1, \widehat{K}_1, \widehat{H}_1, \widehat{I}_1, \widehat{JH}_1, \widehat{C}_1, \widehat{L}_1, \widehat{S}_1, \widehat{E}_1, \widehat{MNAMES}_1, \widehat{REC}_1, \widehat{CG}_1) \sqsubseteq_{Analysis} (\widehat{R}_2, \widehat{K}_2, \widehat{H}_2, \widehat{I}_2, \widehat{JH}_2, \widehat{C}_2, \widehat{L}_2, \widehat{S}_2, \widehat{E}_2, \widehat{MNAMES}_2, \widehat{REC}_2, \widehat{CG}_2) \\ \iff \quad & \widehat{R}_1 \sqsubseteq_R \widehat{R}_2 \ \wedge \\ & \widehat{K}_1 \sqsubseteq_K \widehat{H}_2 \ \wedge \\ & \widehat{H}_1 \sqsubseteq_H \widehat{H}_2 \ \wedge \\ & \widehat{I}_1 \sqsubseteq_I \widehat{I}_2 \ \wedge \\ & \widehat{JH}_1 \sqsubseteq_H \widehat{JH}_2 \ \wedge \\ & \widehat{C}_1 \sqsubseteq_C \widehat{C}_2 \ \wedge \\ & \widehat{L}_1 \sqsubseteq_L \widehat{L}_2 \ \wedge \\ & \widehat{S}_1 \sqsubseteq_S \widehat{S}_2 \ \wedge \\ & \widehat{E}_1 \sqsubseteq_E \widehat{E}_2 \ \wedge \\ & \widehat{MNAMES}_1 \sqsubseteq_{MNAMES} \widehat{MNAMES}_2 \ \wedge \\ & \widehat{REC}_1 \sqsubseteq_{REC} \widehat{REC}_2 \ \wedge \\ & \widehat{CG}_1 \sqsubseteq_{CG} \widehat{CG}_2\end{aligned}$$

and by Proposition 4.2.1 and Paragraph 2.15 of [DP02], this is simply the product/pointwise extension of the ordering on the underlying analysis domains. $\sqsubseteq_{Analysis}$ allows analysis results to be compared for the same program, and in particular to determine whether one analysis result is "less" or "smaller" than another wrt the ordering on $\langle \widehat{\text{Analysis}}, \sqsubseteq_{Analysis} \rangle$.

4.3 Analysis Specification.

The base control-flow analysis is specified using the constraint-based, specification-oriented and implementation-agnostic flow-logic framework of Nielson and Nielson [NN02]. Being a specification-oriented approach, the flow-

logic framework is used to specify what it means for an analysis result (or rather a *proposed* analysis result) to be acceptable (correct) with respect to a program.

The judgements of the flow-logic specification for the analysis of Carmel are of the form:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{instr}$$

where it is implicit:

- $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \in \widehat{\text{Analysis}}$;
- `Base-CFA` is the analysis name;
- m_n.*instructionAt*$(pc_n) =$ `instr`;
- User input parameters:
  - `k` is the maximum abstract context length i.e. the maximum number of stack frames from the concrete callstack to include in the abstract context, as explained in Section 4.2.2.3;
  - `MAX_MOD_COUNT` is the maximum number of times an abstract number may change the numeric interval it contains before it is mapped to the well-defined minimum and maximum values the number may hold according to its type *t* i.e. $(\bot_t, \top_t, \texttt{MAX_MOD_COUNT})$, as explained in Section 4.2.1.2;
  - `MAX_DOM_DYN_ARRAY` is the maximum size of the domain of a dynamic array created via the `new` τ`[]`, as explained in Section 4.2.2.2.

Intuitively the above states that the left-hand side is an *acceptable* analysis for the instruction `instr` at address m_n.*instructionAt*(pc_n) when analysed in any context $\pi \in \widehat{C}(m_n, pc_n)$ and wrt the user supplied parameters `k`, `MAX_MOD_COUNT`, `MAX_DOM_DYN_ARRAY`.

To give the reader a basic grounding in the flow-logic framework and how to read the rules, we shall discuss in detail the flow-logic rules for the following Carmel bytecode instructions:

- `nop`
- `new` τ
- `if` t $cmpop$ `goto` *addr*
- `invokevirtual` w

One of the strengths of the framework is that it is quite intuitive for those familiar with pattern-matching.

4.3.1 The `nop` instruction

The operational semantic rule for the `nop` instruction is:

$$\frac{\begin{array}{c} m_n.\mathit{instructionAt}(pc_n) = \texttt{nop} \\ pc = m_n.\mathit{nextAddress}(pc_n) \end{array}}{P \vdash \begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle\!\rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc), L_n, S_n \rangle\!\rangle \rangle\!\rangle \end{array}}$$

From the semantic rule we see:

- the callstack remains the same length – only the program counter for the top stack frame changes from (m_n, pc_n) to (m_n.*nextAddress*(pc_n));
- the JCVM transitions from (m_n, pc_n) to (m_n.*nextAddress*(pc_n)) with no changes to R, K, H, I, HID, JH
- the JCVM transitions from (m_n, pc_n) is (m_n.*nextAddress*(pc_n)) with no changes to the operand stack S_n or local variable array L_n

In Table 4.3, we detail a line-by-line explanation of the flow-logic clause for the `nop` bytecode instruction.

4.3.2 The `new` τ instruction

In Tables 4.4–4.5 we detail an explanation of the flow-logic for the `new` τ bytecode instruction, and where appropriate a cross-comparison between the flow-logic rule and the corresponding operational semantic rule.

4.3.3 The `if` t $cmpop$ `goto` *addr* instruction

To demonstrate the pattern-matching nature of many of the flow-logic rules, consider the `if` t $cmpop$ `goto` *addr* bytecode instruction shown on page 139:

- the possible successor addresses and contexts of (m_n, pc_n) are bound to π_2 and π_3 in the way familiar from the `nop` and `new` τ examples discussed in great detail in Tables 4.3 and 4.4. π_2 corresponds to the `false` branch and π_3 corresponds to the `true` branch;
- the abstract operand stack is bound and constrained, via:

$$\widehat{S}(m_n, pc_n)(\pi_1) = M :: X_1 :: X_2$$

to be a sequence of at least 2 elements, with the top-most element of the abstract operand stack bound to variable X_2, the second-from-top element bound to variable X_1 and the remainder (which may be the empty sequence) to M;

- search for all possible type-matching operands to the `if` t $cmpop$ `goto` *addr* is pattern-matched, and variables bound, in the lines:

  $\forall \{(t, Y_1, (m_p, pc_p))\} \sqsubseteq_S X_1 :$

  $\forall \{(t, Y_2, (m_q, pc_q))\} \sqsubseteq_S X_2 :$

  and then passed as parameters to the abstract version of the *applyBinary* operator:

  $\big(\textit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{true}\}\big)$

  $\big(\textit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{false}\}\big)$

  Due to the approximation introduced by our representation, the abstract version of *applyBinary* has to be able to produce both `true` and `false` for the same question, as, e.g. in abstract numbers, there may be particular pairs of values i_1 and i_2, $l_1 \leq i_1 \leq h_1$, $l_2 \leq i_2 \leq h_2$ in the intervals (l_1, h_1) and (l_2, h_2) for which i_1 $cmpop$ i_2 is `true` and other pairs of values such that i_1 $cmpop$ i_2 is `false`. For abstract object references, we have maybe-equal-and-definitely-not-equal semantics: when the fields of the abstract object references are identical, they may refer to the same object, and so the abstract version of the *applyBinary* operator produces both `true` and `false`; where at least one field differs between abstract object references, they definitely do not refer to the same object and so the abstract version of the *applyBinary* operator produces `false`. See Lemma B.5.10 on page 434 for more information;

- when *absApplyBinary* determines `false` is a possible value, then the context of the false branch is represented in the analysis: $\{\pi_2\} \sqsubseteq_C \widehat{C}(m_n, m_n.\textit{nextAddress}(pc_n))$

  We ensure the remaining stack M is represented in the analysis: $M \sqsubseteq_S \widehat{S}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2)$ and in the same way familiar from the `nop` and `new` τ examples discussed in great detail in Tables 4.3 and 4.4, we ensure the local variable array. set of methods that may have invoked m_n, and object state changes recorded in the transaction buffer, and the transition from π_1 to π_2 are represented in the analysis results:

  $\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2)$

  $\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2)$

  $\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2)$

$\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$

- similarly, when *absApplyBinary* determines `true` is a possible value, then the context of the true branch is represented in the analysis: $\{\pi_3\} \sqsubseteq_C \widehat{C}((m_n, addr))$

  We ensure the remaining stack M is represented in the analysis: $M \sqsubseteq_S \widehat{S}(m_n, addr)(\pi_3)$
  and in the same way familiar from the `nop` and `new` τ examples discussed in great detail in Tables 4.3 and 4.4, we ensure the local variable array. set of methods that may have invoked m_n, and object state changes recorded in the transaction buffer, and the transition from π_1 to π_3 are represented in the analysis results:

  $\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, addr)(\pi_3)$

  $\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, addr)(\pi_3)$

  $\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, addr)(\pi_3)$

  $\{(\pi_1, \pi_3)\} \sqsubseteq_{CG} \widehat{CG}$

4.3.4 The `invokevirtual` w bytecode instruction

In Tables 4.6–4.7 we detail an explanation of the flow-logic for the `invokevirtual` w bytecode instruction, and where appropriate a cross-comparison between the flow-logic rule and the corresponding operational semantic rule.

Table 4.3: Explanation of flow-logic rule for the `nop` bytecode instruction

Explanation of each flow-logic line	Flow-logic line
For each context/JCVM machine configuration π_1 in which the `nop` instruction may be executed	$\forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$
The next instruction/address will be executed, and its abstract context π_2 may be derived from π_1 simply by changing the last abstract stack frame's program counter to $m_n.\mathit{nextAddress}(pc_n)$)	$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$
Ensure next instruction/address will be recorded in the analysis results in the appropriate context/JCVM machine configuration	$\{\pi_2\} \sqsubseteq_C \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
Ensure the operand stack at the current address and in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{S}(m_n, pc_n)(\pi_1) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the local variable array at the current address and in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of provisional object field and array updates in the transaction buffer in current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of callers to this method in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the transition between the current context/JCVM machine configuration and the next instruction/address in the appropriate context/JCVM machine configuration is recorded.	$\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$

Table 4.4: Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 1

Explanation of each flow-logic line	Flow-logic line	Corresponding section from op sem rule
For each context/JCVM machine configuration π_1 in which the `new` τ instruction may be executed	$\forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$	
The next instruction/address will be executed, and its abstract context π_2 may be derived from π_1 simply by changing the last abstract stack frame's program counter to $m_n.\mathit{nextAddress}(pc_n)$)	$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$	
Ensure next instruction/address will be recorded in the analysis results in the appropriate context/JCVM machine configuration	$\{\pi_2\} \sqsubseteq_C \hat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$	
Create abstract object reference	$\mathsf{O}_q = \left(\mathtt{r}, \left(\begin{array}{l} \mathit{type} : \tau, \\ \mathit{refType} : \tau, \\ \mathit{isArray} : \mathtt{false}, \\ \mathit{owner} : O_n.\mathit{owner}, \\ \mathit{entryPoint} : \mathtt{no}, \\ \mathit{isGlobal} : \mathtt{false}, \\ \mathit{transient} : \mathtt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (m_n, pc_n), \\ \mathit{length} : 0, \end{array} \right), (m_n, pc_n) \right)$	$a \in \mathsf{Object}$ $a.\mathit{type} = \tau$ $a.\mathit{refType} = \tau$ $a.\mathit{isArray} = \mathtt{false}$ $a.\mathit{owner} = \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}$ $a.\mathit{entryPoint} = \mathtt{no}$ $a.\mathit{isGlobal} = \mathtt{false}$ $a.\mathit{transient} = \mathtt{NOT_TRANSIENT}$ $a.\mathit{creationPoint} = (m_n, pc_n)$ $a.\mathit{length} = 0$ $a.\mathit{heapID} = HID$
Initialise object field to default values, in the appropriate object heap	$\forall\, f \in \mathit{instanceFields}(\tau) :$ $v = \begin{cases} \widehat{\sigma}_{\mathtt{Null}}, f.\mathit{type} \in \mathsf{ReferenceType} \\ (\mathtt{b}, (d,d,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{byte}, f.\mathtt{hasInitValue} = \mathtt{true} \wedge f.\mathtt{initValue} = d \\ (\mathtt{b}, (0,0,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{byte}, f.\mathtt{hasInitValue} = \mathtt{false} \\ (\mathtt{s}, (d,d,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{short}, f.\mathtt{hasInitValue} = \mathtt{true} \wedge f.\mathtt{initValue} = d \\ (\mathtt{s}, (0,0,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{short}, f.\mathtt{hasInitValue} = \mathtt{false} \\ (\mathtt{i}, (d,d,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{int}, f.\mathtt{hasInitValue} = \mathtt{true} \wedge f.\mathtt{initValue} = d \\ (\mathtt{i}, (0,0,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{int}, f.\mathtt{hasInitValue} = \mathtt{false} \end{cases}$ $(\xi_n = 0) \;\Rightarrow\; \{v\} \sqsubseteq_H \widehat{H}(O_q).\mathit{values}(f.id)$ $(\xi_n = 1) \;\Rightarrow\; \{v\} \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id)$	$\mathit{loc} \notin \mathit{dom}(H)$ $\mathit{loc} \notin \mathit{dom}(I)$ $\mathit{loc} \notin \mathit{dom}(JH)$ $\mathit{loc} \neq \mathtt{null}$ $dom(a.\mathit{values}) = \{f.\mathit{id} \mid f \in \mathit{instanceFi}$ $\forall\, f \in \mathit{instanceFields}(\tau) :$ $a.\mathit{values}(f.\mathit{id}) = \begin{cases} (\mathtt{r}, \mathtt{null}, (m_n, pc_n) \\ (\mathtt{b}, d, (m_n, pc_n)), \\ (\mathtt{b}, 0, (m_n, pc_n)), \\ (\mathtt{s}, d, (m_n, pc_n)), \\ (\mathtt{s}, 0, (m_n, pc_n)), \\ (\mathtt{i}, d, (m_n, pc_n)), \\ (\mathtt{i}, 0, (m_n, pc_n)), \end{cases}$ $(ctd_n = 0) \;\Rightarrow$ $H' = H[\mathit{loc} \mapsto a]$ $JH' = JH$ $(ctd_n = 1) \;\Rightarrow$ $H' = H$ $JH' = JH[\mathit{loc} \mapsto a]$
Ensure the abstract object reference is pushed on top of a copy of the operand stack at the current address and in the current context/JCVM machine configuration and this is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{S}(m_n, pc_n)(\pi_1)::\{O_q\} \;\sqsubseteq_S\; \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$	$S_n::(\mathtt{r}, \mathit{loc}, (m_n, pc_n))$

Table 4.5: Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 2

Explanation of each flow-logic line	Flow-logic line
Ensure the local variable array at the current address and in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of provisional object field and array updates in the transaction buffer in current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of callers to this method in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the transition between the current context/JCVM machine configuration and the next instruction/address in the appropriate context/JCVM machine configuration is recorded.	$\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$

Table 4.6: Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 1

Explanation of each flow-logic line	Flow-logic line	Corresponding section from op sem rule
For each context/JCVM machine configuration π_1 in which the `invokevirtual w` instruction may be executed	$\forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots,$ $(O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$	
Determine the arity of the method to be invoked	$(\tau_i)_1^q \rightarrow \tau_r = w.\mathit{type}$	$(\tau_i)_1^q \rightarrow \tau_r = w.\mathit{type}$
Pattern-match the target of the method invocation and its potential operands	$\widehat{S}(m_n, pc_n)(\pi_1) = M :: X_0 :: X_1 :: \ldots :: X_q$	$S_n = S' :: (\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0})) :: (t_1, v_1, (m_{r_1}, pc_{r_1}))$
Check target for `null` or invalidated object references and throw `java.lang.NullPointerException` if so	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0 :$ $(\widehat{\sigma}_{\mathtt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}) \Rightarrow$ $\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\mathtt{NullPointerException}})$	$(\mathit{loc} = \mathtt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \Rightarrow$ $SF' = \mathit{catchException}\left(\begin{array}{l}\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (\\ \\ (\mathtt{r}, \mathit{loc}_{\mathtt{NullPointerExce}}\end{array}\right.$
For object references for which the method invocation is well-defined, check the firewall security predicate	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0 :$ $p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$ $((p \neq \bot) \wedge (p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS})) \Rightarrow$ $\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right)$	$(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \Rightarrow$ $p = \mathit{methodLookup}(w.\mathit{id}, \mathtt{getJHorH}(\mathit{loc}).\mathit{refType})$ $p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS}$ $\mathit{checkInvokeVirtual}\left(\begin{array}{l} \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} \end{array}\right)$
For object references for which the method invocation is well-defined and the firewall security predicate passes, check whether the method call is a recursive call and if so, ensure it is registered in the analysis	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0 :$ $p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$ $((p \neq \bot) \wedge (p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS})) \Rightarrow$ $\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$ $(\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, pc_n)(\pi_1)) \Rightarrow \{((m_n, pc_n), p, \pi_1)\} \sqsubseteq_{RECURSIVE} \widehat{REC}$	
For object references for which the method invocation is well-defined, if the firewall security predicate fails throw `java.lang.SecurityException`	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0 :$ $p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$ $((p \neq \bot) \wedge (p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS})) \Rightarrow$ $\neg\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$ $\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\mathtt{SecurityException}})$	$(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \Rightarrow$ $p = \mathit{methodLookup}(w.\mathit{id}, \mathtt{getJHorH}(\mathit{loc}).\mathit{refType})$ $p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS}$ $\neg\mathit{checkInvokeVirtual}\left(\begin{array}{l} \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} \end{array}\right)$ $SF' = \mathit{catchException}\left(\begin{array}{l}\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1 \\ \\ (\mathtt{r}, \mathit{loc}_{\mathtt{SecurityExcep}}\end{array}\right.$

Table 4.7: Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 2

Explanation of each flow-logic line	Flow-logic line	Corresponding section from op sem rule
For object references for which the method invocation is well-defined and the firewall security predicate passes, ensure the method call is added to the set of possible calling methods, along with those possibly calling the current method	$\pi_3 = newAbsContext(\pi_1, k, p, O_q)$ $\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$ $\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$	
Ensure first address of invoked method will be recorded in the analysis results in the appropriate context/JCVM machine configuration	$\{\pi_3\} \sqsubseteq_C \widehat{C}(p, p.\mathit{firstAddress})$	$F = \langle \mathit{loc}, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle$ $SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, i$
Ensure the parameters to the invoked method are represented in the local variable array of the first address of the invoked method in the appropriate context/JCVM machine configuration	$\{O_q\} \sqsubseteq_L \widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(0)$ $\forall j \in \{1, \ldots, q\}:$ $\forall \{(t_j, W_j, (m_j, pc_j))\} \sqsubseteq_S X_j:$ $(\tau_j = \mathtt{r} = t_j) \Rightarrow \{(t_j, W_j, (m_j, pc_j))\} \sqsubseteq_L \widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$ $(\tau_j \neq \mathtt{r}) \Rightarrow \{(t_j, W_j, (p, p.\mathit{firstAddress}))\} \sqsubseteq_L \widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$	$dom(L') = \{0, \ldots, q\}$ $L'(0) = (\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0}))$ $L'(i) = \begin{cases} (t_i, v_i, (m_{r_i}, pc_{r_i})), & \text{if } t_i = \mathtt{r} \\ (t_i, v_i, (p, p.\mathit{firstAddress})), & \text{otherwise} \end{cases}$
Ensure the empty stack is represented in the operand stack of the first address of the invoked method in the appropriate context/JCVM machine configuration	$\{\epsilon\} \sqsubseteq_S \widehat{S}(p, p.\mathit{firstAddress})(\pi_3)$	$F = \langle \mathit{loc}, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle$
Ensure the set of provisional object field and array updates in the transaction buffer in current context/JCVM machine configuration is represented in the first address of the invoked method in the appropriate context/JCVM machine configuration	$\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(p, p.\mathit{firstAddress})(\pi_3)$	
Ensure the transition between the current context/JCVM machine configuration and the next instruction/address in the appropriate context/JCVM machine configuration is recorded.	$\{(\pi_1, \pi_3)\} \sqsubseteq_{CG} \widehat{CG}$	

4.4 Theoretical Properties

In this section two fundamental properties of the base control-flow analysis and its results are formally stated and proved:

- semantic soundness
- a Moore family property

Semantic soundness is used to establish the correctness of the program analysis i.e. that the analysis results do indeed correctly reflect all semantically possible runtime behaviours for the properties of interest being captured, here control-flow. The Moore family property is used to prove all programs may be analysed, and that a least/smallest analysis exists for each program.

4.4.1 Semantic Soundness

As per [Han05, NNH10], the semantic soundness of a flow-logic analysis with a small-steps semantics is proved by establishing a *subject reduction property*. Before doing so, we must formally relate the concrete- and abstract-domains. Again, following [Han05, NNH10], we do so using representation functions and correctness relations. We have already defined the representation functions relating the concrete and abstract and analysis domains in the opening sections of this Chapter.

We express two correctness relations.

The first correctness relation relates the callstack to the analysis results and includes the context-sensitive domains. $SF\,\mathcal{R}^{R,H,JH,k}_{\mathrm{CallStack}}\,(\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{MNAMES},\widehat{CG})$ requires each stack frame, including the operand stack, local variable array, and set of calling methods in the callstack to be represented in the analysis, and the transaction

buffer to be represented in in the current address (m_n, pc_n), to be considered correct:

$$SF\ \mathcal{R}^{R,H,JH,k}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$$

$$\iff \quad SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$\wedge \quad \beta^{H,JH}_{DynamicHeap}(\mathit{JH}) \sqsubseteq_{JH} \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF))$$

$$\wedge \quad \forall i \in \{3, \ldots, n\} :$$

$$\begin{array}{lllll} & \pi_i & = & \beta^{R,H,JH,k}_{Context}\big(\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle\big) \\ \wedge & \{\pi_i\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_i, pc_i) \\ \wedge & \{\mathit{m}_1, \ldots, \mathit{m}_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_i, pc_i)(\pi_i) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(\mathit{m}_i, pc_i)(\pi_i) \\ \wedge & \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(\mathit{m}_i, pc_i)(\pi_i) \end{array}$$

The second correctness relation relates the global entities (i.e. the non-context-sensitive entities) to the analysis results, and requires the applet registry, global heap, static heap and invalidated object references be represented in the analysis results to be considered correct. It also invokes the first correctness relation and so requires the callstack to be represented in the analysis results to be considered correct.

$$\langle R, K, H, I, HID, JH, CHN, SF \rangle\ \mathcal{R}^{R,H,JH,k}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$$

$$\iff \quad SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$$

$$\begin{array}{lll} \wedge & \beta^{R,H,JH}_{Registry}(\mathit{R}) & \sqsubseteq_R \ \widehat{R} \\ \wedge & \beta^{H}_{StaticHeap}(\mathit{K}) & \sqsubseteq_K \ \widehat{K} \\ \wedge & \beta^{H,JH}_{DynamicHeap}(\mathit{H}) & \sqsubseteq_H \ \widehat{H} \\ \wedge & \beta^{R,H,JH}_{Invalidated}(I) & \sqsubseteq_I \ \widehat{I} \end{array}$$

A final preliminary before expressing and proving the subject reduction property: a concrete semantic configuration is said to be *well-formed* if it is either an initial configuration as per Section 3.7.1 or is reachable from an initial configuration by repeated application of the configuration transitions $P\Big|\begin{array}{l} Config \Rightarrow \\ Config \end{array}$ of Section 3.9. This ensures the callstack is well-formed i.e. all the stack frames below the stack frame at the top of the callstack are suspended method invocations.

4.4.2 Subject reduction theorem

Theorem 4.4.1 (Subject Reduction Theorem)**.** Let:

- $P \in$ Program
- $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$
- C = $\langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P\Big|\begin{array}{l} C \Rightarrow \\ C' \end{array}$
- C' = $\langle R', K', H', I', HID', JH', CHN', SF' \rangle$

Then:

$$\begin{array}{llll} & C & \mathcal{R}^{R,H,JH}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \Rightarrow \\ & C' & \mathcal{R}^{R,H,JH}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \\ \wedge & \{(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF'))\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

Proof:

By case inspection. Please see Appendix B.4.1 (page 320) for full details of the proof.

Corollary 4.4.2 (Base CFA calculates graph of all semantically possible transitions)**.**
Let $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$. Then the control-flow graph $\widehat{CG}$ captures all semantically possible transitions, including exceptions, between configurations as directed pairs of abstract contexts.

Proof:

Follows immediately from Theorem 4.4.1.

4.4.3 Set of Acceptable analysis results under $\models_{\texttt{Base-CFA}}$ form a Moore family

The Moore family property is used to prove all programs may be analysed, and that a least/smallest analysis exists for each program.

As per Appendix A.1 of [NNH10]: "a *Moore family* is a subset Y of a complete lattice $L = \langle L, \sqsubseteq_L \rangle$ that is closed under greatest lower bounds: $\forall Y' \subseteq Y : \sqcap Y' \in Y$. It follows a Moore family always contains a least element $\sqcap Y$ and a greatest element $\sqcap \emptyset$, which equals the greatest element $\top_L$ from L; in particular a Moore family is never empty".

Recast in terms of our `Base-CFA` analysis, the Moore family result for a Carmel program P is that the set of acceptable analysis results/solutions of a Carmel program P under $\models_{\texttt{Base-CFA}}$ form a Moore family:

$$\forall\, Y' \subseteq \{Y \mid Y \models_{\texttt{Base-CFA}} P\} : \bigsqcap Y' \in \{Y \mid Y \models_{\texttt{Base-CFA}} P\}$$

Since a Moore family is by definition never empty, this ensures every Carmel program P has at least one solution ($\sqcap \emptyset$) and a best/smallest solution ($\bigsqcap \{Y \mid Y \models_{\texttt{Base-CFA}} P\}$) .

Due to time constraints, we have been unable to spend the time necessary to produce the proof-by-cases for all instructions, and instead state it without proof, and point the interested reader in particular to Theorem 3.16 of [Han05] where a Moore family result for the set of acceptable flow-logic CFA analysis results/solutions presented in that thesis is proved for a simpler variant of Carmel, and more generally to [NNH10].

4.5 Specification

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{nop}$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{push}\ t\ c$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\{\pi_2\}$			$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
$(t=\texttt{r})$	$\Rightarrow$	$\widehat{S}(m_n,pc_n)(\pi_1)::\{\widehat{\sigma}_{\texttt{Null}}\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$(t\in\{\texttt{b},\texttt{s}\})$	$\Rightarrow$	$\widehat{S}(m_n,pc_n)(\pi_1)::\{(\texttt{s},(c,c,0),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$(t=\texttt{i})$	$\Rightarrow$	$\widehat{S}(m_n,pc_n)(\pi_1)::\{(\texttt{i},(c,c,0),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$			$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$			$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$			$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{pop}\ \ n$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\Big\{P_1,\ldots,P_f\Big\} = \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n,pc_n)(\pi_1)\Big)$$

$$\forall\, i \in \{1,\ldots,f\} :$$

$$\big(P_i = M : X \wedge \mathit{absNbWords}(X) = n\big) \ \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{dup}\ \ p\ \ q$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\Big\{P_1,\ldots,P_f\Big\} = \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n,pc_n)(\pi_1)\Big)$$

$$\forall\, i \in \{1,\ldots,f\} :$$

$$\left(\begin{array}{lll} (P_i & = X_2 : X_1) & \wedge \\ (P_i & = X_4 : X_3) & \wedge \\ (\mathit{absNbWords}(X_1) & = p) & \wedge \\ (\mathit{absNbWords}(X_3) & = q) & \end{array}\right) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$X_4 : X_1 : X_3$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{swap}\ \ n_1\ \ n_2$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(\mathit{m}_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(\mathit{m}_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(\mathit{m}_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(\mathit{m}_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(\mathit{m}_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(\mathit{m}_n,\mathit{m}_n.\mathit{nextAddress}(pc_n))))$$

$$\Big\{P_1,\ldots,P_f\Big\} = \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n,pc_n)(\pi_1)\Big)$$

$$\forall\, i \in \{1,\ldots,f\} :$$

$$\left(\begin{array}{lll} (P_i & = X_3 : X_2 : X_1) & \wedge \\ (\mathit{absNbWords}(X_1) & = n_1) & \wedge \\ (\mathit{absNbWords}(X_2) & = n_2) & \end{array}\right) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))$
$X_3 : X_1 : X_2$	$\sqsubseteq_S$	$\widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{numop}\ t\ binop$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X_1::X_2$$

$$\forall\, \{(t,(l_1,h_1,mod_1),(m_a,pc_a))\} \sqsubseteq_S X_1 :$$

$$\forall\, \{(t,(l_2,h_2,mod_2),(m_b,pc_b))\} \sqsubseteq_S X_2 :$$

$$\neg\big(binop \in \{\texttt{div},\texttt{rem}\} \wedge (l_2 = 0 = h_2)\big) \Rightarrow$$

$(t,(l_3,h_3,mod_3),(m_n,pc_n))$	$=$	$absApplyBinary(binop,(t,(l_1,h_1,mod_1),(m_a,pc_a)),(t,(l_2,h_2,mod_2),(m_b,pc_b)),(m_n,pc_n),\mathrm{MAX_MOD_COUNT})$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
$M::\{(t,(l_3,h_3,mod_3),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\big(binop \in \{\texttt{div},\texttt{rem}\} \wedge (l_2 \le 0 \le h_2)\big) \Rightarrow$$

$$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{ArithmeticException}})$$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{numop}\ t\ binop\ t_2$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X_1::X_2$$

$$\forall\, \{(t,(l_1,h_1,mod_1),(m_a,pc_a))\} \sqsubseteq_S X_1 :$$

$$\forall\, \{(t,(l_2,h_2,mod_2),(m_b,pc_b))\} \sqsubseteq_S X_2 :$$

$$\neg\big(binop \in \{\texttt{div},\texttt{rem}\} \wedge (l_2 = 0 = h_2)\big) \Rightarrow$$

$(t,(l_3,h_3,mod_3),(m_n,pc_n))$	$=$	$absApplyBinary(binop,(t,(l_1,h_1,mod_1),(m_a,pc_a)),(t,(l_2,h_2,mod_2),(m_b,pc_b)),(m_n,pc_n),\text{MAX_MOD_COUNT})$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
$M::\{(t,(l_3,h_3,mod_3),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\big(binop \in \{\texttt{div},\texttt{rem}\} \wedge (l_2 \le 0 \le h_2)\big) \Rightarrow$$
$$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{ArithmeticException}})$$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{numop}\ t\ \texttt{neg}$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X$$

$$\forall\, \{(t,(l_1,h_1,mod_1),(m_p,pc_p))\} \sqsubseteq_S X :$$

$(t,(l_3,h_3,mod_3),(m_n,pc_n))$	$=$	$absApplyUnary(\texttt{neg},(t,(l_1,h_1,mod_1),(m_p,pc_p)),(m_n,pc_n),\text{MAX_MOD_COUNT})$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
$M::\{(t,(l_3,h_3,mod_3),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{numop } t \texttt{ to } t_{opt}$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\widehat{S}(m_n,pc_n)(\pi_1) = M::X$

$\forall\, \{(t,(l_1,h_1,mod_1),(m_p,pc_p))\} \sqsubseteq_S X :$

$(t_{opt},(l_3,h_3,mod_3),(m_n,pc_n))$	$=$	$absApplyUnary(\texttt{to},t_{opt},(t,(l_1,h_1,mod_1),(m_p,pc_p)),(m_n,pc_n),\mathrm{MAX_MOD_COUNT})$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
$M::\{(t_{opt},(l_3,h_3,mod_3),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{load } t\ j$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\forall\, \{(t,Y,(m_p,pc_p))\} \sqsubseteq_L \widehat{L}(m_n,pc_n)(\pi_1)(j) :$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n,pc_n)(\pi_1)::\{(t,Y,(m_p,pc_p))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{store}\ t\ j$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$
$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\widehat{S}(m_n,pc_n)(\pi_1) = M::X$
$\forall\, \{(t,Y,(m_p,pc_p))\} \sqsubseteq_S X :$

$\{\pi_2\}$		$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M		$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\forall\, k \in \mathit{dom}(\widehat{L}(m_n,pc_n)(\pi_1))\ .\ k \neq j :$	$\widehat{L}(m_n,pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k)$
$(t = \texttt{r})\ \Rightarrow$	$\{(t,Y,(m_p,pc_p))\}$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j)$
$(t \in \{\texttt{s},\texttt{i}\})\ \Rightarrow$	$\{(t,Y,(m_n,pc_n))\}$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$		$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$		$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$		$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{inc}\ t\ j\ c$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$
$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\forall\, \{(t,(l_1,h_1,mod_1),(m_p,pc_p))\} \sqsubseteq_L \widehat{L}(m_n,pc_n)(\pi_1)(j) :$

$(t,(l_3,h_3,mod_3),(m_n,pc_n))$	$=$	$absApplyBinary(\texttt{add},(t,(l_1,h_1,mod_1),(m_p,pc_p)),(t,(c,c,0),(m_n,pc_n)),(m_n,pc_n),\text{MAX_MOD_COUNT})$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\forall\, k \in \mathit{dom}(\widehat{L}(m_n,pc_n)(\pi_1))\ .\ k \neq j : \widehat{L}(m_n,pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k)$
$\{(t,(l_3,h_3,mod_3),(m_n,pc_n))\}$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{goto } \mathit{addr}$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (\mathit{m}_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (\mathit{m}_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (\mathit{m}_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (\mathit{m}_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (\mathit{m}_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (\mathit{m}_n, \mathit{addr}))$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(\mathit{m}_n, \mathit{addr})$
$\widehat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(\mathit{m}_n, \mathit{addr})(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathit{m}_n, \mathit{addr})(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathit{m}_n, \mathit{addr})(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathit{m}_n, \mathit{addr})(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{if } t \ cmpop \ \texttt{goto } \mathit{addr}$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\pi_3 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,\mathit{addr})))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X_1::X_2$$

$$\forall\, \{(t,Y_1,(m_p,pc_p))\} \sqsubseteq_S X_1 :$$

$$\forall\, \{(t,Y_2,(m_q,pc_q))\} \sqsubseteq_S X_2 :$$

$$(\mathit{absApplyBinary}(cmpop,(t,Y_1,(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{true}\}) \Rightarrow$$

$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,\mathit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,\mathit{addr})(\pi_3)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,\mathit{addr})(\pi_3)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,\mathit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,\mathit{addr})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\mathit{absApplyBinary}(cmpop,(t,Y_1,(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{false}\}) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{if}\ \ t\ \ cmpop\ \ \texttt{null goto}\ \ \mathit{addr}$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(\mathit{m}_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(\mathit{m}_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(\mathit{m}_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(\mathit{m}_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(\mathit{m}_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(\mathit{m}_n,\mathit{m}_n.\mathit{nextAddress}(pc_n))))$$

$$\pi_3 = ((O_1,\psi_1,\gamma_1,\xi_1,(\mathit{m}_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(\mathit{m}_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(\mathit{m}_n,\mathit{addr})))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M :: X_1$$

$$\forall\, \{(\mathit{t},Y_1,(\mathit{m}_p,pc_p))\} \sqsubseteq_S X_1 :$$

$$(\mathit{t},Y_2,(\mathit{m}_q,pc_q)) = \begin{cases} \widehat{\sigma}_{\texttt{Null}} & t = \texttt{r} \\ (\texttt{s},(0,0,0),(\mathit{m}_n,pc_n))) & t = \texttt{s} \end{cases}$$

$$\big(\mathit{absApplyBinary}(cmpop,(\mathit{t},Y_1,(\mathit{m}_p,pc_p)),(\mathit{t},Y_2,(\mathit{m}_q,pc_q))) \supseteq \{\texttt{true}\}\big) \ \Rightarrow$$

$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(\mathit{m}_n,\mathit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(\mathit{m}_n,\mathit{addr})(\pi_3)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathit{m}_n,\mathit{addr})(\pi_3)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathit{m}_n,\mathit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathit{m}_n,\mathit{addr})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\big(\mathit{absApplyBinary}(cmpop,(\mathit{t},Y_1,(\mathit{m}_p,pc_p)),(\mathit{t},Y_2,(\mathit{m}_q,pc_q))) \supseteq \{\texttt{false}\}\big) \ \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(\mathit{m}_n,\mathit{m}_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(\mathit{m}_n,\mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathit{m}_n,\mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathit{m}_n,\mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathit{m}_n,\mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{lookupswitch } t \texttt{ (} k_i \texttt{ => } apc_i)_1^r \texttt{ , default => } apc_{default}$$

$\iff \quad \forall\, \pi_0 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$

$\pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,apc_1)))$

$\vdots$

$\pi_r = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,apc_r)))$

$\pi_{default} = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,apc_{default})))$

$\widehat{S}(m_n,pc_n)(\pi_0) = M::X$

$\forall\, \{(t,(l,h,mod),(m_q,pc_q))\} \sqsubseteq_S X :$

$\forall\, p \in \{1,\ldots,r\} :$

$l \le k_p \le h \;\Rightarrow$

$\{\pi_p\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,apc_p)$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,apc_p)(\pi_p)$
$\widehat{L}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_L$	$\widehat{L}(m_n,apc_p)(\pi_p)$
$\widehat{JH}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,apc_p)(\pi_p)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,apc_p)(\pi_p)$
$\{(\pi_0,\pi_p)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\forall\, v \in \{l,\ldots,h\} :$

$(\nexists\, p \in \{1,\ldots,r\} \,.\, k_p = v) \;\Rightarrow$

$\{\pi_{default}\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,apc_{default})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,apc_{default})(\pi_{default})$
$\widehat{L}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_L$	$\widehat{L}(m_n,apc_{default})(\pi_{default})$
$\widehat{JH}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,apc_{default})(\pi_{default})$
$\widehat{MNAMES}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,apc_{default})(\pi_{default})$
$\{(\pi_0,\pi_{default})\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{tableswitch } t \textit{ low} \texttt{ => } (apc_i)_1^r \texttt{ , default => } apc_{default}$$

$$\iff \quad \forall\, \pi_0 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,apc_1)))$$

$$\vdots$$

$$\pi_r = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,apc_r)))$$

$$\pi_{default} = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,apc_{default})))$$

$$\widehat{S}(m_n,pc_n)(\pi_0) = M::X$$

$$\forall\, \{(t,(l,h,mod),(m_q,pc_q))\} \sqsubseteq_S X :$$

$$\forall\, k \in \{l,\ldots,h\} :$$

$$(low \leq k < (low + r)) \;\Rightarrow$$

p	$=$	$((k - low) + 1)$
$\{\pi_p\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, apc_p)$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, apc_p)(\pi_p)$
$\widehat{L}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_L$	$\widehat{L}(m_n, apc_p)(\pi_p)$
$\widehat{JH}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, apc_p)(\pi_p)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, apc_p)(\pi_p)$
$\{(\pi_0,\pi_p)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$((l < low) \vee (h \geq (low + r)) \;\Rightarrow$$

$\{\pi_{default}\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, apc_{default})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, apc_{default})(\pi_{default})$
$\widehat{L}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_L$	$\widehat{L}(m_n, apc_{default})(\pi_{default})$
$\widehat{JH}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, apc_{default})(\pi_{default})$
$\widehat{MNAMES}(m_n,pc_n)(\pi_0)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, apc_{default})(\pi_{default})$
$\{(\pi_0,\pi_{default})\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{new } \tau\texttt{[]}$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X$$

$$\forall\, \{(\texttt{s},(l,h,mod),(m_q,pc_q))\} \sqsubseteq_S X :$$

$$(h \geq 0) \;\Rightarrow$$

$$min_length = max(0,l)$$
$$max_length = h$$

$$O_q = \left(\texttt{r}, \left(\begin{array}{l} \mathit{type} : \tau, \\ \mathit{refType} : \tau\texttt{[ ]}, \\ \mathit{isArray} : \texttt{true}, \\ \mathit{owner} : O_n.\mathit{owner}, \\ \mathit{entryPoint} : \texttt{no}, \\ \mathit{isGlobal} : \texttt{false}, \\ \mathit{transient} : \texttt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (m_n,pc_n), \\ \mathit{length} : (min_length, max_length), \end{array} \right), (m_n,pc_n) \right)$$

$$v = \begin{cases} \widehat{\sigma}_{\texttt{Null}}, & \tau \in \mathsf{ReferenceType}, \\ (\texttt{b},(0,0,0),(m_n,pc_n)), & \tau : \texttt{byte}, \\ (\texttt{s},(0,0,0),(m_n,pc_n)), & \tau : \texttt{short}, \\ (\texttt{i},(0,0,0),(m_n,pc_n)), & \tau : \texttt{int}, \end{cases}$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$M::\{O_q\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$(\gamma_n = 0) \;\Rightarrow\; \{v\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}(0, \texttt{Short.MAX_VALUE})$
$(\gamma_n = 1) \;\Rightarrow\; \{v\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(0, \texttt{Short.MAX_VALUE})$

$$(l < 0) \;\Rightarrow$$
$$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NegativeArraySizeException}})$$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{new } \tau$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\tau \in$ Class

$$O_q = \left(\texttt{r}, \left(\begin{array}{l} \mathit{type} : \tau, \\ \mathit{refType} : \tau, \\ \mathit{isArray} : \texttt{false}, \\ \mathit{owner} : O_n.\mathit{owner}, \\ \mathit{entryPoint} : \texttt{no}, \\ \mathit{isGlobal} : \texttt{false}, \\ \mathit{transient} : \texttt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (m_n, pc_n), \\ \mathit{length} : 0, \end{array} \right), (m_n, pc_n) \right)$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n,pc_n)(\pi_1)::\{O_q\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\forall\, f \in \mathit{instanceFields}(\tau)$:

$$v = \begin{cases} \widehat{\sigma}_{\texttt{Null}}, & f.\mathit{type} \in \text{ReferenceType} \\ (\texttt{b},(d,d,0),(m_n,pc_n)), & f.\mathit{type} = \texttt{byte}, f.\texttt{hasInitValue} = \texttt{true} \wedge f.\texttt{initValue} = d \\ (\texttt{b},(0,0,0),(m_n,pc_n)), & f.\mathit{type} = \texttt{byte}, f.\texttt{hasInitValue} = \texttt{false} \\ (\texttt{s},(d,d,0),(m_n,pc_n)), & f.\mathit{type} = \texttt{short}, f.\texttt{hasInitValue} = \texttt{true} \wedge f.\texttt{initValue} = d \\ (\texttt{s},(0,0,0),(m_n,pc_n)), & f.\mathit{type} = \texttt{short}, f.\texttt{hasInitValue} = \texttt{false} \\ (\texttt{i},(d,d,0),(m_n,pc_n)), & f.\mathit{type} = \texttt{int}, f.\texttt{hasInitValue} = \texttt{true} \wedge f.\texttt{initValue} = d \\ (\texttt{i},(0,0,0),(m_n,pc_n)), & f.\mathit{type} = \texttt{int}, f.\texttt{hasInitValue} = \texttt{false} \end{cases}$$

$$(\gamma_n = 0) \;\Rightarrow\; \{v\} \sqsubseteq_H \widehat{H}(O_q).\mathit{values}(f.id)$$
$$(\gamma_n = 1) \;\Rightarrow\; \{v\} \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id)$$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{getstatic}\ f$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(f.\mathit{type} \in \{\texttt{boolean},\texttt{byte}\})\ \Rightarrow$

$\forall\, \{(\texttt{b},(l,h,mod),(m_q,pc_q))\} \sqsubseteq_K \widehat{K}(f.id) :$

$\widehat{S}(m_n,pc_n)(\pi_1)::\{(\texttt{s},(l,h,mod),(m_q,pc_q))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$(f.\mathit{type} \notin \{\texttt{boolean},\texttt{byte}\})\ \Rightarrow$

$\widehat{S}(m_n,pc_n)(\pi_1)::\widehat{K}(f.id) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{putstatic}\ f$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M{::}X$$

$f.\mathit{type} \in \mathsf{PrimitiveType}\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(f.\mathit{type} \in \{\texttt{boolean},\texttt{byte}\})\ \Rightarrow$

$\forall\, \{(\mathtt{s},(l,h,mod),(m_q,pc_q))\} \sqsubseteq_S X :$

$(\mathtt{b},(l_2,h_2,mod),(m_q,pc_q))$	$=$	$\mathit{absFromShort}\big((\mathtt{s},(l,h,mod),(m_q,pc_q)),\mathtt{b}\big)$
$\{(\mathtt{b},(l_2,h_2,mod),(m_n,pc_n))\}$	$\sqsubseteq_K$	$\widehat{K}(f.id)$

$(f.\mathit{type} \notin \{\texttt{boolean},\texttt{byte}\})\ \Rightarrow$

$\forall\, \{(t,(l,h,mod),(m_q,pc_q))\} \sqsubseteq_S X :$

$\{(t,(l,h,mod),(m_n,pc_n))\}$	$\sqsubseteq_K$	$\widehat{K}(f.id)$

$f.\mathit{type} \in \mathsf{ReferenceType}\ \Rightarrow$

$\forall\, \{O_q\} = \{(\mathtt{r},Y,(m_r,pc_r))\} \sqsubseteq_S X :$

$\mathit{checkPutObjectStatic}(O_n.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, O_q.\mathit{entryPoint})\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\{O_q\}$	$\sqsubseteq_K$	$\widehat{K}(f.id)$

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{getfield}\ f$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$
$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n, pc_n)(\pi_1) = M::X$$
$$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X :$$

$$(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}) \Rightarrow$$
$$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NullPointerException}})$$
$$\left(\neg O_q.\mathit{isArray} \wedge \left(f.id \in \left(dom\left(\widehat{H}(O_q).\mathit{values}\right) \cup dom\left(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}\right)\right)\right)\right) \Rightarrow$$

$$\mathit{checkGetField}(O_n.\mathit{owner}, O_q.\mathit{owner}) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \Rightarrow$$

$$\left(f.id \in dom(\widehat{H}(O_q).\mathit{values})\right) \Rightarrow$$

$$\forall\, \{(\mathtt{b}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_H \widehat{H}(O_q).\mathit{values}(f.id) :$$
$$M::\{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

$$\left(\gamma_n = 1 \wedge f.id \in dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values})\right) \Rightarrow$$

$$\forall\, \{(\mathtt{b}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_{JH} \widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}(f.id) :$$
$$M::\{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

$$(f.\mathit{type} \notin \{\texttt{boolean}, \texttt{byte}\}) \Rightarrow$$

$$\left(f.id \in dom(\widehat{H}(O_q).\mathit{values})\right) \Rightarrow$$

$$M::\widehat{H}(O_q).\mathit{values}(f.id) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

$$\left(\gamma_n = 1 \wedge f.id \in dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values})\right) \Rightarrow$$

$$M::\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}(f.id) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

$$\neg\, \mathit{checkGetField}(O_n.\mathit{owner}, O_q.\mathit{owner}) \Rightarrow$$
$$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$$

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{getfield this } f$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$\widehat{S}(m_n, pc_n)(\pi_1) = M$
$\widehat{L}(m_n, pc_n)(\pi_1)(0) = X$
$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X :$

$(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}) \Rightarrow$
$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NullPointerException}})$
$\left(\neg O_q.\mathit{isArray} \wedge \left(f.id \in \left(dom\left(\widehat{H}(O_q).\mathit{values}\right) \cup dom\left(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}\right)\right)\right)\right) \Rightarrow$

$\mathit{checkGetField}(O_n.\mathit{owner}, O_q.\mathit{owner}) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\}) \Rightarrow$

$\left(f.id \in dom(\widehat{H}(O_q).\mathit{values})\right) \Rightarrow$

$\forall\, \{(\mathtt{b}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_H \widehat{H}(O_q).\mathit{values}(f.id) :$

$M :: \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\left(\gamma_n = 1 \wedge f.id \in dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values})\right) \Rightarrow$

$\forall\, \{(\mathtt{b}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_{JH} \widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}(f.id) :$

$M :: \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$(f.\mathit{type} \notin \{\texttt{boolean}, \texttt{byte}\}) \Rightarrow$

$\left(f.id \in dom(\widehat{H}(O_q).\mathit{values})\right) \Rightarrow$

$M :: \widehat{H}(O_q).\mathit{values}(f.id) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\left(\gamma_n = 1 \wedge f.id \in dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values})\right) \Rightarrow$

$M :: \widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}(f.id) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\neg \mathit{checkGetField}(O_n.\mathit{owner}, O_q.\mathit{owner}) \Rightarrow$

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$

$$(\hat{R}, \widehat{K}, \widehat{H}, \hat{I}, \widehat{JH}, \hat{C}, \hat{L}, \hat{S}, \hat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{putfield } f$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \hat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$$\hat{S}(m_n, pc_n)(\pi_1) = M::X::A$$
$$\forall\, \{(t, W, (m_q, pc_q))\} \sqsubseteq_S A :$$

$$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X :$$

$$(\hat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \hat{I}) \;\Rightarrow$$
$$\text{HANDLE}(\pi_1, \pi_1, \hat{\sigma}_{\texttt{NullPointerException}})$$
$$\left(\neg O_q.\mathit{isArray} \wedge \left(f.id \in \left(dom\left(\widehat{H}(O_q).\mathit{values}\right) \cup dom\left(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}\right)\right)\right)\right) \;\Rightarrow$$

$$\mathit{checkPutField}(O_n.\mathit{owner}, O_q.\mathit{owner}, (t, W, (m_q, pc_q)), (f.\mathit{type} \in \text{ReferenceType})) \;\Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\hat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\hat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\hat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\hat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\left(f.\mathit{type} \in \{\texttt{boolean}, \texttt{byte}\} \wedge (t = \texttt{s})\right) \;\Rightarrow$$

$$\neg\left(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\right) \;\Rightarrow$$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t, W, (m_q, pc_q)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}(f.id)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\left(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\right) \;\Rightarrow$$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t, W, (m_q, pc_q)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id)$:
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\left(f.\mathit{type} \notin \{\texttt{boolean}, \texttt{byte}\}\right) \;\Rightarrow$$

$$\neg\left(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\right) \;\Rightarrow$$

$(f.\mathit{type} = t = \mathtt{r})$	$\Rightarrow$	$\{(t, W, (m_q, pc_q))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}(f.id)$
$((f.\mathit{type} = t) \wedge (t \in \{\mathtt{s}, \mathtt{i}\}))$	$\Rightarrow$	$\{(t, W, (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}(f.id)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\left(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\right) \;\Rightarrow$$

$(f.\mathit{type} = t = \mathtt{r})$	$\Rightarrow$	$\{(t, W, (m_q, pc_q))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id)$
$((f.\mathit{type} = t) \wedge (t \in \{\mathtt{s}, \mathtt{i}\}))$	$\Rightarrow$	$\{(t, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\neg\mathit{checkPutField}(O_n.\mathit{owner}, O_q.\mathit{owner}, (t, W, (m_q, pc_q)), (f.\mathit{type} \in \text{ReferenceType})) \;\Rightarrow$$
$$\text{HANDLE}(\pi_1, \pi_1, \hat{\sigma}_{\texttt{SecurityException}})$$

$$(\widehat{R}, \widehat{K}, \widehat{H}, \hat{I}, \widehat{JH}, \widehat{C}, \hat{L}, \hat{S}, \hat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{putfield this } f$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$\widehat{S}(m_n, pc_n)(\pi_1) = M::A$
$\widehat{L}(m_n, pc_n)(\pi_1)(0) = X$
$\forall\, \{(t, W, (m_q, pc_q))\} \sqsubseteq_S A :$

$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X :$

$(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \hat{I}) \Rightarrow$
$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NullPointerException}})$
$\Big(\neg O_q.\mathit{isArray} \wedge \Big(f.id \in \Big(dom\big(\widehat{H}(O_q).\mathit{values}\big) \cup dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}\big)\Big)\Big)\Big) \Rightarrow$

$\mathit{checkPutField}(O_n.\mathit{owner}, O_q.\mathit{owner}, (t, W, (m_q, pc_q)), (f.\mathit{type} \in \mathsf{ReferenceType})) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\Big(f.\mathit{type} \in \{\mathtt{boolean}, \mathtt{byte}\} \wedge (t = \mathtt{s})\Big) \Rightarrow$

$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t, W, (m_q, pc_q)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}(f.id)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t, W, (m_q, pc_q)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id) :$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\Big(f.\mathit{type} \notin \{\mathtt{boolean}, \mathtt{byte}\}\Big) \Rightarrow$

$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(f.\mathit{type} = t = \mathtt{r})$	$\Rightarrow$	$\{(t, W, (m_q, pc_q))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}(f.id)$
$((f.\mathit{type} = t) \wedge (t \in \{\mathtt{s}, \mathtt{i}\}))$	$\Rightarrow$	$\{(t, W, (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}(f.id)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(f.\mathit{type} = t = \mathtt{r})$	$\Rightarrow$	$\{(t, W, (m_q, pc_q))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id)$
$((f.\mathit{type} = t) \wedge (t \in \{\mathtt{s}, \mathtt{i}\}))$	$\Rightarrow$	$\{(t, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}(f.id)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\neg\, \mathit{checkPutField}(O_n.\mathit{owner}, O_q.\mathit{owner}, (t, W, (m_q, pc_q)), (f.\mathit{type} \in \mathsf{ReferenceType})) \Rightarrow$

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{checkcast}\ \tau$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\widehat{S}(m_n,pc_n)(\pi_1) = M::X$
$\forall\, \{O_q\} = \{(\mathtt{r},Y,(m_r,pc_r))\} \sqsubseteq_S X :$

$(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\mathit{checkCast}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient}, O_q.\mathit{refType}, \tau \end{array}\right) \Rightarrow$$

$(O_q.\mathit{refType} \preceq \tau) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\neg(O_q.\mathit{refType} \preceq \tau) \Rightarrow$

$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{ClassCastException}})$

$$\neg\mathit{checkCast}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient}, O_q.\mathit{refType}, \tau \end{array}\right) \Rightarrow$$

$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{SecurityException}})$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{instanceof}\ \tau$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\widehat{S}(m_n,pc_n)(\pi_1) = M::X$
$\forall\, \{O_q\} = \{(\mathtt{r},Y,(m_r,pc_r))\} \sqsubseteq_S X :$

$(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$M::\{(\mathtt{s},(0,0,0),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\mathit{checkCast}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient}, O_q.\mathit{refType}, \tau \end{array}\right) \Rightarrow$$

$(O_q.\mathit{refType} \preceq \tau) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$M::\{(\mathtt{s},(1,1,0),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\neg(O_q.\mathit{refType} \preceq \tau) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$M::\{(\mathtt{s},(0,0,0),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\neg\mathit{checkCast}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient}, O_q.\mathit{refType}, \tau \end{array}\right) \Rightarrow$$

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$(\tau_i)_0^q \to \tau_r = p.\textit{type}$

$p.\textit{isStatic}$

$p \notin \texttt{SEPARATELY_HANDLED_API_METHODS}$

$\widehat{S}(m_n, pc_n)(\pi_1) = M :: X_0 :: \ldots :: X_q$

$\left(\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, pc_n)(\pi_1)\right) \Rightarrow \{(p, \pi_1)\}$	$\sqsubseteq_{RECURSIVE}$	$\widehat{REC}$
π_3	$=$	$newAbsContext(\pi_1, k, p)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(p, p.\textit{firstAddress})$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(p, p.\textit{firstAddress})(\pi_3)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(p, p.\textit{firstAddress})(\pi_3)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p, p.\textit{firstAddress})(\pi_3)$
$\{p\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p, p.\textit{firstAddress})(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\forall j \in \{0, \ldots, q\} :$

$\forall\, \{(t_j, W_j, (m_j, pc_j))\} \sqsubseteq_S X_j :$

$(\tau_j = \mathtt{r} = t_j)$	$\Rightarrow \{(t_j, W_j, (m_j, pc_j))\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\textit{firstAddress})(\pi_3)(j)$
$(\tau_j \neq \mathtt{r})$	$\Rightarrow \{(t_j, W_j, (p, p.\textit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\textit{firstAddress})(\pi_3)(j)$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$(\tau_i)_1^q \to \tau_r = p.\mathit{type}$

$\neg p.\mathit{isStatic}$

$\widehat{S}(m_n,pc_n)(\pi_1) = M :: X_0 :: X_1 :: \ldots :: X_q$

$\forall\, \{O_q\} = \{(\mathtt{r},Y,(m_r,pc_r))\} \sqsubseteq_S X_0 :$

$\big(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}\big) \Rightarrow$
$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{NullPointerException}})$

$w = \mathit{methodLookup}(p.\mathit{id}, O_q.\mathit{type})$

$\big((w \neq \bot) \wedge (p \notin \texttt{SEPARATELY_HANDLED_API_METHODS})\big) \Rightarrow$

$$\mathit{checkInvokeDefinite}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$$

$\big(\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n,pc_n)(\pi_1)\big) \Rightarrow \{(p,\pi_1)\}$	$\sqsubseteq_{RECURSIVE}$	$\widehat{REC}$
π_3	$=$	$newAbsContext(\pi_1,k,p)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(p,p.\mathit{firstAddress})$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(p,p.\mathit{firstAddress})(\pi_3)$
$\{O_q\}$	$\sqsubseteq_L$	$\widehat{L}(p,p.\mathit{firstAddress})(\pi_3)(0)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(p,p.\mathit{firstAddress})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p,p.\mathit{firstAddress})(\pi_3)$
$\{p\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p,p.\mathit{firstAddress})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\forall j \in \{1,\ldots,q\} :$

$\forall\, \{(t_j,W_j,(m_j,pc_j))\} \sqsubseteq_S X_j :$

$(\tau_j = \mathtt{r} = t_j)$	$\Rightarrow \{(t_j,W_j,(m_j,pc_j))\}$	$\sqsubseteq_L$	$\widehat{L}(p,p.\mathit{firstAddress})(\pi_3)(j)$
$(\tau_j \neq \mathtt{r})$	$\Rightarrow \{(t_j,W_j,(p,p.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(p,p.\mathit{firstAddress})(\pi_3)(j)$

$$\neg\mathit{checkInvokeDefinite}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$$

$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{SecurityException}})$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokevirtual}\ w$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$(\tau_i)_1^q \to \tau_r = w.\mathit{type}$
$\widehat{S}(m_n,pc_n)(\pi_1) = M :: X_0 :: X_1 :: \ldots :: X_q$

$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0 :$

$\big(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}\big) \Rightarrow$
$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NullPointerException}})$

$p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$
$\big((p \neq \bot) \wedge (p \notin \texttt{SEPARATELY_HANDLED_API_METHODS})\big) \Rightarrow$

$$\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$$

$\big(\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n,pc_n)(\pi_1)\big) \Rightarrow \{(p,\pi_1)\}$	$\sqsubseteq_{RECURSIVE}$	$\widehat{REC}$
π_3	$=$	$newAbsContext(\pi_1,k,p)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(p, p.\mathit{firstAddress})$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(p, p.\mathit{firstAddress})(\pi_3)$
$\{O_q\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(0)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(p, p.\mathit{firstAddress})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$
$\{p\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\forall j \in \{1,\ldots,q\} :$

$\forall\, \{(t_j, W_j, (m_j, pc_j))\} \sqsubseteq_S X_j :$

$(\tau_j = \mathtt{r} = t_j)$	$\Rightarrow \{(t_j, W_j, (m_j, pc_j))\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$
$(\tau_j \neq \mathtt{r})$	$\Rightarrow \{(t_j, W_j, (p, p.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$

$$\neg\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$$

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokeinterface}\ w$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$(\tau_i)_1^q \to \tau_r = w.\mathit{type}$
$\widehat{S}(m_n,pc_n)(\pi_1) = M :: X_0 :: X_1 :: \ldots :: X_q$

$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0 :$

$\big(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}\big) \Rightarrow$
$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NullPointerException}})$

$p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$
$\big((p \neq \bot) \wedge (p \notin \texttt{SEPARATELY_HANDLED_API_METHODS})\big) \Rightarrow$

$O_q.\mathit{owner} = (package_{aid}, applet_{class})$

$\forall\, active \in \{\mathtt{true}, \mathtt{false}\} :$

$$\mathit{checkInvokeInterface}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient}, O_q.\mathit{refType}, p.\mathit{classI}, active, applet_{class} \end{array}\right) \Rightarrow$$

$\big(\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n,pc_n)(\pi_1)\big) \Rightarrow \{(p,\pi_1)\}$	$\sqsubseteq_{RECURSIVE}$	$\widehat{REC}$
π_3	$=$	$newAbsContext(\pi_1,k,p)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(p, p.\mathit{firstAddress})$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(p, p.\mathit{firstAddress})(\pi_3)$
$\{O_q\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(0)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(p, p.\mathit{firstAddress})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$
$\{p\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\forall j \in \{1,\ldots,q\} :$

$\forall\, \{(t_j, W_j, (m_j, pc_j))\} \sqsubseteq_S X_j :$

$(\tau_j = \mathtt{r} = t_j)$	$\Rightarrow \{(t_j, W_j, (m_j, pc_j))\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$
$(\tau_j \neq \mathtt{r})$	$\Rightarrow \{(t_j, W_j, (p, p.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$

$$\neg\mathit{checkInvokeInterface}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient}, O_q.\mathit{refType}, p.\mathit{classI}, active, applet_{class} \end{array}\right) \Rightarrow$$

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{return}$

$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$

$(n = 3) \Rightarrow$

π_3	$=$	$((O_1, \psi_1, \gamma_1, \gamma_1, (m_1, pc_1)), (O_2, \psi_n, \gamma_2, \gamma_n, (m_2, 20)))$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_2, 20)$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_2, 20)(\pi_3)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_2, 20)(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(n > 3) \Rightarrow$

$\forall\, \pi_{invoked} = ((O_1, \psi_1, \gamma_1, \gamma_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \gamma_2, (m_2, pc_2)), \ldots, (O_n, Y_1, \gamma_n, Y_2, (m_n, m_n.\mathit{firstAddress}))) \sqsubseteq_C \widehat{C}(m_n, m_n.\mathit{firstAddress}) :$

$\forall(\pi_{calling}, \pi_{invoked}) \sqsubseteq_{CG} \widehat{CG} :$

$\pi_{calling}$	$=$	$((O_1, \psi_1, \gamma_1, \gamma_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \gamma_2, (m_2, pc_2)), \ldots, (O_b, \psi_b, \gamma_b, \gamma_b, (m_b, pc_b)))$
$\pi_{successor}$	$=$	$((O_1, \psi_1, \gamma_1, \gamma_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \gamma_2, (m_2, pc_2)), \ldots, (O_b, \psi_n, \gamma_b, \gamma_n, (m_b, m_b.\mathit{nextAddress}(pc_b)))$
$(\tau_i)_1^q \rightarrow \tau_r$	$=$	$m_n.\mathit{type}$
$\{\pi_{successor}\}$	$\sqsubseteq_C$	$\widehat{C}(m_b, m_b.\mathit{nextAddress}(pc_b))$
$\widehat{L}(m_b, pc_b)(\pi_{calling})$	$\sqsubseteq_L$	$\widehat{L}(m_b, m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_b, m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$
$\widehat{MNAMES}(m_b, pc_b)(\pi_{calling})$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_b, m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$
$\{(\pi_1, \pi_{successor})\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(m_n.\mathit{isStatic}) \Rightarrow$

$\widehat{S}(m_b, pc_b)(\pi_{calling})$	$=$	$M::X_1::\ldots::X_q$
M	$\sqsubseteq_S$	$\widehat{S}(m_b, m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$

$\neg(m_n.\mathit{isStatic}) \Rightarrow$

$\widehat{S}(m_b, pc_b)(\pi_{calling})$	$=$	$M::X_0::X_1::\ldots::X_q$
M	$\sqsubseteq_S$	$\widehat{S}(m_b, m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{return } t$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$(n=3) \Rightarrow$

π_3	$=$	$((O_1,\psi_1,\gamma_1,\gamma_1,(m_1,pc_1)),(O_2,\psi_n,\gamma_2,\gamma_n,(m_2,20)))$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_2,20)$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_2,20)(\pi_3)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_2,20)(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(n>3) \Rightarrow$

$\widehat{S}(m_n,pc_n)(\pi_1) = N::D$

$\forall\, \pi_{invoked} = ((O_1,\psi_1,\gamma_1,\gamma_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\gamma_2,(m_2,pc_2)),\ldots,(O_n,Y_1,\gamma_n,Y_2,(m_n,m_n.\mathit{firstAddress}))) \sqsubseteq_C \widehat{C}(m_n,m_n.\mathit{firstAddress}) :$

$\forall(\pi_{calling},\pi_{invoked}) \sqsubseteq_{CG} \widehat{CG} :$

$\pi_{calling}$	$=$	$((O_1,\psi_1,\gamma_1,\gamma_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\gamma_2,(m_2,pc_2)),\ldots,(O_b,\psi_b,\gamma_b,\gamma_b,(m_b,pc_b)))$
$\pi_{successor}$	$=$	$((O_1,\psi_1,\gamma_1,\gamma_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\gamma_2,(m_2,pc_2)),\ldots,(O_b,\psi_n,\gamma_b,\gamma_n,(m_b,m_b.\mathit{nextAddress}(pc_b)))$
$(\tau_i)_1^q \rightarrow \tau_r$	$=$	$m_n.\mathit{type}$
$\{\pi_{successor}\}$	$\sqsubseteq_C$	$\widehat{C}(m_b,m_b.\mathit{nextAddress}(pc_b))$
$\widehat{L}(m_b,pc_b)(\pi_{calling})$	$\sqsubseteq_L$	$\widehat{L}(m_b,m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_b,m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$
$\widehat{MNAMES}(m_b,pc_b)(\pi_{calling})$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_b,m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$
$\{(\pi_1,\pi_{successor})\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(m_n.\mathit{isStatic}) \Rightarrow$

$\widehat{S}(m_b,pc_b)(\pi_{calling})$	$=$	$M::X_1::\ldots::X_q$
$M::D$	$\sqsubseteq_S$	$\widehat{S}(m_b,m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$

$\neg(m_n.\mathit{isStatic}) \Rightarrow$

$\widehat{S}(m_b,pc_b)(\pi_{calling})$	$=$	$M::X_0::X_1::\ldots::X_q$
$M::D$	$\sqsubseteq_S$	$\widehat{S}(m_b,m_b.\mathit{nextAddress}(pc_b))(\pi_{successor})$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{arraylength}$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$\widehat{S}(m_n,pc_n)(\pi_1) = M::X$
$\forall\, \{O_q\} = \{(\mathtt{r},Y,(m_r,pc_r))\} \sqsubseteq_S X :$
$(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}) \Rightarrow$
HANDLE$(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{NullPointerException}})$
$(O_q.\mathit{isArray}) \Rightarrow$

checkArrayLoad $(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q\ ) \Rightarrow$

$O_q.\mathit{length}$	$=$	(min,max)
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$M::\{(\mathtt{s},(min,max,0),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\neg$*checkArrayLoad* $(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q\ ) \Rightarrow$

HANDLE$(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{SecurityException}})$

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{arrayload}\ t\ (part\ 1\ of\ 2)$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n, pc_n)(\pi_1) = M::X::A$$
$$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X :$$

$$\left(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}\right) \Rightarrow$$
$$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NullPointerException}})$$

$$(O_q.\mathit{isArray}) \Rightarrow$$

$$O_q.\mathit{length} = (min_array_length, max_array_length)$$

$$\forall\, \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_S A :$$

$$\neg \mathit{checkArrayLoad}\,(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q\ ) \Rightarrow$$
$$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$$

$$\mathit{checkArrayLoad}\,(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q\ ) \Rightarrow$$

$$((l < 0) \vee (h \geq min_array_length)) \Rightarrow$$
$$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{ArrayIndexOutOfBoundsException}})$$

$$(h \geq 0) \Rightarrow$$

$$\forall j \in \{max(0, l), \ldots, min(max_array_length, h)\} :$$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{arrayload}\ t\ (part\ 2\ of\ 2)$

$$\forall (start, end) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}\big) \\ \cup \\ dom\big(\widehat{H}(O_q).\mathit{values}\big) \end{array} \right) . (start \leq j \leq end) :$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(t = \texttt{b})\ \Rightarrow$

$\Big((\gamma_n = 0) \vee (\gamma_n = 1 \wedge (start, end) \notin \mathit{dom}(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}))\Big)\ \Rightarrow$

$\forall\ \{(\texttt{b}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_H \widehat{H}(O_q).\mathit{values}((start, end)) :$

$M::\{(\texttt{s}, (l, h, mod), (m_q, pc_q))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\Big(\gamma_n = 1 \wedge (start, end) \in dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values})\Big)\ \Rightarrow$

$\forall\ \{(\texttt{b}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_{JH} \widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}((start, end)) :$

$M::\{(\texttt{s}, (l, h, mod), (m_q, pc_q))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$(t \neq \texttt{b})\ \Rightarrow$

$\Big((\gamma_n = 0) \vee (\gamma_n = 1 \wedge (start, end) \notin \mathit{dom}(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}))\Big)\ \Rightarrow$

$M::\widehat{H}(O_q).\mathit{values}((start, end)) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\Big(\gamma_n = 1 \wedge (start, end) \in dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values})\Big)\ \Rightarrow$

$M::\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}((start, end)) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{arraystore}\ t\ (part\ 1\ of\ 3)$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X::A_2::A_1$$
$$(\{\widehat{\sigma}_{\texttt{Null}}\} \sqsubseteq_S X) \ \Rightarrow$$
$$\quad \mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{NullPointerException}})$$

$$\forall\, \{O_q\} = \{(\mathtt{r},Y,(m_r,pc_r))\} \sqsubseteq_S X :$$

$$(O_q.\mathit{isArray}) \ \Rightarrow$$

$$O_q.\mathit{length} \quad = \quad (min_array_length, max_array_length)$$

$$array_dom_size \quad = \quad \left| \begin{array}{c} dom\big(\widehat{JH}(m_n,pc_n)(\pi_1)(O_q).\mathit{values}\big) \\ \cup \\ dom\big(\widehat{H}(O_q).\mathit{values}\big) \end{array} \right|$$

$$\forall\, \{(\mathtt{s},(l,h,mod),(m_q,pc_q))\} \sqsubseteq_S A_2 :$$

$$\forall\, \{O_v\} = \{(t_v,W,(m_v,pc_v))\} \sqsubseteq_S A_1 :$$

$$\neg \mathit{checkArrayStore}\,(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t=\mathtt{r})\ ) \ \Rightarrow$$

$$\quad \mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{SecurityException}})$$

$$\mathit{checkArrayStore}\,(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t=\mathtt{r})\ ) \ \Rightarrow$$

$$\Big((t=t_v=\mathtt{r}) \wedge \neg O_v.\mathit{isArray} \wedge (O_v.\mathit{type} \not\preceq O_q.\mathit{type})\Big) \ \Rightarrow$$
$$\quad \mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{ArrayStoreException}})$$

$$((l<0) \vee (h \geq min_array_length)) \ \Rightarrow$$
$$\quad \mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{ArrayIndexOutOfBoundsException}})$$

$$(h \geq 0) \ \Rightarrow$$

$$\quad \forall j \in \{max(0,l),\ldots,min(max_array_length,h)\} :$$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY, RDLV}_{\text{Base-CFA}} (m_n, pc_n)$: arraystore t $(part\ 2\ of\ 3)$

$$\forall (start, end) \in \begin{pmatrix} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).values) \\ \cup \\ dom(\widehat{H}(O_q).values) \end{pmatrix} . (start \leq j \leq end) :$$

$$\Big(\big((t = t_v = \texttt{r}) \wedge \neg O_v.isArray \wedge (O_v.type \preceq O_q.type)\big) \vee \big((t = t_v = \texttt{s})\big) \vee \big((t = t_v = \texttt{i})\big)\Big) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.nextAddress(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\neg\Big(\gamma_n = 1 \wedge O_q.transient = \texttt{NOT_TRANSIENT} \wedge \neg O_v.isGlobal\Big) \Rightarrow$$

$(t = \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).values((start, end))$
$(t \neq \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).values((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)$

$$\Big(\gamma_n = 1 \wedge O_q.transient = \texttt{NOT_TRANSIENT} \wedge \neg O_v.isGlobal\Big) \Rightarrow$$

$(t = \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)(O_q).values((start, end))$
$(t \neq \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)(O_q).values((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)$

$$\Big((t = \texttt{b}) \wedge (t_v = \texttt{s})\Big) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.nextAddress(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\neg\Big(\gamma_n = 1 \wedge O_q.transient = \texttt{NOT_TRANSIENT} \wedge \neg O_v.isGlobal\Big) \Rightarrow$$

$(\texttt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$absFromShort((t_v, W, (m_v, pc_v)), \texttt{b})$
$\{(\texttt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).values((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)$

$$\Big(\gamma_n = 1 \wedge O_q.transient = \texttt{NOT_TRANSIENT} \wedge O_v.isGlobal\Big) \Rightarrow$$

$(\texttt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$absFromShort((t_v, W, (m_v, pc_v)), \texttt{b})$
$\{(\texttt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)(O_q).values((start, end))$:
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY, RDLV}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{arraystore}\ t\ (part\ 3\ of\ 3)$

$$\forall (D_1, D_2) \in \begin{pmatrix} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q).\mathit{values}) \\ \cup \\ dom(\widehat{H}(O_q).\mathit{values}) \end{pmatrix} . (D_1 \leq j \leq D_2) :$$

$$(start, end) = \begin{cases} (0, \texttt{Short.MAX_VALUE}), & array_dom_size \geq MAX_DOM_DYN_ARRAY, \\ (max(0, l), min(max_array_length, h)), & array_dom_size < MAX_DOM_DYN_ARRAY \end{cases}$$

$$\Big(\big((t = t_v = \texttt{r}) \wedge \neg O_v.\mathit{isArray} \wedge (O_v.\mathit{type} \preceq O_q.\mathit{type})\big) \vee \big((t = t_v = \texttt{s})\big) \vee \big((t = t_v = \texttt{i})\big)\Big) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$$

$(t = \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}((start, end))$
$(t \neq \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$$

$(t = \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}((start, end))$
$(t \neq \texttt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\Big((t = \texttt{b}) \wedge (t_v = \texttt{s})\Big) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$$

$(\texttt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t_v, W, (m_v, pc_v)), \texttt{b})$
$\{(\texttt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q).\mathit{values}((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge O_v.\mathit{isGlobal}\Big) \Rightarrow$$

$(\texttt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t_v, W, (m_v, pc_v)), \texttt{b})$
$\{(\texttt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q).\mathit{values}((start, end)) :$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{throw}$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M{::}X$$
$$\forall\, \{O_q\} = \{(\mathbf{r},Y,(m_r,pc_r))\} \sqsubseteq_S X :$$

$$\left(\widehat{\sigma}_{\texttt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}\right) \Rightarrow$$
$$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{NullPointerException}})$$
$$\neg(O_q.\mathit{isArray}) \Rightarrow$$

$$\mathit{checkThrow}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$$

$$\mathsf{HANDLE}(\pi_1,\pi_1,O_q)$$

$$\neg\mathit{checkThrow}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$$

$$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{SecurityException}})$$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n,pc_n) : \texttt{jsr}\ \mathit{addr}$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,\mathit{addr}))$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,\mathit{addr})$
$\widehat{S}(m_n,pc_n)(\pi_1)::\{(\mathrm{ra},\ m_n.\mathit{nextAddress}(pc_n),(m_n,pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n,\mathit{addr})(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,\mathit{addr})(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,\mathit{addr})(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,\mathit{addr})(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{ret}\ i$$

$$\iff \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\forall\, \{(\mathrm{ra},\ addr, (m_r, pc_r))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(i) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \gamma_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \gamma_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \gamma_n, (m_n, addr)))$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, addr)$
$\widehat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, addr)(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, addr)(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, addr)(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, addr)(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

4.6 Whole Program Analysis

In this Section we specify the conditions for being able to analyse a whole Carmel program P. In words, this amounts to:

- ensuring the abstract versions of the valid initial configurations of Section 3.7.1 are represented in the analysis. In particular, we must ensure the correctness relations of Section 4.4.1 hold for each of the initial configurations;

- any static field initialisers in P should be represented in the analysis;

- each bytecode instruction must be analysed;

The next two tables present the above in symbols. Any unfamiliar constant is defined in Appendix $D.7$. The first table contains the initial configurations corresponding to the static `javacard.framework.Applet.install(byte[],short,byte)` methods of Tables 1 and 2 of the valid initial configurations of Section 3.7.1, and the second table contains the remaining three tables, which represents the invocation of Applet lifecycle methods on registered applets with the appropriate machine configuration and parameters for local variable array and operand stack. From inspection, it is clear the whole program analysis conditions correctly relate the initial configurations to the analysis domains of $\widehat{\text{Analysis}}$ i.e. the whole program analysis conditions satisfy the correctness relations of Section 4.4.1 wrt the valid initial configurations of Section 3.7.1. In the second table, we also include the requirement that the analysis results must represent the analysis results of the individual instructions i.e.

$$\forall\, (m_n, pc_n) \in P.addresses :$$
$$(\hat{R}, \widehat{K}, \widehat{H}, \hat{I}, \widehat{JH}, \hat{C}, \hat{L}, \hat{S}, \hat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : m_n.\mathit{instructionAt}(pc_n)$$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} P$ (Part 1 of 2)

$\Longleftrightarrow$

$\beta^{H,JH}_{DynamicHeap}(H_{Init}) \quad \sqsubseteq_H \quad \widehat{H}$

$\beta^{H}_{StaticHeap}(K_{Init}) \quad \sqsubseteq_K \quad \widehat{K}$

$\wedge$

$$C_1 = \left(\left(\left(\mathtt{r}, \left(\begin{array}{l} \mathit{type} : \widehat{\sigma}_{\mathtt{Null}}, \\ \mathit{refType} : \widehat{\sigma}_{\mathtt{Null}}, \\ \mathit{isArray} : \mathtt{false}, \\ \mathit{owner} : (\widehat{\sigma}_{\mathtt{JCRE}}, \widehat{\sigma}_{\mathtt{Null}}), \\ \mathit{entryPoint} : \mathtt{no}, \\ \mathit{isGlobal} : \mathtt{false}, \\ \mathit{transient} : \mathtt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (1, 1), \\ \mathit{length} : 0, \end{array}\right), (1, 1)\right), 0, 0, \mathtt{INITIAL}, (\mathtt{loop}(), 0)\right)$$

$\wedge$

$$C_2 = \left(\left(\left(\mathtt{r}, \left(\begin{array}{l} \mathit{type} : \widehat{\sigma}_{\mathtt{Null}}, \\ \mathit{refType} : \widehat{\sigma}_{\mathtt{Null}}, \\ \mathit{isArray} : \mathtt{false}, \\ \mathit{owner} : (\widehat{\sigma}_{\mathtt{JCRE}}, \widehat{\sigma}_{\mathtt{Null}}), \\ \mathit{entryPoint} : \mathtt{no}, \\ \mathit{isGlobal} : \mathtt{false}, \\ \mathit{transient} : \mathtt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (1, 1), \\ \mathit{length} : 0, \end{array}\right), (1, 1)\right), 0, 0, \mathtt{INITIAL}, (\mathtt{dispatch}(), 0)\right)$$

$\forall\, p \in P.packages :$

$\forall\, c \in p.classes :$

$\forall f \in \mathit{fields}(c)\ .\ f.\mathit{isStatic} : \quad \beta^{R,H,JH}_{Val}(f.\mathit{initValue}) \sqsubseteq_K \widehat{K}(f.\mathit{id})$

$c \prec \mathtt{javacard.framework.Applet} \Rightarrow$

$m_{inst} = \mathit{methodLookup}(\mathtt{javacard.framework.Applet.install(byte[], short, byte)}.\mathit{id}, c)$

$(m_{inst} \neq \bot) \Rightarrow$

$\wedge$

$$C_3 = \left(\left(\left(\mathtt{r}, \left(\begin{array}{l} \mathit{type} : c, \\ \mathit{refType} : c, \\ \mathit{isArray} : \mathtt{false}, \\ \mathit{owner} : (p.packageAID, cl), \\ \mathit{entryPoint} : \mathtt{no}, \\ \mathit{isGlobal} : \mathtt{false}, \\ \mathit{transient} : \mathtt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (1, 1), \\ \mathit{length} : 0, \end{array}\right), (1, 1)\right), 0, 0, \mathtt{INITIAL}, (\mathtt{m_{inst}}, \mathtt{m_{inst}}.\mathit{firstAddress})\right)$$

π	$=$	$C_1 :: C_2 :: C_3$
$\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{inst}, m_{inst}.\mathit{firstAddress})$
$\{m_{inst}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)$
$\{\widehat{\sigma}_{\mathtt{AppletInstallBuffer}}\}$	$\sqsubseteq_L$	$\widehat{L}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)(0)$
$\{(\mathtt{s}, (0, 127, 0), (m_{inst}, m_{inst}.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)(1)$
$\{(\mathtt{b}, (0, 127, 0), (m_{inst}, m_{inst}.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)(2)$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)$
$\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} P$ (Part 2 of 2)

$\wedge \quad \forall \{O_3\} \sqsubseteq_R \widehat{R}:$

$m_{process} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.process(APDU)}.\mathit{id}, O_3.\mathit{type})$

$(m_{process} \neq \bot) \Rightarrow$

C_3	$=$	$\big(O_3, 0, 0, \texttt{INITIAL}, (\texttt{m}_{\texttt{process}}, \texttt{m}_{\texttt{process}}.\mathit{firstAddress})\big)$
π	$=$	$C_1::C_2::C_3$
$\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{process}, m_{process}.\mathit{firstAddress})$
$\{m_{process}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)$
$\{O_3\}$	$\sqsubseteq_L$	$\widehat{L}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)(0)$
$\{\hat{\sigma}_{\texttt{APDUBuffer}}\}$	$\sqsubseteq_L$	$\widehat{L}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)(1)$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)$
$\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)$

$\wedge \quad \forall \{O_3\} \sqsubseteq_R \widehat{R}:$

$\forall m \in \{ \ \texttt{javacard.framework.Applet.select()}, \texttt{javacard.framework.Applet.deselect()} \ \}:$

$m_{singleSoD} = \mathit{methodLookup}(m.\mathit{id}, O_3.\mathit{type}))$

$(m_{singleSoD} \neq \bot) \Rightarrow$

C_3	$=$	$\big(O_3, 0, 0, \texttt{INITIAL}, (\texttt{m}_{\texttt{singleSoD}}, \texttt{m}_{\texttt{singleSoD}}.\mathit{firstAddress})\big)$
π	$=$	$C_1::C_2::C_3$
$\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})$
$\{m_{singleSoD}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})(\pi)$
$\{O_3\}$	$\sqsubseteq_L$	$\widehat{L}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})(\pi)(0)$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})(\pi)$
$\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)$

$\wedge \quad \forall \{O_3\} \sqsubseteq_R \widehat{R}:$

$(O_3.\mathit{type} \preceq \texttt{javacard.framework.MultiSelectable}) \Rightarrow$

$\forall m \in \{ \ \texttt{javacard.framework.MultiSelectable.select()}, \texttt{javacard.framework.MultiSelectable.deselect()} \ \}:$

$m_{multiSoD} = \mathit{methodLookup}(m.\mathit{id}, O_3.\mathit{type}))$

$(m_{multiSoD} \neq \bot) \Rightarrow$

C_3	$=$	$(O_3, 0, 0, \texttt{INITIAL}, (\texttt{m}_{\texttt{multiSoD}}, \texttt{m}_{\texttt{multiSoD}}.\mathit{firstAddress}))$
π	$=$	$C_1::C_2::C_3$
$\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})$
$\{m_{multiSoD}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)$
$\{O_3\}$	$\sqsubseteq_L$	$\widehat{L}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)(0)$
$\{(\texttt{s}, (0, 1, 0), (m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)(1)$
$\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)$
$\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)$

$\wedge \quad \forall (m_n, pc_n) \in P.addresses:$
$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : m_n.\mathit{instructionAt}(pc_n)$

4.7 Constructing a worklist solver

Having specified the program analysis flow-logic clauses for the base control-flow analysis and the program analysis conditions for the whole program analysis, we are able to construct an iterative worklist-solver capable of

computing the least solution to the constraints generated by the base control-flow analysis. Our approach relies on each program analysis flow-logic clause defining a monotone function over a complete lattice (here $\widehat{\text{Analysis}}$), which can easily be verified by inspection, and Tarski's theorem, which guarantees the existence of a least-fixed point to the set of monotone functions in $\widehat{\text{Analysis}}$. Our worklist algorithm then iterates from bottom to the least-fixed point in $\widehat{\text{Analysis}}$ and Tarski's theorem guarantees this least-fixed point will be reached in a finite number of steps.

In this Section, the reader is expected to translate familiar first-order-logic and quantifiers into standard imperative programming parlance, or rather to appreciate that is what is intended e.g. $\forall \{O_3\} \sqsubseteq_R \widehat{R}$, the reader is expected to read this as:

$$\texttt{for each element } O_3 \sqsubseteq_R \widehat{R}$$

statements like $(m_{inst} \neq \bot) \Rightarrow S$ are to be interpreted as conditionals i.e. `if <cond> then do` S, and statements like $\pi = C_1::C_2::C_3$ are assumed to be translated into some suitable stack operations in the language of implementation. I think this is clearer and better than artificially translating the whole program analysis criteria into simple `while` program constructs.

The iterative solver has the following outline form:

1. Initialise analysis domains solution $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \in \widehat{\text{Analysis}}$ to bottom;
2. Ensure the constraints of Table 1 of Section 4.6 are satisfied – this needs to be done just once as these constraints depend only on the static structure of P;
3. Initialise the global boolean flag `shouldIterateOverWorklist` to `true` then enter the main body of the iterative solver, whose guard is `while` $(\texttt{shouldIterateOverWorklist} = \texttt{true})$ `do`, and whose first instruction is to assign `false` to `shouldIterateOverWorklist`. This is a typical high-level control-structure for worklist solvers, where, in the main loop of the loop, if some LUB calculation climbs the lattice, we shall set `shouldIterateOverWorklist` to `true` and force at least one more iteration to check whether a fixed point has been reached;
4. Ensure the constraints beginning with $\forall \{O_3\} \sqsubseteq_R \widehat{R}$ of Table 2 of Section 4.6 are satisfied – this needs to be done each loop to ensure the valid initial configurations for registered applets are represented in the analysis;
5. In the main body of the loop, load all the addresses of the bytecode instructions in P onto a worklist W;

6. Pop the first address (m_n, pc_n) off the worklist W and check whether the current solution satisfies the flow-logic clauses for the instruction at that address. If while checking the solution we increase the cardinality of any analysis relation or otherwise climb the lattice by, for example, widening an abstract number as part of a LUB calculation, one of the constraint-satisfaction-with-equality predicates $\texttt{satisfy}\big(\big)$ defined in Tables 4.8 and 4.9 will set `shouldIterateOverWorklist` to `true` and guarantee the main loop will be executed at least once more;

7. Repeat the previous step until W is empty;

8. Once the body of the loop terminates, which it must do based on the finite height of the complete lattices of the analysis domains under consideration, we must have reached a fixed point. Since all constraints have been satisfied with a LUB operator, either $\sqcup_{Val}$ or $\cup$, this fixed point must be least.

Fuller details on the implementation of our worklist algorithm can be found in Appendix E.

4.7.1 Step 1: Initialisation of Analysis Domains/Components

Let $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \in \widehat{\text{Analysis}}$.

The first step is then the initialisation of each domain of the analysis result to the bottom element of the domain.

$$
\begin{array}{lcll}
\widehat{R} & := & \bot_{\widehat{R}} & := \emptyset \\
\widehat{K} & := & \bot_{\widehat{K}} & := [\,] \\
\widehat{H} & := & \bot_{\widehat{H}} & := [\,] \\
\widehat{I} & := & \bot_{\widehat{I}} & := \emptyset \\
\widehat{JH} & := & \bot_{\widehat{JH}} & := [\,] \\
\widehat{C} & := & \bot_{\widehat{C}} & := [\,] \\
\widehat{L} & := & \bot_{\widehat{L}} & := [\,] \\
\widehat{S} & := & \bot_{\widehat{S}} & := [\,] \\
\widehat{E} & := & \bot_{\widehat{E}} & := \emptyset \\
\widehat{MNAMES} & := & \bot_{\widehat{MNAMES}} & := [\,] \\
\widehat{REC} & := & \bot_{\widehat{REC}} & := \emptyset \\
\widehat{CG} & := & \bot_{\widehat{CG}} & := \emptyset
\end{array}
$$

4.7.2 Step 2: Initialisation of the *Java Card* framework analysis objects and static initialisers of Carmel program P

Using the constraint-satisfaction-with-equality predicates $\texttt{satisfy}\big(\big)$ defined in Tables 4.8 and 4.9, load the abstract equivalent of the `install` methods.

$$
\begin{array}{ll}
\texttt{satisfy}\Big(\beta^{H,JH}_{DynamicHeap}(H_{Init}) & \sqsubseteq_H \quad \widehat{H}\Big) \\
\texttt{satisfy}\Big(\beta^{H}_{StaticHeap}(K_{Init}) & \sqsubseteq_K \quad \widehat{K}\Big)
\end{array}
$$

$$
C_1 = \left\langle \left\langle \begin{array}{l} \mathtt{r}, \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left(\begin{array}{l} \textit{type} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{refType} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{isArray} : \mathtt{false}, \\ \textit{owner} : (\widehat{\sigma}_{\mathtt{JCRE}}, \widehat{\sigma}_{\mathtt{Null}}), \\ \textit{entryPoint} : \mathtt{no}, \\ \textit{isGlobal} : \mathtt{false}, \\ \textit{transient} : \mathtt{NOT_TRANSIENT}, \\ \textit{creationPoint} : (1,1), \\ \textit{length} : 0, \end{array} \right) \right\rangle \left\langle \begin{array}{l} \mathtt{r}, \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left(\begin{array}{l} \textit{type} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{refType} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{isArray} : \mathtt{false}, \\ \textit{owner} : (\widehat{\sigma}_{\mathtt{JCRE}}, \widehat{\sigma}_{\mathtt{Null}}), \\ \textit{entryPoint} : \mathtt{no}, \\ \textit{isGlobal} : \mathtt{false}, \\ \textit{transient} : \mathtt{NOT_TRANSIENT}, \\ \textit{creationPoint} : (1,1), \\ \textit{length} : 0, \end{array} \right) \right\rangle \right\rangle
$$

$$
\begin{array}{l} , (1,1) \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left\rangle \begin{array}{l} , 0, 0, \mathtt{INITIAL}, (\mathtt{loop}(), 0) \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \right\rangle
$$

$$
C_2 = \left\langle \left\langle \begin{array}{l} \mathtt{r}, \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left(\begin{array}{l} \textit{type} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{refType} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{isArray} : \mathtt{false}, \\ \textit{owner} : (\widehat{\sigma}_{\mathtt{JCRE}}, \widehat{\sigma}_{\mathtt{Null}}), \\ \textit{entryPoint} : \mathtt{no}, \\ \textit{isGlobal} : \mathtt{false}, \\ \textit{transient} : \mathtt{NOT_TRANSIENT}, \\ \textit{creationPoint} : (1,1), \\ \textit{length} : 0, \end{array} \right) \right\rangle \left\langle \begin{array}{l} \mathtt{r}, \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left(\begin{array}{l} \textit{type} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{refType} : \widehat{\sigma}_{\mathtt{Null}}, \\ \textit{isArray} : \mathtt{false}, \\ \textit{owner} : (\widehat{\sigma}_{\mathtt{JCRE}}, \widehat{\sigma}_{\mathtt{Null}}), \\ \textit{entryPoint} : \mathtt{no}, \\ \textit{isGlobal} : \mathtt{false}, \\ \textit{transient} : \mathtt{NOT_TRANSIENT}, \\ \textit{creationPoint} : (1,1), \\ \textit{length} : 0, \end{array} \right) \right\rangle \right\rangle
$$

$$
\begin{array}{l} , (1,1) \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left\rangle \begin{array}{l} , 0, 0, \mathtt{INITIAL}, (\mathtt{dispatch}(), 0) \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \right\rangle
$$

$\forall\, p \in P.packages :$

$\quad \forall\, c \in p.classes :$

$\quad\quad \forall f \in \mathit{fields}(c)\ .\ f.\mathit{isStatic} :\quad \texttt{satisfy}\Big(\beta_{Val}^{R,H,JH}(f.\mathit{initValue}) \sqsubseteq_K \widehat{K}(f.\mathit{id})\Big)$

$\quad\quad c \prec \texttt{javacard.framework.Applet} \Rightarrow$

$\quad\quad\quad m_{inst} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.install(byte[], short, byte)}.\mathit{id}, c)$

$\quad\quad\quad (m_{inst} \neq \bot) \Rightarrow$

$$C_3 = \left(\left(\left(\begin{array}{l} \mathit{type} : c, \\ \mathit{refType} : c, \\ \mathit{isArray} : \texttt{false}, \\ \mathit{owner} : (p.packageAID, cl), \\ \mathit{entryPoint} : \texttt{no}, \\ \mathit{isGlobal} : \texttt{false}, \\ \mathit{transient} : \texttt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (1,1), \\ \mathit{length} : 0, \end{array}\right), (1,1)\right), 0, 0, \texttt{INITIAL}, (\texttt{m}_{\texttt{inst}}, \texttt{m}_{\texttt{inst}}.\mathit{firstAddress})\right)$$

π	$=$	$C_1 :: C_2 :: C_3$
$\texttt{satisfy}\big(\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{inst}, m_{inst}.\mathit{firstAddress})\big)$
$\texttt{satisfy}\big(\{m_{inst}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{\widehat{\sigma}_{\texttt{AppletInstallBuffer}}\}$	$\sqsubseteq_L$	$\widehat{L}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)(0)\big)$
$\texttt{satisfy}\big(\{(\texttt{s}, (0,127,0), (m_{inst}, m_{inst}.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)(1)\big)$
$\texttt{satisfy}\big(\{(\texttt{b}, (0,127,0), (m_{inst}, m_{inst}.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)(2)\big)$
$\texttt{satisfy}\big(\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{inst}, m_{inst}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)\big)$

4.7.3 Step 3: Define a global boolean flag `shouldIterateOverWorklist` which controls the repetition of the main loop

We define the variable `shouldIterateOverWorklist` to be `true` immediately prior to entering the main body of the iterative worklist:

`shouldIterateOverWorklist` $:=$ `true`

`while (shouldIterateOverWorklist = true)` do

`shouldIterateOverWorklist` $:=$ `false`

4.7.4 Step 4: Ensure the constraints associated with the initial configurations of registered applets are satisfied

Ensure the initial configurations of registered applets are represented in the analysis.

$\forall\, \{O_3\} \sqsubseteq_R \widehat{R}$:

$m_{process} = \mathit{methodLookup}(\texttt{javacard.framework.Applet.process(APDU)}.\mathit{id}, O_3.\mathit{type})$

$(m_{process} \neq \bot) \Rightarrow$

C_3	$=$	$\big(O_3, 0, 0, \texttt{INITIAL}, (\texttt{m}_{\texttt{process}}, \texttt{m}_{\texttt{process}}.\mathit{firstAddress})\big)$
π	$=$	$C_1 :: C_2 :: C_3$
$\texttt{satisfy}\big(\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{process}, m_{process}.\mathit{firstAddress})\big)$
$\texttt{satisfy}\big(\{m_{process}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{O_3\}$	$\sqsubseteq_L$	$\widehat{L}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)(0)\big)$
$\texttt{satisfy}\big(\{\widehat{\sigma}_{\texttt{APDUBuffer}}\}$	$\sqsubseteq_L$	$\widehat{L}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)(1)\big)$
$\texttt{satisfy}\big(\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{process}, m_{process}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)\big)$

$\forall\, \{O_3\} \sqsubseteq_R \widehat{R}$:

$\forall m \in \{$ $\texttt{javacard.framework.Applet.select()}, \texttt{javacard.framework.Applet.deselect()}$ $\}$:

$m_{singleSoD} = \mathit{methodLookup}(m.\mathit{id}, O_3.\mathit{type}))$

$(m_{singleSoD} \neq \bot) \Rightarrow$

C_3	$=$	$\big(O_3, 0, 0, \texttt{INITIAL}, (\texttt{m}_{\texttt{singleSoD}}, \texttt{m}_{\texttt{singleSoD}}.\mathit{firstAddress})\big)$
π	$=$	$C_1 :: C_2 :: C_3$
$\texttt{satisfy}\big(\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})\big)$
$\texttt{satisfy}\big(\{m_{singleSoD}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{O_3\}$	$\sqsubseteq_L$	$\widehat{L}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})(\pi)(0)\big)$
$\texttt{satisfy}\big(\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{singleSoD}, m_{singleSoD}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)\big)$

$\forall\, \{O_3\} \sqsubseteq_R \widehat{R}$:

$(O_3.\mathit{type} \preceq \texttt{javacard.framework.MultiSelectable}) \Rightarrow$

$\forall m \in \{$ $\texttt{javacard.framework.MultiSelectable.select()}, \texttt{javacard.framework.MultiSelectable.deselect()}$ $\}$:

$m_{multiSoD} = \mathit{methodLookup}(m.\mathit{id}, O_3.\mathit{type}))$

$(m_{multiSoD} \neq \bot) \Rightarrow$

C_3	$=$	$(O_3, 0, 0, \texttt{INITIAL}, (\texttt{m}_{\texttt{multiSoD}}, \texttt{m}_{\texttt{multiSoD}}.\mathit{firstAddress}))$
π	$=$	$C_1 :: C_2 :: C_3$
$\texttt{satisfy}\big(\{\pi\}$	$\sqsubseteq_C$	$\widehat{C}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})\big)$
$\texttt{satisfy}\big(\{m_{multiSoD}\}$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{O_3\}$	$\sqsubseteq_L$	$\widehat{L}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)(0)\big)$
$\texttt{satisfy}\big(\{(\texttt{s}, (0, 1, 0), (m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress}))\}$	$\sqsubseteq_L$	$\widehat{L}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)\big)(1)$
$\texttt{satisfy}\big(\{\epsilon\}$	$\sqsubseteq_S$	$\widehat{S}(m_{multiSoD}, m_{multiSoD}.\mathit{firstAddress})(\pi)\big)$
$\texttt{satisfy}\big(\{\emptyset\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, pc_n)(\pi)\big)$

4.7.5 Step 5: Load all the addresses of the bytecode instructions in P

`W:=` $\{addr \mid addr \in P.addresses\}$

`while (W.peek()` $\neq$ `nil) do`

4.7.6 Step 6: Pop the top address $addr$ from the worklist W and check constraints for Carmel instruction at $addr$

(m_n, pc_n):=W.pop()

switch (instruction at (m_n, pc_n)):

Case instruction labels `nop` instruction at address (m_n, pc_n), check:

$$\forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n):$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\begin{array}{lll}
\texttt{satisfy}\big(\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))\big)\\
\texttt{satisfy}\big(\widehat{S}(m_n,pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\widehat{L}(m_n,pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\widehat{JH}(m_n,pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\widehat{MNAMES}(m_n,pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\{(\pi_1,\pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}\big)
\end{array}$$

Case instruction labels `push t c` instruction at address (m_n, pc_n), check:

$$\forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n):$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$$\begin{array}{llll}
\texttt{satisfy}\big(\{\pi_2\} & & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))\big)\\
(t = \texttt{r}) & \Rightarrow\ \texttt{satisfy}\big(\widehat{S}(m_n,pc_n)(\pi_1)::\{\hat{\sigma}_{\texttt{Null}}\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
(t \in \{\texttt{b},\texttt{s}\}) & \Rightarrow\ \texttt{satisfy}\big(\widehat{S}(m_n,pc_n)(\pi_1)::\{(\texttt{s},(c,c,0),(m_n,pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
(t = \texttt{i}) & \Rightarrow\ \texttt{satisfy}\big(\widehat{S}(m_n,pc_n)(\pi_1)::\{(\texttt{i},(c,c,0),(m_n,pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\widehat{L}(m_n,pc_n)(\pi_1) & & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\widehat{JH}(m_n,pc_n)(\pi_1) & & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\widehat{MNAMES}(m_n,pc_n)(\pi_1) & & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big)\\
\texttt{satisfy}\big(\{(\pi_1,\pi_2)\} & & \sqsubseteq_{CG} & \widehat{CG}\big)
\end{array}$$

. . . [All Carmel bytecode instructions and API methods flow-logic clauses] . . .

4.7.7 Step 7: Repeat until worklist W is empty

The exhaustion of the worklist W is technically part of Step 5. When we reach this point, W will be empty and we will then return to Step 3 to check the condition `shouldIterateOverWorklist`.

`shouldIterateOverWorklist` will be `true` if any of the LUB calculations in the previous iteration over W increased the cardinality of any of the abstract domains or otherwise climbed the lattice e.g. widening an abstract number. If `shouldIterateOverWorklist` is `false`, we must have reached a fixed point and for reasons argued above, this must be the least-fixed point by our choice of LUB operators.

$\texttt{satisfy}\Big(X \sqsubseteq_{Val} Y\Big)$	$\iff$	$(X \not\sqsubseteq_{Val} Y) \Rightarrow$ $Y := Y \sqcup_{Val} X$ $\texttt{shouldIterateOverWorklist} := \texttt{true}$
$\texttt{satisfy}\Big(X \subseteq Y\Big)$	$\iff$	$(X \not\subseteq Y) \Rightarrow$ $Y := Y \cup X$ $\texttt{shouldIterateOverWorklist} := \texttt{true}$

Table 4.8: Fundamental Constraint Satisfaction/Update Functions

$$
\begin{array}{lcl}
\texttt{satisfy}\Big(r_1 \sqsubseteq_R r_2\Big) & \iff & \texttt{satisfy}\Big(r_1 \subseteq r_2\Big) \\
\texttt{satisfy}\Big(k_1 \sqsubseteq_K k_2\Big) & \iff & \forall\, f \in \textit{dom}(k_1) : \texttt{satisfy}\Big(k_1(f.\textit{id}) \sqsubseteq_{Val} k_2(f.\textit{id})\Big) \\
\texttt{satisfy}\Big(h_1 \sqsubseteq_H h_2\Big) & \iff & \forall\, ref \in \textit{dom}(h_1) : \\
 & & \quad o_1 = h_1(ref) \wedge \\
 & & \quad o_2 = h_2(ref) \wedge \\
 & & \quad \forall f \in \textit{dom}(o_1.\textit{values}) : \\
 & & \quad\quad \texttt{satisfy}\Big(o_1.\textit{values}(f) \sqsubseteq_{Val} o_2.\textit{values}(f)\Big) \\
\texttt{satisfy}\Big(i_1 \sqsubseteq_I i_2\Big) & \iff & \texttt{satisfy}\Big(i_1 \subseteq i_2\Big) \\
\texttt{satisfy}\Big(t_1 \sqsubseteq_{JH} t_2\Big) & \iff & \forall\, addr \in \textit{dom}(t_1), \\
 & & \quad \forall\, ctxt \in t_1(addr) : \\
 & & \quad\quad \forall\, ref \in t_1(addr)(ctxt) : \\
 & & \quad\quad\quad o_1 = t_1(addr)(ctxt)(ref) \wedge \\
 & & \quad\quad\quad o_2 = t_2(addr)(ctxt)(ref) \wedge \\
 & & \quad\quad\quad\quad \texttt{satisfy}\Big(o_1.\textit{values}(f) \sqsubseteq_{Val} o_2.\textit{values}(f)\Big) \\
\texttt{satisfy}\Big(c_1 \sqsubseteq_C c_2\Big) & \iff & \forall\, addr \in \textit{dom}(c_1) : \texttt{satisfy}\Big(c_1(addr) \subseteq c_2(addr)\Big) \\
\texttt{satisfy}\Big(l_1 \sqsubseteq_L l_2\Big) & \iff & \forall\, addr \in \textit{dom}(l_1), \\
 & & \quad \forall\, ctxt \in l_1(addr), \\
 & & \quad\quad \forall\, idx \in dom(l_1(addr)(ctxt)) : \\
 & & \quad\quad\quad \texttt{satisfy}\Big(l_1(addr)(ctxt)(idx) \sqsubseteq_{Val} l_2(addr)(ctxt)(idx)\Big) \\
\texttt{satisfy}\Big(s_1 \sqsubseteq_S s_2\Big) & \iff & \forall\, addr \in \textit{dom}(s_1), \\
 & & \quad \forall\, ctxt \in s_1(addr) : \\
 & & \quad\quad s_1(addr)(ctxt) = A_1 :: A_2 :: \ldots :: A_q \wedge \\
 & & \quad\quad s_2(addr)(ctxt) = B_1 :: B_2 :: \ldots :: B_r \wedge \\
 & & \quad\quad r \geq q \wedge \\
 & & \quad\quad \forall i \in \{1, \ldots, q\} : \\
 & & \quad\quad\quad \texttt{satisfy}\Big(A_i \sqsubseteq_{Val} B_i\Big) \\
\texttt{satisfy}\Big(e_1 \sqsubseteq_E e_2\Big) & \iff & \texttt{satisfy}\Big(e_1 \subseteq e_2\Big) \\
\texttt{satisfy}\Big(m_1 \sqsubseteq_{\widehat{MNAMES}} m_2\Big) & \iff & \forall\, addr \in \textit{dom}(m_1), \\
 & & \quad \forall\, ctxt \in m_1(addr) : \\
 & & \quad\quad \texttt{satisfy}\Big(m_1(addr)(ctxt) \subseteq m_2(addr)(ctxt)\Big) \\
\texttt{satisfy}\Big(r_1 \sqsubseteq_{REC} r_2\Big) & \iff & \texttt{satisfy}\Big(r_1 \subseteq r_2\Big) \\
\texttt{satisfy}\Big(cg_1 \sqsubseteq_{CG} cg_2\Big) & \iff & \texttt{satisfy}\Big(cg_1 \subseteq cg_2\Big)
\end{array}
$$

Table 4.9: Higher-Order Constraint Satisfaction/Update Functions

Chapter 5

Extended Control-Flow Analysis

5.1 Introduction

In this Chapter we detail a key deficiency in the base control-flow analysis of Chapter 4 which makes the calculation of loop-bounds so conservative as to be useless, and remedy this deficiency with replacement clauses for the `if` bytecode instructions in an extended control-flow analysis which allows simple loops to be bounded. Moreover, we integrate into the extended control-flow analysis the results of a variant of the classical reaching definitions analysis[1] to further improve the precision of loop-bounds calculation in Carmel.

5.2 Deficiency in base control-flow analysis handling of loops

Having analysed a slew of different types and complexities of loops in Carmel using the base control-flow analysis worklist algorithm of Chapter 4, one distinct issue impacting precision kept presenting itself. We have distilled this issue into its simplest form to aid discussion: a simple bounded loop. The semantics-based solution to this problem, presented in Section 5.3, has proved sufficiently robust to remedy this issue and allowed the calculation of conservative loop bounds for each of the loops we tested, including the Carmel sample given in this Chapter.

5.2.1 Even simple numeric loops cannot be bounded finitely

Consider the Java and corresponding Carmel code of Table 5.1, which shows a simple bounded loop that loops 17 times, where the Java variable i is mapped to the Carmel local variable 11. The $desired$ outcome of applying

[1] See Appendix C or [NNH10] for further information.

Java code:

```
short i=0;
for (i=0; i < 17; i++){

     //body of loop - assume empty statement for simplicity
 }
 //the value of i is 17 at the point the loop terminates
```

Carmel translation:

```
 0: push  s 0
 1: store s 11
 2: load  s 11
 3: push  s 17
 5: if    s ge goto 14
 8: inc   s 11 1
11: goto  2
14: return
```

Table 5.1: Simple bounded loop, in Java and Carmel

our worklist algorithm is given in Table 5.3. The $actual$ results of applying our worklist algorithm to this program can be seen in Table 5.4 and does not make for happy reading. It is extremely disappointing that, for such a simple program, the base control-flow analysis is unable to deduce:

- a useful approximation of the possible values of the local variable 11 at the `if s goto` statement of line 5
- the value of the local variable 11 must be 17 at the `return` statement

In each case, the values map to $(-32768, 32767) = (\bot_{\mathbf{s}}, \top_{\mathbf{s}})$ i.e. all we know about the value is that it is in the range of a `short`.

Looking more closely at the base control-flow analysis flow-logic clauses for the two `if` statement forms in Carmel, the more general of which is reproduced for convenience in Table 5.2, it becomes clear why $(\bot_{\mathbf{s}}, \top_{\mathbf{s}})$ is the least solution for the local variable 11: the relationship between local variables on the operand stack and the local variable array is ignored. In particular, the relationship between local variables `load`ed onto the operand stack *that serve as loop variables* and the local variable array is ignored. Consequently, values for local variables are propagated to branches of `if` statements that are not semantically possible – here for instance, it is not semantically possible for the value of local variable 11 to be ≥ 17 at program counter 8, yet due to the following excerpt from the flow-logic rule:

$$\big(\mathit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\mathtt{true}\}\big) \;\Rightarrow\; \begin{array}{lll} \{\pi_3\} & \sqsubseteq_C & \widehat{C}(m_n, \mathit{addr}) \\ \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, \mathit{addr})(\pi_3) \end{array}$$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{if } t \ cmpop \texttt{ goto } \textit{addr}$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\textit{nextAddress}(pc_n))))$$

$$\pi_3 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,\textit{addr})))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X_1::X_2$$

$$\forall\, \{(t,Y_1,(m_p,pc_p))\} \sqsubseteq_S X_1 :$$

$$\forall\, \{(t,Y_2,(m_q,pc_q))\} \sqsubseteq_S X_2 :$$

$$\big(\textit{absApplyBinary}(cmpop,(t,Y_1,(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{true}\}\big) \Rightarrow$$

$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,\textit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,\textit{addr})(\pi_3)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,\textit{addr})(\pi_3)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,\textit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,\textit{addr})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\big(\textit{absApplyBinary}(cmpop,(t,Y_1,(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{false}\}\big) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\textit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

Table 5.2: Base CFA specification of `if` t $cmpop$ `goto` *addr* statement

$$\big(\textit{absApplyBinary}(cmpop,(t,Y_1,(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{false}\}\big) \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\textit{nextAddress}(pc_n))$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$

if any one value in the interval of a loop-variable's value meets the condition of the `if` statement, then the *whole interval* of a loop-variable's value is propagated to the corresponding branch of the `if` statement. Once a semantically impossible value for a loop variable is propagated down the negative branch of the `if` statement, which contains the loop increment (or loop decrement) in a terminating loop, the program analysis clauses for the `inc` and `if` statements contend to ensure only the $(\bot_{\mathtt{s}}, \top_{\mathtt{s}})$ value can satisfy the constraints for the loop variable.

5.3 Extended control-flow analysis remedies deficiency in base control-flow analysis handling of loops

From Section 5.2 we may summarise the key deficiency of the base control-flow analysis handling of loops as being:

- Not recognising and/or treating loop-variables differently at `if` statements;

5.3.1 Selective propagation of numeric loop-variables

The first question is: how do we recognise loop variables? Firstly, loop variables are numeric local variables whose values are passed on the operand stack as parameters to `if` statements via the `load` *t i* instructions. According to the operational semantics of Carmel as defined in Chapter 3, numeric local variables exhibit the following behaviour:

- whenever a number is saved from the operand stack to the local variable array via a `store` *t i* instruction, the runtime value saved to the local variable array is stamped with the address of the `store` *t i* instruction regardless of its previous runtime address label;
- whenever a number in the local variable array is incremented via a `inc` *t j c* instruction, the runtime value saved to the local variable array is stamped with the address of the `inc` *t j c* instruction regardless of its previous runtime address label;
- whenever a number is loaded onto the operand stack from the local variable array via a `load` *t i*, it is loaded as-is i.e. with the address with which it was saved, which must additionally be either a `store` *t i* or `inc` *t j c* instruction by virtue of the two previous items.

From inspection of the operational semantics of Carmel, no other bytecode instruction stamps a Carmel runtime value with the address of a `store` *t i* or `inc` *t j c* instruction. As the program analysis has been proved to be correct with respect to the operational semantics of Carmel, the above properties have been proved to be preserved by the flow-logic clauses. It follows then that, if the following conditions hold at an `if s` statement:

- the top of the operand stack consists solely of `store` *t i* or `inc` *t j c* values and that only one distinct local variable index is referenced in these values;
- the top of the operand stack is identical to the local variable array at the distinct local variable referenced in the top of the operand stack

we may conclude that the values on top of the operand stack *must* have been `load`ed from the local variable array *and not subsequently changed*. In such a case, the set of values on top of the operand stack are a proxy for the

local variable at the specified index of the local variable array and we should apply the test of the `if` statement against each possible value, from its minimum to maximum value, and propagate the appropriate value(s) of the local variable to the appropriate branch of the `if` statement at the specified index. Equally, where the `if` statement is the variant that takes two operands from the operand stack, and the two conditions above apply to the second from top of the operand stack set of values, these should be similarly tested and propagated to the appropriate branch. Where the two conditions apply to both operands, each pair of possible values from each set should be tested and propagated to the appropriate branch. Utilisation of associated runtime address in the program analyses to identify loop variables loaded from the local variable array is the sole reason we extended the operational semantics of Carmel and constructed the representation functions in the proofs of the base control-flow analysis to include the associated runtime address.

5.3.2 Detecting non-reaching definitions of local variables

An assignment (known as a definition in classical program analysis terms) to a local variable in Carmel (only possible via a `store` *t i* or `inc` *t j c* instruction) is said to be a reaching definition if a path exists from definition to use (via a `load` *t i* instruction) along some path of execution.

We have developed an analogue of the classical reaching definitions analysis[2] which we have named a *reaching definition for local variable analysis*. This intra-method analysis formally guarantees for each program counter pcm of a method m, a set of pairs of (local variable number lv, program counter pc) such that the local variable lv *may* have received its definition at program counter pc of method m.

For illustration purposes, in the Carmel code of Table 5.1, the local variable assignments that are reaching definitions at program counter 5 are:

```
(11, pc:1)
(11, pc:8)
```

which is to say, at program counter 5, the local variable 11 may have received its definition at program counter 1 or 8.

The formal guarantees of the correctness of the *reaching definition for local variable analysis* ensures we can safely identify the reaching definitions:

[2] See Appendix C or [NNH10] for further information.

```
- (s (0,0,0),     pc: 1,   STORE_T_I (StoreTIArguments (t=s), (i=11) )  )
- (s (1,17,17),   pc: 8,   INC_T_I_C (IncTICArguments  (t=s), (i=11),(c=1) )  )
```

and exclude from consideration any other local variable definitions that have reached this point in the program analysis – as they cannot correspond to semantically possible assignments i.e. they are not reaching definitions.

5.3.3 Bounding simple numeric loops and suppressing propagation of non-reaching definitions

Bringing this Section together, the consolidating question is then: under what circumstances at an `if` statement, for the individual values supplied as operands:

- should we attempt to filter and propagate to appropriate branches individual values?
- should we not attempt to filter and propagate to appropriate branches individual values?
- should we discard operand(s)?

Consider first the more general of the `if` statement forms, which is reproduced for convenience in Table 5.2, and which takes two stack operands. Each of the two stack operands may have been loaded from the local variable array, and may be a reaching definition. This implies $2^4 = 16$ possible combinations, as documented in Table 5.3.3. On consideration of each scenario described by the combinations, there are only 4 distinct combinations where we want to act:

- Case 1: No filtering to be applied to either operand. Neither operands have been loaded from the local variable array (and so neither can be reaching definitions);
- Case 2: Filtering is to be applied to both operands. Both operands have been loaded from the local variable array, and both are reaching definitions;
- Case 3: Filtering is to be applied to only the first operand. The first operand has been loaded from the local variable array and is a reaching definition. The second operand has not been loaded from the local variable array (and so cannot be a reaching definition);
- Case 4: Filtering is to be applied to only the second operand. The second operand has been loaded from the local variable array and is a reaching definition. The first operand has not been loaded from the local variable array (and so cannot be a reaching definition);

The remaining 12 combinations are excluded for the reasons stated under the Notes column.

The extended control-flow analysis clause for the more general of the `if` statement forms is then essentially the base control-flow analysis clause for the more general of the `if` statement forms duplicated 4 times, each duplicate

guarded by the conditions for one of the above cases, with filtering of values from minimum to maximum values of appropriate operand(s) for cases 2 - 4, and is documented in Section 5.3.4.

The extended control-flow analysis clause for the less general of the `if` statement forms is then essentially the base control-flow analysis clause for the more general of the `if` statement forms duplicated twice, each duplicate guarded by the conditions for one of the above cases, with filtering of values from minimum to maximum values of appropriate operand(s) for cases 1 and 4, and is documented in Section 5.3.5. Cases 2 and 3 do not apply in this scenario as only one operand is consumed from the operand stack, with an implicit argument of zero or `null` depending on the argument type being used for the second operand.

Table 5.3: Desired least solution to the base control-flow analysis constraints for sample Carmel program

Context	Method	Program Counter	Local Variables	Operand Stack
π_0	m	0	[]	ϵ
π_1	m	1	[]	(s (0,0,0),0(PushTCArguments (t=s), (c=0)))
π_2	m	2	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (1,17,17),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ
π_3	m	3	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (1,17,17),8(IncTICArguments (t=s), (i=11),(c=1)))	(s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (1,17,17),8(IncTICArguments (t=s), (i=11),(c=1)))
π_5	m	5	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (1,17,17),8(IncTICArguments (t=s), (i=11),(c=1)))	(s (17,17,0),3(PushTCArguments (t=s), (c=17))) :: (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (1,17,17),8(IncTICArguments (t=s), (i=11),(c=1)))
π_8	m	8	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (1,16,16),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ
π_{11}	m	11	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (1,17,17),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ
π_{14}	m	14	11↦ (s (17,17,17),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ

Table 5.4: Actual least solution to the base control-flow analysis constraints for sample Carmel program

Context	Method	Program Counter	Local Variables	Operand Stack
π_0	m	0	[]	ϵ
π_1	m	1	[]	(s (0,0,0),0(PushTCArguments (t=s), (c=0)))
π_2	m	2	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ
π_3	m	3	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))	(s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))
π_5	m	5	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))	(s (17,17,0),3(PushTCArguments (t=s), (c=17))) :: (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))
π_8	m	8	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ
π_{11}	m	11	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ
π_{14}	m	14	11↦ (s (0,0,0),1(StoreTIArguments (t=s), (i=11))) (s (-32768,32767,1000),8(IncTICArguments (t=s), (i=11),(c=1)))	ϵ

Table 5.5: Reaching definition and loaded from cross analysis of cases of interest

Value v_1 from X1 `load`ed from local var array?	Value v_2 from X2 `load`ed from local var array?	v_1 is a reaching definition?	v_2 is a reaching definition?	Scenario of interest? False unless otherwise stated.	Notes
false	false	false	false	true	Case 1: No filtering, use base control-flow analysis clause
false	true	false	false		Non-Reaching-v_2
true	false	false	false		Non-Reaching-v_1
true	true	false	false		Non-Reaching-v_1, Non-Reaching-v_2
false	false	false	true		Invalid-loaded-reaching-v_2
false	true	false	true	true	Case 3: Filter v_2
true	false	false	true		Invalid-loaded-reaching-v_2, Non-Reaching-v_1
true	true	false	true		Non-Reaching-v_1
false	false	true	false		Invalid-loaded-reaching-v_1
false	true	true	false		Invalid-loaded-reaching-v_1, Non-Reaching-v_2
true	false	true	false	true	Case 4: Filter v_1
true	true	true	false		Non-Reaching-v_2
false	false	true	true		Invalid-loaded-reaching-v_1, Invalid-loaded-reaching-v_2
false	true	true	true		Invalid-loaded-reaching-v_1
true	false	true	true		Invalid-loaded-reaching-v_2
true	true	true	true	true	Case 2: Filter both v_1 and v_2

5.3.4 Extended CFA for `if` t $cmpop$ `goto` ***addr***

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if}\ t\ cmpop\ \texttt{goto}\ \textit{addr}\ (part\ 1\ of\ 5)$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\textit{nextAddress}(pc_n))))$$

$$\pi_3 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,\textit{addr})))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X_1::X_2$$

$$\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} = \left(\begin{array}{l}(t=\texttt{s}) \wedge X_1 = \{(t,Z,(m_n,pc_p)) \mid m_n.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j, \texttt{inc}\ t\ j\ c\}\} \wedge \\ |\{j \mid X_1 = \{(t,Z,(m_n,pc_p)) \mid m_n.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j, \texttt{inc}\ t\ j\ c\}\}| = 1 \wedge \\ X_1 \sqsubseteq_S \widehat{L}(m_n,pc_n)(\pi_1)(j) \wedge \widehat{L}(m_n,pc_n)(\pi_1)(j) \sqsubseteq_S X_1\end{array}\right)$$

$$\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} = \left(\begin{array}{l}(t=\texttt{s}) \wedge X_2 = \{(t,Z,(m_n,pc_p)) \mid m_n.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j, \texttt{inc}\ t\ j\ c\}\} \wedge \\ |\{j \mid X_2 = \{(t,Z,(m_n,pc_p)) \mid m_n.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j, \texttt{inc}\ t\ j\ c\}\}| = 1 \wedge \\ X_2 \sqsubseteq_S \widehat{L}(m_n,pc_n)(\pi_1)(j) \wedge \widehat{L}(m_n,pc_n)(\pi_1)(j) \sqsubseteq_S X_2\end{array}\right)$$

$$\forall\, \{(t,Y_1,(m_p,pc_p))\} \sqsubseteq_S X_1 :$$

$$\quad \forall\, \{(t,Y_2,(m_q,pc_q))\} \sqsubseteq_S X_2 :$$

$$\texttt{REACHING_DEFN_X1} = \left(\begin{array}{l}(m_p = m_n) \wedge (m_p.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j_5, \texttt{inc}\ t\ j_6\ c_6\}) \wedge \\ \big((j_5,pc_p) \in RDLV(m_n,pc_n) \vee (j_6,pc_p) \in RDLV(m_n,pc_n)\big)\end{array}\right)$$

$$\texttt{REACHING_DEFN_X2} = \left(\begin{array}{l}(m_q = m_n) \wedge (m_q.\textit{instructionAt}(pc_q) \in \{\texttt{store}\ t\ j_7, \texttt{inc}\ t\ j_8\ c_8\}) \wedge \\ \big((j_7,pc_q) \in RDLV(m_n,pc_n) \vee (j_8,pc_q) \in RDLV(m_n,pc_n)\big)\end{array}\right)$$

$$\texttt{IS_CASE_1} = (\neg\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} \wedge \neg\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} \wedge \neg\texttt{REACHING_DEFN_X1} \wedge \neg\texttt{REACHING_DEFN_X2})$$
$$\texttt{IS_CASE_2} = (\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} \wedge \texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} \wedge \texttt{REACHING_DEFN_X1} \wedge \texttt{REACHING_DEFN_X2})$$
$$\texttt{IS_CASE_3} = (\neg\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} \wedge \texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} \wedge \neg\texttt{REACHING_DEFN_X1} \wedge \texttt{REACHING_DEFN_X2})$$
$$\texttt{IS_CASE_4} = (\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} \wedge \neg\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} \wedge \texttt{REACHING_DEFN_X1} \wedge \neg\texttt{REACHING_DEFN_X2})$$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if}\ t\ cmpop\ \texttt{goto}\ \mathit{addr}\ (part\ 2\ of\ 5)$

$\big((t = \texttt{r}) \vee (t = \texttt{s} \wedge \texttt{IS_CASE_1})\big)\ \Rightarrow$

$\big(\mathit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{true}\}\big)\ \Rightarrow$

$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, \mathit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, \mathit{addr})(\pi_3)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, \mathit{addr})(\pi_3)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, \mathit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, \mathit{addr})(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\big(\mathit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{false}\}\big)\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if}\ t\ cmpop\ \texttt{goto}\ \mathsf{addr}\ (part\ 3\ of\ 5)$

$\big((t = \texttt{s} \wedge \texttt{IS_CASE_2})\big)\ \Rightarrow$

$Y1 = (l_1, h_1, mod_1)$
$Y2 = (l_2, h_2, mod_2)$
$\forall\ absval_1 \in \{l_1, \ldots, h_1\} :$
$\quad \forall\ absval_2 \in \{l_2, \ldots, h_2\} :$

$\big(\mathsf{absApplyBinary}(cmpop, (t, (absval_1, absval_1, mod_1), (\mathsf{m}_p, pc_p)), (t, (absval_2, absval_2, mod_2), (\mathsf{m}_q, pc_q))) \supseteq \{\texttt{true}\}\big)\ \Rightarrow$

$(index_1, pc_p)$	$\in$	$RDLV(\mathsf{m}_n, pc_n)$
$(index_2, pc_q)$	$\in$	$RDLV(\mathsf{m}_n, pc_n)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(\mathsf{m}_n, \mathsf{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(\mathsf{m}_n, \mathsf{addr})(\pi_3)$
$\{(absval_1, absval_1, mod_1)\}$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, \mathsf{addr})(\pi_3)(index_1)$
$\{(absval_2, absval_2, mod_2)\}$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, \mathsf{addr})(\pi_3)(index_2)$
$\forall\ k \in \mathsf{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \notin \{index_1, index_2\} : \widehat{L}(m_n, pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_3)(k)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathsf{m}_n, \mathsf{addr})(\pi_3)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathsf{m}_n, \mathsf{addr})(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\big(\mathsf{absApplyBinary}(cmpop, (t, (absval_1, absval_1, mod_1), (\mathsf{m}_p, pc_p)), (t, (absval_2, absval_2, mod_2), (\mathsf{m}_q, pc_q))) \supseteq \{\texttt{false}\}\big)\ \Rightarrow$

$(index_1, pc_p)$	$\in$	$RDLV(\mathsf{m}_n, pc_n)$
$(index_2, pc_q)$	$\in$	$RDLV(\mathsf{m}_n, pc_n)$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_2)$
$\{(absval_1, absval_1, mod_1)\}$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, \mathsf{addr})(\pi_2)(index_1)$
$\{(absval_2, absval_2, mod_2)\}$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, \mathsf{addr})(\pi_2)(index_2)$
$\forall\ k \in \mathsf{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \notin \{index_1, index_2\} : \widehat{L}(m_n, pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_2)(k)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if}\ t\ cmpop\ \texttt{goto}\ \mathit{addr}\ (part\ 4\ of\ 5)$

$\big((t = \texttt{s} \wedge \texttt{IS_CASE_3})\big)\ \Rightarrow$

$Y2 = (l_2, h_2, mod_2)$
$\forall\ absval_2 \in \{l_2, \ldots, h_2\}$:

$\big(\mathit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, (absval_2, absval_2, mod_2), (m_q, pc_q))) \supseteq \{\texttt{true}\}\big)\ \Rightarrow$

$(index_2, pc_q)$	$\in$	$RDLV(m_n, pc_n)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, \mathit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, \mathit{addr})(\pi_3)$
$\{(absval_2, absval_2, mod_2)\}$	$\sqsubseteq_L$	$\widehat{L}(m_n, \mathit{addr})(\pi_3)(index_2)$
$\forall\ k \in \mathit{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \notin \{index_2\} : \widehat{L}(m_n, pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_3)(k)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, \mathit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, \mathit{addr})(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\big(\mathit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, (absval_2, absval_2, mod_2), (m_q, pc_q))) \supseteq \{\texttt{false}\}\big)\ \Rightarrow$

$(index_2, pc_q)$	$\in$	$RDLV(m_n, pc_n)$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(absval_2, absval_2, mod_2)\}$	$\sqsubseteq_L$	$\widehat{L}(m_n, \mathit{addr})(\pi_2)(index_2)$
$\forall\ k \in \mathit{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \notin \{index_2\} : \widehat{L}(m_n, pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if}\ t\ cmpop\ \texttt{goto}\ \mathsf{addr}\ (part\ 5\ of\ 5)$

$\big((t = \texttt{s} \wedge \texttt{IS_CASE_4})\big)\ \Rightarrow$

$Y1 = (l_1, h_1, mod_1)$
$\forall\ absval_1 \in \{l_1, \ldots, h_1\} :$

$\big(\mathsf{absApplyBinary}(cmpop, (t, (absval_1, absval_1, mod_1), (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{true}\}\big)\ \Rightarrow$

$(index_1, pc_p)$	$\in$	$RDLV(m_n, pc_n)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, \mathsf{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, \mathsf{addr})(\pi_3)$
$\{(absval_1, absval_1, mod_1)\}$	$\sqsubseteq_L$	$\widehat{L}(m_n, \mathsf{addr})(\pi_3)(index_1)$
$\forall\ k \in \mathsf{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \notin \{index_1\} : \widehat{L}(m_n, pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_3)(k)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, \mathsf{addr})(\pi_3)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, \mathsf{addr})(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\big(\mathsf{absApplyBinary}(cmpop, (t, (absval_1, absval_1, mod_1), (m_p, pc_p)), (t, (absval_2, absval_2, mod_2), (m_q, pc_q))) \supseteq \{\texttt{false}\}\big)\ \Rightarrow$

$(index_1, pc_p)$	$\in$	$RDLV(m_n, pc_n)$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathsf{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_2)$
$\{(absval_1, absval_1, mod_1)\}$	$\sqsubseteq_L$	$\widehat{L}(m_n, \mathsf{addr})(\pi_2)(index_1)$
$\forall\ k \in \mathsf{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \notin \{index_1\} : \widehat{L}(m_n, pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_2)(k)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

5.3.5 Extended CFA for `if` t $cmpop$ `null goto` ***addr***

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if}\ t\ cmpop\ \texttt{null goto}\ \textit{addr}\ (part\ 1\ of\ 3)$$

$$\iff \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\textit{nextAddress}(pc_n))))$$

$$\pi_3 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,\textit{addr})))$$

$$\widehat{S}(m_n,pc_n)(\pi_1) = M::X_1$$

$$\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} = \left(\begin{array}{l} (t=\texttt{s}) \wedge X_1 = \{(t,Z,(m_n,pc_p)) \mid m_n.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j, \texttt{inc}\ t\ j\ c\}\} \wedge \\ |\{j \mid X_1 = \{(t,Z,(m_n,pc_p)) \mid m_n.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j, \texttt{inc}\ t\ j\ c\}\}| = 1 \wedge \\ X_1 \sqsubseteq_S \widehat{L}(m_n,pc_n)(\pi_1)(j) \wedge \widehat{L}(m_n,pc_n)(\pi_1)(j) \sqsubseteq_S X_1 \end{array}\right)$$

$$\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} = \texttt{false}$$

$$\texttt{REACHING_DEFN_X2} = \texttt{false}$$

$$\forall\, \{(t,Y_1,(m_p,pc_p))\} \sqsubseteq_S X_1 :$$

$$(t,Y_2,(m_q,pc_q)) = \begin{cases} \widehat{\sigma}_{\texttt{Null}}, & t = \texttt{r}, \\ (\texttt{s},(0,0,0),(m_n,pc_n)), & t = \texttt{s} \end{cases}$$

$$\texttt{REACHING_DEFN_X1} = \left(\begin{array}{l} (m_p = m_n) \wedge (m_p.\textit{instructionAt}(pc_p) \in \{\texttt{store}\ t\ j_5, \texttt{inc}\ t\ j_6\ c_6\}) \wedge \\ \big((j_5,pc_p) \in RDLV(m_n,pc_n) \vee (j_6,pc_p) \in RDLV(m_n,pc_n)\big) \end{array}\right)$$

$$\texttt{IS_CASE_1} = (\neg\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} \wedge \neg\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} \wedge \neg\texttt{REACHING_DEFN_X1} \wedge \neg\texttt{REACHING_DEFN_X2})$$
$$\texttt{IS_CASE_4} = (\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X1} \wedge \neg\texttt{LOADED_FROM_LOCAL_VAR_ARRAY_X2} \wedge \texttt{REACHING_DEFN_X1} \wedge \neg\texttt{REACHING_DEFN_X2})$$

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if } t \ \ cmpop \ \texttt{null goto } \textit{addr} \ (part\ 2\ of\ 3)$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\textit{nextAddress}(pc_n))))$$

$$\pi_3 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,\textit{addr})))$$

$$\big((t=\texttt{r}) \vee (t=\texttt{s} \wedge \texttt{IS_CASE_1})\big) \ \Rightarrow$$

$$\big(\textit{absApplyBinary}(cmpop,(t,Y_1,(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{true}\}\big) \ \Rightarrow$$

$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,\textit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,\textit{addr})(\pi_3)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,\textit{addr})(\pi_3)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,\textit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,\textit{addr})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\big(\textit{absApplyBinary}(cmpop,(t,Y_1,(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{false}\}\big) \ \Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\textit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\textit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} (m_n, pc_n) : \texttt{if } t \ \ cmpop \ \texttt{null goto } \mathit{addr} \ (part\ 3\ of\ 3)$

$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$

$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$

$\pi_3 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,\mathit{addr})))$

$\big((t = \texttt{s} \wedge \texttt{IS_CASE_4})\big) \ \Rightarrow$

$Y1 = (l_1,h_1,mod_1)$
$\forall\, absval_1 \in \{l_1,\ldots,h_1\} :$

$\big(\mathit{absApplyBinary}(cmpop,(t,(absval_1,absval_1,mod_1),(m_p,pc_p)),(t,Y_2,(m_q,pc_q))) \supseteq \{\texttt{true}\}\big) \ \Rightarrow$

$(index_1,pc_p)$	$\in$	$RDLV(m_n,pc_n)$
$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,\mathit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,\mathit{addr})(\pi_3)$
$\{(absval_1,absval_1,mod_1)\}$	$\sqsubseteq_L$	$\widehat{L}(m_n,\mathit{addr})(\pi_3)(index_1)$
$\forall\, k \in \mathit{dom}\big(\widehat{L}(m_n,pc_n)(\pi_1)\big)\ .\ k \notin \{index_1\} : \widehat{L}(m_n,pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_3)(k)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,\mathit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,\mathit{addr})(\pi_3)$
$\{(\pi_1,\pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\big(\mathit{absApplyBinary}(cmpop,(t,(absval_1,absval_1,mod_1),(m_p,pc_p)),(t,(absval_2,absval_2,mod_2),(m_q,pc_q))) \supseteq \{\texttt{false}\}\big) \ \Rightarrow$

$(index_1,pc_p)$	$\in$	$RDLV(m_n,pc_n)$
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n,m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(absval_1,absval_1,mod_1)\}$	$\sqsubseteq_L$	$\widehat{L}(m_n,\mathit{addr})(\pi_2)(index_1)$
$\forall\, k \in \mathit{dom}\big(\widehat{L}(m_n,pc_n)(\pi_1)\big)\ .\ k \notin \{index_1\} : \widehat{L}(m_n,pc_n)(\pi_1)(k)$	$\sqsubseteq_L$	$\widehat{L}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k)$
$\widehat{JH}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n,pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n,m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1,\pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

5.4 Modifying the base control-flow analysis worklist solver

Two changes are required to the worklist solver of the base control-flow analysis of Section 4.7 to compute the least solution to the extended control-flow analysis constraints:

- insert a new Step between Step 1 and Step 2, where the global variable $RDLV$ is assigned the results of the Reaching Definitions for Local Variables analysis of Appendix C for Carmel program P;

- at Step 6, replace the existing flow-logic clauses for the `if` statements of the base control-flow analysis with those of the extended control-flow analysis of this Chapter. These replacement clauses are the only ones that access the fresh global variable $RDLV$ so, if memory is a factor, $RDLV$ can have reaching definitions for local variable analysis information for all addresses other than the `if` statements removed once $RDLV$ is computed;

Chapter 6

Worst–Case Resource–Usage Analysis

6.1 Introduction

In this Chapter, we detail the conditions under which we are able to analyse the worst-case resource-usage of a Carmel program P, and present an algorithm for generating from the extended control-flow analysis of Chapter 5 for P a family of integer linear programming problems, one for each applet lifecycle method in P per registered applet whose solution yields the worst-case resource-usage for that applet lifecycle method. While our algorithm is firmly based on the outstanding and pioneering work of [PS97], its presentation is intentionally closer to that of [Sch09].

The general conditions to make the worst-case execution-time decidable (and so make the worst-case resource-usage analysis decidable) for Java applications are [Sch09]:

1. Dynamic class loading is forbidden;
2. Programs must not contain any recursion;
3. The upper bound of each loop has to be known;

Considering these conditions in the context of *Java Card* Classic Edition 3:

1. Dynamic class loading is forbidden in *Java Card* Classic Edition 3[1] so this condition is automatically satisfied for all Carmel programs.
2. We can formally guarantee there are no recursive method calls in any code reachable from any applet

[1] Please see Table 2.1 for Java features supported and not supported in *Java Card* Classic Edition 3.

lifecycle methods in P when

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} P$$

and $\widehat{REC} = \emptyset$.

3. Determining the upper bound of each loop in P and more particularly specifying the conditions under which we can determine the upper bound of each loop requires a few preliminary technical steps[2]:

   - For each applet lifecycle method in P, identify the set of *natural loops*, including the set of basic blocks which form the natural loop, reachable from that applet lifecycle method.
   - For each applet lifecycle method in P and each natural loop reachable from that applet lifecycle method:
     - capture the set of local variables of the operands at the condition of the loop header from the operand stack, and their minimum and maximum values;
     - capture the set of local variables which are written to via the `inc` $t\ i\ c$ in the body of the loop, and the values by which they are increased (or decreased where c is negative);
     - capture the set of local variables which are written to via the `store` $t\ i$ in the body of the loop;
   - Determine from the captured information whether we have the required information to calculate the upper bound for each loop, and that it is safe to do so e.g. that the local variables on which outer loops are dependent are not modified in the body of inner loops.

6.2 Identifying natural loops in Carmel

Let P be a Carmel program such that

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} P$$

Then $\widehat{CG}$ is a control-flow graph of P and more particularly a directed graph where each node in the graph is a basic block in P and the directed edges between nodes represent the control-flow between basic blocks in P including inter-method, intra-method, normal and exceptional control-flows[3]. There is a start node in $\widehat{CG}$ associated with each applet lifecycle method/initial configuration in P.

Intuitively a loop in $\widehat{CG}$ has the following properties:

[2] We cover the required material in Section 6.2; those already familiar with natural loops and dominators may freely skip that section
[3] See Corollary 4.4.2 on page 128

- a loop is a set of nodes $nodes \subseteq \widehat{CG}$;
- a loop has a loop-header node $\in nodes$ such that control-flow to nodes in $nodes$ from outside of $nodes$ in $\widehat{CG}$ must go via the loop-header node;
- a loop has a back-edge from one of the nodes in $nodes$ to the loop-header node

Consider Figure 6.1 which has three nested loops:

- Loop with basic block 5 as the loop-header and block 6 as having the back-edge. The whole loop body is $\{5, 6\}$;
- Loop with basic block 3 as the loop-header and block 7 as having the back-edge. The whole loop body is $\{3, 4, 5, 6, 7\}$;
- Loop with basic block 1 as the loop-header and block 8 as having the back-edge. The whole loop body is $\{1, 2, 3, 4, 5, 6, 7, 8\}$;

It is clear from examining the basic blocks making up the three loops, that the first loop is the innermost loop of the nested loops, and that the second is the middle loop, with the third being the outermost loop. Where there is no intersection between basic blocks in loop-bodies, these loops must be independent. We use natural loops in a similar way as e.g. JOP uses defscopes [Sch09] or SWEET [Gus] uses scope graphs to e.g. ensure the local variables on which outer loops depend should not be written to by child/nested loops. Nestedness and independence of loops can be inferred from the set of basic blocks associated with each loop.

To formalise the specification and identification of natural loops, we introduce the notion of dominators. A node d is said to *dominate* another node n in $\widehat{CG}$ (which we write d dominates n) if every path from a start node to p must go though d. A loop in Carmel then has the following properties:

- A single entry point, the loop header node lh, which dominates all nodes in the loop;
- A back-edge linking a node n to the loop-header node lh.

Considering Figure 6.1 once more:

- Basic block 5 dominates blocks $\{5, 6\}$– block 5 is the loop-header node and block 6 has the back-edge to the loop-header node;
- Basic block 3 dominates blocks $\{3, 4, 5, 6, 7\}$– block 3 is the loop-header node and block 7 has the back-edge to the loop-header node;

- Basic block 1 dominates blocks $\{1, 2, 3, 4, 5, 6, 7, 8\}$– block 1 is the loop-header node and block 8 has the back-edge to the loop-header node.

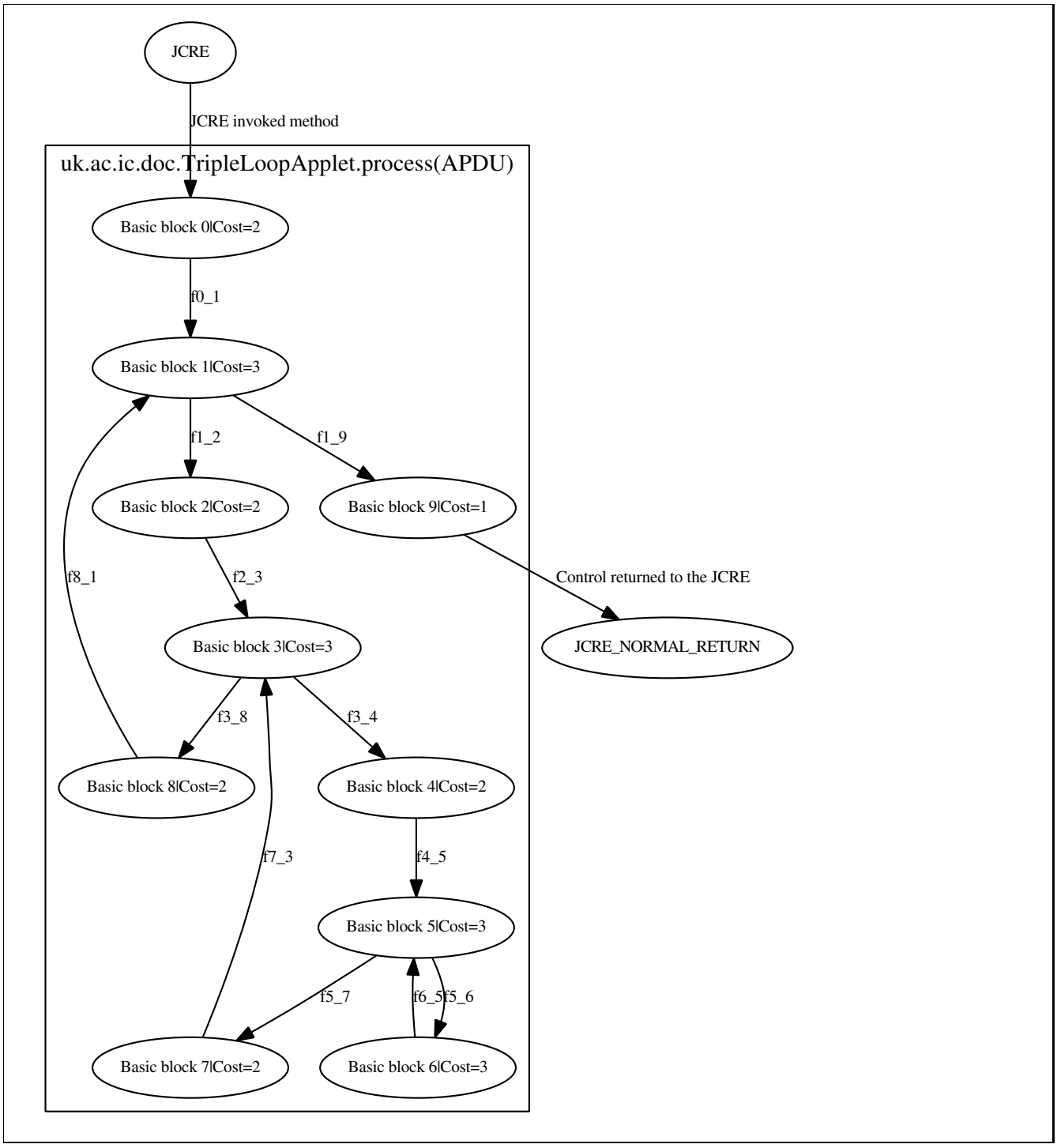


Figure 6.1: Control-flow graph of a Carmel program, with three embedded loops. Loop-header nodes at basic-block nodes 1, 3 and 5.

A *natural loop* in Carmel consists of a back edge $n \rightarrow lh$, where lh dominates n, and the set of nodes x such that lh dominates x and there exists a path from x to n not containing lh; the loop header is lh. Note that we have assumed as part of the thesis' assumptions in Section 1.3 that P is flow-reducible, or has been preprocessed to be be flow-reducible via node-copying/node-splitting techniques, and so $\widehat{CG}$ is flow-reducible. This ensures loops are well-defined and back edges are unique (and consistent with how we have presented them here). An algorithm to detect natural loops in Carmel is given in Table 6.2. We use the graph facilities of the WALA[4] library to compute the dominator relation (using the Tarjan-Lengauer algorithm of [LT79]) and to identify the back edges and compute the natural loop using its depth-first search functions.

```
Algorithm to identify natural loops in a Control Flow Graph:

- Compute dominator relation
- Identify back edges
- Compute the loop for each back edge:

     for each node h in dominator tree
      for each node n for which there exists a back edge n->h
       define the loop with
            header h
            back edge n -> h
            body consisting of all nodes reachable from n by a
            depth first search backwards from n that stops at h
```

Table 6.1: Algorithm to identify natural loops in a control-flow graph such as $\widehat{CG}$, reproduced from [Pin10]

6.3 Capture local/loop variable information from each natural loop

Let P be a Carmel program such that

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY, RDLV}_{\texttt{Extended-CFA}} P$$

[4] The T. J. Watson Libraries for Analysis (WALA) libraries `http://wala.sourceforge.net/wiki/index.php/Main_Page` provides static code analysis capabilities for Java bytecode.

and $\{n_1, \ldots, n_m\}$ be the set of natural loops extracted from $\widehat{CG}$ using the approach and algorithm outlined in Section 6.2 and assuming the required data has been captured to define the following functions:

$$findBlocks: \texttt{NaturalLoop} \longrightarrow \mathcal{P}(\mathbb{N}_0)$$

$$findAddresses: \texttt{NaturalLoop} \longrightarrow \mathcal{P}(\textsf{Address})$$

$$findLoopHeader: \texttt{NaturalLoop} \longrightarrow \mathbb{N}_0$$

$$findLoopHeaderIdom: \texttt{NaturalLoop} \longrightarrow \mathbb{N}_0$$

$$findLoopBackEdge: \texttt{NaturalLoop} \longrightarrow \mathbb{N}_0$$

Then we define and specify the following functions to capture local/loop variable information:

$$findStoreVariables: \texttt{NaturalLoop} \longrightarrow \mathcal{P}(\mathbb{N}_0)$$

$$findStoreVariables(n) = \{i |\ (m, pc) \in findAddresses(n), m.\textsf{instructionAt}(pc) = \texttt{store}\ \ t\ \ i\}$$

$$findIncVariables: \texttt{NaturalLoop} \longrightarrow \mathcal{P}(\mathbb{N}_0)$$

$$findIncVariables(n) = \{i \mid (m, pc) \in findAddresses(n), m.\textsf{instructionAt}(pc) = \texttt{inc}\ \ t\ \ i\ \ c\}$$

$$findIncVariablesAndValues: \texttt{NaturalLoop} \longrightarrow \mathcal{P}(\mathbb{N}_0 \times \mathbb{Z})$$

$$findIncVariablesAndValues(n) = \{(i, c) \mid (m, pc) \in findAddresses(n), m.\textsf{instructionAt}(pc) = \texttt{inc}\ \ t\ \ i\ \ c\}$$

$findLoopVariablesAndValues:$ `NaturalLoop` $\longrightarrow (\mathbb{N}_0 \longrightarrow (\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}))$

$findLoopVariablesAndValues(n) = B$.

n is a loop-header node

n ends with an `if` instruction at (m_n, pc_n)

Start computation of B

$B := [\,]$

$m_n.\mathit{instructionAt}(pc_n) = \texttt{if}\ t\ \mathit{op}\ \texttt{goto}\ \mathit{addr} \Rightarrow$

$\forall \pi_1 \in \widehat{C}(m_n, pc_n):$

$\widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2$

$\forall\ (\mathtt{s}, (l_1, h_1, mod_1), (m_n, pc_2)) \in X_2:$

$m_n.\mathit{instructionAt}(pc_2) \in \{\texttt{store}\ t\ i, \texttt{inc}\ t\ i\ c\} \Rightarrow$

$(B(i)\ \texttt{is nil}) \Rightarrow$

$B(i) := (l_1, h_1, mod_1);$

$(B(i)\ \texttt{is not nil}) \Rightarrow$

$B(i) = (l_2, h_2, mod_2)$

$B(i) := (\texttt{min}\{l_1, l_2\}, \texttt{max}\{h_1, h_2\}, \texttt{max}\{mod_1, mod_2\});$

$\forall\ (\mathtt{s}, (l_1, h_1, mod_1), (m_n, pc_2)) \in X_1:$

$m_n.\mathit{instructionAt}(pc_2) \in \{\texttt{store}\ t\ i, \texttt{inc}\ t\ i\ c\} \Rightarrow$

$(B(i)\ \texttt{is nil}) \Rightarrow$

$B(i) := (l_1, h_1, mod_1);$

$(B(i)\ \texttt{is not nil}) \Rightarrow$

$B(i) = (l_2, h_2, mod_2)$

$B(i) := (\texttt{min}\{l_1, l_2\}, \texttt{max}\{h_1, h_2\}, \texttt{max}\{mod_1, mod_2\});$

$m_n.\mathit{instructionAt}(pc_n) = \texttt{if}\ t\ \mathit{op}\ \texttt{null goto}\ \mathit{addr} \Rightarrow$

$\forall \pi_1 \in \widehat{C}(m_n, pc_n):$

$\widehat{S}(m_n, pc_n)(\pi_1) = M::X_1$

$\forall\ (\mathtt{s}, (l_1, h_1, mod_1), (m_n, pc_2)) \in X_1:$

$m_n.\mathit{instructionAt}(pc_2) \in \{\texttt{store}\ t\ i, \texttt{inc}\ t\ i\ c\} \Rightarrow$

$(B(i)\ \texttt{is nil}) \Rightarrow$

$B(i) := (l_1, h_1, mod_1);$

$(B(i)\ \texttt{is not nil}) \Rightarrow$

$B(i) = (l_2, h_2, mod_2)$

$B(i) := (\texttt{min}\{l_1, l_2\}, \texttt{max}\{h_1, h_2\}, \texttt{max}\{mod_1, mod_2\});$

End computation of B

6.4 Validate conditions for being able to determine loop-bounds

Let P be a Carmel program such that

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \vdash^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY,RDLV}_{\texttt{Extended-CFA}} P$$

and $\{n_1, \ldots, n_m\}$ be the set of natural loops extracted from $\widehat{CG}$ using the approach and algorithm outlined in Section 6.2 and assuming the functions defined in Section 6.3.

Then we require the following conditions/validations to be satisfied to be able to provide upper loop-bounds for each reachable loop in P:

- No recursion:
  $\widehat{REC} = \emptyset$
- All reachable instructions must be on a path that eventually returns control to the `JCRE`:

  $T = \texttt{TRANSITIVE_CLOSURE}(\widehat{CG})$
  $\forall\ (m_n, pc_n) \in P.addresses :$
  $\quad \forall\ \pi \sqsubseteq_C \widehat{C}(m_n, pc_n) :$
  $\quad\quad \Big(\{(\pi, \texttt{JCRE normal}), (\pi, \texttt{JCRE uncaught exception})\} \bigcap T \neq \emptyset\Big)$

  This condition is designed to detect infinite intra-method loops e.g. the simplest/pathological form would be:

  ```
  10 goto 10
  ```

  There is no path to the `JCRE` from instructions for which this condition fails.
- Inner loops must not change the value of loop variables of outer loops:

$$\forall_{i \in \{1,\ldots,m\}} \forall_{j \in \{1,\ldots,m\}, j \neq i} \quad findBlocks(n_i) \cap findBlocks(n_j) \neq \emptyset \ \Rightarrow$$
$$findIncVariables(n_i) \cap findIncVariables(n_j) = \emptyset$$

$$\forall_{i \in \{1,\ldots,m\}} \forall_{j \in \{1,\ldots,m\}, j \neq i} \quad findBlocks(n_i) \cap findBlocks(n_j) \neq \emptyset \ \Rightarrow$$
$$|findBlocks(n_i)| > |findBlocks(n_j)| \ \Rightarrow$$
$$dom(findLoopVariablesAndValues(n_i)) \cap$$
$$(findIncVariables(n_j) \cup findStoreVariables(n_j)) = \emptyset$$

- Last instruction of a loop-header node must be an `if s` statement;

- Loops must be simple numeric loops:

$$\forall_{i \in \{1,\ldots,m\}} \;\; dom(findLoopVariablesAndValues(n_i)) \cap findIncVariables(n_i) \neq \emptyset$$

- Loop variables must be monotonically increasing or decreasing:

$$\begin{gathered}\forall_{i \in \{1,\ldots,m\}} \\ \forall (i_1, c_1) \in findIncVariablesAndValues(n_i) \\ \forall (i_2, c_2) \in findIncVariablesAndValues(n_i) \\ (i_1 = i_2) \;\Rightarrow\; ((c_1 > 0 \wedge c_2 > 0) \vee (c_1 < 0 \wedge c_2 < 0))\end{gathered}$$

- Loop variables must have finite (non-top) interval values:

$$\begin{gathered}\forall_{i \in \{1,\ldots,m\}} \\ \forall j \in dom(findLoopVariablesAndValues(n_i)) \\ \forall (min, max, mod) \in findLoopVariablesAndValues(n_i)(j) : \\ \neg(min = \bot_{\mathbf{S}} \wedge max = \top_{\mathbf{S}} \wedge mod = MAX_MOD_COUNT)\end{gathered}$$

6.5 Calculate loop-bounds

Let P be a Carmel program such that

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY, RDLV}_{\texttt{Extended-CFA}} P$$

and $\{n_1, \ldots, n_m\}$ be the set of natural loops extracted from $\widehat{CG}$ using the approach and algorithm outlined in Section 6.2 and assuming P successfully passes all the validation criteria of Section 6.3, then we are able to calculate an upper loop-bound per loop.

$$findMaxLoopBound: \quad \texttt{NaturalLoop} \longrightarrow \mathcal{P}(\mathbb{N}_0)$$

$$findMaxLoopBound(n) \quad = \quad \left\{ \texttt{max}\{bound_1, \ldots, bound_w\} \;\middle|\; findLoopBounds(n) = \{(index_1, bound_1), \ldots, (index_w, bound_w)\} \right\}$$

$$findLoopBounds : \texttt{NaturalLoop} \longrightarrow \mathcal{P}(\mathbb{N}_0 \times \mathbb{N}_0)$$

$$findLoopBounds(n) = \left\{ \left(index, \left\lceil \frac{(maxVal - minVal)}{step} \right\rceil \right) \;\middle|\; \begin{array}{l} index \in dom(findLoopVariablesAndValues(n)) \\ (minVal, maxVal, modVal) = findLoopVariablesAndValues(n)(index) \\ findIncVariablesAndValues(n) = \{(i, c_1), \ldots, (i, c_m)\}, step = min\{|c_1|, \ldots, |c_m|\} \end{array} \right\}$$

6.6 Generating an integer linear programming problem per registered applet lifecycle method

6.6.1 Introduction

Let P be a Carmel program such that

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY, RDLV}_{\texttt{Extended-CFA}} P$$

and $\{n_1, \ldots, n_m\}$ be the set of natural loops extracted from $\widehat{CG}$ using the approach and algorithm outlined in Section 6.2 and assuming P successfully passes all the validation criteria of Section 6.3, then we are able to generate an integer linear programming problem per registered applet lifecycle method whose solution is the worst-case resource-usage (WCRU) for that registered applet and applet lifecycle method. As mentioned in the outset of this chapter, while our approach is firmly based on the outstanding and pioneering work of [PS97], its presentation is intentionally closer to that of [Sch09].

Remember that $\widehat{CG}$ is a control-flow graph of P and more particularly a directed graph where each node in the graph is a basic block in P and the directed edges between nodes represent the control-flow between basic blocks in P including inter-method, intra-method, normal and exceptional control-flows. There is a start node in $\widehat{CG}$ for each registered applet lifecycle method/initial configuration in P.

In framing the integer linear programming problem for each registered applet lifecycle method (which we'll abbreviate from this point on as the ILP problem):

- We associate a resource usage cost $c_i \in \mathbb{N}_0$ with each basic block/node B_i in $\widehat{CG}$. Each basic block/node in $\widehat{CG}$ is identified by its unique number B_i. For simplicity, our examples in this chapter will mainly focus on examples where the resource cost is the execution time associated with each basic block; we may define any resource cost function that can be expressed as a function of the individual instructions within a basic block and an aggregate function on the instructions within a basic block – for worst-case execution-time and worst-case dynamic memory allocation analysis, the aggregate function is the sum of the cost of the individual instructions. This resource usage cost expresses the cost of enacting the block *once*;

- We associate an execution frequency $e_i \in \mathbb{N}_0$ for each edge in $\widehat{CG}$. Since each basic block/node in $\widehat{CG}$ is assigned a unique number, and there are no self-loops or duplicate edges in $\widehat{CG}$, edges are uniquely identified and labelled as $f < source\ basic\ block/node\ number > _ < destination\ basic\ block/node\ number >$ e.g. $f0_1$ is the directed edge from basic block 0 to basic block 1 in $\widehat{CG}$.

Then the objective function of the ILP for the n basic blocks reachable from the particular registered applet lifecycle method alm in $\widehat{CG}$ is:

$$WCRU = \texttt{max} \sum_{i=1}^{n} t_i = c_i e_i$$

subject to two sets of constraints we generate algorithmically from $\widehat{CG}$:

- *Structural constraints*. The methodology requires:

  - adding a unique entry node S to $\widehat{CG}$ and add an edge fs from S to the basic block corresponding to the first basic block of alm and adding an initial constraint $fs = 1$ to express we must enter the body of alm exactly once;

  - adding a unique exit node T to $\widehat{CG}$ and add an edge ft to T with an initial constraint $ft = 1$ to express we must return to the JCRE exactly once;

  - express that the frequency with which a block is executed must equal the sum of the frequencies of all incoming edges:

$$\begin{aligned} &\forall\, i \in \{1, \ldots, n\} : \\ &\{\texttt{f}X_1\texttt{_i}, \ldots, \texttt{f}X_o\texttt{_i}\} \in edges(\widehat{CG}) : \\ &\text{add constraint} \qquad e_i = \sum_{j \in \{X_1, \ldots, X_o\}} \texttt{f}j\texttt{_}i \end{aligned}$$

  - express that the frequency with which a block is executed must equal the sum of the frequencies of all outgoing edges:

$$\begin{aligned} &\forall\, i \in \{1, \ldots, n\} : \\ &\{\texttt{fi_}X_1, \ldots, \texttt{fi_}X_o\} \in edges(\widehat{CG}) : \\ &\text{add constraint} \qquad e_i = \sum_{j \in \{X_1, \ldots, X_o\}} \texttt{f}i\texttt{_}j \end{aligned}$$

  - for each basic block B_ℓ corresponding to a terminal configuration – returning control to the JCRE either normally or abnormally via an uncaught exception – add an optional edge ft from B_ℓ to T:

$$\forall\, i \in \{1, \ldots, n\} :$$

$$B_i \in \{\texttt{JCRE normal, JCRE uncaught exception}\}$$

$$\text{add constraint} \qquad e_i \leq ft$$

– express the resource cost t_i of each basic block B_i as the product of the frequency of execution and the resource cost of executing the block once. We assume the precalculation of $c_i = \langle some\ resource\ function\ of\ B_i \rangle$ for each basic block prior to generation of the resource cost constraints of each basic block:

$$\forall\, i \in \{1, \ldots, n\} :$$

$$\text{add constraint} \qquad t_i = c_i e_i$$

NB if $B_i \in \{\texttt{JCRE normal, JCRE uncaught exception}\}$ we require $c_i = 0$ (otherwise we'd be including in the ILP a cost to return to the JCRE from an applet lifecycle method).

– express the global logical constraint that, in any valid solution, control must be returned to the JCRE exactly once:

$$\forall\, i \in \{1, \ldots, n\} :$$

$$B_i \in \{\texttt{JCRE normal, JCRE uncaught exception}\}$$

$$\{\texttt{f}X_1\texttt{_i}, \ldots, \texttt{f}X_o\texttt{_i}\} \in edges(\widehat{CG}) :$$

$$\text{add constraint} \qquad \sum_{j \in \{X_1, \ldots, X_o\}} \texttt{f}j\texttt{_}i \; = ft$$

– constrain solution to be integral. Define each block cost, edge frequency and timing variable name to be an integer;

– constrain solution to be non-negative. Define each block cost, edge frequency and timing variable value to be ≥ 0;

- *Loop constraints*. Express the numerical relationship between the frequency of execution of the immediate dominator of

the loop header and the back edge to the loop header:

$$\forall\, i \in \{1, \ldots, m\} :$$

$$loopHeader = findLoopHeader(n_i)$$

$$loopHeaderIdom = findLoopHeaderIdom(n_i)$$

$$loopHeaderTail = findLoopHeaderTail(n_i)$$

$$maxLoopBound = findMaxLoopBound(n_i)$$

$$backEdge = f\langle loopHeaderTail\rangle_\langle loopHeader\rangle$$

$$idomEdge = f\langle loopHeaderIdom\rangle_\langle loopHeader\rangle$$

$$\text{add constraint} \qquad backEdge = maxLoopBound * idomEdge$$

6.6.2 Sample generation of an integer programming problem for an overridden method in an entry point method

In this subsection we demonstrate how our approach to generating ILP handles polymorphism/overridden methods for an applet with rival implementations of a polymorphic method and how the solution of the ILP selects the rival implementation with the highest resource cost.

Dynamic dispatch of method invocation is a fundamental feature of an object-oriented language like Java. Dynamic dispatch occurs at method invocation in Java and Carmel when the runtime system has to select/resolve which version of a polymorphic method call to invoke, and is based principally on the runtime class of the object and other runtime types, including interfaces, it possesses. Dynamic dispatch occurs in Carmel when either of the following bytecode instructions are executed:

- `invokevirtual` w
- `invokeinterface` w

and the semantics of resolving dynamic dispatch are abstracted into the function *methodLookup* defined in Section 3.3.2.1. Note that dynamic dispatch does not occur with `invokedefinite` w instructions. The semantics of method invocation in Carmel are defined in Section 3.9.3.

In Tables 6.2–6.3, we present the simplified Carmel code for an applet `AnimalApplet` which has an attribute $animal$ of abstract type `uk.ac.imperial.doc.Animal` and includes annotations of which basic block instructions belong to. The control-flow graph for this program is shown in Figure 6.2 and includes the basic blocks and their resource cost. The entry point is block 0, the first block of the `AnimalApplet install` method. In a fuller applet, the installation byte array would be used to select one particular

animal and a variety of fields and methods would be provided. For simplicity, the abstract type `uk.ac.imperial.doc.Animal` defines only one method, the abstract method `doSomething()`. The extended control-flow analysis determines this field may be populated by concrete classes: `uk.ac.imperial.doc.Hawk` and `uk.ac.imperial.doc.Dog` as a result of blocks 2, 5 and 8. Block 9 invokes the `doSomething()` on the $animal$ instance and dynamic dispatch adds directed edges from block 9 to blocks 11 (Hawk), 12 (Dog) and to the uncaught JCRE exception block 10 (since $animal$ *may* have been `null` according to the extended control-flow analysis). The is modelled in the ILP as:

$$e9 = f9_10 + f9_11 + f9_12;$$

with the alternative paths represented due to the rule on the sum of the outgoing edges. Since the objective function is to maximise a sum involving $t9$, and the definition of $t9 = 1e9$, the solver selects the edge which leads to the highest resource cost. This is straightforward in our example since each rival implementation of `doSomething()` does not invoke any other methods or have any loops or conditionals, but in a real-life example, rival implementations of a polymorphic method may very well do so, and the beauty of the ILP formulation and ILP solvers is the ability to find a *global* maximum of the objective function i.e. to select the worst-case resource-usage across *all* paths in a Carmel program P.

The ILP is documented in Tables 6.4 – 6.5 and the syntax of the ILP is tailored for the open-source linear integer programming solver lpsolve5.5[5]. The solution to the ILP is given in Table 6.6 and aspects included in the control-flow graph given in Figure 6.3. From the solution we learn a number of interesting things:

- the worst-case resource-usage is 30 (with the unit of measurement here being the number of Carmel instructions executed);
- the edges not forming part of the worst-case resource-usage have a frequency of zero;
- the blocks B_i not forming part of the worst-case resource-usage have an execution frequency $e_i = 0$. We see that $e_{11} = 0$ and $e_{10} = 0$ and so the corresponding blocks 11 (the Hawk implementation of the `doSomething()` method) and block 10 (an unhandled exception return to the JCRE) are not executed in the worst-case resource-usage path, and see that the Dog implementation of the `doSomething()` method (block 12) and normal return to the JCRE (block 17) are part of the worst-case resource-usage path, since $e_{12} = 1 \neq 0$ and $e_{17} = 1 \neq 0$. This is shown in Figure 6.3.

[5]and we like very much the lpsolve package and its easy-to-use lpsolve integrated development environment.

Table 6.2: Carmel program demonstrating overriding, part 1

Block number	Carmel code

	`package uk.ac.imperial.doc { 0xa0, 0x24, 0x24, 0x24, 0x24, 0x2 };`
	`public abstract class Animal {`
	`public abstract void doSomething();`
	`}`
	`public class Hawk extends uk.ac.imperial.doc.Animal {`
	`public void doSomething() {`

11	`0: nop` `1: return`
	`}` `public void <init>(){`
4	`0: return`

	`}`
	`}`
	`public class Dog extends uk.ac.imperial.doc.Animal {`
	`public void doSomething() {`

12	`0: nop`
	`1: nop`
	`2: nop`
	`3: nop`
	`4: return`

	`}` `public void <init>(){`
7	`0: return`
	`}` `}`

Table 6.3: Carmel program demonstrating overriding, part 2

Block number	Carmel code

	public class AnimalApplet extends javacard.framework.Applet
	{ 0xa0, 0x24, 0x24, 0x24, 0x24, 0x2, 0x01 } {
	
	private uk.ac.imperial.doc.Animal animal;
	
	public void <init>(byte[]){

	0: load r 0 16: new uk.ac.imperial.doc.Hawk
2	19: dup 1 0
3	20: invokedefinite uk.ac.imperial.doc.Hawk.<init>()
	23: putfield this uk.ac.imperial.doc.NewApplet.animal 30: new uk.ac.imperial.doc.Dog
5	33: dup 1 0
6	34: invokedefinite uk.ac.imperial.doc.Dog.<init>()
	37: putfield this uk.ac.imperial.doc.NewApplet.animal 40: load r 0
8	41: getfield uk.ac.imperial.doc.NewApplet.animal
9	44: invokevirtual uk.ac.imperial.doc.Animal.doSomething()
13	47: return

	}
	
	public static void install(byte[], short, byte){

	0: load r 0 3: store r 4 4: new uk.ac.imperial.doc.AnimalApplet 5: store r 5 7: load r 5
0	8: load r 4
1	9: invokevirtual uk.ac.imperial.doc.AnimalApplet.<init>(byte[])
14	12: load r 5
15	13: invokedefinite javacard.framework.Applet.register()
16	14: return
	} }

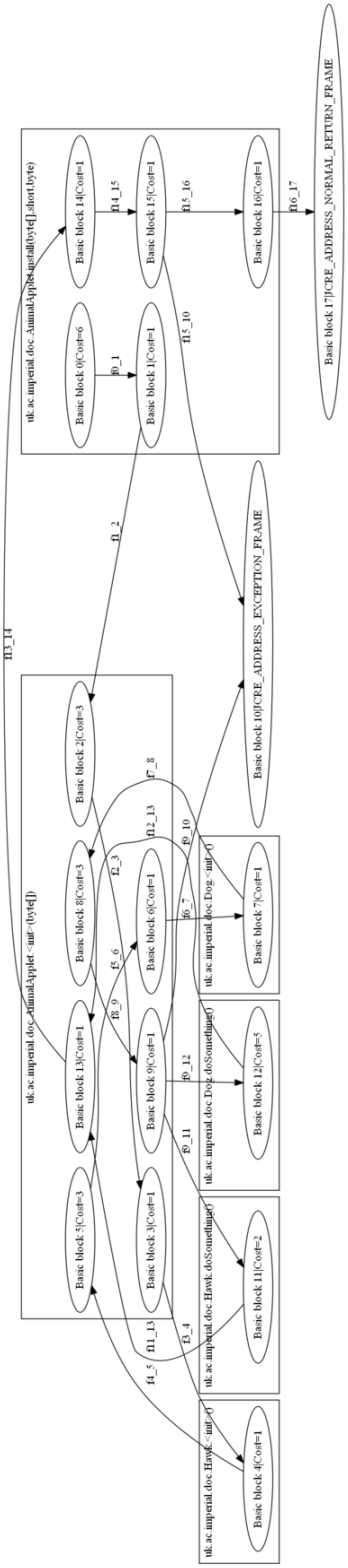


Figure 6.2: Control-flow graph with resource costs of a Carmel program showing overridden versions of a `doSomething` method

Table 6.4: ILP generated from Carmel program demonstrating overriding, Part 1

/*Objective function Carmel address: (0,uk.ac.imperial.doc.AnimalApplet.install(byte[],short,byte))*/
max: t0 + t1 + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10 + t11 + t12 + t13 + t14 + t15 + t16 + t17;

/*Flow constraints*/
S: fs = 1;
T: ft = 1;
e0 = fs;
e0 = f0_1;
e1 = f0_1;
e1 = f1_2;
e2 = f1_2;
e2 = f2_3;
e3 = f2_3;
e3 = f3_4;
e4 = f3_4;
e4 = f4_5;
e5 = f4_5;
e5 = f5_6;
e6 = f5_6;
e6 = f6_7;
e7 = f6_7;
e7 = f7_8;
e8 = f7_8;
e8 = f8_9;
e9 = f8_9;
e9 = f9_10 + f9_11 + f9_12;
e10 = f9_10 + f15_10;
e10 $\leq$ ft;
e11 = f9_11;
e11 = f11_13;
e12 = f9_12;
e12 = f12_13;
e13 = f11_13 + f12_13;
e13 = f13_14;
e14 = f13_14;
e14 = f14_15;
e15 = f14_15;
e15 = f15_10 + f15_16;
e16 = f15_16;
e16 = f16_17;
e17 = f16_17;
e17 $\leq$ ft;

/*one terminal exit*/
TERMINAL_CONSTRAINT: f9_10 + f15_10 + f16_17 = ft;

/*resource usage cost*/
t0 = 6 e0;
t1 = 1 e1;
t2 = 3 e2;
t3 = 1 e3;
t4 = 1 e4;
t5 = 3 e5;
t6 = 1 e6;
t7 = 1 e7;
t8 = 3 e8;
t9 = 1 e9;
t10 = 0 e10;
t11 = 2 e11;
t12 = 5 e12;
t13 = 1 e13;
t14 = 1 e14;
t15 = 1 e15;
t16 = 1 e16;
t17 = 0 e17;

Table 6.5: ILP generated from Carmel program demonstrating overriding, Part 2

```
/*non-negativity clauses*/
e0 ≥ 0;
e1 ≥ 0;
e10 ≥ 0;
e11 ≥ 0;
e12 ≥ 0;
e13 ≥ 0;
e14 ≥ 0;
e15 ≥ 0;
e16 ≥ 0;
e17 ≥ 0;
e2 ≥ 0;
e3 ≥ 0;
e4 ≥ 0;
e5 ≥ 0;
e6 ≥ 0;
e7 ≥ 0;
e8 ≥ 0;
e9 ≥ 0;
f0_1 ≥ 0;
f11_13 ≥ 0;
f12_13 ≥ 0;
f13_14 ≥ 0;
f14_15 ≥ 0;
f15_10 ≥ 0;
f15_16 ≥ 0;
f16_17 ≥ 0;
f1_2 ≥ 0;
f2_3 ≥ 0;
f3_4 ≥ 0;
f4_5 ≥ 0;
f5_6 ≥ 0;
f6_7 ≥ 0;
f7_8 ≥ 0;
f8_9 ≥ 0;
f9_10 ≥ 0;
f9_11 ≥ 0;
f9_12 ≥ 0;
t0 ≥ 0;
t1 ≥ 0;
t10 ≥ 0;
t11 ≥ 0;
t12 ≥ 0;
t13 ≥ 0;
t14 ≥ 0;
t15 ≥ 0;
t16 ≥ 0;
t17 ≥ 0;
t2 ≥ 0;
t3 ≥ 0;
t4 ≥ 0;
t5 ≥ 0;
t6 ≥ 0;
t7 ≥ 0;
t8 ≥ 0;
t9 ≥ 0;

/*integral clauses*/
int e0;
int e1;
int e10;
int e11;
int e12;
int e13;
int e14;
int e15;
int e16;
int e17;
int e2;
int e3;
int e4;
int e5;
int e6;
int e7;
int e8;
int e9;
int f0_1;
int f11_13;
int f12_13;
int f13_14;
int f14_15;
int f15_10;
int f15_16;
int f16_17;
int f1_2;
int f2_3;
int f3_4;
int f4_5;
int f5_6;
int f6_7;
int f7_8;
int f8_9;
int f9_10;
int f9_11;
int f9_12;
int t0;
int t1;
int t10;
int t11;
int t12;
int t13;
int t14;
int t15;
int t16;
int t17;
int t2;
int t3;
int t4;
int t5;
int t6;
int t7;
int t8;
int t9;
```

Table 6.6: Solution to ILP generated from Carmel program demonstrating overriding

Variables	MILP Feasible	result
	30	30
f0_1	1	1
f1_2	1	1
f2_3	1	1
f3_4	1	1
f4_5	1	1
f5_6	1	1
f6_7	1	1
f7_8	1	1
f8_9	1	1
f9_10	0	0
f9_11	0	0
f9_12	1	1
f11_13	0	0
f12_13	1	1
f13_14	1	1
f14_15	1	1
f15_10	0	0
f15_16	1	1
f16_17	1	1
fs	1	1
ft	1	1

Variables	MILP Feasible	result
e0	1	1
e1	1	1
e2	1	1
e3	1	1
e4	1	1
e5	1	1
e6	1	1
e7	1	1
e8	1	1
e9	1	1
e10	0	0
e11	0	0
e12	1	1
e13	1	1
e14	1	1
e15	1	1
e16	1	1
e17	1	1
t0	6	6
t1	1	1
t2	3	3
t3	1	1
t4	1	1
t5	3	3
t6	1	1
t7	1	1
t8	3	3
t9	1	1
t10	0	0
t11	0	0
t12	5	5
t13	1	1
t14	1	1
t15	1	1
t16	1	1
t17	0	0

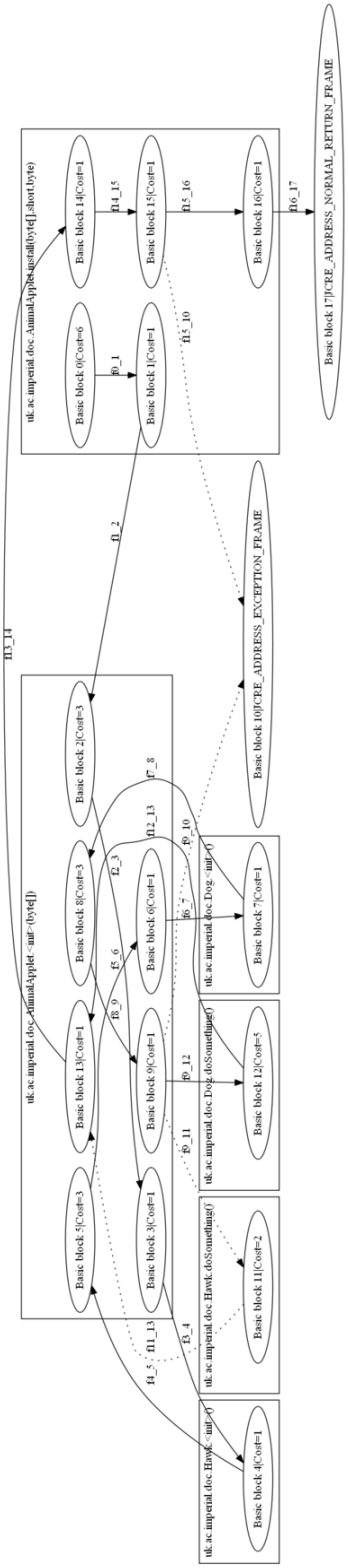


Figure 6.3: Control-flow graph with resource costs of a Carmel program showing overridden versions of a `doSomething` method, dotted lines show edges NOT taken in worst-case resource-usage, all other edges are executed exactly once

6.6.3 Sample generation of an integer programming problem for a triangular triple-nested loop

In this subsection we demonstrate how our approach to generating ILP handles triangular and more generally nested loops. A triangular loop is a loop in which the the number of executions of an inner loop is dependent on a variable in an outer loop. For example in the following Java code:

```
int i, j, k;
for (i=0; i < 100; i++){          // max iterations of loop 100
   for (j=i; j < 75; j=j+2){      // max iterations of loop 38  when i =0
        for (k=200; k > 0; k--){ // max iterations of loop 200
         ....
        }
    }
}
```

the middle loop (variable j) depends on the outer loop (variable i) and so this is a triangular loop.

In Table 6.7, we present the simplified Carmel code for an applet `TripleLoopApplet`, corresponding to the above Java code, including annotations of which basic block instructions belong to. The control-flow graph for this program is shown in Figure 6.4 and includes the basic blocks and their resource cost. The entry point is block 0, the first block of the `TripleLoopApplet install` method. This is admittedly not a good example of applet code: it is intended to demonstrate how we generate an ILP for nested loops. In the Carmel code, index 11 corresponds to i in the Java code, index 12 corresponds to j in the Java code, and index 13 corresponds to k in the Java code. Block 2 corresponds to $j = i$.

Table 6.8 shows how we calculate the loop bounds from the extended control-flow analysis. While we are able to calculate exact loop bounds for simple loop variables i and k, our approach to calculating loop bounds conservatively (over-) approximates the loop bounds for j – further investigation is needed to reduce the degree of over–approximation. On the other hand, other tools such as JOP and SWEET state they are unable to handle triangular loops.

The ILP is documented in Table 6.9 and the syntax of the ILP is tailored for the open-source linear integer programming solver lpsolve5.5[6]. The solution to the ILP is given in Tables 6.10 and aspects included in the control-flow graph given in Figure 6.5. From the solution we learn a number of interesting things:

- the worst-case resource-usage is 6051006 (with the unit of measurement here being the number of Carmel instructions executed);

[6] and we like very much the lpsolve package and its easy-to-use lpsolve integrated development environment.

- all edges and blocks are visited (all fx_y and e_i are positive integers).

Table 6.7: Carmel program demonstrating triangular triple-nested loop

Block number	Carmel code

	`package uk.ac.imperial.doc { 0xa0, 0x24, 0x24, 0x24, 0x24, 0x2 };`
	
	`public class TripleLoopApplet extends javacard.framework.Applet`
	`{ 0xa0, 0x24, 0x24, 0x24, 0x24, 0x2, 0x02 } {`
	
	`public static void install(byte[], short, byte) {`

0	`0: push s 0` `1: store s 11`

1	`2: load s 11`
	`3: push s 100`
	`5: if s ge goto 51`

2	`8: load s 11` `10: store s 12`

3	`11: load s 12`
	`12: push s 75`
	`14: if s ge goto 45`

4	`17: push s 200` `20: store s 13`

5	`21: load s 13`
	`215: push s 0`
	`22: if s le goto 39`

6	`25: nop`
	`33: inc s 13 -1`
	`36: goto 21`

7	`39: inc s 12 2` `42: goto 11`
8	`45: inc s 11 1` `48: goto 2`
9	`51: return`
	`}` `}`

Table 6.8: Calculation of loop bounds for Carmel program demonstrating triangular triple-nested loop

Loop header at basic block 1, pc=5, back edge at block 8, immediate dominator block 0, stack loop value operands:
- (s (0,0,0), pc: 1,(StoreTIArguments (t=s), (i=11)))
- (s (1,100,100), pc: 45,(IncTICArguments (t=s), (i=11),(c=1)))
$\Rightarrow$ max=100.0,min=0.0,step=1.0,bound=100.0
$\Rightarrow$ $f8_1 = 100\ f0_1$

Loop header at basic block 3, pc=14, back edge at block 7, immediate dominator block 2, stack loop value operands:
- (s (0,99,99), pc: 10,(StoreTIArguments (t=s), (i=12)))
- (s (2,76,38), pc: 39,(IncTICArguments (t=s), (i=12),(c=2)))
$\Rightarrow$ max=99.0,min=0.0,step=2.0,bound=50.0
$\Rightarrow$ $f7_3 = 50\ f2_3$

Loop header at basic block 5, pc=22, back edge at block 6, immediate dominator block 4, stack loop value operands:
- (s (200,200,0), pc: 20,(StoreTIArguments (t=s), (i=13)))
- (s (0,199,200), pc: 33,(IncTICArguments (t=s), (i=13),(c=-1)))
$\Rightarrow$ max=200.0,min=0.0,step=1.0,bound=200.0
$\Rightarrow$ $f6_5 = 200\ f4_5$

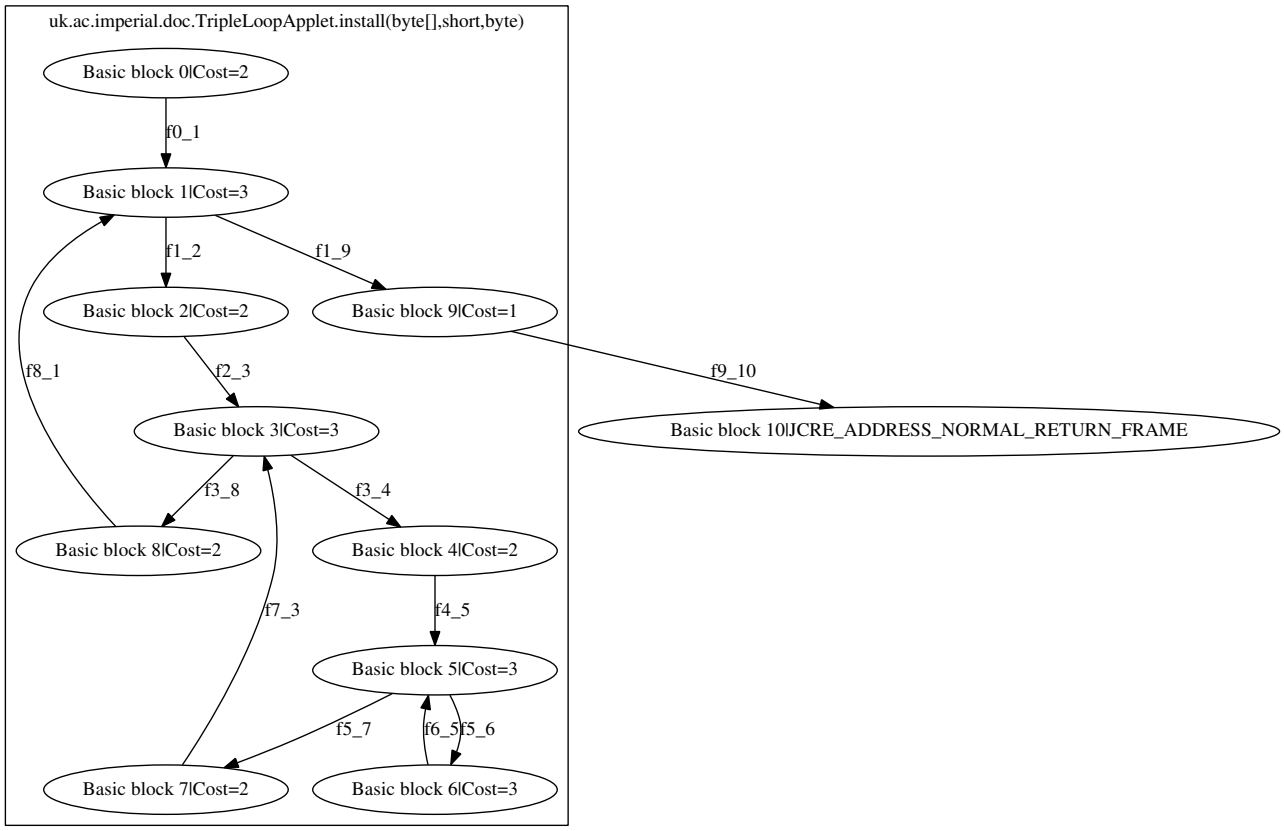


Figure 6.4: Control-flow graph of a Carmel program showing a triple-nested triangular loop.

Table 6.9: ILP generated from Carmel program demonstrating a triangular triple-nested loop

/*Objective function Carmel address: (0,uk.ac.imperial.doc.TripleLoopApplet.install(byte[],short,byte))*/
max: t0 + t1 + t2 + t3 + t4 + t5 + t6 + t7 + t8 + t9 + t10;

/*Flow constraints*/
S: fs = 1;
T: ft = 1;
e0 = fs;
e0 = f0_1;
e1 = f0_1 + f8_1;
e1 = f1_2 + f1_9;
e2 = f1_2;
e2 = f2_3;
e3 = f2_3 + f7_3;
e3 = f3_4 + f3_8;
e4 = f3_4;
e4 = f4_5;
e5 = f4_5 + f6_5;
e5 = f5_6 + f5_7;
e6 = f5_6;
e6 = f6_5;
e7 = f5_7;
e7 = f7_3;
e8 = f3_8;
e8 = f8_1;
e9 = f1_9;
e9 = f9_10;
e10 = f9_10;
e10 $\leq$ ft;

/*one terminal exit*/
TERMINAL_CONSTRAINT: f9_10 = ft;

/*loop bounds*/
f8_1 = 100 f0_1;
f7_3 = 50 f2_3;
f6_5 = 200 f4_5;

/*non-negativity clauses*/
e0 $\geq$ 0;
e1 $\geq$ 0;
e10 $\geq$ 0;
e2 $\geq$ 0;
e3 $\geq$ 0;
e4 $\geq$ 0;
e5 $\geq$ 0;
e6 $\geq$ 0;
e7 $\geq$ 0;
e8 $\geq$ 0;
e9 $\geq$ 0;
f0_1 $\geq$ 0;
f1_2 $\geq$ 0;
f1_9 $\geq$ 0;
f2_3 $\geq$ 0;
f3_4 $\geq$ 0;
f3_8 $\geq$ 0;
f4_5 $\geq$ 0;
f5_6 $\geq$ 0;
f5_7 $\geq$ 0;
f6_5 $\geq$ 0;
f7_3 $\geq$ 0;
f8_1 $\geq$ 0;
f9_10 $\geq$ 0;
t0 $\geq$ 0;
t1 $\geq$ 0;
t10 $\geq$ 0;
t2 $\geq$ 0;
t3 $\geq$ 0;
t4 $\geq$ 0;
t5 $\geq$ 0;
t6 $\geq$ 0;
t7 $\geq$ 0;
t8 $\geq$ 0;
t9 $\geq$ 0;

/*integral clauses*/
int e0;
int e1;
int e10;
int e2;
int e3;
int e4;
int e5;
int e6;
int e7;
int e8;
int e9;
int f0_1;
int f1_2;
int f1_9;
int f2_3;
int f3_4;
int f3_8;
int f4_5;
int f5_6;
int f5_7;
int f6_5;
int f7_3;
int f8_1;
int f9_10;
int t0;
int t1;
int t10;
int t2;
int t3;
int t4;
int t5;
int t6;
int t7;
int t8;
int t9;

/*resource usage cost*/
t0 = 2 e0;
t1 = 3 e1;
t2 = 2 e2;
t3 = 3 e3;
t4 = 2 e4;
t5 = 3 e5;
t6 = 3 e6;
t7 = 2 e7;
t8 = 2 e8;
t9 = 1 e9;
t10 = 0 e10;

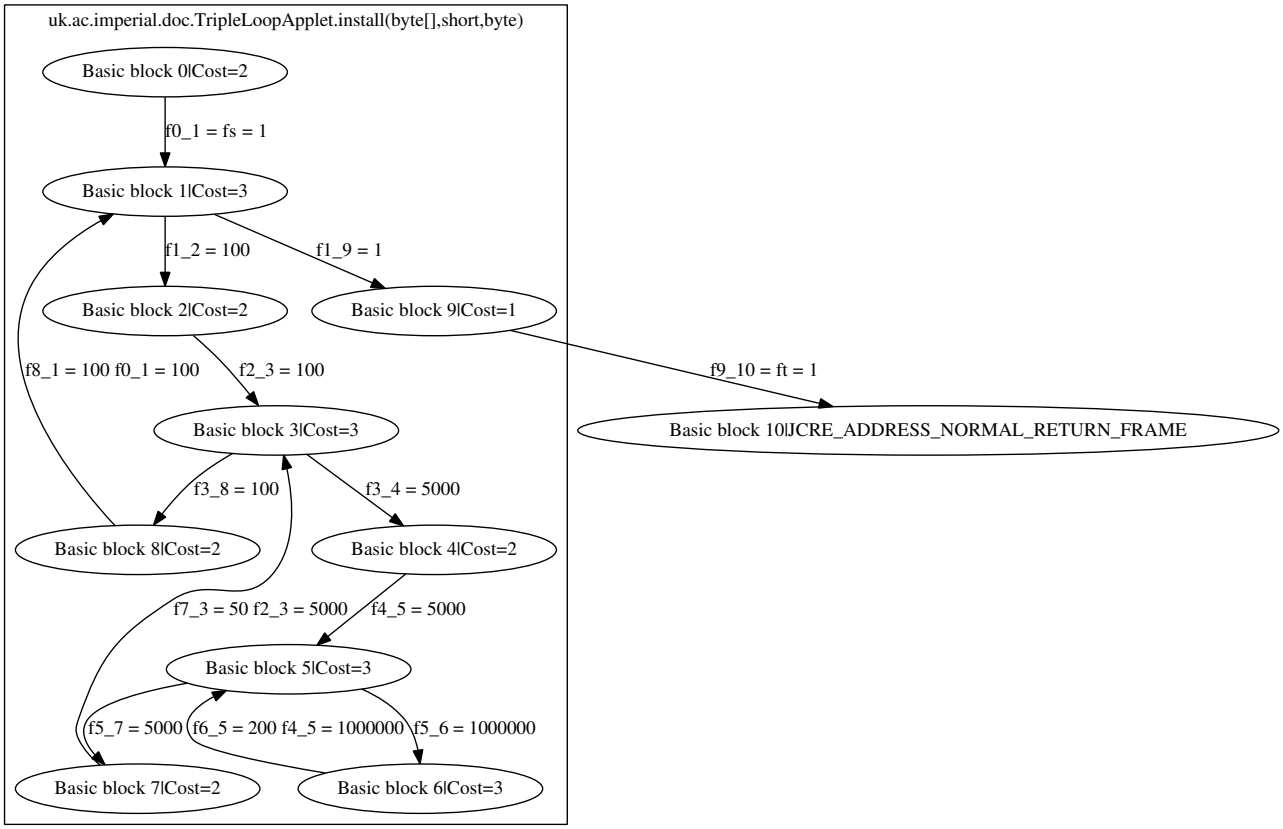


Figure 6.5: Control-flow graph of a Carmel program showing a triple-nested triangular loop, annotated with frequencies derived from solution of the corresponding ILP.

Table 6.10: Solution to ILP generated from Carmel program demonstrating triangular loops

Variables	MILP Feasible	result
	6051006	6051006
e0	1	1
e1	101	101
e2	100	100
e3	5100	5100
e4	5000	5000
e5	1005000	1005000
e6	1000000	1000000
e7	5000	5000
e8	100	100
e9	1	1
e10	1	1
f0_1	1	1
f1_2	100	100
f1_9	1	1
f2_3	100	100
f3_4	5000	5000
f3_8	100	100
f4_5	5000	5000
f5_6	1000000	1000000
f5_7	5000	5000
f6_5	1000000	1000000
f7_3	5000	5000
f8_1	100	100
f9_10	1	1
fs	1	1
ft	1	1
t0	2	2
t1	303	303
t2	200	200
t3	15300	15300
t4	10000	10000
t5	3015000	3015000
t6	3000000	3000000
t7	10000	10000
t8	200	200
t9	1	1
t10	0	0

6.7 Evaluation of the Precision of the Worst-Case Resource-Usage Analysis

Calculating the worst-case execution-time of a program is in general an undecidable problem, equivalent to the halting problem [WEE$^+$08], and so evaluation of the precision of the worst-case resource-usage of a Carmel program is best conducted in terms of analysing a suite (or suites) of benchmark programs with known resource usage through our tool. Whilst implementing the

worklist solver, I composed a suite of my own input Carmel programs with manually computed known resource usage, mostly based on analysing a variety of different loops with different levels of nestedness and loop conditions, principally to test the correctness of the implementation.

The formal evaluation of our worst-case resource-usage tool has been conducted in terms of cross-comparison with the mature test suite of Mälardalen worst-case execution-time benchmarks[7] [GBEL10]. This suite is a collection of C programs for which staff at the Mälardalen Real-Time Research Centre have provided loop-bounds computed by their mature SWEET[8] tool. In particular, our evaluation of our worst-case resource-usage tool consists of two parts per test program, namely whether or not our tool:

- is able to provide loop bounds;
- is able to match the precision of the loop bounds provided by the SWEET tool;

for all reachable loops in the Carmel translation of the C test program in the suite of Mälardalen worst-case execution-time benchmarks [GBEL10]. The Mälardalen test suite consists, at the time of writing this, of 35 C programs. Our suite of equivalent Carmel programs consists of 26 of the original 35, the remaining tests were dropped for one of a number of reasons:

- they test recursion;
- the pointer arithmetic was too complex or impossible to translate into Carmel;
- the test program relies on floating-point arithmetic, which is not supported by the *Java Card* virtual machine;

Note in terms of translating each C program into the equivalent Carmel, we first transform the C program syntactically into valid Java code, and then translate from the generated Java bytecode the corresponding Carmel code.

Table 6.16 shows the results of cross-comparing the loop-bounds our tool generates against those generated by SWEET and the results are very encouraging. We were able to match SWEET for 18 of the 26 programs in terms of the number of loops detected and the loop bounds computed. There is a slight difference in how loop bounds are reported, in that SWEET reports a loop as having a bound of zero when it determines the loop will not be executed i.e. when SWEET determines the loop is executed zero times $\equiv$ the loop is not executed according to SWEET $\equiv$ the loop is not reachable in the control-flow graph our tool computes, and so no loop bound is given for it in our corresponding generated integer-linear programming problem, since the integer-linear programming problem is generated from the control-flow graph. Of the 8 programs for which we were not able to provide loop-bounds, 5 of these 8 were due to array indexes not moving monotonically forwards or backwards[9], a fundamental requirement in our approach to the calculation of conservative loop-bounds, and one of the validation conditions of Section 6.4. Of the other 3, one test case had a loop condition which depended on a non-constant global variable, and our tool couldn't bound its possible values. This code pattern is not a common pattern in *Java Card* applet code. The other two

[7] http://www.mrtc.mdh.se/projects/wcet/benchmarks.html
[8] http://www.mrtc.mdh.se/projects/wcet/sweet/index.html
[9] This is a natural/necessary part of the algorithm being used in some of the test programs e.g. the binary search and quicksort algorithms.

test programs have loop conditions which are not currently able to be bounded by our tool.

Careful analysis and reflection on the test suite showed three things:

- the design, use and integration of our reaching definitions for local variables analysis[10] to filter out non-reaching definitions and so increase the precision of loop-bounds was the right decision. The Mälardalen worst-case execution-time benchmarks extensively reuse the same local variable for different purposes in the same function/method. Consider Table 6.11, which shows a sequence of three loops using the same local variable, whose values in each loop range over distinct subranges of the `short` datatype. Without reaching definitions[11], the loop bounds of the second loop would be substantially less accurate as the maximum value of the local variable would be coarsened to 100 from the flow from the first loop and the loop bound computed as $[-50, 100, |1|] = 150$. Similarly the third loop bounds would be coarsened to $[-50, 200, |1|] = 250$;
- whilst designed with bounding iteration over arrays in mind, our tool is also able to bound iteration of a class of simple looping algorithms and so is able to provide accurate resource-usage of e.g. the iterative version of the calculation of the Fibonacci sequence[12] as shown in Table 6.13. The class of simple looping algorithms are those that iterate from a constant monotonically up or down towards a parameter n e.g. the iterative version of the Fibonacci sequence iterates monotonically from 2 to n and the loop-bound is computed correctly as $(n - 1)$ for $2 \leq n \leq 30$. [The upper-limit is due to the first condition in the C for loop, and so replicated in the Carmel code];
- our tool shows great promise of being able to be extended to calculate loop bounds for more complex algorithms, by analysing the product of arithmetic operations used as loop conditions. Consider Table 6.12, which presents the Java and Carmel versions of the Mälardalen C implementation[13] of the classic algorithm for determining whether or not a positive integer $n \geq 2$ is a prime number. Its essence is that if n is even, then if $n = 2$, it is prime, otherwise it is not prime. Otherwise, we iterate over the odd integers from 3 to $\lceil \sqrt{n} \rceil$ and check whether it is a factor of n - if it a factor, then n cannot be prime. If iteration to $\lceil \sqrt{n} \rceil$ completes with no factors have been found, then n is prime. This second part is expressed in this implementation with the loop:

  ```
  for (i = 3; i * i <= n; i += 2)
  ```

  with the key part of the corresponding Carmel code being:

  ```
  31: inc s 1 2


  36: load s 1
  37: load s 1
  38: numop s mul
  ```

[10] or equivalent techniques
[11] or equivalent techniques
[12] http://www.mrtc.mdh.se/projects/wcet/wcet_bench/fibcall/fibcall.c
[13] http://www.mrtc.mdh.se/projects/wcet/wcet_bench/prime/prime.c

```
39: load s 0
40: if s le  goto    21
```

The static analysis underlying our tool is not able to bound this loop since it cannot determine the relationship between i, the result of the binary operator $i * i = i^2$ and n. In particular, the `if` statement at program counter (pc) 40 does not recognise one of its operands as being a derived function (via pc 38) of local variable 1 at pc 31. Note that I consider it to be a derived function because the operand to the `if` statement at pc 40 that is a function of local variable 1 $(= i)$ at pc 31 remains a reaching definition at pc 40, and so can be expressed as a function of the local variable 1 at pc 40. This opens the door to filtering the values of local variable 1 at pc 40 to its successor addresses according to whether the operator at pc 38 succeeds or fails. Such filtering prevents the local variable 1 at pc 31 being forced to top.

For exploratory reasons only, as a foray into the feasibility of the approach, we extended the implementation of each abstract number to add the set of its operands, and for the multiplication operation to capture its operands and to add filtering as described above. The Java code implementing this change is given in Tables 6.14 and 6.15 and the results have been astonishingly accurate. Trying around 30 prime and non prime numbers with the new code, at the analysis results for local variable 1 at pc 40, the values have ranged from 3 to $(\lceil \sqrt{n} \rceil + 1)$ and the `boolean` value of the primes method returned to pc 13 of the `install` method has been correct for the value passed in at pc 6 has been true $(1, 1, X)$ or false $(0, 0, Y)$ and never maybe-true-and-false $(0, 1, W)$. Much further analysis is required before adopting this approach, but our change makes logical sense and produces the right set of values for each input parameter I tried. From this set of values, we can produce the loop-bound as the loop increment is known (two).

Table 6.11: Illustration on how the use of reaching definitions improve the prevision of loop bounds

C Code	Carmel translation	Loop bounds
short i = 0;	0: push s 0 1: store s 11	

for (i = 0; i < 100; i++) {
No changes to i
}
	2: push s 0
	3: store s 11
	4: goto 15
	7: nop
	
	10: inc s 11 1
	
	15: load s 11
	16: push s 100
	18: if s lt goto 7
		(s (0,0,0), (3,store (t=s), (i=11)))
		(s (1,100,100),(10,inc (t=s), (i=11),(c=1)))
		loop bound, max=100.0,min=0.0,step=1.0,bound=100.0

for (i = -50; i < -20; i ++) {
No changes to i
}
	102: push s -50
	103: store s 11
	104: goto 1015
	107: nop
	
	1010: inc s 11 1
	
	1015: load s 11
	1016: push s -20
	1018: if s lt goto 107
		(s (-50,-50,0),(103,store (t=s), (i=11)))
		(s (-49,-20,30),(1010,inc (t=s), (i=11),(c=1)))
		loop bound, max=-20.0,min=-50.0,step=1.0,bound=30.0

for (i = 250; i > 200; i–) {
No changes to i
}
	202: push s 250
	203: store s 11
	204: goto 2015
	207: nop
	
	2020: inc s 11 -1
	
	2015: load s 11
	2016: push s 200
	2018: if s gt goto 207
	
	21: return
		(s (250,250,0),(203,store (t=s), (i=11)))
		(s (200,249,50),(2020,inc (t=s), (i=11),(c=-1)))
		loop bound, max=250.0,min=200.0,step=1.0,bound=50.0

Table 6.12: An algorithm for determining whether a given positive integer is prime

Java form of C Code	Carmel translation	Values

public static boolean divides(short n, short m) {
return (m % n == 0);
}
 	
public static boolean even(short n) {
return (divides((short) 2, n));
}
 	
public static boolean prime(short n) {
short i;
 	
if (even(n))
return (n == 2);
for (i = 3; i * i ¡= n; i += 2) {
if (divides(i, n))
return false;
}
return true;
}
 	
public static void main(String[] args) {
short x = 6217;
 	
boolean prime = prime(x);
}
	public static void install(byte[], short, byte){
	
	0: push s 6217
	6: invokedefinite uk.ac.imperial.PrimesApplet.prime(short)
	13: return
	}
	
	public static boolean prime(short){
	
	0: load s 0
	1: invokedefinite uk.ac.imperial.PrimesApplet.even(short)
	4: if s eq null goto 16
	7: load s 0
	8: push s 2
	9: if s ne goto 14
	12: push s 1
	13: return s
	14: push s 0
	15: return s
	16: push s 3
	17: store s 1
	18: goto 36
	21: load s 1
	22: load s 0
	23: invokedefinite uk.ac.imperial.PrimesApplet.divides(short,short)
	26: if s eq null goto 31
	29: push s 0
	30: return s
	31: inc s 1 2
	
	36: load s 1
	37: load s 1
	38: numop s mul
	
	39: load s 0
	40: if s le goto 21
	43: push s 1
	44: return s
	}
	
	public static boolean even(short){
	
	0: push s 2
	1: load s 0
	2: invokedefinite uk.ac.imperial.PrimesApplet.divides(short,short)
	5: return s
	}
	
	public static boolean divides(short, short){
	
	0: load s 1
	1: load s 0
	2: numop s rem
	3: if s ne null goto 8
	6: push s 1
	7: return s
	8: push s 0
	9: return s
	}
		(s (3,3,0), (17,store (t=s), (i=1)))
		(s (5,80,78), (31,inc (t=s), (i=1),(c=2)))
		(s (3,78,77), (38,binop (t=s), (op=MUL)))

Table 6.13: Illustration on how Fulgurite is able to calculate loop bounds for a class of simple looping algorithms such as computing the Fibonacci sequence

C Code	Carmel translation	Loop bound for fib(30)

int fib(int n){
 	
int i, Fnew, Fold, temp,ans;
 	
Fnew = 1; Fold = 0;
for (i = 2;
i ¡= 30 && i ¡= n;
i++)
{
temp = Fnew;
Fnew = Fnew + Fold;
Fold = temp;
}
ans = Fnew;
return ans;
}
 	
int main()
{
int a;
 	
a = 30;
fib(a);
return a;
}
	public static short fib(short){
	
	0: push s 1
	1: store s 2
	2: push s 0
	3: store s 3
	4: push s 2
	5: store s 1
	6: goto 25
	9: load s 2
	10: store s 4
	12: load s 2
	13: load s 3
	14: numop s add
	16: store s 2
	17: load s 4
	19: store s 3
	20: inc s 1 1
	25: load s 1
	26: push s 30
	28: if s gt goto 36
	31: load s 1
	32: load s 0
	33: if s le goto 9
	36: load s 2
	37: store s 5
	39: load s 5
	41: return s
	
	}
		(s (2,2,0), (5, store (t=s), (i=1)))
		(s (3,31,29), (20, inc (t=s), (i=1),(c=1)))
		loop bound, max=31.0,min=2.0,step=1.0,bound=29.0

```
SortedSet<AbstractCarmelNumber> absX2Operands = absX2.getOperands();
filter4 = absX2.getOperands().size() == 1
     && absX2.getCarmelAddress().getInstruction().isNumericBinopInstruction()
     && absX2.getOperands().first().getCarmelAddress().getInstruction()
          .isLocalVariableStoreOrIncInstruction()
     && absX2.getOperands().first().getOperands().size() == 0;

if (filter4) {

 AbstractCarmelNumber abx2operand1 = absX2Operands.first();

 ExtendedCarmelAddress addr = new ExtendedCarmelAddress(abx2operand1.getCarmelAddress(),
      (((ReachingDefinitionInstuctionArguments) abx2operand1.getCarmelAddress()
           .getInstructionArguments()).getI()));

 boolean b = ((rdlv.contains(addr)));

 BINOP_NO_OPT_T_NUMERIC_Arguments instructionArguments2 = (BINOP_NO_OPT_T_NUMERIC_Arguments) absX2
      .getCarmelAddress().getInstructionArguments();

 BinaryNumericOperator binop = instructionArguments2.getBinop();

 i2 = addr.getIndex();

 CarmelLUBTreeSet i2Lub = localVarArray.get(new CarmelIntegerIndex(i2));
 AbstractShortNumber[] abstractShortNumbers = i2Lub.getAbstractShortNumbers()
      .toArray(new AbstractShortNumber[0]);

 for (acn1low = absX1.getMin(); b && acn1low <= acn1high; acn1low++) {
     for (AbstractShortNumber sNumber : abstractShortNumbers) {
      for (acn2low = sNumber.getMin(); acn2low <= sNumber.getMax(); acn2low++) {

          temp1 = new AbstractShortNumber((short) acn1low, (short) acn1low,
               acn1CurrentModCount, absX1.getCarmelAddress());

          temp2 = new AbstractShortNumber((short) acn2low, (short) acn2low,
               acn2CurrentModCount, absX2.getCarmelAddress());

          AbstractShortNumber temp3 = (AbstractShortNumber) temp2.apply(binop, temp2, temp2,
               instructionArguments2, carmelAddress);

          bool = absX2.apply(compOp, temp3, temp1, instructionArguments, carmelAddress);

          if (bool.containsTrueBooleanValue()) {

           testPassed = true;

           trueJ.add(temp2, carmelAddress);
          }

          if (bool.containsFalseBooleanValue()) {

           testFailed = true;

           falseJ.add(temp2, carmelAddress);
          }
      }
     }
 }
```

Table 6.14: Exploratory Fulgurite code handling primes scenario, part 1

```
if (testFailed) {

    falseLocal = falseLocal
         .LUBGeneral(localVarArray.duplicateLocalVarArrayReplacingMappingForMappableKey(
              new CarmelIntegerIndex(i2), falseJ, carmelAddress), carmelAddress);
}

if (testPassed) {

    trueLocal = trueLocal
         .LUBGeneral(localVarArray.duplicateLocalVarArrayReplacingMappingForMappableKey(
              new CarmelIntegerIndex(i2), trueJ, carmelAddress), carmelAddress);
}

if (trueLocal.keySet().size() > 0) {

    addCarmelOpStackAndLocalVarArray(opStack, trueLocal, gotoAddrContext, gotoAddr, jHMap,
         carmelAddress, context);

}

if (falseLocal.keySet().size() > 0) {

    addCarmelOpStackAndLocalVarArray(opStack, falseLocal, nextAddressContextSequence,
         nextAddress, jHMap, carmelAddress, context);

}
```

Table 6.15: Exploratory Fulgurite code handling primes scenario, part 2

Table 6.16: Cross-comparison of our tool with the Mälardalen worst-case execution-time benchmarks [GBEL10]

Benchmark	Description	Can determine loop bounds	Reason why loop cannot be bound
adpcm*	Adaptive pulse code modulation algorithm.	**FALSE**	**algorithmically too complex for Fulgurite presently**
bs	Binary search for the array of 15 integer elements.	**FALSE**	**non-monotonic array index movements**
bsort100	Bubblesort program.	**TRUE**	
cnt*	Counts non-negative numbers in a matrix.	**TRUE**	
compress*	Data compression program.	**FALSE**	**uses a global non-constant variable to control iteration**
cover*	Program for testing many paths.	**TRUE**	
crc*	Cyclic redundancy check computation on 40 bytes of data.	**TRUE**	
duff*	Using ”Duff’s device” from the Jargon file to copy 43 byte array.	**TRUE**	
edn*	Finite Impulse Response (FIR) filter calculations.	**TRUE**	
expint	Series expansion for computing an exponential integral function.	**TRUE**	
fdct	Fast Discrete Cosine Transform.	**TRUE**	
fibcall	Simple iterative Fibonacci calculation, used to calculate fib(30).	**TRUE**	
insertsort*	Insertion sort on a reversed array of size 10.	**FALSE**	**non-monotonic array index movements**
janne_complex*	Nested loop program.	**FALSE**	**non-monotonic array index movements**
lcdnum	Read ten values, output half to LCD.	**TRUE**	
ludcmp	LU decomposition algorithm.	**TRUE**	
matmult*	Matrix multiplication of two 20x20 matrices.	**TRUE**	
ndes*	Complex embedded code.	**TRUE**	
ns*	Search in a multi-dimensional array.	**TRUE**	
nsichneu*	Simulate an extended Petri Net.	**TRUE**	
prime	Calculates whether numbers are prime.	**FALSE**	**algorithmically too complex for Fulgurite presently**
qsort-exam	Non-recursive version of quick sort algorithm.	**FALSE**	**non-monotonic array index movements**
qurt	Root computation of quadratic equations.	**TRUE**	
select	A function to select the Nth largest number in a floating point array.	**FALSE**	**non-monotonic array index movements**
sqrt	Square root function implemented by Taylor series.	**TRUE**	
st	Statistics program.	**TRUE**	
ud	Calculation of matrixes.	**TRUE**	

Chapter 7

Related Work

7.1 Introduction

We begin this Chapter by honouring the foundations of this thesis and a related PhD thesis which has proved invaluable in the context of constructing proofs for flow-logic [NN02] program analyses of Carmel:

- This thesis is firmly rooted in the SecSafe project[1] [HSN+03, Pro02], a successfully completed EC-funded investigation whose main focus was on the development of static techniques to analyse security properties of realistic languages. Among the achievements of this project, the specification of the language *Java Card* Virtual Machine Language equivalent language Carmel, and the provision of an operational semantics and program analyses for Carmel, rate highly;
- In his PhD thesis, Hansen [Han05], a member of the original SecSafe project, extends the program analyses of Carmel from SecSafe and includes novel program analyses for security and information flow. We have referred to this thesis many times for the methodology of how to construct proofs in the flow-logic framework [NN02].

By related work we mean (most relevant) projects and published papers on the topics of analysis or certification[2] or estimation of resource-usage or resource cost or memory-usage or execution-time of computer languages, at three different levels of similarity to our thesis:

- *Java Card*: Despite extensive literature searches, we have been able to locate only one research project – the Castles[3] project – and a handful of published papers [CJPS05, GS05, SCJ06, PTTC08, PTT08, JJ05,

[1] Project IST-1999-29075, *Secure and Safe Systems based on Static Analysis*, funded by the European Community under the "Information Society Technologies" Programme (1998-2002)

[2] certified in the sense of formal certificates of program behaviour generated by proof-carrying code approaches inspired by [Nec97] or certified meaning proved by theorem provers

[3] http://raweb.inria.fr/rapportsactivite/RA2006/lande/uid44.html

LSWH09]. For the reader interested in where most research effort has been spent on *Java Card*, it has focussed on security considerations:

  - Formalising the Java Card platform and being able to prove security properties of *Java Card* applets and extract certified bytecode verifiers e.g. (Verificard [BDdS+02] and SecSafe [HSN+03] projects);
  - Co-evolution of attacks and defences against attacks on *Java Card* smartcards:
    * Hardware attacks and particularly fault analysis on specific *Java Card* smartcards
    * Software attacks against implementation deficiencies of *Java Card* API methods or on–card bytecode verifiers on particular *Java Card* smartcards
    * Platform weakness in relying on off-card bytecode validation allowing a window of opportunity to introduce corruptions e.g. defects in CAP files

    to introduce type-confusion, non-deterministic behaviour and the leakage of sensitive/secret information held on the smartcard e.g. [LBL+15, LBR+13, BG14, LBL+13, BLLL15, BTL13a, SFL13, BTL13b, BICL11].

- *Embedded systems*: If we widen our search to Java Micro Edition, a version of the Java platform designed for embedded systems such as TV set-top boxes, gateways, printers and mobile 'phones, which have greater computational resources than *Java Card* smartcards, there exists a number of projects devoted to proof-carrying code (including proofs of resource-usage properties) including:
  - the Mobile Resource Guarantee project[4]
  - the Mobius project[5]
  - the Lande project[6]
  - the COSTA project[7]

- *General programming languages*:

  We will cover projects including the C and Java programming languages.

[4] http://groups.inf.ed.ac.uk/mrg/
[5] http://software.imdea.org/∼gbarthe/mobius/bin/view/Mobius/WebHome.html
[6] http://raweb.inria.fr/rapportsactivite/RA2006/lande/uid4.html
[7] http://costa.ls.fi.upm.es/∼costa/costa/costa.php

7.2 *Java Card*

7.2.1 The Castles project and other *Java Card* works

The Castles project focused on creating a unified automated environment for formally certifying the *Java Card* platform and *Java Card* applets by integrating static analysis with automated proof checkers and software testing methodologies. Such integration was intended to provide an architecture capable of formally connecting and supporting specification, implementation and testing in one system. The Castles project partnered with, among others, the Lande project; whereas Castles' focus was on *Java Card*, the Lande project provided more supporting research and tooling in formal methods and static analysis.

Two papers from the Castles project are of particular interest to us, and both use variants of the Carmel programming language. The first [CJPS05, GS05] presents a certified[8] algorithm capable of determining whether any bytecode instruction in a Carmel program may dynamically allocate memory using the `new` operator in a loop. If there are no calls to the `new` operator in a loop, the program is deemed to execute in bounded memory; otherwise, it is deemed to execute in an unbounded amount of memory. The distinction between applets operating in bounded memory (applet is fine) and unbounded memory (applet is rejected) appears to be due to a judgement on the availability of an object deletion mechanism (which requires an explicit API call to reclaim unreachable objects; garbage collection remains optional in all Classic editions):

> "Indeed, for Java Card up to version 2.1 there is no garbage collector and starting with version 2.2 the machine includes a garbage collector which may be activated invoking an API function at the end of the execution of the applet. This has lead to a rather restrictive programming discipline for smart cards in which the programmer must avoid memory allocation in parts of the code that are within loops".

which is not in the spirit of the official intended use of the object deletion mechanism[9]:

> "The object deletion mechanism is not like garbage collection in standard Java technology applications due to space and time constraints. The amount of available RAM on the card is limited. In addition, because the object deletion mechanism is applied to objects stored in persistent memory, it must be used sparingly. EEPROM writes are very time-consuming operations and only a limited number of writes can be performed on a card.

[8] Here proved with the Coq theorem prover https://coq.inria.fr/
[9] https://docs.oracle.com/javacard/3.0.5/prognotes/object_deletion_mechanism.htm

> Due to these limitations, the object deletion mechanism in Java Card technology is not automatic: it is performed only when an applet requests it. Use the object deletion mechanism sparingly and only when other Java Card technology-based facilities are cumbersome or inadequate".

nor does the presented analysis ensure the applet under consideration actually calls the referenced API method to reclaim unreachable objects.

In contrast to our analysis, the algorithm presented in [CJPS05, GS05] does not provide any basis for *quantification* of the dynamic memory allocation using the `new` operator and excludes from consideration the API methods which create dynamic objects via API calls e.g. to any of the `JCSystem.makeXXXArray` methods.

The second paper [SCJ06] is interesting because it declares itself based on the *probabilistic* abstract interpretation framework of [DHW03]. Certainly it takes the central notions of viewing programs and systems as transitions systems with a quantity attached to each transition, and by taking a richer algebraic structure than a lattice (here a semi-idempotent ring) and using linear operators in the abstract space – but deviates from it by attaching numerical quantities between transitions representing the resource-usage to be measured[10] which in the context of this paper is the cost of a cache lookup failure for a local variable. It is shown how to compute correct (over-)approximations of the concrete resource cost.

[PTTC08, PTT08] presents an algorithm suitable for on–card calculation/validation of upper bounds on heap usage. However, it assumes loop-bounds are known and passed into the applet or are constants. Additionally, it only considers dynamic memory allocation via a `new` operator and excludes from consideration the API methods which create dynamic objects via API calls e.g. to any of the `JCSystem.makeXXXArray` methods.

[JJ05, BG02] are similar papers in that they share a common aim to reduce the size of applets bytecode on–card using elements familiar from compression techniques. The former observes that in *Java Card*, a range of the opcodes from full Java bytecode is unused and could therefore be repurposed to combine multiple bytecode instructions into new single bytecode instructions, saving space on-card whilst (speculatively) simultaneously improving performance. The paper reports a 10% saving in space saving using this approach. [BG02] is one of a number of papers which propose full compression of bytecode in a variety of schemes to save space with a corresponding change to the *Java Card Virtual Machine* to decompress the instruction

[10] rather than a probability between 0 and 1 as in the original work

prior to executing it. This paper measures a 2% to 30% drop in execution speed, for a space saving of 15%. Other papers e.g. [LSWH09] present approaches for improving aspects of the performance of *Java Card*, here transaction performance; since our emphasis is on language-based resource-usage analysis and static analysis, we do not discuss them further here.

7.3 Embedded systems

The Mobile Resource Guarantee project[11] developed a proof-carrying framework for a resource-aware functional language with object handling extensions (Camelot and O'Camelot for the version extended with object support) which is capable of generating from a Camelot or O'Camelot program a provably equivalent Java bytecode program and provably correct resource certificate(s). This separation of the source functional language from the target Java bytecode was designed to allow for future extensions to target other languages e.g. the Common Intermediate Language for .NET or Mono. None of the papers from this thesis have been helpful in analysing *Java Card* bytecode. The EmBounded project[12] is a successor to the Mobile Resource Guarantee project that aims to quantify and certify the resource-usage of another functional language Hume[13], a domain-specific language for real-time embedded systems; again, no papers in this thesis have been useful in understanding how to analyse *Java Card* bytecode.

The COSTA[14] project[15] uses cost-relations (extended forms of recurrence relations) to analyse Java bytecode and automatically generate and solve systems of cost equations to derive upper bounds on resource-usage for a variety of resource metrics e.g. number of instructions executed, dynamic memory allocated. COSTA can handle standard Java as well as micro Java versions and applies an admirable array of semantics-based transformation and analysis techniques, including partial evaluation, determination of loop invariants and ranking functions. Sometimes the system of cost equations COSTA generates is not solvable and no upper bound can be found [AAGP08]. This happens when, as part of the solution process, the recursive relations are being replaced with equivalent non-recursive *closed form*, non-determinacy in a relation is found. The COSTA system is used as part of the Mobius project and in the case-studies evaluation of the Mobius deliverable[16] where they conclude it can analyse "a relatively large class of Java bytecode programs, and gives reasonable results in terms of precision and efficiency".

[11] http://groups.inf.ed.ac.uk/mrg/
[12] http://www.embounded.org/
[13] http://www.hume-lang.org/
[14] COSt and Termination Analyzer for Java Bytecode
[15] http://costa.ls.fi.upm.es/∼costa/costa/costa.php
[16] http://software.imdea.org/∼gbarthe/mobius/bin/view/DeliverablesList/D5.2.pdf

As mentioned in Section 7.2.1, the Lande project[17] provided supporting research and tooling in formal methods and static analysis, and especially in terms of integrating static analysis with automated proof checkers and software testing methodologies. Most of the papers of the Lande project relate to certified proof–carrying code frameworks including how program/certificate consumers can verify on–device the certificate claims in a space efficient manner [BJP06b, BJP06a]. We have referenced in Section 7.2.1 the two resource-related papers from the Lande and Castles projects.

The Mobius project[18] is a successor to the Mobile Resource Guarantee project and its aims were to develop a proof-carrying framework (including resource-usage, information-flow and security policies) targeting the Java Micro Edition at the bytecode level. Combining an impressive battery of techniques and technologies, including certified[19] certificate-generating compilers (both interactive for ad-hoc policies and fully automatic predefined policies) and certified[20] certificate-checkers, theorem provers, type systems, Java Markup Language, Bytecode Markup Language, Resource Aware Java, cost-relations, BoogiePL, static analysis – including polyhedral analysis to recover intervals of possible values for variables and linear relations between variables – and ways to represent proof certificates compactly. A full list of their deliverables (including their published papers at the end of each deliverable) can be found here[21]. Based on Chapter 3 of Mobius deliverable[22] the *Source Specification in the Figure* is currently limited to Java Markup Language, and the Bytecode Specification is Bytecode Markup Language. The COSTA system is used to generate upper bounds for Java bytecode to verify compliance with resource certificate policies, and a number of novel analyses were added to COSTA as described in Chapter 5 of Mobius deliverable[23]. A number of case studies are presented, mostly MIDlets for the Java Micro Edition, evaluating the end-to-end behaviour of final software is given in Mobius deliverable[24].

[17] http://raweb.inria.fr/rapportsactivite/RA2006/lande/uid4.html
[18] http://software.imdea.org/∼gbarthe/mobius/bin/view/Mobius/WebHome.html
[19] in Coq https://coq.inria.fr/
[20] in Coq https://coq.inria.fr/
[21] http://software.imdea.org/∼gbarthe/mobius/bin/view/DeliverablesList/WebHome.html
[22] http://software.imdea.org/∼gbarthe/mobius/bin/view/DeliverablesList/D4.6.pdf
[23] http://software.imdea.org/∼gbarthe/mobius/bin/view/DeliverablesList/D2.7.pdf
[24] http://software.imdea.org/∼gbarthe/mobius/bin/view/DeliverablesList/D5.2.pdf

7.4 General programming languages

7.4.0.1 Hard Real-Time C Programs

An excellent survey paper covering the worst-case execution-time problem in hard real-time systems, its history and tooling is given in [WEE+08]. These tools all analyse C or some subset of C programs. The cited paper includes a discussion of the SWEET tool, which analyses an intermediate language called ALF (Artist Flow Analysis Language), and a number of translators, from other languages, including C and C++, to ALF exist. However on closer examination of the ALF language, support for variable stack unwinding, necessary for exception handling, is not currently part of the language – only support for popping a named/literal number of stack frames is supported and so it unclear whether, and if so how, support for exceptions could be built into this tool. Nor can SWEET handle triangular loops, as our analysis does.

7.4.0.2 CompCert Certified C compiler

INRIA has developed a certified C compiler[25] for most of ANSI C which as well as formally guaranteeing generation of executables whose behaviour is semantically equivalent to the C program is capable of producing formally correct/safe WCET estimates [MBPP14a].

7.4.0.3 The Java Optimized Processor (JOP) project

The Java Optimized Processor (JOP) project[26] tackles a central problem in hard real–time worst-case execution-time analysis – the growing complexity of processors which have been designed to make the *average* case fast but which have architectural features difficult to analyse for the worst-case, and which suffer from timing anomalies [RWT+06], causing unduly pessimistic WCET estimates – with a time-predictable[27] Java Virtual Machine built in hardware which has been designed from first principles to facilitate worst-case execution-time analysis and to be free from timing anomalies. JOP targets Java bytecode. In terms of this primary aim, the JOP project has been a massive success, is in use in a number of industrial applications, being used as the root of a number of different processors, and being used as a foundation by several other research projects. JOP has been extended to facilitate:

- WCET of chip-multiprocessors with shared memory [SAA+15, Sch10b];
- WCET of safety-critical Java [SDH+14a, HHL+09, SR12];

[25] http://compcert.inria.fr/ certification using the Coq proof assistant
[26] http://www.jopdesign.com/
[27] There are no time dependencies between bytecodes

– WCET of real-time Java [Sch04a, Sch04b];

JOP employs the IPET integer linear programming approach and obtains loop bounds using a data-flow analysis to automatically detect simple loop bounds. For loops for which it is not able to determine loop bounds automatically, JOP expects loop bounds to be annotated in the source Java program (JOP recovers these annotations using the bytecode engineering library). According to [SPPH10a], the data-flow analysis for JOP cannot handle exceptions or triangular loops (both of which our work is able to conservatively handle).

Should we ever want to target a real-time architecture, we would choose JOP, for the reasons outlined above, especially its timing predictability.

7.4.0.4 Operational Semantics and Bytecode Verification for the Java Virtual Machine Language

In composing our operational semantics for Carmel, our principal concerns were:

– correctness, including the modelling of novel aspects of the *Java Card* platform, e.g. the applet firewall;

– expression, including the use of auxiliary predicates, that most lends itself to being converted into a compositional verbose flow-logic analysis [NN02] without stratified constraints to simplify:

  * formulation of proofs;

  * implementation of a worklist algorithm to solve with equality all the constraints of the compositional verbose flow-logic analysis without stratified constraints;

These concerns have lead to clear, and well-structured, operational semantics which incorporate the many low-level details of the *Java Card* platform [Ora11c, Ora11b, Ora11a].

These concerns have also lead to the simplifying assumption of all Carmel code under consideration having been bytecode verified[28]. When we consider our operational semantics for Carmel wrt to the operational semantics, and indeed the type systems, of foundational papers around modelling full Java bytecode and resolving prevailing ambiguities and inconsistencies[29] of bytecode verification, we realise how much is owed to such pioneering research [SA99, Fre98, FM99, DEK99]. In particular, when one considers the elegance and compactness of [FM99, SA99], our operational semantics seem bloated and ugly in comparison. Our

[28] As outlined in Section 1.3.

[29] or perhaps more clearly, specifying the correctness

operational semantics has great fidelity to the *Java Card* platform, and makes it is easy to see the different conditions that lead to different transitions, particularly around the novel applet firewall rules.

7.5 Evaluation of the loop-bound precision of our tool wrt SWEET, JOP and COSTA

From our literature survey, we have identified three tools which are most like our timing tool in terms of their primary focus being the determination of loop-bounds for input programs: SWEET, COSTA and JOP. We would like to evaluate the relative precision of the loop-bounds of each of these tools with our own[30].

COSTA provides much more sophisticated/comprehensive loop-bound analysis than our tool. In addition, and unlike JOP and SWEET, COSTA – like our tool – is able to provide resource estimates for code that may cause exceptions to be thrown [AAG+07]. Since COSTA operates on Java bytecode, some customisation to be able to accurately analyse *Java Card* applet bytecode would be required. Whether the program structure of commercial *Java Card* applet code is complex enough to benefit from COSTA, and so to justify the time / development effort of specialising COSTA to the *Java Card* platform, is unclear. It might make more sense to strengthen our tool's ability to handle more complex algorithms and loop-conditions, as explored tentatively with encouraging results in Section 6.7. While objectionable in principle to the current writer, a hybrid/augmenting approach of using COSTA to provide potential loop-bounds to our tool, would also be possible, leveraging the loop-bounds analysis strengths of COSTA, to provide information to our tool, which it can consult according to its better understanding of reachability/control-flow and data-flow of *Java Card* applet bytecode.

Section 6.7 provides a detailed cross-comparison of our timing tool and SWEET wrt the Mälardalen worst-case execution-time benchmarks[31] [GBEL10]. In terms of programs based on array processing, our tool and SWEET were of equal standing. SWEET is able to handle recursion and is able to provide loop-bounds for more complex programs than our tool. However, as is shown in Section 6.7, our tool shows great promise of being able to be developed to provide tight loop-bounds for more complex programs. SWEET is based on

[30] *NB* all three tools are more mature tools than ours, and in particular have benefitted from more time and more developers being poured into them than our tool.

[31] http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

analysing ALF programs, and significant effort would be required to customise it to the *Java Card* platform. Since, as outlined in Section 7.4.0.1 above, it does not look possible for SWEET to handle variable stack unwinding, it does not look possible for SWEET to be able to handle Java style exception handling, and so I do not believe it worth the time or effort to extend SWEET to handle *Java Card* applets.

JOP has been a massive success in its intention to produce a timing-predictable / timing-anomaly-free Java processor, as outlined in Section 7.4.0.3. However, the dataflow analysis underlying JOP's worst-case execution-time analysis is weak, and JOP typically relies on manual annotations of worst-case / maximum loop-bounds in the Java code to be made available in the Java bytecode to complete analysis. Table 7.5 shows such annotations. Further, JOP cannot handle exceptions or triangular loops (both of which our work is able to conservatively handle). The inability to handle exceptions is a particular weakness for a tool targetting an object-oriented language, as it is very easy for buggy code to hide in rarely-executed `catch` statements, with the real worst-case resource-usage path tied to exceptional flows. Direct comparison between the JOP dataflow and our tool is further complicated by the JOP benchmarks of Section 6.2 of [Sch09] being full Java applications including IP/UDP libraries, and so our tool cannot analyse these. JOP also performs its own cross-comparison with the Mälardalen worst-case execution-time benchmarks, but only gives its final measured cycle-time which includes the cost of memory-cache-misses and different weightings for different instructions, making it next to impossible to compare like-for-like with our tool. For the simple running example/method given in Listing 6.1 of [Sch09], reproduced for convenience in Table 7.5, our tool matches the accuracy of the loop-bounds JOP determines. The only difference is that we have to call the method twice with different `boolean` arguments to explore all the paths. This implies our tool is more accurate than JOP, since we never "guess" the values of method arguments.

Table 7.1: Reproduction of [Listing 6.1: The example used for WCET analysis] from [Sch09]

```
public static int loop(boolean b, int val ) {

    int i , j ;
    for ( i=0; i<10; ++i) {        // @WCA loop=10
        if (b) {
            for ( j=0; j<3; ++j) { // @WCA loop=3
               val *= val;
            }
        } else {
            for ( j=0; j<4; ++j) { // @WCA loop=4
                   val += val;
            }
        }
    }

    return val ;
}
```

Chapter 8

Conclusions and further work

8.1 Introduction

In this Chapter, we review the main achievements of this thesis, provide the expected scope for future work, and reach a conclusion on whether the thesis statement has been proved.

8.2 Main Achievements of this Thesis

The main achievement of this thesis has been the rigorous, systematic and dogged engagement with the complexities of a real-world, commercially important/successful platform like the *Java Card* Classic Edition 3.0.1 and its bytecode language[1] which we have rationalised into the Carmel language and its associated programming structures. From this systematic engagement:

- Our first major achievements were the product of thorough immersion in the *Java Card* platform specification 3.0.1 [Ora11c, Ora11b, Ora11a] and the Sun official *Java Card* book [Che00], and the development of:
  - high-level arguments, detailed in Section 1.1.2, as to why we might expect *Java Card* Classic Editions to be amenable to worst-case resource-usage analysis, based on architecture, programming-conventions and ultimately the limited resources of a typical smartcard;
  - the existing SecSafe operational semantics and program analyses to better support loop-bounds calculation and detect potentially recursive method calls, whilst integrating full transaction- and

[1] A utility developed as part of the SecSafe project is capable of generating equivalent Carmel from a compressed applet (CAP) file.

APDU I/O- semantics. These translate into the operational semantics of Chapter 3 and the base control-flow analysis of Chapter 4;

  - a worklist algorithm capable of generating the least solution to the base control-flow analysis of Chapter 4.

- Development of the extended control-flow analysis of Chapter 5 from the base control-flow analysis of Chapter 4 to better handle loop-variables and discard non-reaching local variable definitions[2] via the refinement of the `if` statements from the base control-flow analysis integrated with the results of novel analogues of classical loop-induction analyses and reaching definitions analyses, with a corresponding change to the handling of the `if` statements in the worklist algorithm of Chapter 4. Our extended control-flow analysis is the most precise static analysis of *Java Card* bytecode of which we are aware;
- Implementation in Java in a tool which we have named Fulgurite, for a Carmel program P:
  - the worklist algorithm for computing the least solution to the constraints of the extended control-flow analysis of P;
  - the generation of a family of integer linear programming problems per registered applet in P and their solution via the open-source mixed-integer linear-programming solver `lpsolve`[3];
  - the production of DOT control-flow/resource graphs from P;
- Use of Fulgurite to analyse:
  - whole- and key fragments- of various applets from Oracle's *Java Card* development kits;
  - assorted loops and recursive methods to various depths, translated from Java classes to Carmel via `javap`, `sed` and manual translation;

  from which we have verified Fulgurite/the extended control-flow analysis does indeed bound simple loops over arrays using their `arraylength`, or bounded loops from numeric literals, and triangular loops of various depths, all successfully;
- Development of the formulation of the integer-linear programming problem from [PS97, Sch09] to Carmel to include exceptional flows and inclusion of constraints ensuring exactly one return path to the *Java Card* Runtime Environment, whether via a normal/exception-free path or abnormal/uncaught exception path. Utilisation of exception behaviour, and the ability to handle triangular loops makes our worst-case resource-usage tool superior to other more mature known worst-case execution-time tools e.g. JOP and SWEET. We can still learn from other tools, such as COSTA, which analyses full Java bytecode, on

[2] See [NNH10] and Appendix C and for further information on reaching definitions, non-reaching definitions and reaching definitions for local variables in Carmel.

[3] http://lpsolve.sourceforge.net/5.5/

how to analyse more complex loops. Further analysis of commercial applet code would be needed to determine whether applet code control-flow/data-flow/loop-conditions are complex enough to be able to benefit from COSTA's more advanced loop analysis techniques, and so to justify the effort to transplant these to Carmel.

8.3 Future Work

There are a number of different directions in which the work in this thesis could be developed:

1. The material in this thesis could be updated in line with the latest version of *Java Card* (at time of writing, this is 3.0.5). Having reviewed the *Java Card* platform specification 3.0.5, no changes to the *Java Card* Runtime Environment or *Java Card* Virtual Machine have been made since v 2.2.2[4]. The changes are all in the *Java Card* Application Programmer Interface (API), the developer libraries.

   The hard part is the modelling of each new API method as an operational semantic rule, its translation into a flow-logic clause and completion of the subject reduction theorem proof of correctness. Adding the Java implementation of the new abstract API methods to Fulgurite is straight-forward, the method name is added to the set of API methods `SEPARATELY_HANDLED_API_METHODS` and the implementation is added to a `case` of a `switch` statement against the method name of the API name, next to the implementation of the other intercepted API methods;
2. Our static analysis could be extended to handle more complex arithmetic expressions. In particular, our static analysis could be extended to handle more complex arithmetic expressions forming part of the loop conditions of Carmel programs. Initial efforts in this regard, as recorded in Section 6.7, have proved very encouraging;
3. While we have a high degree of confidence in the correctness of our manual proofs and in the correctness of Fulgurite, our implementation of the worklist algorithm capable of solving the constraints of the extended control-flow analysis of Chapter 5, we would prefer the greater assurance of machine-checked proofs via e.g. a certified theorem prover or model-checking. To this end, given that:
   - certified defensive and offensive *Java Card* virtual machines have been developed in Coq as part of [BDdS$^+$02, G. , BDJ$^+$01];
   - a certified proof of the correctness of the worst-case execution-time estimates for the CompCert C compiler has also been developed in Coq as part of [BMP13, MBPP14a];

[4] Confirmed by http://www.oracle.com/technetwork/java/embedded/javacard/downloads/default-1970005.html

it would seem both feasible and appropriate to implement in Coq a certified proof of the correctness of the worst-case resource-usage analysis estimates for Carmel. Further supporting this, it is reported in [CJPR04] that the expression of a flow-logic analysis lends itself to being encoded in Coq. From the certified proof, Coq supports the "proof-as-program" extraction facility [Let08] into O'Caml, Scheme and Haskell;

4. Fulgurite could be developed into a plugin for Eclipse for developers to use while developing *Java Card* applications for immediate feedback on analysability and worst-case resource-usage, or developed into a plugin for a continuous build system such as Jenkins or Hudson to allow per-build and series analysability and worst-case resource-usage data to be collected.

8.4 Thesis statement evaluation

From Section 1.1, our formal thesis statement was:

"The *Java Card* Classic edition platform is worst-case resource-usage analysis-*friendly*. In particular, the *Java Card* platform is *highly amenable* to worst-case execution-time analysis and worst-case dynamic memory allocation analysis through a combination of techniques from the fields of static analysis and integer linear programming".

I believe we have proved our thesis statement emphatically, as the achievements summarised in this Chapter attest.

Bibliography

[AAG+07] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Costa: Design and implementation of a cost and termination analyzer for java bytecode. In *of Lecture Notes in Computer Science*, pages 113–132. Springer, 2007.

[AAGP08] Elvira Albert, Puri Arenas, Samir Genaim, and Germn Puebla. Automatic inference of upper bounds for recurrence relations in cost analysis. In Mara Alpuente and Germn Vidal, editors, *Static Analysis*, volume 5079 of *Lecture Notes in Computer Science*, pages 221–237. Springer Berlin Heidelberg, 2008.

[AGB07] Nadia Bel Hadj Aissa, Gilles Grimaud, and Vincent Benony. Bringing worst case execution time awareness to an open smart card OS. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*. Institute of Electrical & Electronics Engineers (IEEE), aug 2007.

[AGH+04] David Aspinall, Stephen Gilmore, Martin Hofmann, Donald Sannella, and Ian Stark. Mobile resource guarantees for smart devices. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, International Workshop, CASSIS 2004, Marseille, France, March 10-14, 2004, Revised Selected Papers*, pages 1–26, 2004.

[AMM14] RajaNaeem Akram, Konstantinos Markantonakis, and Keith Mayes. An introduction to java card programming. In Konstantinos Markantonakis and Keith Mayes, editors, *Secure Smart Embedded Devices, Platforms and Applications*, pages 497–513. Springer New York, 2014.

[BCP09] Samia Bouzefrane, Julien Cordry, and Pierre Paradinas. MESURE tool to benchmark java card platforms. *CoRR*, abs/0909.2103, 2009.

[BDdS+02] Gilles Barthe, Guillaume Dufay, Simo Melo de Sousa, Pierre Courtieu, and Daniel Le Metayer. Verificard. 1999-2002.

[BDJ+01] G. Barthe, G. Dufay, L. Jakubiec, B. Serpette, and S. Melo de Sousa. A Formal Executable Semantics of the JavaCard Platform. In D. Sands, editor, *Proceedings of ESOP'01*, volume 2028, pages 302–319, 2001.

[BFH+13] Thomas Bogholm, Christian Frost, René Rydhof Hansen, Casper Svenning Jensen, Kasper Soe Luckow, Anders P. Ravn, Hans Sondergaard, and Bent Thomsen. Towards harnessing theories through tool support for hard real-time java programming. *Innovations in Systems and Software Engineering*, 9(1):17–28, 2013.

[BG02] Gabriel Bizzotto and Gilles Grimaud. Practical java card bytecode compression, 2002.

[BG14] Guillaume Barbu and Christophe Giraud. New countermeasures against fault and software type confusion attacks on java cards. In David Naccache and Damien Sauveron, editors, *Information Security Theory and Practice. Securing the Internet of Things*, volume 8501 of *Lecture Notes in Computer Science*, pages 57–75. Springer Berlin Heidelberg, 2014.

[BICL11] Guillaume Bouffard, Julien Iguchi-Cartigny, and Jean-Louis Lanet. Combined software and hardware attacks on the java card control flow. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications*, volume 7079 of *Lecture Notes in Computer Science*, pages 283–296. Springer Berlin Heidelberg, 2011.

[BJP06a] Frdric Besson, Thomas Jensen, and David Pichardie. A pcc architecture based on certified abstract interpretation, 2006.

[BJP06b] Frédéric Besson, Thomas Jensen, and David Pichardie. Proof-carrying code from certified abstract interpretation and fixpoint compression. *Theoretical Computer Science*, 364(3):273–291, nov 2006.

[BJV14] Frdric Besson, Thomas Jensen, and Pierre Vittet. Sawjacard: A static analysis tool for certifying java card applications. In Markus Mller-Olm and Helmut Seidl, editors, *Static Analysis*, volume 8723 of *Lecture Notes in Computer Science*, pages 51–67. Springer International Publishing, 2014.

[BLLL15] Guillaume Bouffard, Michael Lackner, Jean-Louis Lanet, and Johannes Loinig. Heap . . . hop! heap is also vulnerable. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications*, volume 8968 of *Lecture Notes in Computer Science*, pages 18–31. Springer International Publishing, 2015.

[BMP13] Sandrine Blazy, André Maroneze, and David Pichardie. Formal verification of loop bound estimation for WCET analysis. In *Proc. of the 5th Working Conference on Verified Software: Theories,*

Tools, and Experiments (VSTTE 2013), volume 8164 of *Lecture Notes in Computer Science*, pages 281–303. Springer-Verlag, 2013.

[BTL13a] Guillaume Bouffard, BhagyalekshmyN. Thampi, and Jean-Louis Lanet. Detecting laser fault injection for smart cards using security automata. In SabuM. Thampi, PradeepK. Atrey, Chun-I Fan, and GregorioMartinez Perez, editors, *Security in Computing and Communications*, volume 377 of *Communications in Computer and Information Science*, pages 18–29. Springer Berlin Heidelberg, 2013.

[BTL13b] Guillaume Bouffard, BhagyalekshmyN. Thampi, and Jean-Louis Lanet. Vulnerability analysis on smart cards using fault tree. In Friedemann Bitsch, Jrmie Guiochet, and Mohamed Kaniche, editors, *Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 82–93. Springer Berlin Heidelberg, 2013.

[BVGV14] Pablo Basanta-Val and Marisol Garca-Valls. A simple distributed garbage collector for distributed real-time java. *The Journal of Supercomputing*, 70(3):1588–1616, 2014.

[Che00] Zhiqun Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[CJPR04] David Cachera, Thomas Jensen, David Pichardie, and Vlad Rusu. *Programming Languages and Systems: 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004. Proceedings*, chapter Extracting a Data Flow Analyser in Constructive Logic, pages 385–400. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[CJPS05] David Cachera, Thomas Jensen, David Pichardie, and Gerardo Schneider. Certified Memory Usage Analysis. In *Procóf 13th International Symposium on Formal Methods (FM'05)*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[CnW09] Néstor Cataño and Tim Wahls. Executing jml specifications of java card applications: A case study. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 404–408, New York, NY, USA, 2009. ACM.

[Cor09] Julien Cordry. *La mesure de performance dans les cartes à puce*. Theses, Conservatoire national des arts et metiers - CNAM, November 2009.

[DEK99] Sophia Drossopoulou, Susan Eisenbach, and Sarfraz Khurshid. The Java Type System is Sound. volume 5, pages 3–24, October 1999.

[DHW03] Pierro A Di, C Hankin, and H Wiklicky. Quantitative relations and approximate process equivalences. pages 508–522, 2003.

[DP02] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[FM99] Stephen N. Freund and John C. Mitchell. A formal framework for the java bytecode language and verifier. *SIGPLAN Not.*, 34(10):147–166, October 1999.

[Fre98] Stephen N. Freund. The costs and benefits of java bytecode subroutines. In *In Formal Underpinnings of Java Workshop at OOPSLA*, 1998.

[G.] url = http://www-sop.inria.fr/lemme/verificard/certicartes/ year = 1999-2002 G. Barthe and G. Dufay and L. Jakubiec and B. Serpette and S. Melo de Sousa, title = CertiCartes.

[GBEL10] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET benchmarks – past, present and future. pages 137–147, Brussels, Belgium, July 2010. OCG.

[GBS06] G. Le Guernic, A. Banerjee, and D. Schmidt. Automaton-based non-interference monitoring. Technical Report 2006-1, Department of Computing and Information Sciences, College of Engineering, Kansas State University, 234 Nichols Hall, Manhattan, KS 66506, USA, April 2006.

[GE98] Jan Gustafsson and Andreas Ermedahl. Automatic derivation of path and loop annotations in object-oriented real-time programs. *Journal of Parallel and Distributed Computing Practices*, 1, 1998.

[GESL06] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper. Automatic derivation of loop bounds and infeasible paths for wcet analysis using abstract execution. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 57–66, Dec 2006.

[GS05] Pablo Giambiagi and Gerardo Schneider. Memory consumption analysis of java smart cards. 2005.

[Gus] Jan Gustafsson. *SWEET Reference Handbook*. Available at `http://www.mrtc.mdh.se/projects/wcet/sweet/manual/SWEET_manual.pdf`.

[Han05] R. R. Hansen. *Flow logic for language-based safety and security*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2005. Supervised by Prof. Flemming Nielson.

[HHL+09] Thomas Henties, James J. Hunt, Doug Locke, Kelvin Nilsen, Martin Schoeberl, and Jan Vitek. Java for safety-critical applications. In *2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems (SafeCert 2009)*, 2009.

[HK07] Trevor Harmon and Raymond Klefstad. A survey of worst-case execution time analysis for real-time java. In *2007 IEEE International Parallel and Distributed Processing Symposium*. Institute of Electrical & Electronics Engineers (IEEE), 2007.

[HMP06] Engelbert Hubbers, Wojciech Mostowski, and Erik Poll. Tearing java cards. In *IN PROCEEDINGS, E-SMART 2006, SOPHIA-ANTIPOLIS*, 2006.

[HS05] René Rydhof Hansen and Igor A. Siveroni. Towards verification of well-formed transactions in Java Card bytecode. *Electron. Notes Theor. Comput. Sci.*, 141(1):145–162, December 2005.

[HSN+03] Chris Hankin, Igor Siveroni, Flemming Nielson, Hanne Nielson, and Rene Rydof Hansen. Secsafe. 2000-2003.

[HSST06] James J. Hunt, Fridtjof B. Siebert, Peter H. Schmitt, and Isabel Tonin. Provably correct loops bounds for realtime java programs. In *Proceedings of the 4th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '06, pages 162–169, New York, NY, USA, 2006. ACM.

[INR06] INRIA. The castles rntl project (http://raweb.inria.fr/rapportsactivite/ra2006/lande/uid44.html). 2006.

[JC97] Johan Janssen and Henk Corporaal. Making graphs reducible with controlled node splitting. *ACM Trans. Program. Lang. Syst.*, 19(6):1031–1052, November 1997.

[JJ05] Min-Sik Jin and Min-Soo Jung. A study on how to reduce time and space by redefining new bytecode for java card. In *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*, pages 551–554, Aug 2005.

[KKP+11] Raimund Kirner, Jens Knoop, Adrian Prantl, Markus Schordan, and Albrecht Kadlec. Beyond loop bounds: comparing annotation languages for worst-case execution time analysis. *Software Systems Modeling*, 10(3):411–437, 2011.

[LBL+13] Michael Lackner, Reinhard Berlach, Johannes Loinig, Reinhold Weiss, and Christian Steger. Towards the hardware accelerated defensive virtual machine type and bound protection. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications*, volume 7771 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2013.

[LBL+15] Jean-Louis Lanet, Guillaume Bouffard, Rokia Lamrani, Ranim Chakra, Afef Mestiri, Mohammed Monsif, and Abdellatif Fandi. Memory forensics of a java card dump. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications*, volume 8968 of *Lecture Notes in Computer Science*, pages 3–17. Springer International Publishing, 2015.

[LBR+13] Michael Lackner, Reinhard Berlach, Wolfgang Raschke, Reinhold Weiss, and Christian Steger. A defensive virtual machine layer to counteract fault attacks on java cards. In Lorenzo Cavallaro and Dieter Gollmann, editors, *Information Security Theory and Practice. Security of Mobile and Cyber-Physical Systems*, volume 7886 of *Lecture Notes in Computer Science*, pages 82–97. Springer Berlin Heidelberg, 2013.

[Let08] Pierre Letouzey. Extraction in coq: An overview. In *Logic and Theory of Algorithms*, pages 359–369. Springer, 2008.

[LGZ+09] Mingsong Lv, Nan Guan, Yi Zhang, Qingxu Deng, Ge Yu, and Jianming Zhang. A survey of WCET analysis of real-time operating systems. In *2009 International Conference on Embedded Software and Systems*. Institute of Electrical & Electronics Engineers (IEEE), 2009.

[LM95] Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the 32Nd Annual ACM/IEEE Design Automation Conference*, DAC '95, pages 456–461, New York, NY, USA, 1995. ACM.

[LSWH09] J. Loinig, C. Steger, R. Weiss, and E. Haselsteiner. Java card performance optimization of secure transaction atomicity based on increasing the class field locality. In *Secure Software Integration and Reliability Improvement, 2009. SSIRI 2009. Third IEEE International Conference on*, pages 342–347, July 2009.

[LT79] Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, January 1979.

[Mar01] Renaud Marlet. Syntax of the jcvm language to be studied in the secsafe project. May 2001.

[MBPP14a] André Maroneze, Sandrine Blazy, David Pichardie, and Isabelle Puaut. A Formally Verified WCET Estimation Tool. In Heiko Falk, editor, *14th International Workshop on Worst-Case Execution Time Analysis*, volume 39 of *OpenAccess Series in Informatics (OASIcs)*, pages 11–20, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[MBPP14b] André Maroneze, Sandrine Blazy, David Pichardie, and Isabelle Puaut. A formally verified wcet estimation tool. In *14th International Workshop on Worst-Case Execution Time Analysis, WCET 2014, July 8, 2014, Ulm, Germany*, volume 39, pages 11–20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[MM01] Renaud Marlet and Daniel Le Metayer. First note on industrial needs in terms of language, api and security properties of java card. Technical report, Trusted Logic, 2001.

[Mos07] Wojciech Mostowski. From sequential java to java card. In Bernhard Beckert, Reiner Hhnle, and PeterH. Schmitt, editors, *Verification of Object-Oriented Software. The KeY Approach*, volume 4334 of *Lecture Notes in Computer Science*, pages 375–405. Springer Berlin Heidelberg, 2007.

[Nec97] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '97, pages 106–119, New York, NY, USA, 1997. ACM.

[NN02] HanneRiis Nielson and Flemming Nielson. Flow logic: A multi-paradigmatic approach to static analysis. In Torben. Mogensen, DavidA. Schmidt, and I.Hal Sudborough, editors, *The Essence of Computation*, volume 2566 of *Lecture Notes in Computer Science*, pages 223–244. Springer Berlin Heidelberg, 2002.

[NNH10] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Publishing Company, Incorporated, 2010.

[Ora11a] *Application Programming Interface for the Java Card Platform, Classic Edition*. Oracle, September 2011.

[Ora11b] *Runtime Environment Specification for the Java Card Platform, Classic Edition*. Oracle, September 2011.

[Ora11c] *Virtual Machine Specification for the Java Card Platform, Classic Edition*. Oracle, September 2011.

[PCB09] Pierre Paradinas, Julien Cordry, and Samia Bouzefrane. Measurement analysis when benchmarking java card platforms. In Olivier Markowitch, Angelos Bilas, Jaap-Henk Hoepman, ChrisJ. Mitchell, and Jean-Jacques Quisquater, editors, *Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, volume 5746 of *Lecture Notes in Computer Science*, pages 84–94. Springer Berlin Heidelberg, 2009.

[PHS10] Wolfgang Puffitsch, Benedikt Huber, and Martin Schoeberl. Worst-case analysis of heap allocations. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *Lecture Notes in Computer Science*, pages 464–478. Springer Berlin Heidelberg, 2010.

[Pin10] Keshav Pingali. *Loop Optimizations*. University of Texas, 2010. Available at `http://www.cs.utexas.edu/~pingali/CS375/2010Sp/lectures/LoopOptimizations.pdf`.

[Pro02] Project IST-1999-29075. Final report of the secsafe (secure and safe systems based on static analysis) project. 2002.

[PS97] PeterP. Puschner and AntonV. Schedl. Computing maximum task execution times a graph-based approach. *Real-Time Systems*, 13(1):67–91, 1997.

[PTT08] Tuan-Hung Pham, Anh-Hoang Truong, and Ninh-Thuan Truong. Computing heap space cost of java card applets. In Akim Demaille, Tru Cao, and Bao Ho, editors, *Contributions to the 2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing & Communication Technologies*, page 190–196, University of Science –- Vietnam National University Ho Chi Minh City, July 2008.

[PTTC08] Tuan-Hung Pham, Anh-Hoang Truong, Ninh-Thuan Truong, and Wei-Ngan Chin. A fast algorithm to compute heap memory bounds of java card applets. In *Software Engineering and Formal Methods, 2008. SEFM '08. Sixth IEEE International Conference on*, pages 259–267, Nov 2008.

[RWT+06] Jan Reineke, Bjrn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A definition and classification of timing anomalies. In *6th Intl Workshop on Worst-Case Execution Time (WCET) Analysis*, 2006.

[SA99] Raymie Stata and Martin Abadi. A type system for java bytecode subroutines. *ACM Trans. Program. Lang. Syst.*, 21(1):90–137, January 1999.

[SAA+15] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, Andr Rocha, Cludio Silva, Jens Spars, and Alessandro Tocchi. T-crest: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, (0):accepted for publication, 2015.

[Sch03] Martin Schoeberl. JOP: A Java Optimized Processor. In Robert Meersman and Zahir Tari, editors, *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, volume 2889 of *Lecture Notes in Computer Science*, pages 346–359. Springer Berlin Heidelberg, 2003.

[Sch04a] Martin Schoeberl. Design rationale of a processor architecture for predictable real-time execution of java programs. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2004)*, 2004.

[Sch04b] Martin Schoeberl. Restrictions of java for embedded real-time systems. In *Proceedings of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2004)*, pages 93–100. IEEE, 2004.

[Sch09] Martin Schoeberl. *JOP Reference Handbook*. Number ISBN 978-1438239699. CreateSpace, 2009. Available at `http://www.jopdesign.com/doc/handbook.pdf`.

[Sch10a] Martin Schoeberl. Scheduling of hard real-time garbage collection. *Real-Time Systems*, 45(3):176–213, 2010.

[Sch10b] Martin Schoeberl. Time-predictable chip-multiprocessor design. In *Asilomar Conference on Signals, Systems, and Computers*, 2010.

[SCJ06] Pascal Sotin, David Cachera, and Thomas Jensen. Quantitative Static Analysis over semirings: analysing cache behaviour for Java Card. In *Quantitative Aspects of Programming Languages*, 2006.

[SDH+14a] Martin Schoeberl, Andreas Engelbredt Dalsgaard, Ren Rydhof Hansen, Stephan E. Korsholm, Anders P. Ravn, Juan Ricardo Rios Rivas, Trur Biskopst Strm, and Hans Sndergaard. Certifiable java for embedded systems. In *Proceedings of the 12th International Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2014)*, pages 10–19. ACM, 2014.

[SDH+14b] Martin Schoeberl, Andreas Engelbredt Dalsgaard, René Rydhof Hansen, Stephan E. Korsholm, Anders P. Ravn, Juan Ricardo Rios Rivas, Tórur Biskopstø Strøm, and Hans Søndergaard. Certifiable java for embedded systems. In *Proceedings of the 12th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '14, pages 10:10–10:19, New York, NY, USA, 2014. ACM.

[SFL13] Aymerick Savary, Marc Frappier, and Jean-Louis Lanet. Detecting vulnerabilities in java-card bytecode verifiers using model-based testing. In EinarBroch Johnsen and Luigia Petre, editors, *Integrated Formal Methods*, volume 7940 of *Lecture Notes in Computer Science*, pages 223–237. Springer Berlin Heidelberg, 2013.

[SH01] Igor Siveroni and Chris Hankin. A proposal for the JCVMLe operational semantics. February 17 2001.

[Siv04] Igor A. Siveroni. Operational Semantics of the Java Card Virtual Machine. *The Journal of Logic and Algebraic Programming*, 58(1-2):3–25, Jan 2004.

[SPK09] Martin Schoeberl, Peter Puschner, and Raimund Kirner. A single-path chip-multiprocessor system. In Sunggu Lee and Priya Narasimhan, editors, *Software Technologies for Embedded and Ubiquitous Systems*, volume 5860 of *Lecture Notes in Computer Science*, pages 47–57. Springer Berlin Heidelberg, 2009.

[SPPH10a] Martin Schoeberl, Wolfgang Puffitsch, Rasmus Ulslev Pedersen, and Benedikt Huber. Worst-case execution time analysis for a java processor. *Software: Practice and Experience*, 40/6:507–542, 2010.

[SPPH10b] Martin Schoeberl, Wolfgang Puffitsch, Rasmus Ulslev Pedersen, and Benedikt Huber. Worst-case execution time analysis for a java processor. *Software: Practice and Experience*, 40(6):507–542, 2010.

[SR12] Martin Schoeberl and Juan Ricardo Rios. Safety-critical java on a java processor. In *Proceedings of the 10th International Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2012)*, pages 54–61. ACM, 2012.

[The06] The Common Criteria Recognition Agreement Members. Common Criteria for Information Technology Security Evaluation. September 2006.

[VHU92] Jan Vitek, R. Nigel Horspool, and James S. Uhl. *Compiler Construction: 4th International Conference, CC '92 Paderborn, FRG, October 5–7, 1992 Proceedings*, chapter Compile-time analysis of object-oriented programs, pages 236–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 1992.

[WEE+08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[Wil14] Reinhard Wilhelm. Software quality assurance by static program analysis. In Dietmar Winkler, Stefan Biffl, and Johannes Bergsmann, editors, *Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering*, volume 166 of *Lecture Notes in Business Information Processing*, pages 1–11. Springer International Publishing, 2014.

[ZCS15] Maciej Zielenkiewicz, Jacek Chrzszcz, and Aleksy Schubert. Java loops are mainly polynomial. In GiuseppeF. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorn, Jean-Jacques Quisquater, and Roger Wattenhofer, editors, *SOFSEM 2015: Theory and Practice of Computer Science*, volume 8939 of *Lecture Notes in Computer Science*, pages 603–614. Springer Berlin Heidelberg, 2015.

[Zha] Jianjun Zhao. Analyzing Control Flow in Java Bytecode.

[ZK15] Michael Zolda and Raimund Kirner. Calculating wcet estimates from timed traces. *Real-Time Systems*, pages 1–50, 2015.

Appendix A

API methods

A.1 Introduction

In this Chapter we present for each of the key *Java Card* API methods:

- the operational semantic rule;
- the corresponding flow-logic clause;
- proof of the subject reduction theorem[1] for that API method.

Due to time-constraints we have not been able to type up the complete set of *Java Card* API methods for which we have written proofs, though we can supply 'photocopies on request. All of the supported API methods have been implemented in our worklist solver.

[1] See Theorem B.4.1 on page 320 for the Subject Reduction Theorem for the base control-flow analysis.

A.2 Transaction Methods

A.2.1 Case API method `javacard.framework.JCSystem.getTransactionDepth()`:

A.2.1.1 Operational Semantic rule

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p \\ p = \texttt{javacard.framework.JCSystem.getTransactionDepth}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ SF = \text{«}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{»} :: \text{«}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{»} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n \rangle \\ SF' = \text{«}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{»} :: \text{«}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{»} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), \mathit{L}_n, S \rangle \\ S = \mathit{S}_n :: (s, ctd_n, (\mathit{m}_n, pc_n)) \end{array}}{P \left| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF \rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF' \rangle \end{array} \right.}$$

A.2.1.2 Flow-Logic clause

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (\mathit{m}_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (\mathit{m}_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (\mathit{m}_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (\mathit{m}_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (\mathit{m}_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))))$$

p	$=$	`javacard.framework.JCSystem.getTransactionDepth()`
$p.isStatic$		
$p \in$ `SEPARATELY_HANDLED_API_METHODS`		
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1) :: \{(\mathtt{s}, (\gamma_n, \gamma_n, 0), (\mathit{m}_n, pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

A.2.1.3 Subject Reduction Theorem

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p \\ p = \texttt{javacard.framework.JCSystem.getTransactionDepth}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \end{array}}{{}_P\Big|\ \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = {}_{\langle\!\langle \mathit{loc}_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1,\, pc_1),\, L_1,\, S_1\rangle\!\rangle}::{}_{\langle\!\langle \mathit{loc}_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2,\, pc_2),\, L_2,\, S_2\rangle\!\rangle}::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{s}_n\rangle$$

$$SF' = {}_{\langle\!\langle \mathit{loc}_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1,\, pc_1),\, L_1,\, S_1\rangle\!\rangle}::{}_{\langle\!\langle \mathit{loc}_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2,\, pc_2),\, L_2,\, S_2\rangle\!\rangle}::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), \mathit{L}_n, S\rangle$$

$$S = \mathit{S}_n::(s, ctd_n, (\mathit{m}_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for this API method the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n :: (s, ctd_n, (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \forall\ \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

p	$=$	`javacard.framework.JCSystem.getTransactionDepth()`
$p.isStatic$		
$p \in$ `SEPARATELY_HANDLED_API_METHODS`		
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1) :: \{(\mathtt{s}, (\gamma_n, \gamma_n, 0), (m_n, pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

By definition of $\beta^{R,H,JH,k}_{Context}$, $ctd_n = \gamma_n \ \Rightarrow\ \beta_{Num}(s, ctd_n, (m_n, pc_n)) \sqsubseteq_{Val} \{(\mathtt{s}, (\gamma_n, \gamma_n, 0), (m_n, pc_n))\}$.

Combining terms:

$$\begin{aligned}
\beta^{R,H,JH}_{Stack}(S) &= \beta^{R,H,JH}_{Stack}(S_n{::}(s, ctd_n, (\mathit{m}_n, pc_n))) \\
&\sqsubseteq_S \beta^{R,H,JH}_{Stack}(S_n){::}\beta_{Num}(s, ctd_n, (\mathit{m}_n, pc_n))) \\
&\sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1){::}\{(\mathtt{s}, (\gamma_n, \gamma_n, 0), (\mathit{m}_n, pc_n))\} \\
&\sqsubseteq_S \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{aligned}$$

and the result follows.

A.2.2 Case API method `javacard.framework.JCSystem.getUnusedCommitCapacity()`:

A.2.2.1 Operational Semantic rule

$\mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p$

$p = \texttt{javacard.framework.JCSystem.getUnusedCommitCapacity}()$

$p.isStatic$

$0 \leq \texttt{CARD_SPECIFIC_UNUSED_CAPACITY} \leq \texttt{Short.MAX_VALUE}$

$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$

$SF = {}_{\langle\!\langle}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1{}_{\rangle\!\rangle} :: {}_{\langle\!\langle}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2{}_{\rangle\!\rangle} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), L_n, S_n \rangle$

$SF' = {}_{\langle\!\langle}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1{}_{\rangle\!\rangle} :: {}_{\langle\!\langle}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2{}_{\rangle\!\rangle} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$

$S = \mathit{S}_n{::}(\mathtt{s}, \texttt{CARD_SPECIFIC_UNUSED_CAPACITY}, (\mathit{m}_n, pc_n))$

$$P \Big| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF \rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF' \rangle \end{array}$$

A.2.2.2 Flow-Logic clause

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite } p$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

p	$=$	`javacard.framework.JCSystem.getUnusedCommitCapacity()`
$p.isStatic$		
$p \in$ `SEPARATELY_HANDLED_API_METHODS`		
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)::\{(\texttt{s}, (0, \texttt{Short.MAX_VALUE}, 0), (m_n, pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

A.2.2.3 Subject Reduction Theorem

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite } p \\ p = \texttt{javacard.framework.JCSystem.getUnusedCommitCapacity}() \\ p.isStatic \\ 0 \leq \texttt{CARD_SPECIFIC_UNUSED_CAPACITY} \leq \texttt{Short.MAX_VALUE} \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \end{array}}{P \left| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF \rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF' \rangle \end{array} \right.}$$

whence:

$$SF = \text{«}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{»} :: \text{«}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{»} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$SF' = \text{«}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{»} :: \text{«}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{»} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$$

$$S = S_n::(\texttt{s}, \texttt{CARD_SPECIFIC_UNUSED_CAPACITY}, (m_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we

have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lll}
\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for this API method the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n{::}(\texttt{s}, \texttt{CARD_SPECIFIC_UNUSED_CAPACITY}, (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite } p$$

$$\iff \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$
$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

p	$=$	javacard.framework.JCSystem.getUnusedCommitCapacity()
$p.isStatic$		
$p \in$ SEPARATELY_HANDLED_API_METHODS		
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)::\{(\mathtt{s}, (0, \mathtt{Short.MAX_VALUE}, 0), (m_n, pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

Given:

$$0 \leq \mathtt{CARD_SPECIFIC_UNUSED_CAPACITY} \leq \mathtt{Short.MAX_VALUE} \Rightarrow$$
$$\beta_{Num}(s, \mathtt{CARD_SPECIFIC_UNUSED_CAPACITY}, (m_n, pc_n)) \sqsubseteq_{Val} \{(\mathtt{s}, (0, \mathtt{Short.MAX_VALUE}, 0), (m_n, pc_n))\}$$

Combining terms:

$$\begin{aligned}
\beta^{R,H,JH}_{Stack}(S) &= \beta^{R,H,JH}_{Stack}(S_n::(s, \mathtt{CARD_SPECIFIC_UNUSED_CAPACITY}, (m_n, pc_n))) \\
&\sqsubseteq_S \beta^{R,H,JH}_{Stack}(S_n)::\beta_{Num}(s, \mathtt{CARD_SPECIFIC_UNUSED_CAPACITY}, (m_n, pc_n)) \\
&\sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)::\{(\mathtt{s}, (0, \mathtt{Short.MAX_VALUE}, 0), (m_n, pc_n\} \\
&\sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{aligned}$$

and the result follows.

A.2.3 Case API method `javacard.framework.JCSystem.getMaxCommitCapacity()`:

A.2.3.1 Operational Semantic rule

$m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p$

$p = \texttt{javacard.framework.JCSystem.getMaxCommitCapacity}()$

$p.isStatic$

$0 \leq \texttt{CARD_SPECIFIC_MAX_CAPACITY} \leq \texttt{Short.MAX_VALUE}$

$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$

$SF = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$

$SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$

$S = S_n :: (\texttt{s}, \texttt{CARD_SPECIFIC_MAX_CAPACITY}, (m_n, pc_n))$

$$P \Big| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF \rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF' \rangle \end{array}$$

A.2.3.2 Flow-Logic clause

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$
$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

p	$=$	`javacard.framework.JCSystem.getMaxCommitCapacity()`
$p.isStatic$		
$p \in$ `SEPARATELY_HANDLED_API_METHODS`		
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1) :: \{(\texttt{s}, (0, \texttt{Short.MAX_VALUE}, 0), (m_n, pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

A.2.3.3 Subject Reduction Theorem

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p \\ p = \texttt{javacard.framework.JCSystem.getMaxCommitCapacity}() \\ p.isStatic \\ 0 \leq \texttt{CARD_SPECIFIC_MAX_CAPACITY} \leq \texttt{Short.MAX_VALUE} \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \end{array}}{{}_P\Big|\ \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = {}_{\langle\!\langle \mathit{loc}_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1, pc_1),\, L_1,\, S_1\rangle\!\rangle} :: {}_{\langle\!\langle \mathit{loc}_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2, pc_2),\, L_2,\, S_2\rangle\!\rangle} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$$

$$SF' = {}_{\langle\!\langle \mathit{loc}_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1, pc_1),\, L_1,\, S_1\rangle\!\rangle} :: {}_{\langle\!\langle \mathit{loc}_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2, pc_2),\, L_2,\, S_2\rangle\!\rangle} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), \mathit{L}_n, S\rangle$$

$$S = \mathit{S}_n :: (\texttt{s}, \texttt{CARD_SPECIFIC_MAX_CAPACITY}, (\mathit{m}_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{\mathit{R,H,JH},k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{\mathit{R,H,JH},k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{\mathit{R,H,JH}}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for this API method the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n::(\texttt{s}, \texttt{CARD_SPECIFIC_MAX_CAPACITY}, (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \forall\ \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

p	$=$	javacard.framework.JCSystem.getMaxCommitCapacity()
$p.isStatic$		
$p \in$ SEPARATELY_HANDLED_API_METHODS		
$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)::\{(\texttt{s}, (0, \texttt{Short.MAX_VALUE}, 0), (m_n, pc_n))\}$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

Given:

$$0 \leq \texttt{CARD_SPECIFIC_MAX_CAPACITY} \leq \texttt{Short.MAX_VALUE}\ \Rightarrow$$

$$\beta_{Num}(s, \texttt{CARD_SPECIFIC_MAX_CAPACITY}, (m_n, pc_n)) \sqsubseteq_{Val} \{(\texttt{s}, (0, \texttt{Short.MAX_VALUE}, 0), (m_n, pc_n))\}$$

Combining terms:

$$\begin{array}{rcl} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n::(s, \texttt{CARD_SPECIFIC_MAX_CAPACITY}, (m_n, pc_n))) \\ & \sqsubseteq_S & \beta^{R,H,JH}_{Stack}(S_n)::\beta_{Num}(s, \texttt{CARD_SPECIFIC_MAX_CAPACITY}, (m_n, pc_n)) \\ & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1)::\{(\texttt{s}, (0, \texttt{Short.MAX_VALUE}, 0), (m_n, pc_n\} \\ & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

A.2.4 Case API method `javacard.framework.JCSystem.beginTransaction()`:

A.2.4.1 Operational Semantic rule

$$
\frac{
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p \\
p = \texttt{javacard.framework.JCSystem.beginTransaction}() \\
p.isStatic \\
p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\
SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle \\
(1 = ctd_n)\ \Rightarrow \\
\quad SF' = \mathit{catchException}\left(SF, (\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)), \texttt{java.framework.TransactionException}, ctd_n, io_n\right) \\
\quad H' = H[\mathit{loc}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id}) \mapsto \widehat{\sigma}_{\texttt{TransactionException}}.\texttt{IN_PROGRESS}] \\
\quad JH' = JH \\
(0 = ctd_n)\ \Rightarrow \\
\quad JH' = [] \\
\quad SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, 1, io_n, (m_n, m_n.\mathit{nextAddress}(pc)), L_n, S_n\rangle \\
\quad H' = H
\end{array}
}{
{}_P\vert\ \begin{array}{l}\langle R, K, H, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H', HID, JH, CHN, SF'\rangle\end{array}
}
$$

A.2.4.2 Flow-Logic clause

$$
(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p
$$

$$
\iff \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :
$$

$$
\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))
$$

$p = \texttt{javacard.framework.JCSystem.beginTransaction}()$
$p.isStatic$
$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$
$(\gamma_n = 1)\ \Rightarrow$
$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$
$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{TransactionException}}, \emptyset)$
$(\gamma_n = 0)\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{[]\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

A.2.4.3 Subject Reduction Theorem

Possible transition 1 of 2

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ \ p \\ p = \texttt{javacard.framework.JCSystem.beginTransaction}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ \big(1 = ctd_n\big) \end{array}}{{}_P\Big|\ \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H', HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

SF= $\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$

SF'=catchException$\Big(SF, (\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)), \texttt{java.framework.TransactionException}, ctd_n, io_n\Big)$

$H' = H[\mathit{loc}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id}) \mapsto \widehat{\sigma}_{\texttt{TransactionException}}.\texttt{IN_PROGRESS}]$

The form of C and C' meet the preconditions for Lemma B.5.2 and so we have only to show:

- $\beta^{R,H,JH}_{Val}\quad (\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{IN_PROGRESS}) \sqsubseteq_{Val}$

  $\widehat{H}\big(\beta^{R,H,JH}_{Ref}((\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)))\big).\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$

- HANDLE$(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH}_{Ref}(\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)), \emptyset)$

to conclude.

From the flow logic rule we have:

$$(\widehat{R},\widehat{K},\widehat{H},\widehat{I},\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{E},\widehat{MNAMES},\widehat{REC},\widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \quad \forall\, \pi_1 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) \sqsubseteq_C \widehat{C}(m_n,pc_n) :$$

$$\pi_2 = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,1,\xi_n,(m_n,m_n.\mathit{nextAddress}(pc_n))))$$

$p = \texttt{javacard.framework.JCSystem.beginTransaction}()$

$p.isStatic$

$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$

$(\gamma_n = 1) \Rightarrow$

$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$

$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{TransactionException}},\emptyset)$

combined with the definition of $\beta^{R,H,JH,k}_{Context}$, $ctd_n = \gamma_n = 1$ we must have:

$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$
$\mathsf{HANDLE}(\pi_1,\pi_1,\widehat{\sigma}_{\texttt{TransactionException}},\emptyset)$

From Appendix D.7 we have:

$$\widehat{\sigma}_{\texttt{TransactionException}} \;=\; \beta^{R,H,JH}_{Ref}(\mathit{loc}_{\texttt{TransactionException}})$$

and from $SF\ \mathcal{R}^{R,H,JH}_{\mathrm{CallStack}}\ (\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{MNAMES},\widehat{CG})$ we must have:

$$\exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n,pc_n)\ .\ \pi_1 \quad = \beta^{R,H,JH,k}_{Context}(SF)$$

and the result follows.

Possible transition 2 of 2

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ \ p \\ p = \texttt{javacard.framework.JCSystem.beginTransaction}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ \big(0 = ctd_n\big) \end{array}}{{}_P\Big| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle \ \Rightarrow \\ \langle R, K, H, HID, JH', CHN, SF'\rangle \end{array}}$$

whence:

$\mathit{SF} = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$

$\mathit{SF}' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, 1, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc)), \mathit{L}_n, \mathit{S}_n\rangle$

JH'=[]

$S = \mathit{S}_n$

Using Lemma B.5.3 on page 414 as a basis, and factoring in the requirements for handling *JH*, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{\mathit{R,H,JH},k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{\mathit{R,H,JH},k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{\mathit{R,H,JH}}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{[]\} & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for this API method the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \forall\ \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$p = \texttt{javacard.framework.JCSystem.beginTransaction}()$

$p.isStatic$

$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$

$(\gamma_n = 1)\ \Rightarrow$

$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{TransactionException}}, \emptyset)$

$(\gamma_n = 0)\ \Rightarrow$

$$\begin{array}{lll} \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\ \widehat{S}(m_n, pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \{[]\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

combined with the definition of $\beta^{R,H,JH,k}_{Context},\ ctd_n = \gamma_n = 0$ we must have:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.nextAddress(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\{[]\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.nextAddress(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

Combining terms:

$$\begin{aligned}\beta^{R,H,JH}_{Stack}(S) &= \beta^{R,H,JH}_{Stack}(S_n)\\ &\sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)\\ &\sqsubseteq_S \widehat{S}(m_n, m_n.nextAddress(pc_n))(\pi_2)\end{aligned}$$

and the result follows.

A.2.5 Case API method `javacard.framework.JCSystem.commitTransaction()`:

A.2.5.1 Operational Semantic rule

$$\frac{\begin{array}{l} m_n.instructionAt(pc_n) = \texttt{invokedefinite}\ p\\ p = \texttt{javacard.framework.JCSystem.commitTransaction}()\\ p.isStatic\\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS}\\ SF = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots:: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle\\ (0 = ctd_n)\ \Rightarrow\\ \quad SF' = catchException\left(SF, (\texttt{r}, loc_{\texttt{TransactionException}}, (1,1)), \texttt{java.framework.TransactionException}, ctd_n, io_n\right)\\ \quad H' = H[loc_{\texttt{TransactionException}}.values(\texttt{javacard.framework.TransactionException.reason}.id) \mapsto \widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS}]\\ \quad JH' = JH\\ (1 = ctd_n)\ \Rightarrow\\ \quad SF' = \langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle\!\rangle :: \langle\!\langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle\!\rangle ::\ldots:: \langle loc_n, itd_n, 1, io_n, (m_n, m_n.nextAddress(pc)), L_n, S_n\rangle\\ \quad H' = H\\ \quad \forall\ loc \in dom(JH)\ .\ \forall\ f\ \in dom(JH(loc).values) : H[loc.values(f.id) \mapsto JH(loc).values(f.id)]\\ \quad JH' = []\end{array}}{{}_P\left|\begin{array}{l}\langle R, K, H, HID, JH, CHN, SF\rangle \Rightarrow\\ \langle R, K, H', HID, JH', CHN, SF'\rangle\end{array}\right.}$$

A.2.5.2 Flow-Logic clause

$$(\hat{R}, \hat{K}, \hat{H}, \hat{I}, \widehat{JH}, \hat{C}, \hat{L}, \hat{S}, \hat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \hat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$p = \texttt{javacard.framework.JCSystem.commitTransaction()}$
$p.isStatic$
$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$
$(\gamma_n = 0) \Rightarrow$
$\{(\hat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \hat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$
$\texttt{HANDLE}(\pi_1, \pi_1, \hat{\sigma}_{\texttt{TransactionException}}, \emptyset)$
$(\gamma_n = 1) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\hat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\hat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\hat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\hat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\hat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_H$	$\widehat{H}$
$\{[]\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

A.2.5.3 Subject Reduction Theorem

Possible transition 1 of 2

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p \\ p = \texttt{javacard.framework.JCSystem.commitTransaction}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ (0 = ctd_n) \end{array}}{P \left| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF \rangle \Rightarrow \\ \langle R, K, H', HID, JH, CHN, SF' \rangle \end{array} \right.}$$

whence:

SF= $\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$

$SF' = \mathit{catchException}\Big(SF, (\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1, 1)), \texttt{java.framework.TransactionException}, ctd_n, io_n\Big)$

$H' = H[\mathit{loc}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id}) \mapsto \hat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS}]$

The form of C and C' meet the preconditions for Lemma B.5.2 and so we have only to show:

- $\beta_{Val}^{R,H,JH}$ $(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS}) \sqsubseteq_{Val}$
  $\widehat{H}(\beta_{Ref}^{R,H,JH}((\mathtt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)))).\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$
- HANDLE$(\beta_{Context}^{R,H,JH,k}(SF), \beta_{Context}^{R,H,JH,k}(SF), \beta_{Ref}^{R,H,JH}(\mathtt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)), \emptyset)$

to conclude.

From the flow logic rule we have:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$
$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$$p = \texttt{javacard.framework.JCSystem.commitTransaction()}$$
$$p.isStatic$$
$$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$$
$$(\gamma_n = 0) \Rightarrow$$
$$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$$
$$\text{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{TransactionException}}, \emptyset)$$

combined with the definition of $\beta_{Context}^{R,H,JH,k}$, $ctd_n = \gamma_n = 0$ we must have:

$$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$$
$$\text{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{TransactionException}}, \emptyset)$$

From Appendix D.7 we have:

$$\widehat{\sigma}_{\texttt{TransactionException}} = \beta_{Ref}^{R,H,JH}(\mathit{loc}_{\texttt{TransactionException}})$$

and from $SF\ \mathcal{R}_{\text{CallStack}}^{R,H,JH}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 = \beta_{Context}^{R,H,JH,k}(SF)$$

and the result follows.

Possible transition 2 of 2

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ \ p \\ p = \texttt{javacard.framework.JCSystem.commitTransaction}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ (1 = ctd_n) \\ \forall\ \mathit{loc} \in \mathit{dom}(\mathit{JH})\ .\ \forall\ f\ \in \mathit{dom}(\mathit{JH}(\mathit{loc}).\mathit{values}) : \mathit{H}[\mathit{loc}.\mathit{values}(f.\mathit{id})\ \mapsto\ \mathit{JH}(\mathit{loc}).\mathit{values}(f.\mathit{id})] \end{array}}{{}_P\Big|\ \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H, HID, JH', CHN, SF'\rangle \end{array}}$$

whence:

$\mathit{SF} = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$

$\mathit{SF}' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::\ldots::\langle \mathit{loc}_n, itd_n, 0, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc)), \mathit{L}_n, \mathit{S}_n\rangle$

JH'=[]

$S = \mathit{S}_n$

Using Lemma B.5.3 on page 414 as a basis, and factoring in the requirements for handling *JH* and *H*, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{\mathit{R,H,JH},k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{\mathit{R,H,JH},k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{\mathit{R,H,JH}}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_H & \widehat{H} \\ \wedge & \{[]\} & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

From the flow-logic rule, we have:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k,MAX_MOD_COUNT,MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\iff \forall\ \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$
$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$p = \texttt{javacard.framework.JCSystem.commitTransaction()}$
$p.isStatic$
$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$
$(\gamma_n = 0)\ \Rightarrow$
$\{(\hat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \hat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$
$\mathsf{HANDLE}(\pi_1, \pi_1, \hat{\sigma}_{\texttt{TransactionException}}, \emptyset)$
$(\gamma_n = 1)\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_H$	$\widehat{H}$
$\{[]\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

combined with the definition of $\beta^{R,H,JH,k}_{Context}$, $ctd_n = \gamma_n = 1$ we must have:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_H$	$\widehat{H}$
$\{[]\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

Combining terms:

$$\begin{array}{rcl}\beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n) \\ & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \\ & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\end{array}$$

and the result follows.

A.2.6 Case API method `javacard.framework.JCSystem.abortTransaction()`:

A.2.6.1 Operational Semantic rule

$m_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ p$

$p = \texttt{javacard.framework.JCSystem.abortTransaction()}$

$p.isStatic$

$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$

$SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$

$(0 = ctd_n)\ \Rightarrow$

$\mathit{SF}' = \mathit{catchException}\left(SF, (\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)), \texttt{java.framework.TransactionException}, ctd_n, io_n\right)$

$H' = H[\mathit{loc}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id}) \mapsto \widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS}]$

$JH' = JH$

$(1 = ctd_n)\ \Rightarrow$

$\mathit{SF}' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle ::: \ldots ::: \langle \mathit{loc}_n, itd_n, 1, io_n, (m_n, m_n.\mathit{nextAddress}(pc)), L_n, S_n\rangle$

$H' = H$

$JH' = []$

$$P \Big| \begin{array}{l}\langle R, K, H, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H', HID, JH', CHN, SF'\rangle\end{array}$$

A.2.6.2 Flow-Logic clause

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p$$

$$\Longleftrightarrow \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$p = \texttt{javacard.framework.JCSystem.abortTransaction()}$

$p.isStatic$

$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$

$(\gamma_n = 0)\ \Rightarrow$

$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{TransactionException}}, \emptyset)$

$(\gamma_n = 1)\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\widehat{S}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{[]\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

A.2.6.3 Subject Reduction Theorem

Possible transition 1 of 2

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ \ p \\ p = \texttt{javacard.framework.JCSystem.abortTransaction}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ \big(0 = ctd_n\big) \end{array}}{{}_P\Big| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle \ \Rightarrow \\ \langle R, K, H', HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

SF= $\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$

SF'=catchException$\Big(SF, (\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)), \texttt{java.framework.TransactionException}, ctd_n, io_n\Big)$

$\mathit{H'} = \mathit{H}[\mathit{loc}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id}) \ \mapsto\ \widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS}]$

The form of C and C' meet the preconditions for Lemma B.5.2 and so we have only to show:

- $\beta_{Val}^{R,H,JH}\quad (\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS}) \sqsubseteq_{Val}$
  $\widehat{H}\big(\beta_{Ref}^{R,H,JH}((\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)))\big).\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$
- HANDLE$(\beta_{Context}^{R,H,JH,k}(SF), \beta_{Context}^{R,H,JH,k}(SF), \beta_{Ref}^{R,H,JH}(\texttt{r}, \mathit{loc}_{\texttt{TransactionException}}, (1,1)), \emptyset)$

to conclude.

From the flow logic rule we have:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ \ p$$

$$\Longleftrightarrow \quad \forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$$

$$\begin{array}{l} p = \texttt{javacard.framework.JCSystem.abortTransaction}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ (\gamma_n = 0)\ \Rightarrow \\ \quad \{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id}) \\ \quad \text{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{TransactionException}}, \emptyset) \end{array}$$

combined with the definition of $\beta^{R,H,JH,k}_{Context}$, $ctd_n = \gamma_n = 0$ we must have:

$\{(\widehat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \widehat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$
$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{TransactionException}}, \emptyset)$

From Appendix D.7 we have:

$$\widehat{\sigma}_{\texttt{TransactionException}} \quad = \quad \beta^{R,H,JH}_{Ref}(\mathit{loc}_{\texttt{TransactionException}})$$

and from $SF\ \mathcal{R}^{R,H,JH}_{\mathrm{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 \quad = \beta^{R,H,JH,k}_{Context}(SF)$$

and the result follows.

Possible transition 2 of 2

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{invokedefinite}\ \ p \\ p = \texttt{javacard.framework.JCSystem.abortTransaction}() \\ p.isStatic \\ p \in \texttt{SEPARATELY_HANDLED_API_METHODS} \\ (1 = ctd_n) \end{array}}{{}_P\Big|\ \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H, HID, JH', CHN, SF'\rangle \end{array}}$$

whence:

$\mathit{SF} = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$

$\mathit{SF}' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, 0, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc)), \mathit{L}_n, \mathit{S}_n\rangle$

$\mathit{JH}' = []$

$S = \mathit{S}_n$

Using Lemma B.5.3 on page 414 as a basis, and factoring in the requirements for handling *JH* and *H*, we have

only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{[]\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lll}
\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

From the flow-logic rule, we have:

$$
(\hat{R}, \widehat{K}, \widehat{H}, \hat{I}, \widehat{JH}, \hat{C}, \hat{L}, \hat{S}, \hat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\text{Base-CFA}} (m_n, pc_n) : \texttt{invokedefinite}\ p
$$

$$
\iff \forall\ \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :
$$

$$
\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, 1, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))
$$

$p = \texttt{javacard.framework.JCSystem.abortTransaction()}$

$p.isStatic$

$p \in \texttt{SEPARATELY_HANDLED_API_METHODS}$

$(\gamma_n = 0) \Rightarrow$

$\{(\hat{\sigma}_{\texttt{TransactionException}}.\texttt{NOT_IN_PROGRESS})\} \sqsubseteq_{Val} \hat{\sigma}_{\texttt{TransactionException}}.\mathit{values}(\texttt{javacard.framework.TransactionException.reason}.\mathit{id})$

$\mathsf{HANDLE}(\pi_1, \pi_1, \hat{\sigma}_{\texttt{TransactionException}}, \emptyset)$

$(\gamma_n = 1) \Rightarrow$

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\widehat{S}(m_n, pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{[]\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

combined with the definition of $\beta^{R,H,JH,k}_{Context}$, $ctd_n = \gamma_n = 1$ we must have:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\widehat{S}(m_n, pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{[]\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

Combining terms:

$$
\begin{array}{rcl}
\beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n) \\
& \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \\
& \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

and the result follows.

A.3 APDU methods

Appendix B

Base Control-Flow Analysis

B.1 Introduction

In this Appendix we present the full proofs for our base control-flow analysis for Carmel and prove it correct with respect to the operational semantics of Chapter 3. The base control-flow analysis is specified using the constraint-based, specification-oriented and implementation-agnostic flow-logic framework of Nielson and Nielson [NN02]. Further, we show how to systematically construct from the flow-logic specification a worklist algorithm capable of generating the least solution to the constraints of the base control-flow analysis. In proving the correctness of the base control-flow analysis, we have been greatly aided by Chapter 3 of [Han05] with the approach and structure of this chapter very much based on that one.

B.2 Abstract Domains including Analysis Domains

Following the original SecSafe material and [Han05], in this thesis we use the notation of "overlined" domains, e.g. $\overline{\mathrm{RetAddr}}$ for abstract return addresses, to indicate abstract counterparts of concrete domains, and domains with a hat, e.g. $\widehat{\mathrm{Val}}$, to indicate complete lattices over abstract domains.

It is customary to present the abstract domains first and then later the choice of representation functions. However, we shall be presenting each abstract domain and its representation function together as we believe it is helpful to understand immediately the nature of the correspondence between concrete and abstract entities.

B.2.1 Abstract Domains for Values

We define abstract domains and their corresponding representation functions for the values that can occur in a Carmel program: numbers, object references (for class instances and array objects), and return addresses (for subroutines).

$$\overline{\mathsf{Val}} \quad = \quad \overline{\mathsf{Num}} + \overline{\mathsf{Ref}} + \overline{\mathsf{RetAddr}}$$

The high-level representation function for values in given before; the representation functions on the right-hand-side are defined in the next few pages.

$$\beta^{R,H,JH}_{Val}(t, Y, (m_n, pc_n)) = \begin{cases} \beta^{R,H,JH}_{Ref}(t, Y, (m_n, pc_n)) & t = \texttt{r} \\ \beta_{ReturnAddress}(t, Y, (m_n, pc_n)) & t = \texttt{ra} \\ \beta_{Num}(t, Y, (m_n, pc_n)) & t \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \end{cases}$$

B.2.1.1 Return Addresses

We simply enclose the return address value in a set for its abstract representation. There is no need for approximation since the value can be determined statically from the `jsr` *addr* instruction, the only instruction that can produce such values.

$$\overline{\mathsf{RetAddr}} \quad : \quad \mathsf{OpType} \times \mathbb{N}_0 \times \mathsf{Address}$$

$$\overline{\mathsf{RetAddr}} \quad = \quad \left\{ \; (\texttt{ra}, apc, (m, pc)) \quad | \quad (m, apc), (m, pc) \in \mathsf{Address} \; \right\}$$

$$\beta_{ReturnAddress}(\texttt{ra}, addr, (m_n, pc_n)) \quad = \quad \{(\mathrm{ra}, \; addr, (m_n, pc_n))\}$$

B.2.1.2 Numbers

The abstract domain for numbers is motivated by the desire to be able to represent and track a runtime value number $(t, n, (m, pc))$ as an interval (inclusive each end) from lowest number l to highest number h, such that $l \leq n \leq h$, associated with (m, pc), and augmented by a *modification count*. The intention is that we will track changes to numbers represented as intervals up to a maximum number of changes, `MAX_MOD_COUNT`, parametric to

the analysis, and then map the number to the well-defined minimum and maximum values the number may hold according to its type i.e. $(t, (\bot_t, \top_t, \texttt{MAX_MOD_COUNT}), (m, pc))$.

$$\overline{\mathsf{Num}} \quad : \quad \mathsf{OpType} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{N}_0 \times \mathsf{Address}$$

$$\overline{\mathsf{Num}} \quad = \quad \left\{ (t, (l, h, mod_{count}), (m, pc)) \;\middle|\; \begin{array}{c} \bot_t \leq l \leq h \leq \top_t \\ 0 \leq mod_{count} \leq \text{MAX_MOD_COUNT} \\ \bot_t, l, h, \top_t \in \mathbb{Z} \\ mod_{count} \in \mathbb{N}_0 \\ (m, pc) \in \mathsf{Address} \\ t \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \\ \bot_{\texttt{b}} = -2^7, \top_{\texttt{b}} = 2^7 - 1 \\ \bot_{\texttt{s}} = -2^{15}, \top_{\texttt{s}} = 2^{15} - 1 \\ \bot_{\texttt{i}} = -2^{31}, \top_{\texttt{i}} = 2^{31} - 1 \end{array} \right\}$$

$$\beta_{Num}(t, c, (m_n, pc_n)) \quad = \quad \{(t, (c, c, 0), (m_n, pc_n))\}$$

B.2.1.3 Object references

Abstract object references include 9 of the 11 attributes of concrete objects defined in Section 3.5.1. Of the 9 included attributes, only *length* and *owner* attributes differ from their concrete counterparts:

- *length* is now an interval, in line with the abstract representation of numbers, since the program analysis clause for the `new` τ`[]` operator takes an abstract number off the stack;

- *owner* records the package AID as per the concrete context, for use in the security predicates, but does not take the applet AID, since the applet could be registered with a multitude of different AIDs. Instead the second component of the abstract context is the applet's fully qualified class name. The second component of the concrete context – the applet AID – is not used in any of the security predicates at present. So currently, it is more meaningful to record the class name of the applet – this can always be changed to the default AID that should be provided with each concrete applet in the Carmel program if the security predicates ever change to additionally use the applet AID.

This approach is similar in spirit to the *textual object graphs* of [VHU92]. There is sufficient information in abstract object references to be able to check firewall predicates and array lengths for the associated object.

$$
\begin{aligned}
\overline{\mathsf{Ref}} \quad = \quad & (\mathit{type} : \mathsf{Type}) \\
\times \quad & (\mathit{refType} : \mathsf{ReferenceType}) \\
\times \quad & (\mathit{isArray} : \mathsf{boolean}) \\
\times \quad & (\mathit{owner} : (\mathit{context} : \mathsf{Context}) \times (\mathit{type} : \mathsf{Type})) \\
\times \quad & (\mathit{entryPoint} : \mathsf{EntryPoint}) \\
\times \quad & (\mathit{isGlobal} : \mathsf{boolean}) \\
\times \quad & (\mathit{transient} : \mathsf{Transient}) \\
\times \quad & (\mathit{creationPoint} : \mathsf{Address}) \\
\times \quad & (\mathit{length} : (min : \mathbb{N}_0) \times (max : \mathbb{N}_0))
\end{aligned}
$$

$$\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (\mathit{m}_n, \mathit{pc}_n)) = \begin{cases} \{\widehat{\sigma}_{\mathtt{Null}}\} & \mathit{loc} = \mathtt{null} \\ \left\{ \left(\mathtt{r}, \left(\begin{array}{l} \mathit{type} : \tau_1, \\ \mathit{refType} : \tau_2, \\ \mathit{isArray} : is, \\ \mathit{owner} : own, \\ \mathit{entryPoint} : ep, \\ \mathit{isGlobal} : global, \\ \mathit{transient} : trans, \\ \mathit{creationPoint} : (\mathit{m}, pc), \\ \mathit{length} : (min, max) \end{array} \right), (\mathit{m}_n, \mathit{pc}_n) \right) \middle| \begin{array}{lcl} \tau_1 & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{type} \\ \tau_2 & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{refType} \\ is & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray} \\ own & = & \beta^{R,H,JH}_{Own}(\mathtt{getJHorH}(\mathit{loc}).\mathit{owner}) \\ ep & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{entryPoint} \\ global & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal} \\ trans & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} \\ (\mathit{m}, pc) & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{creationPoint} \\ min & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \\ max & = & \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \end{array} \right\} & \mathit{loc} \neq \mathtt{null} \end{cases}$$

$$\beta^{R,H,JH}_{Own}(package_{aid}, applet_{aid}) = (package_{aid}, \mathtt{getJHorH}(\mathit{loc}_{applet}).\mathit{type}),\ \mathit{R}(applet_{aid}) = (\mathit{loc}_{aid}, \mathit{loc}_{applet})$$

B.2.2 Abstract Domains for Analysis

B.2.2.1 Fundamental Complete Lattice Underpinning Analysis Domains

Having defined the abstract domains for values:

$$\overline{\mathsf{Val}} \quad = \quad \overline{\mathsf{Num}} + \overline{\mathsf{Ref}} + \overline{\mathsf{RetAddr}}$$

corresponding closely to the concrete domains, we define the fundamental complete lattice underpinning our analysis and analysis domains:

$$\widehat{\mathsf{Val}} = \left\langle \mathcal{P}(\overline{\mathsf{Val}}), \ \sqsubseteq_{Val} \right\rangle$$

where partial order $\sqsubseteq_{Val}$ is defined for $v_1.v_2 \in \widehat{\mathsf{Val}}$:

$$
\begin{aligned}
v_1 \sqsubseteq_{Val} v_2 \quad \Longleftrightarrow \quad & \forall\, (t, Y, (m, pc)) \in v_1 : \\
& (t \in \{\mathtt{r}, \mathtt{ra}\}) \ \Rightarrow \\
& \quad \{(t, Y, (m, pc))\} \subseteq v_2 \\
& (t \in \{\mathtt{b}, \mathtt{s}, \mathtt{i}\}) \ \Rightarrow \\
& \quad \exists\, \{(t, Y_2, (m, pc))\} \subseteq v_2 \ . \\
& \quad Y = (l, h, mod_{count1}) \\
& \quad Y_2 = (l_2, h_2, mod_{count2}) \\
& \quad \bot_t \leq l_2 \leq l \leq h \leq h_2 \leq \top_t \\
& \quad 0 \leq mod_{count1} \leq mod_{count2} \leq \mathrm{MAX_MOD_COUNT} \\
& \quad \bot_t, l, l_2, h, h_2, \top_t \in \mathbb{Z} \\
& \quad mod_{count1}, mod_{count2} \in \mathbb{N}_0
\end{aligned}
$$

By Lemma A.2 of Appendix A of [NNH10], $\langle \widehat{\mathsf{Val}}, \sqsubseteq_{Val} \rangle$ is a complete lattice with bottom $\bot_{\mathsf{Val}} = \emptyset$ and $\top_{\mathsf{Val}} = \overline{\mathsf{Val}}$ and binary least-upper bounds $\sqcup_{Val}\{v_1, v_2\}$, written infix as $v_1 \sqcup_{Val} v_2$, defined in Table 4.1. In words, the least upper bound of two values in $\widehat{\mathsf{Val}}$ depends on whether both values are numbers or not:

- if both values are not numbers sharing the same type and associated address, the least upper bound is simply the union of the two values;
- if both values are numbers sharing the same type and associated address, the least upper bound is then:
  - if the modification count of one or both of the values is $\geq$ `MAX_MOD_COUNT`, then the numeric interval component of the resulting abstract number is widened to the minimum and maximum values supported by that type i.e. $(\bot_t, \top_t, \texttt{MAX_MOD_COUNT})$;
  - otherwise, the numeric interval component of the resulting abstract number is widened to:

$$\left(\begin{array}{ll} \texttt{minimum}(first\ value\ left\ interval\ number, & second\ value\ left\ interval\ number), \\ \texttt{maximum}(first\ value\ right\ interval\ number, & second\ value\ right\ interval\ number), \\ \texttt{maximum}(first\ value\ modification\ count, & second\ value\ modification\ count) \end{array}\right)$$

$$
(t_1, Y_1, (m_1, pc_1)) \sqcup_{Val} (t_2, Y_2, (m_2, pc_2)) \quad = \quad \begin{cases} \{(t_1, (min, max, mod), (m_1, pc_1))\} & \begin{array}{l} t_1 = t_2 \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \\ m_1 = m_2 \\ pc_1 = pc_2 \\ Y_1 = (l_1, h_1, mod_1) \\ Y_2 = (l_2, h_2, mod_2) \\ min = \texttt{minimum}(l_1, l_2) \\ max = \texttt{maximum}(h_1, h_2) \\ mod = \texttt{maximum}(mod_1, mod_2)) \\ mod < \texttt{MAX_MOD_COUNT} \end{array} \\ \{(t_1, (\bot_t, \top_t, \texttt{MAX_MOD_COUNT}), (m_1, pc_1))\} & \begin{array}{l} t_1 = t_2 \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \\ m_1 = m_2 \\ pc_1 = pc_2 \\ Y_1 = (l_1, h_1, mod_1) \\ Y_2 = (l_2, h_2, mod_2) \\ min = \texttt{minimum}(l_1, l_2) \\ max = \texttt{maximum}(h_1, h_2) \\ mod = \texttt{maximum}(mod_1, mod_2)) \\ mod \geq \texttt{MAX_MOD_COUNT} \end{array} \\ \{(t_1, Y_1, (m_1, pc_1)), (t_2, Y_2, (m_2, pc_2))\} & \text{otherwise} \end{cases}
$$

Table B.1: Binary least upper-bounds operator $\bigsqcup_{Val}$ over the complete lattice $\langle \widehat{\mathsf{Val}}, \ \sqsubseteq_{Val} \rangle$

B.2.2.2 Object State - Class instances and Array objects

The fields of a class instance may be determined statically and in a finite program is a finite (and typically small) number of fields. Therefore a map from fields to elements of $\widehat{\text{Val}}$ is an appropriate representation of a class instance's object state.

In contrast, according to the semantics of arrays in Carmel, an array's length is allowed to be up to $\top_{\mathtt{s}} = 2^{15} - 1$. While such a length is unlikely in a Carmel program, due to resource limitations, it does highlight the question of how best to represent the object state of an array object. In line with the representation of abstract numbers in Carmel, and desiring a compact representation of array object state, a map from intervals of array index numbers to elements of $\widehat{\text{Val}}$ is an appropriate representation of array object state. In particular, the default value of zero or `null` for an abstract array would have the abstract representation of zero or `null` mapped from the interval $(0, \top_{\mathtt{s}})$. To ensure finiteness in the analysis results, and to allow a trade-off between memory costs and precision, parametric to the analysis is `MAX_DOM_DYN_ARRAY` – the maximum size of the domain of a dynamic array created via the `new` τ`[]`. When a value is loaded from an array object, the array index is passed as an abstract number and the interval (l, h) extracted, and each value i in (l, h) is checked against each of the array's domain of intervals $(dl_1, dh_1), (dl_2, dh_2), \ldots, (dl_n, dh_n)$, and where $dl_j \leq i \leq dh_j$ the mapped value is added to the set of values whose LUB is pushed onto the stack as the result of the array load instruction. When a value is to be stored to an array object, the array index is passed as an abstract number and the interval (l, h) extracted. If (l, h) matches exactly one of the array's domain of intervals $(dl_1, dh_1), (dl_2, dh_2), \ldots, (dl_n, dh_n)$, then the LUB of the value to be stored and the existing value mapped against (l, h) is stored against (l, h). Otherwise, if $n <$ `MAX_DOM_DYN_ARRAY`, (l, h) is added to the array's domain of intervals and the value to be stored is mapped against (l, h). When $n \geq$ `MAX_DOM_DYN_ARRAY`, the LUB of the value to be stored and the current value mapped against the interval $(0, \top_{\mathtt{s}})$ is stored against the interval $(0, \top_{\mathtt{s}})$.

We define across class instances and array objects:

$$\widehat{\text{Object}} = (\text{Field} \cup (\mathbb{N}_0 \times \mathbb{N}_0)) : \textit{values} \longrightarrow \widehat{\text{Val}}$$

and extend point-wise the $\sqsubseteq_{Val}$ ordering to objects $o_1, o_2 \in \widehat{\text{Object}}$:

$$o_1 \sqsubseteq_{\widehat{\text{Object}}} o_2 \iff \textit{dom}(o_1.\textit{values}) \subseteq \textit{dom}(o_2.\textit{values}) \ \wedge \forall f \in \textit{dom}(o_1.\textit{values}) : o_1.\textit{values}(f) \sqsubseteq_{Val} o_2.\textit{values}(f)$$

Note that in a Carmel program P, where an object or static field specifies an array's elements e.g.

```
private static byte[] bootstrapSENCKeyData = { 77, 118, 55, 118, -86, -94, -80, 91, -87, 91, 88, -82, 11, 76, 40, -93 };
```

since we can determine statically the length and elements of the array, regardless of the value of the `MAX_DOM_DYN_ARRAY` parameter, the length is set correctly and the array's *values* accordingly. For the example above *values* would be set to:

$$[(0,0) \mapsto (\mathtt{b}, (77, 77, 0), (m, pc)), \ldots, (15,15) \mapsto (\mathtt{b}, (-93, -93, 0), (m, pc))]$$

For the avoidance of doubt, the parameter `MAX_DOM_DYN_ARRAY` is only ever consulted during an `arraystore` *t* instruction and both statically declared arrays such as `bootstrapSENCKeyData` above and any dynamically created instruction created by the `new` τ`[ ]` instruction will have new intervals added to the array's domain of intervals only if the current size of the array's domain of interval is less than `MAX_DOM_DYN_ARRAY`, otherwise the LUB of the value to be stored and the current value mapped against the interval $(0, \top_{\mathtt{s}})$ is stored against the interval $(0, \top_{\mathtt{s}})$, as described above.

NB it is worth reiterating the heightened importance of arrays in Java Card (and so in Carmel) as a result of the lack of the Collections framework of discrete mathematical structures available in standard Java. Consequently, we have taken particular care in our choice of representation of abstract array objects including state.

B.2.2.3 Context-Sensitive Domains

To facilitate loop-bounds calculations and to reduce false positives in the detection of potentially recursive method calls, we require our analysis to be precise as possible *at each Carmel address wrt the possible*:

- JCVM machine states including transaction mechanism state and IO/APDU state (for precision of control-flow and data-flow);
- callstacks (for checking possibly recursive calls);
- class instance and array object *values* in the transaction buffer (for precision of control-flow and data-flow);
- operand stack (for analysing loops);
- local variable array (for analysing loops);

To this end, our choice of abstract context, defined in Table 4.2, includes the machine state and sufficient information from the abstract callstack to be able to check the firewall security predicates. Since infinite callstacks are a possibility, due to recursion, to ensure finiteness in the analysis results, and to allow a trade-off between memory costs and precision, parametric to the analysis is $\mathtt{k}$ – the maximum number of stack frames from the concrete callstack to include in the abstract context. For the firewall security predicates, and for a minimal level of acceptable precision, we require always $\mathtt{k} \geq 5$ and our worklist algorithm checks this parameter. In program analysis parlance, our base control-flow analysis is a k-CFA analysis, as is the extended control-flow analysis of Chapter 5.

Context information is recorded for each address in a context-cache: $\widehat{\mathsf{ContextCache}} = \mathsf{Address} \to \mathcal{P}(\overline{\mathsf{Context}})$ and we calculate, for each address (m, pc), and for each context associated with that address:

- a possible over-approximation of the set of all method names that have invoked the method m (for checking possibly recursive calls);
- a possible over-approximation of the set of all class instance and array object *values* in the transaction buffer (for precision of control-flow and data-flow);
- abstract operand stack (for analysing loops);
- abstract local variable array (for analysing loops);

Each of the context-sensitive domains is defined in the following way:

$$\widehat{\mathsf{AbstractDomain}} = \mathsf{Address} \to \overline{\mathsf{Context}} \to X$$

that is the abstract domain is defined as a map from address to contexts in which the address may be executed, to (the abstract version of the concrete domain). For example, the abstract local variable array is defined as a

map from address to contexts in which the address may be executed, and from each context to a map from local variable array indices to abstract values:

$$\widehat{\mathsf{LocalVar}} = \mathsf{Address} \to \overline{\mathsf{Context}} \to \mathbb{N}_0 \to \widehat{\mathsf{Val}}$$

where $\mathbb{N}_0 \to \widehat{\mathsf{Val}}$ is the abstract version of the local variable array i.e. as a map from natural numbers to abstract values. The representation function for each of the context-sensitive domains then shows how to produce the abstract version X of the concrete domain.

$\overline{\mathsf{Context}}: \quad (\overline{\mathsf{Ref}} \times \mathsf{TransactionDepth} \times \mathsf{TransactionDepth} \times \mathsf{IOState} \times \mathsf{Address})^*$

$$\beta^{R,H,JH,k}_{Context}(\quad \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle :: \ldots :: \langle \mathit{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i\rangle) =$$

$$\begin{cases} \left(\begin{array}{l} \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_1, \mathtt{getJHorH}(\mathit{loc}_1).\mathit{creationPoint}), imtd_1, cmtd_1, io_1, (m_1, pc_1))\right), \\ \ldots, \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_i, \mathtt{getJHorH}(\mathit{loc}_i).\mathit{creationPoint}), imtd_i, cmtd_i, io_i, (m_i, pc_i))\right) \end{array} \right), & i < k \wedge k \geq 5 \\ \\ \left(\begin{array}{l} \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_1, \mathtt{getJHorH}(\mathit{loc}_1).\mathit{creationPoint}), imtd_1, cmtd_1, io_1, (m_1, pc_1))\right), \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_2, \mathtt{getJHorH}(\mathit{loc}_2).\mathit{creationPoint}), imtd_2, cmtd_2, io_2, (m_2, pc_2))\right), \\ \ldots, \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_{(k-2)}, \mathtt{getJHorH}(\mathit{loc}_{(k-2)}).\mathit{creationPoint}), itd_{(k-2)}, ctd_{(k-2)}, io_{(k-2)}, (m_{(k-2)}, pc_{(k-2)}))\right), \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_{(i-1)}, \mathtt{getJHorH}(\mathit{loc}_{(i-1)}).\mathit{creationPoint}), itd_{(i-1)}, ctd_{(i-1)}, io_{(i-1)}, (m_{(i-1)}, pc_{(i-1)}))\right), \\ \left(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_i, \mathtt{getJHorH}(\mathit{loc}_i).\mathit{creationPoint}), imtd_i, cmtd_i, io_i, (m_i, pc_i))\right) \end{array} \right), & i \geq k \wedge k \geq 5 \end{cases}$$

Table B.2: Abstract context representation

B.2.2.3.1 Abstract Context Cache

The abstract context cache is modelled as a map from address to contexts in which the address may be executed:

$$\widehat{\mathsf{ContextCache}} = \mathsf{Address} \to \widehat{\mathsf{Context}}$$

and define the ordering $\sqsubseteq_C$ to abstract context caches $c_1, c_2 \in \widehat{\mathsf{ContextCache}}$:

$$c_1 \sqsubseteq_C c_2 \iff \forall\, addr \in \mathit{dom}(c_1) :\ c_1(addr) \subseteq c_2(addr)$$

B.2.2.3.2 Local Variable Array

Modelling of the abstract local variable array is as a map from address to contexts in which the address may be executed, and from each context to local variable array indices to abstract values.

$$\widehat{\mathsf{LocalVar}} = \mathsf{Address} \to \overline{\mathsf{Context}} \to \mathbb{N}_0 \to \widehat{\mathsf{Val}}$$

with representation function:

$$\begin{aligned} \beta^{R,H,JH}_{LocalVar}(L) \quad = \quad & M\ . \\ & \forall\, i \in \mathit{dom}(L) : \\ & \beta^{R,H,JH}_{Val}(L(i)) \sqsubseteq_{Val} M(i) \end{aligned}$$

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract local variable arrays $l_1, l_2 \in \widehat{\mathsf{LocalVar}}$:

$$\begin{aligned} l_1 \sqsubseteq_L l_2 \iff & \forall\, addr \in \mathit{dom}(l_1), \\ & \forall\, ctxt \in l_1(addr), \\ & \forall\, idx \in dom(l_1(addr)(ctxt)) : \\ & l_1(addr)(ctxt)(idx) \sqsubseteq_{Val} l_2(addr)(ctxt)(idx) \end{aligned}$$

B.2.2.3.3 Operand Stack

Modelling of the abstract operand stack is as a map from address to contexts in which the address may be executed, and from each context to a sequence of abstract values. Since each Carmel program under consideration has been bytecode verified, abstract operand stacks must be of finite length.

$$\widehat{\mathsf{Stack}} = \mathsf{Address} \rightarrow \overline{\mathsf{Context}} \rightarrow (\widehat{\mathsf{Val}})^*$$

with representation function:

$$\beta^{R,H,JH}_{Stack}\big((t_1, v_1, (m_1, pc_1))::\ldots::(t_o, v_o, (m_o, pc_o))\big) = \beta^{R,H,JH}_{Val}(t_1, v_1, (m_1, pc_1))::\ldots::\beta^{R,H,JH}_{Val}(t_o, v_o, (m_o, pc_o))$$

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract operand stacks $s_1, s_2 \in \widehat{\mathsf{Stack}}$:

$$\begin{aligned}
s_1 \sqsubseteq_S s_2 \quad \Longleftrightarrow \quad & \forall\, addr \in \mathit{dom}(s_1), \\
& \forall\, ctxt \in s_1(addr) : \\
& \quad s_1(addr)(ctxt) = A_1::A_2::\ldots::A_q \wedge \\
& \quad s_2(addr)(ctxt) = B_1::B_2::\ldots::B_r \wedge \\
& \quad r \geq q \wedge \\
& \quad \forall i \in \{1, \ldots, q\} : \\
& \qquad A_i \sqsubseteq_{Val} B_i
\end{aligned}$$

B.2.2.3.4 Transactional Heap

Modelling of the abstract transactional heap is as a map from address to contexts in which the address may be executed, from each context to a map from object references to object (both class instance and array object) *values* map from fields or numeric intervals to abstract values.

$$\begin{aligned}\widehat{\mathsf{TransactionDynamicHeap}} &= \mathsf{Address} \to \overline{\mathsf{Context}} \to \overline{\mathsf{Ref}} \to (\mathsf{Field} \cup (\mathbb{N}_0 \times \mathbb{N}_0)) : \textit{values} \longrightarrow \widehat{\mathsf{Val}} \\ &= \mathsf{Address} \to \overline{\mathsf{Context}} \to \overline{\mathsf{Ref}} \to \widehat{\mathsf{Object}}\end{aligned}$$

with representation function[1]:

$$\begin{aligned}
& \beta^{H,JH}_{DynamicHeap}(HEAP) = M \\
\iff \quad & \forall\, \textit{loc} \in \textit{dom}(HEAP) : \\
& \quad o = \beta^{R,H,JH}_{Ref}(\mathtt{r}, \textit{loc}, HEAP(\textit{loc}).\textit{creationPoint}) \\
\wedge \quad & \forall\, f \in \textit{dom}(HEAP(\textit{loc}).\textit{values}) : \\
& \quad \beta^{R,H,JH}_{Val}(HEAP(\textit{loc}).\textit{values}(f.\textit{id})) \sqsubseteq_{Val} M(o).\textit{values}(f.\textit{id})
\end{aligned}$$

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract transactional heaps $t_1, t_2 \in \widehat{\mathsf{TransactionDynamicHeap}}$:

$$\begin{aligned}
t_1 \sqsubseteq_{JH} t_2 \quad \iff \quad & \forall\, addr \in \textit{dom}(t_1), \\
& \forall\, ctxt \in t_1(addr) : \\
& \forall\, ref \in t_1(addr)(ctxt) : \\
& o_1 = t_1(addr)(ctxt)(ref) \ \wedge \\
& o_2 = t_2(addr)(ctxt)(ref) \ \wedge \\
& dom(o_1.\textit{values}) \subseteq dom(o_2.\textit{values}) \ \wedge \\
& \forall f \in \textit{dom}(o_1.\textit{values}) : \\
& \quad o_1.\textit{values}(f) \sqsubseteq_{Val} o_2.\textit{values}(f)
\end{aligned}$$

[1] Here HEAP is JH; we use the same representation function for both the object heap and the transactional object heap, and supply JH or H as a parameter as appropriate.

consistent with the ordering on $\sqsubseteq_{\widehat{\text{Object}}}$.

B.2.2.3.5 Method Names Cache

A purely semantic component, to overcome potential loss of callstack information due to our choice of context (see Table 4.2), and so to ensure at each address (m, pc) we have a conservative (i.e. a possible over-approximation) of the set of methods that may have invoked m, we developed relation $\widehat{\mathsf{MethodNamesCache}}$. Modelling of the method name cache $\widehat{\mathsf{MethodNamesCache}}$ is as a map from address (m, pc) to contexts in which the address may be executed, and from each context to the set of Carmel methods that may have invoked m.

$$\widehat{\mathsf{MethodNamesCache}} = \mathsf{Address} \to \overline{\mathsf{Context}} \to \mathcal{P}(\mathsf{Method})$$

$\widehat{\mathsf{MethodNamesCache}}$ admits the following ordering $\sqsubseteq_{\widehat{MNAMES}}$ to method name caches $mnames_1, mnames_2 \in \widehat{\mathsf{MethodNamesCache}}$:

$$\begin{aligned} mnames_1 \sqsubseteq_{\widehat{MNAMES}} mnames_2 \quad \iff \quad & \forall\, addr \in \mathit{dom}(mnames_1), \\ & \forall\, ctxt \in mnames_1(addr) : \\ & \quad mnames_1(addr)(ctxt) \subseteq mnames_2(addr)(ctxt) \end{aligned}$$

B.2.2.4 Context-Insensitive Domains

B.2.2.4.1 Applet Registry

The abstract applet registry records the set of abstract object references that are registered via one of the applet `register` methods:

$$\widehat{\mathsf{Registry}} \quad = \quad \mathcal{P}(\overline{\mathsf{Ref}})$$

with representation function:

$$\beta^{R,H,JH}_{Registry}(R) = \quad M \ .$$

$$\forall\ aid \in \mathit{dom}(R) :$$

$$R(aid) = (\mathit{loc}_{JCREOwnedAID}, \mathit{loc}_{applet}) \wedge$$

$$\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_{applet}, \mathtt{getJHorH}(\mathit{loc}_{applet}).\mathit{creationPoint}) \sqsubseteq_{Val} M$$

and define the ordering on elements $r_1, r_2 \in \widehat{\text{Registry}}$:

$$r_1 \sqsubseteq_R r_2 \quad \Longleftrightarrow \quad r_1 \sqsubseteq_{Val} r_2$$

B.2.2.4.2 Static Heap

The abstract static heap simply maps static fields to abstract values:

$$\widehat{\mathsf{StaticHeap}} = \mathsf{Field} \rightarrow \widehat{\mathsf{Val}}$$

with representation function:

$$\begin{aligned}\beta^{H}_{StaticHeap}(STATIC_HEAP) = \quad & M. \\ & \forall\ f \in \mathit{dom}(STATIC_HEAP) : \\ & \beta^{R,H,JH}_{Val}(STATIC_HEAP(f.\mathit{id})) \sqsubseteq_{Val} M(f.\mathit{id})\end{aligned}$$

and define the ordering on elements $k_1, k_2 \in \widehat{\mathsf{StaticHeap}}$:

$$k_1 \sqsubseteq_K k_2 \quad \Longleftrightarrow \quad \mathit{dom}(k_1) \subseteq \mathit{dom}(k_2)\ \wedge \forall\ f \in \mathit{dom}(k_1) :\ k_1(f.\mathit{id}) \sqsubseteq_{Val} k_2(f.\mathit{id})$$

B.2.2.4.3 Object Heap

Modelling of the abstract object heap is a map from object references to object (both class instance and array object) *values* map from fields or numeric intervals to abstract values.

$$\begin{aligned}\widehat{\mathsf{ObjectHeap}} &= \overline{\mathsf{Ref}} \rightarrow (\mathsf{Field} \cup (\mathbb{N}_0 \times \mathbb{N}_0)) : \mathit{values} \longrightarrow \widehat{\mathsf{Val}} \\ &= \overline{\mathsf{Ref}} \rightarrow \widehat{\mathsf{Object}}\end{aligned}$$

with representation function[2]:

[2]Here HEAP is *H*; we use the same representation function for both the object heap and the transactional object heap, and supply JH or H as a parameter as appropriate.

$$\beta^{H,JH}_{DynamicHeap}(HEAP) = M$$

$$\iff \quad \forall\ \mathit{loc} \in \mathit{dom}(HEAP) :$$

$$o = \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, HEAP(\mathit{loc}).\mathit{creationPoint})$$

$$\wedge \qquad \forall\ f \in \mathit{dom}(HEAP(\mathit{loc}).\mathit{values}) :$$

$$\beta^{R,H,JH}_{Val}(HEAP(\mathit{loc}).\mathit{values}(f.\mathit{id})) \sqsubseteq_{Val} M(o).\mathit{values}(f.\mathit{id})$$

and extend the point-wise ordering $\sqsubseteq_{Val}$ to abstract object heaps $h_1, h_2 \in \widehat{\mathsf{ObjectHeap}}$:

$$h_1 \sqsubseteq_H h_2 \quad \iff \quad \forall\ ref \in \mathit{dom}(h_1) :$$

$$o_1 = h_1(ref)\ \wedge$$

$$o_2 = h_2(ref)\ \wedge$$

$$dom(o_1.\mathit{values}) \subseteq dom(o_2.\mathit{values})\ \wedge$$

$$\forall f \in \mathit{dom}(o_1.\mathit{values}) :$$

$$o_1.\mathit{values}(f) \sqsubseteq_{Val} o_2.\mathit{values}(f)$$

consistent with the ordering on $\sqsubseteq_{\widehat{\mathsf{Object}}}$.

B.2.2.4.4 Invalidated Objects Cache

The abstract invalidated objects cache records the set of abstract object references whose related object may have been created inside a transaction that was subsequently aborted and so members of this set should be treated – for safety – simultaneously as being both equal to a `null` object reference and as a regular non-`null` object reference[3].

$$\widehat{\mathsf{InvalidatedReferences}} \quad = \quad \mathcal{P}(\overline{\mathsf{Ref}})$$

[3] See Section 2.3.3.9 for fuller information.

with representation function:

$$\beta^{R,H,JH}_{Invalidated}(I) = \quad M \ .$$

$$\forall\ \mathit{loc} \in \mathit{dom}(I) :$$

$$\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, \mathtt{getJHorH}(\mathit{loc}).\mathit{creationPoint}) \sqsubseteq_{Val} M$$

and define the ordering on elements $i_1, i_2 \in \widehat{\mathsf{InvalidatedReferences}}$:

$$i_1 \sqsubseteq_I i_2 \iff i_1 \sqsubseteq_{Val} i_2$$

B.2.2.4.5 Exceptions Cache

The exceptions cache records the abstract reference of the exception object, the abstract context in which the exception was thrown, and the abstract context in which the exception was caught.

$$\widehat{\mathsf{ExceptionsCache}} \quad = \quad \mathcal{P}(\overline{\mathsf{Ref}} \times \overline{\mathsf{Context}} \times \overline{\mathsf{Context}})$$

Lemma B.5.1 (on page 407) proves that whenever the concrete semantics throws an exception, the program analysis ensures, among other actions, $\widehat{\mathsf{ExceptionsCache}}$ is updated appropriately. In particular, when:

$$SF' = \mathit{catchException}(SF, (\mathtt{r}, \mathit{loc}, (\mathit{m}_r, pc_r)), \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (\mathit{m}_r, pc_r)).\mathit{refType}), ctd_n, io_n)$$

Then the program analysis ensures via the HANDLE predicate:

$$\begin{array}{lll} & \{(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (\mathit{m}_r, pc_r)), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF'))\} \quad \sqsubseteq_E & \widehat{\mathsf{ExceptionsCache}} \\ \iff & \{(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (\mathit{m}_r, pc_r)), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF'))\} \quad \subseteq & \widehat{\mathsf{ExceptionsCache}} \end{array}$$

and define the ordering on elements $e_1, e_2 \in \widehat{\mathsf{ExceptionsCache}}$:

$$e_1 \sqsubseteq_E e_2 \iff e_1 \subseteq e_2$$

B.2.2.4.6 Recursive Method Calls Cache

When an attempt to invoke a potentially recursive method m is detected at an address $addr$, we store the address $addr$, method to invoke m and the abstract context in which $addr$ was attempted to invoke m.

$$\widehat{\mathsf{RecursiveMethodsCache}} = \mathcal{P}(\mathsf{Address} \times \mathsf{Method} \times \overline{\mathsf{Context}})$$

$\widehat{\mathsf{RecursiveMethodsCache}}$ admits the following ordering $\sqsubseteq_{REC}$ to recursive method name caches $rmc_1, rmc_2 \in \widehat{\mathsf{RecursiveMethodsCache}}$:

$$rmc_1 \sqsubseteq_{\widehat{REC}} rmc_2 \quad \Longleftrightarrow \quad rmc_1 \subseteq rmc_2$$

B.2.2.4.7 Context Graph

Whenever a possible transition from one abstract context $current$ to another abstract context $next$ occurs in the program analysis clause, then $(current, next)$ is recorded in the context graph:

$$\widehat{\mathsf{ContextGraph}} = \mathcal{P}(\overline{\mathsf{Context}} \times \overline{\mathsf{Context}})$$

$\widehat{\mathsf{ContextGraph}}$ admits the following ordering $\sqsubseteq_{CG}$ to abstract context graphs $cg_1, cg_2 \in \widehat{\mathsf{ContextGraph}}$:

$$cg_1 \sqsubseteq_{CG} cg_2 \quad \Longleftrightarrow \quad cg_1 \subseteq cg_2$$

B.2.3 Abstract Analysis Domains Define Complete Lattices

Proposition B.2.1 (Analysis domains define complete lattices)**.** The following domains are each complete lattices:

- $\widehat{\mathsf{Registry}} = \langle \widehat{\mathsf{Registry}}, \sqsubseteq_R \rangle$
- $\widehat{\mathsf{StaticHeap}} = \langle \widehat{\mathsf{StaticHeap}}, \sqsubseteq_S \rangle$
- $\widehat{\mathsf{ObjectHeap}} = \langle \widehat{\mathsf{ObjectHeap}}, \sqsubseteq_H \rangle$
- $\widehat{\mathsf{InvalidatedReferences}} = \langle \widehat{\mathsf{InvalidatedReferences}}, \sqsubseteq_I \rangle$
- $\widehat{\mathsf{TransactionDynamicHeap}} = \langle \widehat{\mathsf{TransactionDynamicHeap}}, \sqsubseteq_{JH} \rangle$
- $\widehat{\mathsf{Context}} = \langle \widehat{\mathsf{Context}}, \sqsubseteq_C \rangle$
- $\widehat{\mathsf{LocalVar}} = \langle \widehat{\mathsf{LocalVar}}, \sqsubseteq_L \rangle$
- $\widehat{\mathsf{Stack}} = \langle \widehat{\mathsf{Stack}}, \sqsubseteq_S \rangle$

- $\widehat{\text{ExceptionsCache}} = \langle \widehat{\text{ExceptionsCache}}, \sqsubseteq_E \rangle$
- $\widehat{\text{MethodNamesCache}} = \langle \widehat{\text{MethodNamesCache}}, \sqsubseteq_{MNAMES} \rangle$
- $\widehat{\text{RecursiveMethodsCache}} = \langle \widehat{\text{RecursiveMethodsCache}}, \sqsubseteq_{REC} \rangle$
- $\widehat{\text{ContextGraph}} = \langle \widehat{\text{ContextGraph}}, \sqsubseteq_{CG} \rangle$

Proof: $\widehat{\text{Val}}$ is a complete lattice under ordering $\sqsubseteq_{Val}$ as shown in Section 4.2.2.1. The orderings on domains $\widehat{\text{Registry}}$, $\widehat{\text{StaticHeap}}$, $\widehat{\text{ObjectHeap}}$, $\widehat{\text{InvalidatedReferences}}$, $\widehat{\text{TransactionDynamicHeap}}$, $\widehat{\text{LocalVar}}$, $\widehat{\text{Stack}}$ are pointwise extensions of the ordering on $\widehat{\text{Val}}$ i.e. $\sqsubseteq_{Val}$ and the result follows from Paragraph 2.15 of [DP02]. The remaining domains are powersets of finite sets, ordered by $\subseteq$, which are complete lattices by Examples 2.6 (2) on [DP02]

B.2.4 Abstract Domain for High-Level Base CFA Analysis

Having defined the abstract value- and analysis- domains, we may now define the high-level domain for the base control-flow analysis analysis as the cross-product of the analysis domains:

$$\begin{aligned}\widehat{\text{Analysis}} \quad = \quad & \widehat{\text{Registry}} \times \widehat{\text{StaticHeap}} \times \widehat{\text{ObjectHeap}} \times \\ & \widehat{\text{InvalidatedReferences}} \times \widehat{\text{TransactionDynamicHeap}} \times \\ & \widehat{\text{Context}} \times \widehat{\text{LocalVar}} \times \widehat{\text{Stack}} \times \widehat{\text{ExceptionsCache}} \times \\ & \widehat{\text{MethodNamesCache}} \times \widehat{\text{RecursiveMethodsCache}} \times \widehat{\text{ContextGraph}}\end{aligned}$$

$\widehat{\text{Analysis}}$ admits the following ordering $\sqsubseteq_{Analysis}$ to abstract analysis results:
Let $(\widehat{R}_1, \widehat{K}_1, \widehat{H}_1, \widehat{I}_1, \widehat{JH}_1, \widehat{C}_1, \widehat{L}_1, \widehat{S}_1, \widehat{E}_1, \widehat{MNAMES}_1, \widehat{REC}_1, \widehat{CG}_1), (\widehat{R}_2, \widehat{K}_2, \widehat{H}_2, \widehat{I}_2, \widehat{JH}_2, \widehat{C}_2, \widehat{L}_2, \widehat{S}_2, \widehat{E}_2, \widehat{MNAMES}_2, \widehat{REC}_2, \widehat{CG}_2) \in \widehat{\text{Analysis}}$. Then:

$$\begin{aligned}&(\widehat{R}_1, \widehat{K}_1, \widehat{H}_1, \widehat{I}_1, \widehat{JH}_1, \widehat{C}_1, \widehat{L}_1, \widehat{S}_1, \widehat{E}_1, \widehat{MNAMES}_1, \widehat{REC}_1, \widehat{CG}_1) \sqsubseteq_{Analysis} (\widehat{R}_2, \widehat{K}_2, \widehat{H}_2, \widehat{I}_2, \widehat{JH}_2, \widehat{C}_2, \widehat{L}_2, \widehat{S}_2, \widehat{E}_2, \widehat{MNAMES}_2, \widehat{REC}_2, \widehat{CG}_2) \\ \iff \quad & \widehat{R}_1 \sqsubseteq_R \widehat{R}_2 \ \wedge \\ & \widehat{K}_1 \sqsubseteq_K \widehat{H}_2 \ \wedge \\ & \widehat{H}_1 \sqsubseteq_H \widehat{H}_2 \ \wedge \\ & \widehat{I}_1 \sqsubseteq_I \widehat{I}_2 \ \wedge \\ & \widehat{JH}_1 \sqsubseteq_H \widehat{JH}_2 \ \wedge \\ & \widehat{C}_1 \sqsubseteq_C \widehat{C}_2 \ \wedge \\ & \widehat{L}_1 \sqsubseteq_L \widehat{L}_2 \ \wedge \\ & \widehat{S}_1 \sqsubseteq_S \widehat{S}_2 \ \wedge \\ & \widehat{E}_1 \sqsubseteq_E \widehat{E}_2 \ \wedge \\ & \widehat{MNAMES}_1 \sqsubseteq_{MNAMES} \widehat{MNAMES}_2 \ \wedge \\ & \widehat{REC}_1 \sqsubseteq_{REC} \widehat{REC}_2 \ \wedge \\ & \widehat{CG}_1 \sqsubseteq_{CG} \widehat{CG}_2\end{aligned}$$

and by Proposition 4.2.1 and Paragraph 2.15 of [DP02], this is simply the product/pointwise extension of the ordering on the underlying analysis domains. $\sqsubseteq_{Analysis}$ allows analysis results to be compared for the same program, and in particular to determine whether one analysis result is "smaller" than another wrt the ordering on $\langle \widehat{\text{Analysis}}, \sqsubseteq_{Analysis} \rangle$.

B.3 Analysis Specification.

The base control-flow analysis is specified using the constraint-based, specification-oriented and implementation-agnostic flow-logic framework of Nielson and Nielson [NN02]. Being a specification-oriented approach, the flow-logic framework is used to specify what it means for an analysis result (or rather a *proposed* analysis result) to be acceptable (correct) with respect to a program.

The judgements of the flow-logic specification for the analysis of Carmel are of the form:

$$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models^{k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY}_{\texttt{Base-CFA}} (m_n, pc_n) : \texttt{instr}$$

where it is implicit:

- $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \in \widehat{\text{Analysis}}$;
- `Base-CFA` is the analysis name;
- m_n.*instructionAt*$(pc_n) =$ `instr`;
- User input parameters:
  - `k` is the maximum abstract context length i.e. the maximum number of stack frames from the concrete callstack to include in the abstract context, as explained in Section 4.2.2.3;
  - `MAX_MOD_COUNT` is the maximum number of times an abstract number may change the numeric interval it contains before it is mapped to the well-defined minimum and maximum values the number may hold according to its type *t* i.e. $(\bot_t, \top_t, \texttt{MAX_MOD_COUNT})$, as explained in Section 4.2.1.2;
  - `MAX_DOM_DYN_ARRAY` is the maximum size of the domain of a dynamic array created via the `new` τ`[]`, as explained in Section 4.2.2.2.

Intuitively the above states that the left-hand side is an *acceptable* analysis for the instruction `instr` at address m_n.*instructionAt*(pc_n) when analysed in any context $\pi \in \widehat{C}(m_n, pc_n)$ and wrt the user supplied parameters

k, MAX_MOD_COUNT, MAX_DOM_DYN_ARRAY.

To give the reader a basic grounding in the flow-logic framework and how to read the rules, we shall discuss in detail the flow-logic rules for the following Carmel bytecode instructions:

- `nop`
- `new` τ
- `if` t $cmpop$ `goto` *addr*
- `invokevirtual` w

One of the strengths of the framework is that it is quite intuitive for those familiar with pattern-matching.

B.3.1 The `nop` instruction

The operational semantic rule for the `nop` instruction is:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{nop} \\ pc = m_n.\mathit{nextAddress}(pc_n) \end{array}}{\begin{array}{l} P \vdash \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle\!\rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc), L_n, S_n \rangle\!\rangle \rangle\!\rangle \end{array}}$$

From the semantic rule we see:

- the callstack remains the same length – only the program counter for the top stack frame changes from (m_n, pc_n) to (m_n.*nextAddress*(pc_n));
- the JCVM transitions from (m_n, pc_n) to (m_n.*nextAddress*(pc_n)) with no changes to R, K, H, I, HID, JH
- the JCVM transitions from (m_n, pc_n) is (m_n.*nextAddress*(pc_n)) with no changes to the operand stack S_n or local variable array L_n

In Table 4.3, we detail a line-by-line explanation of the flow-logic for the `nop` bytecode instruction.

B.3.2 The `new` τ instruction

In Tables 4.4–4.5 we detail an explanation of the flow-logic for the `new` τ bytecode instruction, and where appropriate a cross-comparison between the flow-logic rule and the corresponding operational semantic rule.

B.3.3 The `if` t $cmpop$ `goto` *addr* instruction

To demonstrate the pattern-matching nature of many of the flow-logic rules, consider the `if` t $cmpop$ `goto` *addr* bytecode instruction shown on page 139:

- the possible successor addresses and contexts of (m_n, pc_n) are bound to π_2 and π_3 in the way familiar from the `nop` and `new` τ examples discussed in great detail in Tables 4.3 and 4.4. π_2 corresponds to the `false` branch and π_3 corresponds to the `true` branch;
- the abstract operand stack is bound and constrained, via:

  $$\widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2$$

  to be a sequence of at least 2 elements, with the top-most element of the abstract operand stack bound to variable X_2, the second-from-top element bound to variable X_1 and the remainder (which may be the empty sequence) to M;
- search for all possible type-matching operands to the `if` t $cmpop$ `goto` *addr* is pattern-matched, and variables bound, in the lines:

  $$\forall\, \{(t, Y_1, (m_p, pc_p))\} \sqsubseteq_S X_1 :$$

  $$\forall\, \{(t, Y_2, (m_q, pc_q))\} \sqsubseteq_S X_2 :$$

  and then passed as parameters to the abstract version of the *applyBinary* operator:

  $$\big(\textit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{true}\}\big)$$

  $$\big(\textit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{false}\}\big)$$

  Due to the approximation introduced by our representation, the abstract version of *applyBinary* has to be able to produce both `true` and `false` for the same question, as, e.g. in abstract numbers, there may be particular pairs of values i_1 and i_2, $l_1 \leq i_1 \leq h_1$, $l_2 \leq i_2 \leq h_2$ in the intervals (l_1, h_1) and (l_2, h_2) for which i_1 $cmpop$ i_2 is `true` and other pairs of values such that i_1 $cmpop$ i_2 is `false`. For abstract object references, we have maybe-equal-and-definitely-not-equal semantics: when the fields of the abstract object references are identical, they may refer to the same object, and so the abstract version of the *applyBinary* operator produces both `true` and `false`; where at least one field differs between abstract object references, they definitely do not refer to the same object and so the abstract version of the *applyBinary* operator produces `false`. See Lemma B.5.10 on page 434 for more information;
- when *absApplyBinary* determines `false` is a possible value, then the context of the false branch is repre-

sented in the analysis: $\{\pi_2\} \sqsubseteq_C \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$

We ensure the remaining stack M is represented in the analysis: $M \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$ and in the same way familiar from the `nop` and `new` τ examples discussed in great detail in Tables 4.3 and 4.4, we ensure the local variable array. set of methods that may have invoked m_n, and object state changes recorded in the transaction buffer, and the transition from π_1 to π_2 are represented in the analysis results:

$\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$

- similarly, when *absApplyBinary* determines `true` is a possible value, then the context of the true branch is represented in the analysis: $\{\pi_3\} \sqsubseteq_C \widehat{C}((m_n, \mathit{addr}))$

  We ensure the remaining stack M is represented in the analysis: $M \sqsubseteq_S \widehat{S}(m_n, \mathit{addr})(\pi_3)$
  and in the same way familiar from the `nop` and `new` τ examples discussed in great detail in Tables 4.3 and 4.4, we ensure the local variable array. set of methods that may have invoked m_n, and object state changes recorded in the transaction buffer, and the transition from π_1 to π_3 are represented in the analysis results:

  $\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, \mathit{addr})(\pi_3)$

  $\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, \mathit{addr})(\pi_3)$

  $\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, \mathit{addr})(\pi_3)$

  $\{(\pi_1, \pi_3)\} \sqsubseteq_{CG} \widehat{CG}$

B.3.4 The `invokevirtual` w bytecode instruction

In Tables 4.6–4.7 we detail an explanation of the flow-logic for the `invokevirtual` w bytecode instruction, and where appropriate a cross-comparison between the flow-logic rule and the corresponding operational semantic rule.

Table B.3: Explanation of flow-logic rule for the `nop` bytecode instruction

Explanation of each flow-logic line	Flow-logic line
For each context/JCVM machine configuration π_1 in which the `nop` instruction may be executed	$\forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$
The next instruction/address will be executed, and its abstract context π_2 may be derived from π_1 simply by changing the last abstract stack frame's program counter to $m_n.\mathit{nextAddress}(pc_n)$)	$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$
Ensure next instruction/address will be recorded in the analysis results in the appropriate context/JCVM machine configuration	$\{\pi_2\} \sqsubseteq_C \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
Ensure the operand stack at the current address and in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{S}(m_n, pc_n)(\pi_1) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the local variable array at the current address and in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of provisional object field and array updates in the transaction buffer in current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of callers to this method in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the transition between the current context/JCVM machine configuration and the next instruction/address in the appropriate context/JCVM machine configuration is recorded.	$\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$

Table B.4: Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 1

Explanation of each flow-logic line	Flow-logic line	Corresponding section from op sem rule
For each context/JCVM machine configuration π_1 in which the `new` τ instruction may be executed	$\forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n) :$	
The next instruction/address will be executed, and its abstract context π_2 may be derived from π_1 simply by changing the last abstract stack frame's program counter to $m_n.\mathit{nextAddress}(pc_n)$)	$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, m_n.\mathit{nextAddress}(pc_n))))$	
Ensure next instruction/address will be recorded in the analysis results in the appropriate context/JCVM machine configuration	$\{\pi_2\} \sqsubseteq_C \hat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$	
Create abstract object reference	$\mathsf{O}_q = \left(\mathtt{r}, \left(\begin{array}{l} \mathit{type} : \tau, \\ \mathit{refType} : \tau, \\ \mathit{isArray} : \mathtt{false}, \\ \mathit{owner} : O_n.\mathit{owner}, \\ \mathit{entryPoint} : \mathtt{no}, \\ \mathit{isGlobal} : \mathtt{false}, \\ \mathit{transient} : \mathtt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : (m_n, pc_n), \\ \mathit{length} : 0, \end{array} \right), (m_n, pc_n) \right)$	$a \in \mathsf{Object}$ $a.\mathit{type} = \tau$ $a.\mathit{refType} = \tau$ $a.\mathit{isArray} = \mathtt{false}$ $a.\mathit{owner} = \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}$ $a.\mathit{entryPoint} = \mathtt{no}$ $a.\mathit{isGlobal} = \mathtt{false}$ $a.\mathit{transient} = \mathtt{NOT_TRANSIENT}$ $a.\mathit{creationPoint} = (m_n, pc_n)$ $a.\mathit{length} = 0$ $a.\mathit{heapID} = HID$
Initialise object field to default values, in the appropriate object heap	$\forall\, f \in \mathit{instanceFields}(\tau) :$ $v = \begin{cases} \widehat{\sigma}_{\mathtt{Null}}, f.\mathit{type} \in \mathsf{ReferenceType} \\ (\mathtt{b}, (d,d,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{byte}, f.\mathtt{hasInitValue} = \mathtt{true} \wedge f.\mathtt{initValue} = d \\ (\mathtt{b}, (0,0,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{byte}, f.\mathtt{hasInitValue} = \mathtt{false} \\ (\mathtt{s}, (d,d,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{short}, f.\mathtt{hasInitValue} = \mathtt{true} \wedge f.\mathtt{initValue} = d \\ (\mathtt{s}, (0,0,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{short}, f.\mathtt{hasInitValue} = \mathtt{false} \\ (\mathtt{i}, (d,d,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{int}, f.\mathtt{hasInitValue} = \mathtt{true} \wedge f.\mathtt{initValue} = d \\ (\mathtt{i}, (0,0,0), (m_n, pc_n)), f.\mathit{type} = \mathtt{int}, f.\mathtt{hasInitValue} = \mathtt{false} \end{cases}$ $(\xi_n = 0) \;\Rightarrow\; \{v\} \sqsubseteq_H \widehat{H}(O_q)(\mathit{values})(f.id)$ $(\xi_n = 1) \;\Rightarrow\; \{v\} \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})(f.id)$	$\mathit{loc} \notin \mathit{dom}(H)$ $\mathit{loc} \notin \mathit{dom}(I)$ $\mathit{loc} \notin \mathit{dom}(JH)$ $\mathit{loc} \neq \mathtt{null}$ $dom(a.\mathit{values}) = \{f.\mathit{id} \mid f \in \mathit{instanceFi}$ $\forall\, f \in \mathit{instanceFields}(\tau) :$ $a.\mathit{values}(f.\mathit{id}) = \begin{cases} (\mathtt{r}, \mathtt{null}, (m_n, pc_n) \\ (\mathtt{b}, d, (m_n, pc_n)), \\ (\mathtt{b}, 0, (m_n, pc_n)), \\ (\mathtt{s}, d, (m_n, pc_n)), \\ (\mathtt{s}, 0, (m_n, pc_n)), \\ (\mathtt{i}, d, (m_n, pc_n)), \\ (\mathtt{i}, 0, (m_n, pc_n)), \end{cases}$ $(ctd_n = 0) \;\Rightarrow$ $H' = H[\mathit{loc} \mapsto a]$ $JH' = JH$ $(ctd_n = 1) \;\Rightarrow$ $H' = H$ $JH' = JH[\mathit{loc} \mapsto a]$
Ensure the abstract object reference is pushed on top of a copy of the operand stack at the current address and in the current context/JCVM machine configuration and this is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{S}(m_n, pc_n)(\pi_1)::\{O_q\} \;\sqsubseteq_S\; \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$	$S_n::(\mathtt{r}, \mathit{loc}, (m_n, pc_n))$

Table B.5: Explanation of flow-logic rule for the `new` τ bytecode instruction – Part 2

Explanation of each flow-logic line	Flow-logic line
Ensure the local variable array at the current address and in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{L}(m_n, pc_n)(\pi_1) \sqsubseteq_L \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of provisional object field and array updates in the transaction buffer in current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the set of callers to this method in the current context/JCVM machine configuration is represented in the next instruction/address in the appropriate context/JCVM machine configuration	$\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
Ensure the transition between the current context/JCVM machine configuration and the next instruction/address in the appropriate context/JCVM machine configuration is recorded.	$\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$

Table B.6: Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 1

Explanation of each flow-logic line	Flow-logic line	Corresponding section from op sem rule
For each context/JCVM machine configuration π_1 in which the `invokevirtual w` instruction may be executed	$\forall\, \pi_1 = ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots,$ $(O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \sqsubseteq_C \widehat{C}(m_n, pc_n)$:	
Determine the arity of the method to be invoked	$(\tau_i)_1^q \rightarrow \tau_r = w.\mathit{type}$	$(\tau_i)_1^q \rightarrow \tau_r = w.\mathit{type}$
Pattern-match the target of the method invocation and its potential operands	$\widehat{S}(m_n, pc_n)(\pi_1) = M :: X_0 :: X_1 :: \ldots :: X_q$	$S_n = S' :: (\mathtt{r}, \mathit{loc}, (m_{r_0}, pc_{r_0})) :: (t_1, v_1, (m_{r_1}, pc_{r_1}))$
Check target for `null` or invalidated object references and throw `java.lang.NullPointerException` if so	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0$: $(\widehat{\sigma}_{\mathtt{Null}} = O_q \vee \{O_q\} \sqsubseteq_I \widehat{I}) \Rightarrow$ $\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\mathtt{NullPointerException}})$	$(\mathit{loc} = \mathtt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \Rightarrow$ $SF' = \mathit{catchException}\left(\begin{array}{l}\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (\\ \\ (\mathtt{r}, \mathit{loc}_{\mathtt{NullPointerExce}}\end{array}\right.$
For object references for which the method invocation is well-defined, check the firewall security predicate	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0$: $p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$ $((p \neq \bot) \wedge (p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS})) \Rightarrow$ $\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right)$	$(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \Rightarrow$ $p = \mathit{methodLookup}(w.\mathit{id}, \mathtt{getJHorH}(\mathit{loc}).\mathit{refType})$ $p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS}$ $\mathit{checkInvokeVirtual}\left(\begin{array}{l} \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} \end{array}\right.$
For object references for which the method invocation is well-defined and the firewall security predicate passes, check whether the method call is a recursive call and if so, ensure it is registered in the analysis	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0$: $p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$ $((p \neq \bot) \wedge (p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS})) \Rightarrow$ $\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$ $(\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_n, pc_n)(\pi_1)) \Rightarrow \{((m_n, pc_n), p, \pi_1)\} \sqsubseteq_{RECURSIVE} \widehat{REC}$	
For object references for which the method invocation is well-defined, if the firewall security predicate fails throw `java.lang.SecurityException`	$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X_0$: $p = \mathit{methodLookup}(w.\mathit{id}, O_q.\mathit{type})$ $((p \neq \bot) \wedge (p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS})) \Rightarrow$ $\neg\mathit{checkInvokeVirtual}\left(\begin{array}{l} O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q.\mathit{isArray}, O_q.\mathit{isGlobal}, \\ \\ O_q.\mathit{entryPoint}, O_q.\mathit{transient} \end{array}\right) \Rightarrow$ $\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\mathtt{SecurityException}})$	$(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \Rightarrow$ $p = \mathit{methodLookup}(w.\mathit{id}, \mathtt{getJHorH}(\mathit{loc}).\mathit{refType})$ $p \notin \mathtt{SEPARATELY_HANDLED_API_METHODS}$ $\neg\mathit{checkInvokeVirtual}\left(\begin{array}{l} \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{entryPoint}, \\ \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} \end{array}\right.$ $SF' = \mathit{catchException}\left(\begin{array}{l}\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1 \\ \\ (\mathtt{r}, \mathit{loc}_{\mathtt{SecurityExcep}}\end{array}\right.$

Table B.7: Explanation of flow-logic rule for the `invokevirtual` w bytecode instruction – Part 2

Explanation of each flow-logic line	Flow-logic line	Corresponding section from op sem rule
For object references for which the method invocation is well-defined and the firewall security predicate passes, ensure the method call is added to the set of possible calling methods, along with those possibly calling the current method	$\pi_3 = newAbsContext(\pi_1, k, p, O_q)$ $\{p\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$ $\widehat{MNAMES}(m_n, pc_n)(\pi_1) \sqsubseteq_{MNAMES} \widehat{MNAMES}(p, p.\mathit{firstAddress})(\pi_3)$	
Ensure first address of invoked method will be recorded in the analysis results in the appropriate context/JCVM machine configuration	$\{\pi_3\} \sqsubseteq_C \widehat{C}(p, p.\mathit{firstAddress})$	$F = \langle \mathit{loc}, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle$ $SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, i$
Ensure the parameters to the invoked method are represented in the local variable array of the first address of the invoked method in the appropriate context/JCVM machine configuration	$\{O_q\} \sqsubseteq_L \widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(0)$ $\forall j \in \{1, \ldots, q\}:$ $\forall \{(t_j, W_j, (\mathit{m}_j, pc_j))\} \sqsubseteq_S X_j:$ $(\tau_j = \mathtt{r} = t_j) \Rightarrow \{(t_j, W_j, (\mathit{m}_j, pc_j))\} \sqsubseteq_L \widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$ $(\tau_j \neq \mathtt{r}) \Rightarrow \{(t_j, W_j, (p, p.\mathit{firstAddress}))\} \sqsubseteq_L \widehat{L}(p, p.\mathit{firstAddress})(\pi_3)(j)$	$dom(L') = \{0, \ldots, q\}$ $L'(0) = (\mathtt{r}, \mathit{loc}, (\mathit{m}_{r_0}, pc_{r_0}))$ $L'(i) = \begin{cases} (t_i, v_i, (\mathit{m}_{r_i}, pc_{r_i})), & \text{if } t_i = \mathtt{r} \\ (t_i, v_i, (p, p.\mathit{firstAddress})), & \text{otherwise} \end{cases}$
Ensure the empty stack is represented in the operand stack of the first address of the invoked method in the appropriate context/JCVM machine configuration	$\{\epsilon\} \sqsubseteq_S \widehat{S}(p, p.\mathit{firstAddress})(\pi_3)$	$F = \langle \mathit{loc}, ctd_n, ctd_n, io_n, (p, p.\mathit{firstAddress}), L', \epsilon \rangle$
Ensure the set of provisional object field and array updates in the transaction buffer in current context/JCVM machine configuration is represented in the first address of the invoked method in the appropriate context/JCVM machine configuration	$\widehat{JH}(m_n, pc_n)(\pi_1) \sqsubseteq_{JH} \widehat{JH}(p, p.\mathit{firstAddress})(\pi_3)$	
Ensure the transition between the current context/JCVM machine configuration and the next instruction/address in the appropriate context/JCVM machine configuration is recorded.	$\{(\pi_1, \pi_3)\} \sqsubseteq_{CG} \widehat{CG}$	

B.4 Theoretical Properties

In this section two fundamental properties of the base control-flow analysis and its results are formally stated and proved:

- semantic soundness
- a Moore family property

Semantic soundness is used to establish the correctness of the program analysis i.e. that the analysis results do indeed correctly reflect all semantically possible runtime behaviours for the properties of interest being captured, here control-flow. The Moore family property is used to prove all programs may be analysed, and that a least/smallest analysis exists for each program.

B.4.1 Semantic Soundness

As per [Han05, NNH10], the semantic soundness of a flow-logic analysis with a small-steps semantics is proved by establishing a *subject reduction property*. Before doing so, we must formally relate the concrete- and abstract-domains. Again, following [Han05, NNH10], we do so using representation functions and correctness relations. We have already defined the representation functions relating the concrete and abstract and analysis domains in the opening sections of this Chapter.

We express two correctness relations.

The first correctness relation relates the callstack to the analysis results and includes the context-sensitive domains. $SF\,\mathcal{R}^{R,H,JH,k}_{\mathrm{CallStack}}\,(\widehat{JH},\widehat{C},\widehat{L},\widehat{S},\widehat{MNAMES},\widehat{CG})$ requires each stack frame, including the operand stack, local variable array, and set of calling methods in the callstack to be represented in the analysis, and the transaction

buffer to be represented in in the current address (m_n, pc_n), to be considered correct:

$$SF\ \mathcal{R}^{R,H,JH,k}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$$

$$\iff \quad SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$\wedge \quad \beta^{H,JH}_{DynamicHeap}(\mathit{JH}) \sqsubseteq_{JH} \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF))$$

$$\wedge \quad \forall i \in \{3, \ldots, n\} :$$

$$\begin{array}{llll} & \pi_i & = & \beta^{R,H,JH,k}_{Context}\big(\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle\big) \\ \wedge & \{\pi_i\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_i, pc_i) \\ \wedge & \{\mathit{m}_1, \ldots, \mathit{m}_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_i, pc_i)(\pi_i) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(\mathit{m}_i, pc_i)(\pi_i) \\ \wedge & \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(\mathit{m}_i, pc_i)(\pi_i) \end{array}$$

The second correctness relation relates the global entities (i.e. the non-context-sensitive entities) to the analysis results, and requires the applet registry, global heap, static heap and invalidated object references be represented in the analysis results to be considered correct. It also invokes the first correctness relation and so requires the callstack to be represented in the analysis results to be considered correct.

$$\langle R, K, H, I, HID, JH, CHN, SF \rangle\ \mathcal{R}^{R,H,JH,k}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$$

$$\iff \quad SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$$

$$\begin{array}{lll} \wedge & \beta^{R,H,JH}_{Registry}(\mathit{R}) & \sqsubseteq_R \ \widehat{R} \\ \wedge & \beta^{H}_{StaticHeap}(\mathit{K}) & \sqsubseteq_K \ \widehat{K} \\ \wedge & \beta^{H,JH}_{DynamicHeap}(\mathit{H}) & \sqsubseteq_H \ \widehat{H} \\ \wedge & \beta^{R,H,JH}_{Invalidated}(I) & \sqsubseteq_I \ \widehat{I} \end{array}$$

A final preliminary before expressing and proving the subject reduction property: a concrete semantic configuration is said to be *well-formed* if it is either an initial configuration as per Section 3.7.1 or is reachable from an initial configuration by repeated application of the configuration transitions $P \Big| \begin{matrix} Config \Rightarrow \\ Config \end{matrix}$ of Section 3.9. This ensures the callstack is well-formed i.e. all the stack frames below the stack frame at the top of the callstack are suspended method invocations.

B.4.2 Subject reduction theorem

Theorem B.4.1 (Subject Reduction Theorem)**.** Let:

- $P \in$ Program
- $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$
- C = $\langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P \Big| \begin{matrix} C \Rightarrow \\ C' \end{matrix}$
- C' = $\langle R', K', H', I', HID', JH', CHN', SF' \rangle$

Then:

$$\begin{array}{llll} & C & \mathcal{R}^{R,H,JH}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \Rightarrow \\ & C' & \mathcal{R}^{R,H,JH}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \\ \wedge & \{(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF'))\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

Proof:

By case inspection. Due to time-constraints we have not been able to type up the proofs for all the atomic bytecode instructions, though we can supply 'photocopies on request. We have typed up 17 of the atomic bytecode instructions below.

B.4.2.1 Case `nop`:

By assumption:

$$\frac{\mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{nop}}{{}_P\Big|\ \begin{array}{l}\langle R, K, H, I, HID, JH, CHN, SF\rangle \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle\end{array}}$$

whence:

$$SF = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$$

$$SF' = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S\rangle$$

$$S = S_n$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll}
 & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `nop` the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$\widehat{S}(m_n, pc_n)(\pi_1) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

Combining terms:

$$\begin{array}{lll} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n) \\ & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \\ & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

B.4.2.2 Case `push` *t c*:

By assumption:

$$\frac{\begin{array}{lll} m_n.\mathit{instructionAt}(pc_n) & = & \texttt{push } t \ c \\ t \in \{\texttt{b}, \texttt{s}\} & \Rightarrow & t_2 = \texttt{s} \\ t = \texttt{i} & \Rightarrow & t_2 = \texttt{i} \\ t = \texttt{r} & \Rightarrow & t_2 = \texttt{r} \end{array}}{P \left| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF \rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF' \rangle \end{array} \right.}$$

whence:

$$SF = \langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$SF' = \langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$$

$$S = S_n :: (t_2, c, (m_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll}
\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `push` t c the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n :: (t_2, c, (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$\begin{array}{llll}
(t = \texttt{r}) & \Rightarrow \widehat{S}(m_n, pc_n)(\pi_1) :: \{\widehat{\sigma}_{\texttt{Null}}\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
(t \in \{\texttt{b}, \texttt{s}\}) & \Rightarrow \widehat{S}(m_n, pc_n)(\pi_1) :: \{(\texttt{s}, (c, c, 0), (m_n, pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
(t = \texttt{i}) & \Rightarrow \widehat{S}(m_n, pc_n)(\pi_1) :: \{(\texttt{i}, (c, c, 0), (m_n, pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}$$

which can readily be seen as the expanded form of:

$$\widehat{S}(m_n, pc_n)(\pi_1) :: \beta^{R,H,JH}_{Val}(t_2, c, (m_n, pc_n)) \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

Combining terms:

$$\begin{array}{rcl}
\beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n{::}(t_2, c, (m_n, pc_n))) \\
& = & \beta^{R,H,JH}_{Stack}(S_n){::}\beta^{R,H,JH}_{Val}(t_2, c, (m_n, pc_n))) \\
& \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1){::}\beta^{R,H,JH}_{Val}(t_2, c, (m_n, pc_n))) \\
& \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}$$

and the result follows.

B.4.2.3 Case `pop` *p*:

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{pop}\ \ n \\ n \in \mathbb{N}_0 \\ S_n = T_2 : T_1 \\ n = \mathit{nbWords}(T_1) \end{array}}{P \left| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle \ \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array} \right.}$$

whence:

$$SF = {\scriptstyle \langle\!\langle loc_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1,\, pc_1),\, L_1,\, S_1\rangle\!\rangle :: \langle\!\langle loc_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2,\, pc_2),\, L_2,\, S_2\rangle\!\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle}$$

$$SF' = {\scriptstyle \langle\!\langle loc_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1,\, pc_1),\, L_1,\, S_1\rangle\!\rangle :: \langle\!\langle loc_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2,\, pc_2),\, L_2,\, S_2\rangle\!\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S\rangle}$$

$$S = T_2$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lll}
\exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `pop` *p* the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(T_2) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

complicated by having to prove the conditional is also satisfied:

$$\Big\{P_1, \ldots, P_f\Big\} = \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big)$$

$$\forall\, i \in \{1, \ldots, f\} :$$

$$\big(P_i = M : X \wedge \mathit{absNbWords}(X) = n\big) \;\Rightarrow$$

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(T_2) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

By definition of $\texttt{SINGLE_VALUE_STACKS}\big()$[4]:

$$\begin{aligned}\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \;\Rightarrow\; & \exists \quad Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big) \,.\, \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \; Z \\ & \wedge \quad |Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big) \,.\, \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \; Z| \geq 1\end{aligned}$$

and so there exists P_i such that:

$$\begin{gathered}\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S P_i \wedge P_i = M : X \wedge \mathit{absNbWords}(X) = n \;\Rightarrow \\ \beta^{R,H,JH}_{Stack}(T_2) \sqsubseteq_S M \wedge M \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \;\Rightarrow \\ \beta^{R,H,JH}_{Stack}(T_2) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\end{gathered}$$

Combining terms:

$$\begin{aligned}\beta^{R,H,JH}_{Stack}(S) \;&=\; \beta^{R,H,JH}_{Stack}(T_2) \\ &\sqsubseteq_S \; M \\ &\sqsubseteq_S \; \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\end{aligned}$$

and the result follows.

B.4.2.4 Case `dup` *p q*:

By assumption:

[4] See Section B.5.1 for further information.

$$
\frac{\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{dup}\ \ p\ \ q \\
p, q \in \mathbb{N}_0 \\
S_n = T_2 : T_1 \\
S_n = T_4 : T_3 \\
\mathit{nbWords}(T_1) = p \\
\mathit{nbWords}(T_3) = q \\
S = T_4 : T_1 : T_3
\end{array}}{{}_P\Big|\ \begin{array}{l}\langle R, K, H, I, HID, JH, CHN, SF\rangle \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle\end{array}}
$$

whence:

$$SF = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$$

$$SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S\rangle$$

$$S = T_4 : T_1 : T_3$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lccl}
 & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `dup` *p q* the proof obligation reduces to:

$$
\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(T_4 : T_1 : T_3) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
$$

complicated by having to prove the conditional is also satisfied:

$$
\Big\{P_1, \ldots, P_f\Big\} = \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big)
$$

$$
\forall\, i \in \{1, \ldots, f\} :
$$

$$
\left(\begin{array}{lll}
(P_i & = X_2 : X_1) & \wedge \\
(P_i & = X_4 : X_3) & \wedge \\
(\mathit{absNbWords}(X_1) & = p) & \wedge \\
(\mathit{absNbWords}(X_3) & = q) &
\end{array}\right) \Rightarrow
$$

$$
\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(T_4 : T_1 : T_3) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
$$

By definition of $\texttt{SINGLE_VALUE_STACKS}\Big(\Big)$[5]:

$$
\begin{array}{rcll}
\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \ \Rightarrow & \exists & Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big)\ .\ \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S\ Z \\
& \wedge & |Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big)\ .\ \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S\ Z| \geq 1
\end{array}
$$

and so there exists P_i such that:

[5] See Section B.5.1 for further information.

$\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S P_i \;\Rightarrow$

$(P_i = X_2 : X_1 \wedge \mathit{absNbWords}(X_1) = p) \wedge (P_i = X_4 : X_3 \wedge \mathit{absNbWords}(X_3) = q) \;\Rightarrow$

$\beta^{R,H,JH}_{Stack}(T_1) \sqsubseteq_S X_1 \wedge$

$\beta^{R,H,JH}_{Stack}(T_2) \sqsubseteq_S X_2 \wedge$

$\beta^{R,H,JH}_{Stack}(T_3) \sqsubseteq_S X_3 \wedge$

$\beta^{R,H,JH}_{Stack}(T_4) \sqsubseteq_S X_4 \wedge$

$X_4 : X_1 : X_3 \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \;\Rightarrow$

$\beta^{R,H,JH}_{Stack}(T_4)::\beta^{R,H,JH}_{Stack}(T_1)::\beta^{R,H,JH}_{Stack}(T_3) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \;\Rightarrow$

$\beta^{R,H,JH}_{Stack}(T_4::T_1::T_3) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Combining terms:

$$\begin{array}{rcl} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(T_4::T_1::T_3) \\ & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

B.4.2.5 Case `swap` p_1 p_2:

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{swap}\ \ p_1\ \ p_2 \\ p_1, p_2 \in \mathbb{N}_0 \\ S_n = T_3 : T_2 : T_1 \\ p_1 = \mathit{nbWords}(T_1) \\ p_2 = \mathit{nbWords}(T_2) \\ S = T_3 : T_1 : T_2 \end{array}}{P \left| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle \;\Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array} \right.}$$

whence:

$$SF = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$$

$$S = T_3 : T_1 : T_2$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll}
\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `swap` p_1 p_2 the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(T_3 : T_1 : T_2) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

complicated by having to prove the conditional is also satisfied:

$$\Big\{P_1,\ldots,P_f\Big\} = \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n,pc_n)(\pi_1)\Big)$$

$$\forall\, i \in \{1,\ldots,f\}:$$

$$\left(\begin{array}{lll} (P_i & = X_3 : X_2 : X_1) & \wedge \\ (\textit{absNbWords}(X_1) & = p_1) & \wedge \\ (\textit{absNbWords}(X_2) & = p_2) & \end{array}\right) \Rightarrow$$

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(T_3 : T_1 : T_2) \sqsubseteq_S \widehat{S}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2)$$

By definition of $\texttt{SINGLE_VALUE_STACKS}\Big(\Big)$[6]:

$$\begin{array}{rcl} \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \widehat{S}(m_n,pc_n)(\pi_1) \;\Rightarrow\; & \exists & Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n,pc_n)(\pi_1)\Big)\;.\;\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S\; Z \\ & \wedge & |Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n,pc_n)(\pi_1)\Big)\;.\;\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S\; Z| \geq 1 \end{array}$$

and so there exists P_i such that:

$$\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S P_i \;\Rightarrow$$

$$((P_i = X_3 : X_2 : X_1) \wedge (\textit{absNbWords}(X_1) = p_1) \wedge (\textit{absNbWords}(X_2) = p_2)) \;\Rightarrow$$

$$\beta^{R,H,JH}_{Stack}(T_1) \sqsubseteq_S X_1 \wedge$$

$$\beta^{R,H,JH}_{Stack}(T_2) \sqsubseteq_S X_2 \wedge$$

$$\beta^{R,H,JH}_{Stack}(T_3) \sqsubseteq_S X_3 \wedge$$

$$X_3 : X_1 : X_2 \sqsubseteq_S \widehat{S}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2) \;\Rightarrow$$

$$\beta^{R,H,JH}_{Stack}(T_3)::\beta^{R,H,JH}_{Stack}(T_1)::\beta^{R,H,JH}_{Stack}(T_2) \sqsubseteq_S \widehat{S}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2) \;\Rightarrow$$

$$\beta^{R,H,JH}_{Stack}(T_3 : T_1 : T_2) \sqsubseteq_S \widehat{S}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2)$$

Combining terms:

$$\begin{array}{rcl} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(T_3 : T_1 : T_2) \\ & \sqsubseteq_S & \widehat{S}(m_n, m_n.\textit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

[6] See Section B.5.1 for further information.

B.4.2.6 Cases numop *t* *binop* and numop *t* *binop* t_{opt}:

B.4.2.6.1 Possible transition 1 of 2

By assumption:

$$\frac{\begin{array}{l}\big((m_n.\mathit{instructionAt}(pc_n) = \texttt{numop}\ \ t\ \ binop) \vee (m_n.\mathit{instructionAt}(pc_n) = \texttt{numop}\ \ t\ \ binop\ \ t_{opt})\big) \\ t \in \{\texttt{s}, \texttt{i}\} \\ S_n = A{::}(t, c_1, (m_p, pc_p)){::}(t, c_2, (m_q, pc_q)) \\ \neg\big(binop \in \{\texttt{div}, \texttt{rem}\} \wedge c_2 = 0\big) \\ (t, r, (m_n, pc_n)) = \mathit{applyBinary}(binop, (t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q)), (m_n, pc_n))\end{array}}{{}_P\Big|\ \begin{array}{l}\langle R, K, H, I, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle\end{array}}$$

whence:

$$SF = \text{《}loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$$

$$SF' = \text{《}loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S\rangle$$

$$S = A{::}(t, r, (m_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for $\texttt{numop}\ \ t\ \ binop$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A::(t, r, (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$\begin{array}{l} \widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2 \\ \forall\, \{(t, (l_1, h_1, mod_1), (m_a, pc_a))\} \sqsubseteq_S X_1 : \\ \quad \forall\, \{(t, (l_2, h_2, mod_2), (m_b, pc_b))\} \sqsubseteq_S X_2 : \\ \quad\quad \neg(binop \in \{\texttt{div}, \texttt{rem}\} \wedge (l_2 = 0 = h_2)) \ \Rightarrow \\ \\ \quad\quad\quad (t, (l_3, h_3, mod_3), (m_n, pc_n)) \quad = \quad absApplyBinary(binop, (t, (l_1, h_1, mod_1), (m_a, pc_a)), (t, (l_2, h_2, mod_2), (m_b, pc_b)), (m_n, pc_n)) \\ \quad\quad\quad M::\{(t, (l_3, h_3, mod_3), (m_n, pc_n))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \\ \quad\quad\quad (binop \in \{\texttt{div}, \texttt{rem}\} \wedge (l_2 \leq 0 \leq h_2)) \ \Rightarrow \\ \quad\quad\quad \mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{ArithmeticException}}) \end{array}$$

Now:

$$\begin{array}{lll} \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2 \ \Rightarrow \\ \beta^{R,H,JH}_{Stack}((t, c_1, (m_p, pc_p))) & \sqsubseteq_S & X_1 \wedge \\ \beta^{R,H,JH}_{Stack}((t, c_2, (m_q, pc_q))) & \sqsubseteq_S & X_2 \wedge \\ \beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M \ \Rightarrow \\ \exists\, \{(t, (l_1, h_1, mod_1), (m_a, pc_a))\} \sqsupseteq_{Val} \beta_{Num}((t, c_1, (m_p, pc_p))) & \sqsubseteq_{Val} & X_1 \wedge \\ \exists\, \{(t, (l_2, h_2, mod_2), (m_b, pc_b))\} \sqsupseteq_{Val} \beta_{Num}((t, c_2, (m_q, pc_q))) & \sqsubseteq_{Val} & X_2 \ \Rightarrow \\ \neg(l_2 = 0 = h_2) & & \end{array}$$

We have from the op. sem. rule, by assumption: $\neg(binop \in \{\texttt{div}, \texttt{rem}\})$ and utilising Lemma B.5.7 on page 430:

$$\neg(l_2 = 0 = h_2) \ \wedge \neg\big(binop \in \{\texttt{div}, \texttt{rem}\}\big) \quad \Rightarrow$$

$$\neg\big(binop \in \{\texttt{div}, \texttt{rem}\} \wedge (l_2 = 0 = h_2)\big) \quad \Rightarrow$$

$$M{::}\{(t, (l_3, h_3, mod_3), (\mathit{m}_n, pc_n))\} \quad \sqsubseteq_S \quad \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \ \Rightarrow$$

$$\beta^{R,H,JH}_{Stack}(A) :: \beta_{Num}((t, r, (\mathit{m}_n, pc_n))) \quad \sqsubseteq_S \quad \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \ \Rightarrow$$

$$\beta^{R,H,JH}_{Stack}(A :: (t, r, (\mathit{m}_n, pc_n))) \quad \sqsubseteq_S \quad \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

Combining terms:

$$\begin{array}{rcl} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(A :: (t, r, (\mathit{m}_n, pc_n))) \\ & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

B.4.2.6.2 Possible transition 2 of 2

By assumption:

$$\big((\mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{numop} \ \ t \ \ binop) \vee (\mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{numop} \ \ t \ \ binop \ \ t_{opt})\big)$$

$t \in \{\texttt{s}, \texttt{i}\}$

$\mathit{S}_n = A{::}(t, c_1, (\mathit{m}_p, pc_p)){::}(t, c_2, (\mathit{m}_q, pc_q))$

$\big(binop \in \{\texttt{div}, \texttt{rem}\} \wedge c_2 = 0\big)$

$$\mathit{SF'} = \mathit{catchException}\left(\begin{array}{l} SF, (\texttt{r}, \mathit{loc}_{\texttt{ArithmeticException}}, (1, 1)), \\ \\ \texttt{java.lang.ArithmeticException}, ctd_n, io_n \end{array}\right)$$

$$\frac{}{{}_P\Big| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle \ \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = {}_{\langle\!\langle \mathit{loc}_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1,\, pc_1),\, L_1,\, S_1\rangle\!\rangle} :: {}_{\langle\!\langle \mathit{loc}_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2,\, pc_2),\, L_2,\, S_2\rangle\!\rangle} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{s}_n\rangle$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$, from the flow-logic rule, we have:

$$\begin{array}{l} \widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2 \\ \forall\, \{(t, (l_1, h_1, mod_1), (m_a, pc_a))\} \sqsubseteq_S X_1 : \\ \quad \forall\, \{(t, (l_2, h_2, mod_2), (m_b, pc_b))\} \sqsubseteq_S X_2 : \\ \quad\quad \neg(binop \in \{\texttt{div}, \texttt{rem}\} \wedge (l_2 = 0 = h_2)) \ \Rightarrow \\ \\ \quad\quad\quad (t, (l_3, h_3, mod_3), (m_n, pc_n)) \quad = \quad absApplyBinary(binop, (t, (l_1, h_1, mod_1), (m_a, pc_a)), (t, (l_2, h_2, mod_2), (m_b, pc_b)), (m_n, pc_n)) \\ \quad\quad\quad M::\{(t, (l_3, h_3, mod_3), (m_n, pc_n))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \\ \quad\quad\quad (binop \in \{\texttt{div}, \texttt{rem}\} \wedge (l_2 \leq 0 \leq h_2)) \ \Rightarrow \\ \quad\quad\quad \mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{ArithmeticException}}) \end{array}$$

Now:

$$\begin{array}{lll} \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2 \ \Rightarrow \\ \beta^{R,H,JH}_{Stack}((t, c_1, (m_p, pc_p))) & \sqsubseteq_S & X_1 \wedge \\ \beta^{R,H,JH}_{Stack}((t, c_2, (m_q, pc_q))) & \sqsubseteq_S & X_2 \wedge \\ \beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M \ \Rightarrow \\ \exists\, \{(t, (l_1, h_1, mod_1), (m_a, pc_a))\} \sqsupseteq_{Val} \beta_{Num}((t, c_1, (m_p, pc_p))) & \sqsubseteq_{Val} & X_1 \wedge \\ \exists\, \{(t, (l_2, h_2, mod_2), (m_b, pc_b))\} \sqsupseteq_{Val} \beta_{Num}((t, c_2, (m_q, pc_q))) & \sqsubseteq_{Val} & X_2 \ \Rightarrow \\ (l_2 \leq 0 \leq h_2) & & \end{array}$$

We have from the op. sem. rule, by assumption: $(binop \in \{\texttt{div}, \texttt{rem}\})$ and utilising Lemma B.5.7 on page 430:

$$\begin{array}{l} ((l_2 \leq 0 \leq h_2) \ \wedge (binop \in \{\texttt{div}, \texttt{rem}\})) \quad \Rightarrow \\ \mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{ArithmeticException}}) \end{array}$$

and the result follows from Lemma B.5.1 on page 407.

B.4.2.7 Cases `numop` *t* $unop$:

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{numop}\ \ t\ \ \texttt{neg} \\ t \in \{\texttt{s}, \texttt{i}\} \\ \mathit{S}_n = A::(t, c, (\mathit{m}, apc)) \\ (t, v, (\mathit{m}_n, pc_n)) = \mathit{applyUnary}(\texttt{neg}, (t, c, (\mathit{m}, apc)), (\mathit{m}_n, pc_n)) \end{array}}{{}_P\Big| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$$

$$SF' = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), \mathit{L}_n, S\rangle$$

$$S = A::(t, v, (\mathit{m}_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `numop` t $unop$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A{::}(t, v, (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$$\begin{array}{l} \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X \\ \forall\, \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \sqsubseteq_S X : \\ \quad \begin{array}{lcl} (t, (l_3, h_3, mod_3), (m_n, pc_n)) & = & absApplyUnary(\texttt{neg}, (t, (l_1, h_1, mod_1), (m_p, pc_p)), (m_n, pc_n)) \\ M{::}\{(t, (l_3, h_3, mod_3), (m_n, pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array} \end{array}$$

Now:

$$\begin{array}{lll} \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}((t, c, (m, apc))) & \sqsubseteq_S & X\ \wedge \\ \beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M\ \Rightarrow \\ \exists\, \{(t, (l_1, h_1, mod_1), (m_a, pc_a))\} \sqsupseteq_{Val} \beta_{Num}((t, c, (m, apc))) & \sqsubseteq_{Val} & X \end{array}$$

Utilising Lemma B.5.7 on page 430:

$$\begin{array}{lll} M{::}\{(t, (l_3, h_3, mod_3), (m_n, pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}(A) :: \beta_{Num}((t, v, (m_n, pc_n))) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}(A :: (t, v, (m_n, pc_n))) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

Combining terms:

$$\begin{array}{lll} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(A :: (t, v, (m_n, pc_n))) \\ & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

B.4.2.8 Cases `numop` t `to` t_{opt}:

By assumption:

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{numop } t \texttt{ to } t_{opt} \\
t \in \{\texttt{s}, \texttt{i}\} \\
S_n = A::(t, c, (m, apc)) \\
(t_{opt}, v, (m_n, pc_n)) = \mathit{applyUnary}(\texttt{to}, t_{opt}, (t, c, (m, apc)), (m_n, pc_n))
\end{array}
$$

$$
\frac{}{P \left| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF \rangle \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF' \rangle \end{array} \right.}
$$

whence:

$$SF = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$SF' = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$$

$$S = A::(t_{opt}, v, (m_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `numop` t $unop$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A::(t_{opt}, v, (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$\widehat{S}(m_n, pc_n)(\pi_1) = M::X$

$\forall\, \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \sqsubseteq_S X :$

$$\begin{array}{lcl} (t_{opt}, (l_3, h_3, mod_3), (m_n, pc_n)) & = & absApplyUnary(\texttt{to}, t_{opt}, (t, (l_1, h_1, mod_1), (m_p, pc_p)), (m_n, pc_n), \text{MAX_MOD_COUNT}) \\ M::\{(t_{opt}, (l_3, h_3, mod_3), (m_n, pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

Now:

$$\begin{array}{lcl} \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M::X\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}((t, c, (m, apc))) & \sqsubseteq_S & X\ \wedge \\ \beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M\ \Rightarrow \\ \exists\, \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \sqsupseteq_{Val} \beta_{Num}((t, c, (m, apc))) & \sqsubseteq_{Val} & X \end{array}$$

Utilising Lemma B.5.9 on page 433:

$$\begin{array}{lcl} M::\{(t_{opt}, (l_3, h_3, mod_3), (m_n, pc_n))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}(A) :: \beta_{Num}((t_{opt}, v, (m_n, pc_n))) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}(A :: (t_{opt}, v, (m_n, pc_n))) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

Combining terms:

$$\begin{array}{lcl} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(A::(t_{opt}, v, (m_n, pc_n))) \\ & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

B.4.2.9 Case `load` *t i*:

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{load } t \ j \\ j \in \mathbb{N}_0 \\ t \in \{\texttt{s}, \texttt{i}, \texttt{r}\} \\ (t, c, (m_n, pc_a)) = L_n(j) \end{array}}{{}_P\Big| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = \text{《}loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$$

$$SF' = \text{《}loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S\rangle$$

$$S = S_n :: (t, c, (m_n, pc_a))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lccl} & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_L & \widehat{L}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `load` t i the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n{::}(t, c, (m_n, pc_a))) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

By assumption:

$$\begin{array}{lcll} \beta^{R,H,JH}_{LocalVar}(L_n) & = & M\ . & \\ & & \forall\, i \in \mathit{dom}(L_n) : & \\ & & \beta^{R,H,JH}_{Val}(L_n(i)) & \sqsubseteq_{Val}\ M(i) \\ & & & \sqsubseteq_{Val}\ \widehat{L}(m_n, pc_n)(\pi_1)(i) \\ & \Rightarrow & \beta^{R,H,JH}_{Val}(L_n(j)) & \sqsubseteq_{Val}\ \widehat{L}(m_n, pc_n)(\pi_1)(j) \\ & \Rightarrow & \beta^{R,H,JH}_{Val}((t, c, (m_n, pc_a))) & \sqsubseteq_{Val}\ \widehat{L}(m_n, pc_n)(\pi_1)(j) \end{array}$$

From the flow-logic rule, we have:

$$\forall\, \{(t, Y, (m_p, pc_p))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(j) :$$

$$\widehat{S}(m_n, pc_n)(\pi_1){::}\{(t, Y, (m_p, pc_p))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

Combining the above:

$$\begin{array}{lll} \exists\, \{(t, Y, (m_p, pc_p))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(j)\ .\ (t, Y, (m_p, pc_p)) & \sqsupseteq_{Val} & \beta^{R,H,JH}_{Val}((t, c, (m_n, pc_a)))\ \Rightarrow \\ \widehat{S}(m_n, pc_n)(\pi_1){::}\{(t, Y, (m_p, pc_p))\} & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}(S_n){::}\beta^{R,H,JH}_{Val}((t, c, (m_n, pc_a))) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}(S_n{::}(t, c, (m_n, pc_a))) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

and the result follows.

B.4.2.10 Case `store` *t i*:

By assumption:

$$\frac{\begin{array}{lll} m_n.\mathit{instructionAt}(pc_n) & = & \texttt{store } t \ j \\ t \in \{\texttt{s},\texttt{i},\texttt{r}\} & & \\ j \in \mathbb{N}_0 & & \\ S_n = A::(t, v, (m, apc)) & & \\ \big(t \in \{\texttt{s},\texttt{i}\} & \Rightarrow & (m_q = m_n \wedge pc_q = pc_n)\big) \\ \big(t = \texttt{r} & \Rightarrow & (m_q = m \wedge pc_q = apc)\big) \end{array}}{{}_P\Big| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = {}_{\langle\!\langle}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1{}_{\rangle\!\rangle} :: {}_{\langle\!\langle}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2{}_{\rangle\!\rangle} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$$

$$SF' = {}_{\langle\!\langle}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1{}_{\rangle\!\rangle} :: {}_{\langle\!\langle}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2{}_{\rangle\!\rangle} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L, S\rangle$$

$$S = A$$

$$L = L_n[j \mapsto (t, v, (m_q, pc_q))]$$

The form of C and C' meet the preconditions for Lemma B.5.5 on page 420 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `store` t i the proof obligation reduces to:

$$\begin{array}{lcll} & \beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

From the flow-logic rule, we have:

$$\begin{array}{l} \widehat{S}(m_n, pc_n)(\pi_1) = M :: X \\ \forall\, \{(t, Y, (m_p, pc_p))\} \sqsubseteq_S X : \\ \quad \begin{array}{llll} M & & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \forall\, k \in \mathit{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \neq j: & \widehat{L}(m_n, pc_n)(\pi_1)(k) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k) \\ (t = \texttt{r})\ \Rightarrow & \{(t, Y, (m_p, pc_p))\} & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \\ (t \in \{\texttt{s}, \texttt{i}\})\ \Rightarrow & \{(t, Y, (m_n, pc_n))\} & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \end{array} \end{array}$$

Now:

$$
\begin{array}{lll}
\beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X \;\Rightarrow \\
\beta^{R,H,JH}_{Stack}((t, v, (m, apc))) & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M \;\Rightarrow \\
\exists\, \{(t, Y, (m_p, pc_p))\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t, v, (m, apc))) & \sqsubseteq_{Val} & X \;\Rightarrow \\
M & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \wedge \\
\forall\, k \in \mathit{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \neq j : \widehat{L}(m_n, pc_n)(\pi_1)(k) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k) \wedge \\
(t = \mathtt{r}) \;\Rightarrow\; \{(t, Y, (m_p, pc_p))\} & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \wedge \\
(t \in \{\mathtt{s}, \mathtt{i}\}) \;\Rightarrow\; \{(t, Y, (m_n, pc_n))\} & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \;\Rightarrow \\
(t \in \{\mathtt{s}, \mathtt{i}\}) \;\Rightarrow\; \beta^{R,H,JH}_{Val}((t, v, (m_n, pc_n))) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \wedge \\
(t = \mathtt{r}) \;\Rightarrow\; \beta^{R,H,JH}_{Val}((t, v, (m, apc))) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j)
\end{array}
$$

Combining the above:

$$
\begin{array}{lll}
\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M \\
 & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

From inspection of the op. sem. rule: $L = L_n \left[j \mapsto \begin{cases} (t, v, (m_n, pc_n)) & t \in \{\mathtt{s}, \mathtt{i}\} \\ (t, v, (m, apc)) & t = \mathtt{r} \end{cases} \right]$

Given we have:

$$
\begin{array}{lll}
\beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_L & \widehat{L}(m_n, pc_n)(\pi_1) \wedge \\
\forall\, k \in \mathit{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big)\ .\ k \neq j : \widehat{L}(m_n, pc_n)(\pi_1)(k) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k) \wedge \\
(t \in \{\mathtt{s}, \mathtt{i}\}) \;\Rightarrow\; \beta^{R,H,JH}_{Val}((t, v, (m_n, pc_n))) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \wedge \\
(t = \mathtt{r}) \;\Rightarrow\; \beta^{R,H,JH}_{Val}((t, v, (m, apc))) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \;\Rightarrow \\
 & & \\
\beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

the result follows.

B.4.2.11 Case `inc` *t j c*:

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{inc}\ \ t\ \ j\ \ c \\ t \in \{\texttt{s}, \texttt{i}\} \\ j \in \mathbb{N}_0 \\ c \in \mathbb{Z} \\ (t, a, (m, apc)) = L_n(j) \\ (t, r, (m_n, pc_n)) = \mathit{applyBinary}(\texttt{add}, (t, c, (m_n, pc_n)), (t, a, (m, apc)), (m_n, pc_n)) \end{array}}{{}_P\Big| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = \text{《}loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle$$

$$SF' = \text{《}loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S\rangle$$

$$S = S_n$$

$$L = L_n[j \mapsto (t, r, (m_n, pc_n))]$$

The form of C and C' meet the preconditions for Lemma B.5.5 on page 420 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll} & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\ \wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `inc t j c` the proof obligation reduces to:

$$\begin{array}{lcll} & \beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \wedge & \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \end{array}$$

By assumption:

$$\begin{array}{lcll} \beta^{R,H,JH}_{LocalVar}(L_n) & = & M\ . & \\ & & \forall\, i \in \mathit{dom}(L_n) : & \\ & & \beta^{R,H,JH}_{Val}(L_n(i)) & \sqsubseteq_{Val} M(i) \\ & & & \sqsubseteq_{Val} \widehat{L}(m_n, pc_n)(\pi_1)(i) \\ & \Rightarrow & \beta^{R,H,JH}_{Val}(L_n(j)) & \sqsubseteq_{Val} \widehat{L}(m_n, pc_n)(\pi_1)(j) \\ & \Rightarrow & \beta^{R,H,JH}_{Val}(t, a, (m, apc)) & \sqsubseteq_{Val} \widehat{L}(m_n, pc_n)(\pi_1)(j) \end{array}$$

and so we know :

$$\exists\, \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(j)\ .\ \beta_{Num}(t, a, (m, apc)) \sqsubseteq_{Val} \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\}$$

From the flow-logic rule, we have:

$$\begin{array}{lcl} \forall\, \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(j) : & & \\ \quad (t, (l_3, h_3, mod_3), (m_n, pc_n)) & = & absApplyBinary(\texttt{add}, (t, (l_1, h_1, mod_1), (m_p, pc_p)), (t, (c, c, 0), (m_n, pc_n)), (m_n, pc_n)) \\ \quad \widehat{S}(m_n, pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\ \quad \forall\, k \in \mathit{dom}(\widehat{L}(m_n, pc_n)(\pi_1))\ .\ k \neq j : \widehat{L}(m_n, pc_n)(\pi_1)(k) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k) \\ \quad \{(t, (l_3, h_3, mod_3), (m_n, pc_n))\} & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \end{array}$$

and so we may conclude:

$\exists\, \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(j) :$

$$
\begin{array}{lcl}
(t, (l_3, h_3, mod_3), (m_n, pc_n)) & = & absApplyBinary(\mathsf{add}, (t, (l_1, h_1, mod_1), (m_p, pc_p)), (t, (c, c, 0), (m_n, pc_n)), (m_n, pc_n)) \\
\widehat{S}(m_n, pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\forall\, k \in \mathit{dom}(\widehat{L}(m_n, pc_n)(\pi_1)) \,.\, k \neq j : \widehat{L}(m_n, pc_n)(\pi_1)(k) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k) \\
\{(t, (l_3, h_3, mod_3), (m_n, pc_n))\} & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j)
\end{array}
$$

which immediately gives us:

$$
\begin{array}{lcl}
\beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \\
 & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

and utilising Lemma B.5.7 on page 430:

$$
\beta_{Num}(t, r, (m_n, pc_n))) \quad \sqsubseteq_{Val} \quad \{(t, (l_3, h_3, mod_3), (m_n, pc_n))\}
$$

and combining with:

$$
\begin{array}{lcl}
\beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_L & \widehat{L}(m_n, pc_n)(\pi_1) \wedge \\
\forall\, k \in \mathit{dom}\big(\widehat{L}(m_n, pc_n)(\pi_1)\big) \,.\, k \neq j : \widehat{L}(m_n, pc_n)(\pi_1)(k) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(k) \wedge \\
\{(t, (l_3, h_3, mod_3), (m_n, pc_n))\} & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(j) \;\Rightarrow \\
\\
\beta^{R,H,JH}_{LocalVar}(L) = \beta^{R,H,JH}_{LocalVar}(L_n[j \;\mapsto\; (t, r, (m_n, pc_n))]) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

the result follows.

B.4.2.12 Case `goto` $addr$:

By assumption:

$$
\frac{m_n.\mathit{instructionAt}(pc_n) = \texttt{goto}\ \ addr}{P \Big| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle \;\Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array}}
$$

whence:

$$
SF = \langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle\!\rangle :: \langle\!\langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle\!\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n\rangle
$$

$$
SF' = \langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\rangle\!\rangle :: \langle\!\langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\rangle\!\rangle :: \ldots :: \langle loc_n, itd_n, ctd_n, io_n, (m_n, addr), L_n, S\rangle
$$

$$
S = S_n
$$

The form of C and C' meet the preconditions for Lemma B.5.4 on page 419 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, addr) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, addr)(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, addr)(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, addr)(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, addr)(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll}
\exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `goto` $addr$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \widehat{S}(m_n, addr)(\pi_2)$$

From the flow-logic rule, we have:

$$\widehat{S}(m_n, pc_n)(\pi_1) \quad \sqsubseteq_S \quad \widehat{S}(m_n, addr)(\pi_2)$$

Combining terms:

$$\begin{array}{lll}
\beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n) \\
 & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \\
 & \sqsubseteq_S & \widehat{S}(m_n, addr)(\pi_2)
\end{array}$$

and the result follows.

B.4.2.13 Case `if` *t op* `goto` *addr*:

B.4.2.13.1 Possible transition 1 of 2

By assumption:

$$\frac{\begin{array}{l}\mathit{m.instructionAt}(pc_n) = \texttt{if}\ \ t\ \ \mathit{op}\ \ \texttt{goto}\ \ \mathit{addr} \\ t \in \{\texttt{r}, \texttt{s}\} \\ \mathit{S}_n = A{::}(t, c_2, (\mathit{m}_q, pc_q)){::}(t, c_1, (\mathit{m}_p, pc_p)) \\ \mathit{applyBinary}(\mathit{op}, (t, c_1, (\mathit{m}_p, pc_p)), (t, c_2, (\mathit{m}_q, pc_q)), (\mathit{m}_n, pc_n))\end{array}}{{}_P\left|\begin{array}{l}\langle R, K, H, I, HID, JH, CHN, SF\rangle \ \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle\end{array}\right.}$$

whence:

$$SF = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$$

$$SF' = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, addr), \mathit{L}_n, S\rangle$$

$$S = A$$

The form of C and C' meet the preconditions for Lemma B.5.4 on page 419 and so to conclude for this case, we have only to

show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_3 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_3\} & \sqsubseteq_C & \widehat{C}(m_n, addr) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, addr)(\pi_3) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, addr)(\pi_3) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, addr)(\pi_3) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, addr)(\pi_3) \\
\wedge & \{(\pi_1, \pi_3)\} & \sqsubseteq_{CG} & \widehat{CG} \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lll}
\exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \quad \pi_3 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \quad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `if` *t op* `goto` *addr* the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A) \sqsubseteq_S \widehat{S}(m_n, addr)(\pi_3)$$

From the flow-logic rule, we have:

$\widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2$

$\forall\ \{(t, Y_1, (m_p, pc_p))\} \sqsubseteq_S X_1 :$

$\forall\ \{(t, Y_2, (m_q, pc_q))\} \sqsubseteq_S X_2 :$

$\big(\mathit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{true}\}\big)\ \Rightarrow$

$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, \mathit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, \mathit{addr})(\pi_3)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, \mathit{addr})(\pi_3)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, \mathit{addr})(\pi_3)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, \mathit{addr})(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\big(\mathit{absApplyBinary}(cmpop, (t, Y_1, (m_p, pc_p)), (t, Y_2, (m_q, pc_q))) \supseteq \{\texttt{false}\}\big)\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

Now:

$\beta^{R,H,JH}_{Stack}(S_n)$	$\sqsubseteq_S$	$\widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2\ \Rightarrow$
$\beta^{R,H,JH}_{Stack}((t, c_1, (m_p, pc_p)))$	$\sqsubseteq_S$	$X_1 \wedge$
$\beta^{R,H,JH}_{Stack}((t, c_2, (m_q, pc_q)))$	$\sqsubseteq_S$	$X_2 \wedge$
$\beta^{R,H,JH}_{Stack}(A)$	$\sqsubseteq_S$	$M\ \Rightarrow$
$\exists\ \{(t, (l_1, h_1, mod_1), (m_a, pc_a))\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t, c_1, (m_p, pc_p)))$	$\sqsubseteq_{Val}$	$X_1 \wedge$
$\exists\ \{(t, (l_2, h_2, mod_2), (m_b, pc_b))\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t, c_2, (m_q, pc_q)))$	$\sqsubseteq_{Val}$	X_2

From Lemma B.5.10 on page 434, we have

$$\mathit{applyBinary}(binop, (t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q)), (m_n, pc_n)) \in absApplyBinary\left(binop,\ \begin{matrix}\beta^{R,H,JH}_{Val}((t, c_1, (m_p, pc_p))),\\ \beta^{R,H,JH}_{Val}((t, c_2, (m_q, pc_q))), (m_n, pc_n)\end{matrix}\right)$$

and so we may conclude: $M \sqsubseteq_S \widehat{S}(\mathit{m}_n, \mathit{addr})(\pi_3)$

Combining terms:

$$\begin{aligned} \beta^{R,H,JH}_{Stack}(S) &= \beta^{R,H,JH}_{Stack}(A) \\ &\sqsubseteq_S M \\ &\sqsubseteq_S \widehat{S}(\mathit{m}_n, addr)(\pi_3) \end{aligned}$$

and the result follows.

B.4.2.13.2 Possible transition 2 of 2

By assumption:

$$\frac{\begin{array}{l} \mathit{m}.\mathit{instructionAt}(pc_n) = \texttt{if}\ \ \mathit{t}\ \ \mathit{op}\ \ \texttt{goto}\ \ \mathit{addr} \\ \mathit{t} \in \{\texttt{r}, \texttt{s}\} \\ \mathit{S}_n = A{::}(t, c_2, (\mathit{m}_q, pc_q)){::}(t, c_1, (\mathit{m}_p, pc_p)) \\ \texttt{false} = \mathit{applyBinary}(\mathit{op}, (t, c_1, (\mathit{m}_p, pc_p)), (t, c_2, (\mathit{m}_q, pc_q)), (\mathit{m}_n, pc_n)) \end{array}}{{}_P\Big| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle \ \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = {}_{\langle\!\langle \mathit{loc}_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1,\, pc_1),\, L_1,\, S_1\rangle\!\rangle} :: {}_{\langle\!\langle \mathit{loc}_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2,\, pc_2),\, L_2,\, S_2\rangle\!\rangle} ::\ldots:: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$$

$$SF' = {}_{\langle\!\langle \mathit{loc}_1,\, itd_1,\, ctd_1,\, io_1,\, (m_1,\, pc_1),\, L_1,\, S_1\rangle\!\rangle} :: {}_{\langle\!\langle \mathit{loc}_2,\, itd_2,\, ctd_2,\, io_2,\, (m_2,\, pc_2),\, L_2,\, S_2\rangle\!\rangle} ::\ldots:: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)), \mathit{L}_n, S\rangle$$

$$S = A$$

The form of C and C' meet the preconditions for Lemma B.5.4 on page 419 and so to conclude for this case, we

have only to show: whence:

$$SF = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$$

$$S = A::(t, r, (m_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.3 on page 414 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll}
 & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `numop` t $binop$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A) \sqsubseteq_S \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$$

From the flow-logic rule, we have:

$\widehat{S}(m_n, pc_n)(\pi_1) = M::X_1::X_2$

$\forall\ \{(\mathit{t}, Y_1, (\mathit{m}_p, pc_p))\} \sqsubseteq_S X_1 :$

$\forall\ \{(\mathit{t}, Y_2, (\mathit{m}_q, pc_q))\} \sqsubseteq_S X_2 :$

$\big(\mathit{absApplyBinary}(cmpop, (\mathit{t}, Y_1, (\mathit{m}_p, pc_p)), (\mathit{t}, Y_2, (\mathit{m}_q, pc_q))) \supseteq \{\texttt{true}\}\big)\ \Rightarrow$

$\{\pi_3\}$	$\sqsubseteq_C$	$\widehat{C}(\mathit{m}_n, \mathit{addr})$
M	$\sqsubseteq_S$	$\widehat{S}(\mathit{m}_n, \mathit{addr})(\pi_3)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathit{m}_n, \mathit{addr})(\pi_3)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathit{m}_n, \mathit{addr})(\pi_3)$
$M\widehat{NAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$M\widehat{NAMES}(\mathit{m}_n, \mathit{addr})(\pi_3)$
$\{(\pi_1, \pi_3)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\big(\mathit{absApplyBinary}(cmpop, (\mathit{t}, Y_1, (\mathit{m}_p, pc_p)), (\mathit{t}, Y_2, (\mathit{m}_q, pc_q))) \supseteq \{\texttt{false}\}\big)\ \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$M\widehat{NAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$M\widehat{NAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

Now:

$$
\begin{array}{lll}
\beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X_1{::}X_2 \;\Rightarrow \\
\beta^{R,H,JH}_{Stack}((t, c_1, (\mathit{m}_p, pc_p))) & \sqsubseteq_S & X_1 \wedge \\
\beta^{R,H,JH}_{Stack}((t, c_2, (\mathit{m}_q, pc_q))) & \sqsubseteq_S & X_2 \wedge \\
\beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M \;\Rightarrow \\
\exists\, \{(\mathit{t}, (l_1, h_1, mod_1), (\mathit{m}_a, pc_a))\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t, c_1, (\mathit{m}_p, pc_p))) & \sqsubseteq_{Val} & X_1 \wedge \\
\exists\, \{(\mathit{t}, (l_2, h_2, mod_2), (\mathit{m}_b, pc_b))\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t, c_2, (\mathit{m}_q, pc_q))) & \sqsubseteq_{Val} & X_2
\end{array}
$$

From Lemma B.5.10 on page 434, we have

$$
\mathit{applyBinary}(binop, (t, c_1, (\mathit{m}_p, pc_p)), (t, c_2, (\mathit{m}_q, pc_q)), (\mathit{m}_n, pc_n)) \in absApplyBinary\left(binop,\ \begin{array}{l}\beta^{R,H,JH}_{Val}((t, c_1, (\mathit{m}_p, pc_p))),\\ \beta^{R,H,JH}_{Val}((t, c_2, (\mathit{m}_q, pc_q))), (\mathit{m}_n, pc_n)\end{array}\right)
$$

and so we may conclude: $M \sqsubseteq_S \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Combining terms:

$$
\begin{array}{rcl}
\beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(A) \\
& \sqsubseteq_S & M \\
& \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

and the result follows.

B.4.2.14 Case `lookupswitch` *t* $(\mathit{k}_i$ `=>` $apc_i)_1^r$, `default =>` $apc_{default}$:

B.4.2.14.1 Possible transition 1 of 2

By assumption:

$$\frac{\begin{array}{l} \mathit{m}_n.\mathit{instructionAt}(pc_n) = \texttt{lookupswitch}\ \mathit{t}\ \ (\mathit{k}_i\ \texttt{=>}\ apc_i)_1^r\ ,\ \texttt{default =>}\ apc_{default} \\ \mathit{t} \in \{\texttt{s}, \texttt{i}\} \\ k_1, \ldots, k_r \in \mathbb{N}_0 \\ \mathit{S}_n = A::(t, key, (\mathit{m}, apc)) \\ (\exists i \in \{1, \ldots, r\}\ .\ \mathit{key} = \mathit{k}_i) \end{array}}{{}_P\Big|\ \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF'\rangle \end{array}}$$

whence:

$$SF = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, pc_n), \mathit{L}_n, \mathit{S}_n\rangle$$

$$SF' = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (\mathit{m}_n, apc_i), \mathit{L}_n, S\rangle$$

$$S = A$$

The form of C and C' meet the preconditions for Lemma B.5.4 on page 419 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll} & \pi_0 & = & \beta_{Context}^{R,H,JH,k}(SF) \\ \wedge & \pi_i & = & \beta_{Context}^{R,H,JH,k}(SF') \\ \wedge & \{\pi_i\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, apc_i) \\ \wedge & \beta_{Stack}^{R,H,JH}(S) & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, apc_i)(\pi_i) \\ \wedge & \widehat{L}(m_n, pc_n)(\pi_0) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, apc_i)(\pi_i) \\ \wedge & \widehat{JH}(m_n, pc_n)(\pi_0) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, apc_i)(\pi_i) \\ \wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_0) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, apc_i)(\pi_i) \\ \wedge & \{(\pi_0, \pi_i)\} & \sqsubseteq_{CG} & \widehat{CG} \\ \Rightarrow & C' & \mathcal{R}_{\text{Config}}^{R,H,JH,k} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_0 \sqsubseteq_C \widehat{C}_i\ .\ \pi_0 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_i & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(\mathsf{m}_n, pc_n)(\pi_0) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `lookupswitch` t $(\mathsf{k}_i$ `=>` $apc_i)_1^r$, `default =>` $apc_{default}$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A) \sqsubseteq_S \widehat{S}(\mathsf{m}_n, apc_i)(\pi_i)$$

From the flow-logic rule, we have:

$$\begin{array}{l} \widehat{S}(m_n, pc_n)(\pi_0) = M{::}X \\ \forall\ \{(\mathsf{t}, (l, h, mod), (\mathsf{m}_q, pc_q))\} \sqsubseteq_S X : \\ \quad \forall\ p \in \{1, \ldots, r\} : \\ \qquad l \leq k_p \leq h\ \Rightarrow \end{array}$$

$$\begin{array}{lll} \{\pi_p\} & \sqsubseteq_C & \widehat{C}(\mathsf{m}_n, apc_p) \\ M & \sqsubseteq_S & \widehat{S}(\mathsf{m}_n, apc_p)(\pi_p) \\ \widehat{L}(m_n, pc_n)(\pi_0) & \sqsubseteq_L & \widehat{L}(\mathsf{m}_n, apc_p)(\pi_p) \\ \widehat{JH}(m_n, pc_n)(\pi_0) & \sqsubseteq_{JH} & \widehat{JH}(\mathsf{m}_n, apc_p)(\pi_p) \\ \widehat{MNAMES}(m_n, pc_n)(\pi_0) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathsf{m}_n, apc_p)(\pi_p) \\ \{(\pi_0, \pi_p)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

From the op. sem. we have: $(\exists i \in \{1, \ldots, r\}\ .\ \mathsf{key} = \mathsf{k}_i)$. Now:

$\beta^{R,H,JH}_{Stack}(S_n)$	$\sqsubseteq_S$	$\widehat{S}(\mathsf{m}_n, pc_n)(\pi_1) \wedge \widehat{S}_i(\pi_1) = M::X \ \Rightarrow$
$\beta^{R,H,JH}_{Stack}((t, key, (\mathsf{m}, apc)))$	$\sqsubseteq_S$	$X \wedge$
$\beta^{R,H,JH}_{Stack}(A)$	$\sqsubseteq_S$	$M \ \Rightarrow$
$\exists\, \{(\mathsf{t}, (l_1, h_1, mod_1), (\mathsf{m}_q, pc_q))\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t, key, (\mathsf{m}, apc)))$	$\sqsubseteq_{Val}$	$X \ \Rightarrow$

$l \leq key \leq h \ \Rightarrow \ \exists p \in \{1, \ldots, r\} \,.\, l \leq k_p \leq h \ . \ k_p = \mathsf{k}_i \ \Rightarrow$

$\{\pi_i\}$	$\sqsubseteq_C$	$\widehat{C}(\mathsf{m}_n, apc_i)$
M	$\sqsubseteq_S$	$\widehat{S}(\mathsf{m}_n, apc_i)(\pi_i)$
$\widehat{L}(m_n, pc_n)(\pi_0)$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, apc_i)(\pi_i)$
$\widehat{JH}(m_n, pc_n)(\pi_0)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathsf{m}_n, apc_i)(\pi_i)$
$M\widehat{NAM}ES(m_n, pc_n)(\pi_0)$	$\sqsubseteq_{MNAMES}$	$M\widehat{NAM}ES(\mathsf{m}_n, apc_i)(\pi_i)$
$\{(\pi_0, \pi_i)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

and since we have $\beta^{R,H,JH}_{Stack}(A) \sqsubseteq_S M$, the result follows.

B.4.2.14.2 Possible transition 2 of 2

By assumption:

$\mathsf{m}_n.\mathit{instructionAt}(pc_n) = \texttt{lookupswitch}\ \mathsf{t}\ \ (\mathsf{k}_i\ \texttt{=>}\ apc_i)_1^r\ ,\ \texttt{default =>}\ apc_{default}$

$\mathsf{t} \in \{\texttt{s}, \texttt{i}\}$

$k_1, \ldots, k_r \in \mathbb{N}_0$

$\mathsf{S}_n = A::(t, key, (\mathsf{m}, apc))$

$\neg(\exists i \in \{1, \ldots, r\} \,.\, \mathsf{key} = \mathsf{k}_i)$

$$P \Big| \begin{array}{l} \langle R, K, H, I, HID, JH, CHN, SF \rangle \ \Rightarrow \\ \langle R, K, H, I, HID, JH, CHN, SF' \rangle \end{array}$$

whence:

$$SF = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, apc_{default}), L_n, S \rangle$$

$$S = A$$

The form of C and C' meet the preconditions for Lemma B.5.4 on page 419 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_0 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_{default} & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_{default}\} & \sqsubseteq_C & \widehat{C}(m_n, apc_{default}) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, apc_{default})(\pi_{default}) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_0) & \sqsubseteq_L & \widehat{L}(m_n, apc_{default})(\pi_{default}) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_0) & \sqsubseteq_{JH} & \widehat{JH}(m_n, apc_{default})(\pi_{default}) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_0) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, apc_{default})(\pi_{default}) \\
\wedge & \{(\pi_0, \pi_{default})\} & \sqsubseteq_{CG} & \widehat{CG} \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\mathrm{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\mathrm{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll}
\exists\, \pi_0 \sqsubseteq_C \widehat{C}_{default} \,.\, \pi_0 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_{default} & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_0)
\end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `lookupswitch` t $(k_i$ `=>` $apc_{default})_1^r$, `default =>` $apc_{default}$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(A) \sqsubseteq_S \widehat{S}(m_n, apc_{default})(\pi_{default})$$

From the flow-logic rule, we have:

$$\widehat{S}(m_n, pc_n)(\pi_0) = M::X$$

$$\forall\, \{(t, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_S X :$$

$$\forall\, v \in \{l, \ldots, h\} :$$

$$(\nexists\, p \in \{1, \ldots, r\} \,.\, k_p = v) \;\Rightarrow$$

$$\begin{array}{lll}
\{\pi_{default}\} & \sqsubseteq_C & \widehat{C}(m_n, apc_{default}) \\
M & \sqsubseteq_S & \widehat{S}(m_n, apc_{default})(\pi_{default}) \\
\widehat{L}(m_n, pc_n)(\pi_0) & \sqsubseteq_L & \widehat{L}(m_n, apc_{default})(\pi_{default}) \\
\widehat{JH}(m_n, pc_n)(\pi_0) & \sqsubseteq_{JH} & \widehat{JH}(m_n, apc_{default})(\pi_{default}) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_0) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, apc_{default})(\pi_{default}) \\
\{(\pi_0, \pi_{default})\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

From the op. sem. we have: $\neg(\exists i \in \{1, \ldots, r\} \,.\, key = k_i)$. Now:

$$\begin{array}{lll}
\beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}_{default}(\pi_1) = M::X \;\Rightarrow \\
\beta^{R,H,JH}_{Stack}((t, key, (m, apc))) & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(A) & \sqsubseteq_S & M \;\Rightarrow \\
\exists\, \{(t, (l_1, h_1, mod_1), (m_q, pc_q))\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t, key, (m, apc))) & \sqsubseteq_{Val} & X \;\Rightarrow
\end{array}$$

$$l \leq key \leq h \;\Rightarrow\; \neg\exists p \in \{1, \ldots, r\} \,.\, l \leq k_p \leq h \;.\; k_p = k_i \;\Rightarrow$$

$$\begin{array}{lll}
\{\pi_{default}\} & \sqsubseteq_C & \widehat{C}(m_n, apc_{default}) \\
M & \sqsubseteq_S & \widehat{S}(m_n, apc_{default})(\pi_{default}) \\
\widehat{L}(m_n, pc_n)(\pi_0) & \sqsubseteq_L & \widehat{L}(m_n, apc_{default})(\pi_{default}) \\
\widehat{JH}(m_n, pc_n)(\pi_0) & \sqsubseteq_{JH} & \widehat{JH}(m_n, apc_{default})(\pi_{default}) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_0) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, apc_{default})(\pi_{default}) \\
\{(\pi_0, \pi_{default})\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

and since we have $\beta^{R,H,JH}_{Stack}(A) \sqsubseteq_S M$, the result follows.

B.4.2.15 Cases `arraystore` *t*:

B.4.2.15.1 Possible transition 1 of 10

By assumption:

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{arraystore}\ \ t \\
S_n = S{::}(\texttt{r}, \mathit{loc}, (m_p, pc_p)){::}(\texttt{s}, i, (m_q, pc_q)){::}(t_1, v, (m, apc_3)) \\
((\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true})) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\
\mathit{checkArrayStore}\Big(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ \Big) \\
(0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length}) \\
(t = \texttt{b}) \\
((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \\
(\texttt{b}, w, (m, apc_3)) = \mathit{fromShort}((t_1, v, (m, apc_3)), \texttt{b}) \\
H' = H[\mathit{loc}.\mathit{values}(i) \ \mapsto\ (\texttt{b}, w, (m_n, pc_n))] \\
SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle ::\ldots:: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle\!\rangle
\end{array}
$$

$$
{}_P| \quad \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle ::\ldots:: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle\!\rangle \ \Rightarrow \\
\langle\!\langle R, K, H', I, HID, JH, CHN, SF' \rangle\!\rangle
$$

The form of C and C' meet the preconditions for Lemma B.5.6 on page 425 and so to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{Val}(\texttt{b}, w, (m_n, pc_n)) & \sqsubseteq_{Val} & \widehat{H}\big(\beta^{R,H,JH}_{Ref}(\texttt{r}, \mathit{loc}, (m_p, pc_p))\big).\mathit{values}((i, i)) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

Table B.8 gives the relevant excerpt from the flow-logic clause for this instruction. It is worth noting that sections two and three may be expressed as:

$$\forall(start, end) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\textit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)\textit{values})\big) \end{array} \right)\ .\ (start \leq j \leq end) : \langle BLOCK \rangle$$

and:

$$\nexists(D_1, D_2) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\textit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)(\textit{values})\big) \end{array} \right)\ .\ (D_1 \leq j \leq D_2) : \langle BLOCK \rangle$$

and:

- the two conditions are mutually exclusive;
- one of the two conditions must be true;
- the same set of constraints ($\langle BLOCK \rangle$) are to be satisfied regardless of which condition is triggered;

in line with the explanation in Section 4.2.2.2 on how the `arraystore` instruction functions in the abstract analysis i.e. array index i in the concrete semantics is represented as an abstract number $(l, h, mod)\ .\ l \leq i \leq h$ storing the value against an existing exactly-matching array index interval is preferred to adding either a fresh array index interval or storing the value against the default all-elements array index interval, otherwise.

From the operational semantic rule, it must be $t_1 = \mathtt{s} \Leftarrow (\mathtt{b}, w, (\textit{m}, apc_3)) = \textit{fromShort}((t_1, v, (\textit{m}, apc_3)), \mathtt{b})$.

$$
\begin{array}{llll}
\beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1 \;\Rightarrow \\
\beta^{R,H,JH}_{Stack}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & & \sqsubseteq_S & A_1 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & & \sqsubseteq_S & A_2 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M \;\Rightarrow \\
\exists\, \{O_q\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & \sqsubseteq_{Val} & X \wedge \\
\exists\, \{(\mathtt{s}, (l, h, mod), (\mathit{m}_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\
\exists\, \{O_v\} = \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & \sqsubseteq_{Val} & A_2
\end{array}
$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(\mathit{loc}_n)$ and $O_3 = \mathtt{getJHorH}(\mathit{loc}_3)$. Combined with the above:

$$
\mathit{checkArrayStore}\left(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p)), (\mathit{t}_1, v, (\mathit{m}, apc_3)), (t = \mathtt{r})\ \right)
$$

=

$$
\mathit{checkArrayStore}\left(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ \right)
$$

From the op. sem. we have $(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length})$.

Since $\{O_q\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) \wedge O_q.\mathit{length} = (min_array_length, max_array_length) \;\Rightarrow$

$$
\begin{array}{lllll}
min_array_length \leq & 0 & \leq & i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \leq max_array_length \wedge \\
& l & \leq & i < h \;\Rightarrow \\
& & & i \in \{max(0,l), \ldots, min(max_array_length, h)\} \;\Rightarrow \\
& & \exists & j \in \{max(0,l), \ldots, min(max_array_length, h)\} \,.\, i = j
\end{array}
$$

Combining all of the above we may conclude:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
M & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

$$
\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{lll}
(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q)) & = & \mathit{absFromShort}\big((t_v, W, (m_v, pc_v)), \mathtt{b}\big) \\
\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

$$
\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{lll}
(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q)) & = & \mathit{absFromShort}\big((t_v, W, (m_v, pc_v)), \mathtt{b}\big) \\
\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) : \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

Utilising B.5.9[7] on page 430: $\beta_{Num}((\mathtt{b}, w, (m, apc_3))) \sqsubseteq_{Val} \{(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$. From the assumptions of the operational semantics and by definition of $\beta^{R,H,JH,k}_{Context}$,$\gamma_n = ctd_n$, we may conclude:

[7] since *absFromShort* calls $absApplyUnary$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}\big((t_v, W, (m_v, pc_v)), \mathtt{b}\big)$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Combining the above:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\beta^{R,H,JH}_{Stack}(S) \sqsubseteq_S M$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\beta_{Num}((\mathtt{b}, w, (m, apc_3))) \sqsubseteq_{Val} \{(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))\}$	$\sqsubseteq_{Val}$	$\widehat{H}(O_q = \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (m_p, pc_p))))(\mathit{values})((i, i))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

and the result follows.

B.4.2.15.2 Possible transition 2 of 10

By assumption:

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \mathtt{arraystore}\ \ t \\
S_n = S{::}(\mathtt{r}, \mathit{loc}, (m_p, pc_p)){::}(\mathtt{s}, i, (m_q, pc_q)){::}(t_1, v, (m, apc_3)) \\
((\mathit{loc} = \mathtt{null}) \vee (\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray} = \mathtt{true})) \\
(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\
\mathit{checkArrayStore}\Big(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \mathtt{r})\ \Big) \\
(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length}) \\
(t = \mathtt{b})((ctd_n = 1 \wedge \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal}))(\mathtt{b}, w, (m, apc_3)) = \mathit{fromShort}((t_1, v, (m, apc_3)), \mathtt{b}) \\
H' = H \\
JH' = JH[\mathit{loc}.\mathit{values}(i) \ \mapsto\ (\mathtt{b}, w, (m_n, pc_n))] \\
SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle\!\rangle
\end{array}
$$

$$
P\Big|\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \Rightarrow
\langle\!\langle R, K, H', I, HID, CHN, SF' \rangle\!\rangle
$$

$\widehat{S}(m_n, pc_n)(\pi_1) = M::X::A_2::A_1$
$\forall \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X :$

$(O_q.\mathit{isArray}) \Rightarrow$

$O_q.\mathit{length} = (min_array_length, max_array_length)$

$array_dom_size = \left| \begin{array}{c} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})) \\ \cup \\ dom(\widehat{H}(O_q)(\mathit{values})) \end{array} \right|$

$\forall \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_S A_2 :$

$\forall \{O_v\} = \{(t_v, W, (m_v, pc_v))\} \sqsubseteq_S A_1 :$

$\mathit{checkArrayStore}(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ ) \Rightarrow$

$(h \geq 0) \Rightarrow$

$\forall j \in \{max(0, l), \ldots, min(max_array_length, h)\} :$

$\forall (start, end) \in \left(\begin{array}{c} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})) \\ \cup \\ dom(\widehat{H}(O_q)(\mathit{values})) \end{array} \right) . (start \leq j \leq end) :$

$\Big((t = \mathtt{b}) \wedge (t_v = \mathtt{s})\Big) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t_v, W, (m_v, pc_v)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t_v, W, (m_v, pc_v)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) :$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\forall (D_1, D_2) \in \left(\begin{array}{c} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})) \\ \cup \\ dom(\widehat{H}(O_q)(\mathit{values})) \end{array} \right) . (D_1 \leq j \leq D_2) :$

$(start, end) = \begin{cases} (0, max_array_length), & array_dom_size \geq MAX_DOM_DYN_ARRAY, \\ (max(0, l), min(max_array_length, h)), & array_dom_size < MAX_DOM_DYN_ARRAY \end{cases}$

$\Big((t = \mathtt{b}) \wedge (t_v = \mathtt{s})\Big) \Rightarrow$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t_v, W, (m_v, pc_v)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge O_v.\mathit{isGlobal}\Big) \Rightarrow$

$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}((t_v, W, (m_v, pc_v)), \mathtt{b})$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) :$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Table B.8: arraystore t first two cases

$$\widehat{S}(m_n, pc_n)(\pi_1) = M::X::A_2::A_1$$
$$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (m_r, pc_r))\} \sqsubseteq_S X :$$

$$(O_q.\mathit{isArray}) \;\Rightarrow$$

$$O_q.\mathit{length} \quad = \quad (min_array_length, max_array_length)$$

$$array_dom_size \quad = \quad \left| \begin{array}{c} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})) \\ \cup \\ dom(\widehat{H}(O_q)(\mathit{values})) \end{array} \right|$$

$$\forall\, \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} \sqsubseteq_S A_2 :$$

$$\forall\, \{O_v\} = \{(t_v, W, (m_v, pc_v))\} \sqsubseteq_S A_1 :$$

$$\mathit{checkArrayStore}\,(\;\; O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\;\;) \;\Rightarrow$$

$$(h \geq 0) \;\Rightarrow$$

$$\forall j \in \{max(0, l), \ldots, min(max_array_length, h)\} :$$

$$\forall (start, end) \in \left(\begin{array}{c} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})) \\ \cup \\ dom(\widehat{H}(O_q)(\mathit{values})) \end{array} \right) . (start \leq j \leq end) :$$

$$\Big(\big((t = t_v = \mathtt{r}) \wedge \neg O_v.\mathit{isArray} \wedge (O_v.\mathit{type} \preceq O_q.\mathit{type})\big) \vee \big((t = t_v = \mathtt{s})\big) \vee \big((t = t_v = \mathtt{i})\big)\Big) \;\Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$MN\widehat{AM}ES(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$MN\widehat{AM}ES(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \;\Rightarrow$$

$(t = \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q)(\mathit{values})((start, end))$
$(t \neq \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \;\Rightarrow$$

$(t = \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end))$
$(t \neq \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\not\exists (D_1, D_2) \in \left(\begin{array}{c} dom(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})) \\ \cup \\ dom(\widehat{H}(O_q)(\mathit{values})) \end{array} \right) . (D_1 \leq j \leq D_2) :$$

$$(start, end) = \begin{cases} (0, max_array_length), & array_dom_size \geq MAX_DOM_DYN_ARRAY, \\ (max(0, l), min(max_array_length, h)), & array_dom_size < MAX_DOM_DYN_ARRAY \end{cases}$$

$$\Big(\big((t = t_v = \mathtt{r}) \wedge \neg O_v.\mathit{isArray} \wedge (O_v.\mathit{type} \preceq O_q.\mathit{type})\big) \vee \big((t = t_v = \mathtt{s})\big) \vee \big((t = t_v = \mathtt{i})\big)\Big) \;\Rightarrow$$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$MN\widehat{AM}ES(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$MN\widehat{AM}ES(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

$$\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \;\Rightarrow$$

$(t = \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q)(\mathit{values})((start, end))$
$(t \neq \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_H$	$\widehat{H}(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

$$\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \;\Rightarrow$$

$(t = \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end))$
$(t \neq \mathtt{r})$	$\Rightarrow$	$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$			$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Table B.9: arraystore t third until sixth cases

Based on the Lemma B.5.6 on page 425 to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{Val}(\mathtt{b}, w, (m_n, pc_n)) & \sqsubseteq_{Val} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_p, pc_p))\big).\mathit{values}((i,i)) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lllll}
 & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

Table B.8 gives the relevant excerpt from the flow-logic clause for this instruction. It is worth noting that sections two and three may be expressed as:

$$
\forall (start, end) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)\mathit{values})\big) \end{array} \right)\ .\ (start \leq j \leq end) : \langle BLOCK \rangle
$$

and:

$$\nexists(D_1, D_2) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\textit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)(\textit{values})\big) \end{array} \right) . (D_1 \leq j \leq D_2) : \langle BLOCK \rangle$$

and:

- the two conditions are mutually exclusive;
- one of the two conditions must be true;
- the same set of constraints ($\langle BLOCK \rangle$) are to be satisfied regardless of which condition is triggered;

in line with the explanation in Section 4.2.2.2 on how the `arraystore` instruction functions in the abstract analysis i.e. array index i in the concrete semantics is represented as an abstract number (l, h, mod) . $l \leq i \leq h$ storing the value against an existing exactly-matching array index interval is preferred to adding either a fresh array index interval or storing the value against the default all-elements array index interval, otherwise.

From the operational semantic rule, it must be $t_1 = \mathtt{s} \Leftarrow (\mathtt{b}, w, (\textit{m}, apc_3)) = \textit{fromShort}((\textit{t}_1, v, (\textit{m}, apc_3)), \mathtt{b})$.

$\beta^{R,H,JH}_{Stack}(S_n)$		$\sqsubseteq_S$	$\widehat{S}(\textit{m}_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M::X::A_2::A_1 \Rightarrow$
$\beta^{R,H,JH}_{Stack}((\textit{t}_1, v, (\textit{m}, apc_3)))$		$\sqsubseteq_S$	$A_1 \wedge$
$\beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (\textit{m}_q, pc_q)))$		$\sqsubseteq_S$	$A_2 \wedge$
$\beta^{R,H,JH}_{Stack}((\mathtt{r}, \textit{loc}, (\textit{m}_p, pc_p)))$		$\sqsubseteq_S$	$X \wedge$
$\beta^{R,H,JH}_{Stack}(S)$		$\sqsubseteq_S$	$M \Rightarrow$
$\exists \{O_q\}$	$\sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, \textit{loc}, (\textit{m}_p, pc_p)))$	$\sqsubseteq_{Val}$	$X \wedge$
$\exists \{(\mathtt{s}, (l, h, mod), (\textit{m}_q, pc_q))\}$	$\sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (\textit{m}_q, pc_q)))$	$\sqsubseteq_{Val}$	A_1
$\exists \{O_v\} = \{(\textit{t}_v, W, (\textit{m}_v, pc_v))\}$	$\sqsupseteq_{Val}\ \beta_{Num}((\textit{t}_1, v, (\textit{m}, apc_3)))$	$\sqsubseteq_{Val}$	A_2

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(\textit{loc}_n)$ and $O_3 = \mathtt{getJHorH}(\textit{loc}_3)$. Combined with the above:

$$\textit{checkArrayStore}\ \Big(\ \mathtt{getJHorH}(\textit{loc}_n).\textit{owner}, \mathtt{getJHorH}(\textit{loc}_3).\textit{owner}, \mathtt{getJHorH}(\textit{loc}).\textit{owner}, (\mathtt{r}, \textit{loc}, (\textit{m}_p, pc_p)), (\textit{t}_1, v, (\textit{m}, apc_3)), (t = \mathtt{r})\ \Big)$$

=

$$\textit{checkArrayStore}\ \Big(\ O_n.\textit{owner}, O_3.\textit{owner}, O_q.\textit{owner}, O_q, O_v, (t = \mathtt{r})\ \Big)$$

From the op. sem. we have $(0 \leq i < \texttt{getJHorH}(\textit{loc}).\textit{length})$.

Since $\{O_q\} \sqsupseteq_{Val} \beta_{Ref}^{R,H,JH}((\texttt{r}, \textit{loc}, (\textit{m}_p, pc_p))) \wedge O_q.\textit{length} = (min_array_length, max_array_length) \Rightarrow$

$$\begin{array}{rrll} min_array_length \leq & 0 & \leq & i < \texttt{getJHorH}(\textit{loc}).\textit{length} \leq max_array_length \wedge \\ & l & \leq & i < h \Rightarrow \\ & & & i \in \{max(0,l), \ldots, min(max_array_length, h)\} \Rightarrow \\ & & \exists & j \in \{max(0,l), \ldots, min(max_array_length, h)\} \,.\, i = j \end{array}$$

Combining all of the above we may conclude:

$$\begin{array}{lll} \{\pi_2\} & \sqsubseteq_C & \widehat{C}(\textit{m}_n, \textit{m}_n.\textit{nextAddress}(pc_n)) \\ M & \sqsubseteq_S & \widehat{S}(\textit{m}_n, \textit{m}_n.\textit{nextAddress}(pc_n))(\pi_2) \\ \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\textit{m}_n, \textit{m}_n.\textit{nextAddress}(pc_n))(\pi_2) \\ \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\textit{m}_n, \textit{m}_n.\textit{nextAddress}(pc_n))(\pi_2) \\ \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

$$\neg\Big(\gamma_n = 1 \wedge O_q.\textit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\textit{isGlobal}\Big) \Rightarrow$$

$$\begin{array}{lll} (\texttt{b}, (l_2, h_2, mod), (\textit{m}_q, pc_q)) & = & \textit{absFromShort}\big((\textit{t}_v, W, (\textit{m}_v, pc_v)), \texttt{b}\big) \\ \{(\texttt{b}, (l_2, h_2, mod), (\textit{m}_n, pc_n))\} & \sqsubseteq_H & \widehat{H}(O_q)(\textit{values})((start, end)) \\ \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\textit{m}_n, \textit{m}_n.\textit{nextAddress}(pc_n))(\pi_2) \end{array}$$

$$\Big(\gamma_n = 1 \wedge O_q.\textit{transient} = \texttt{NOT_TRANSIENT} \wedge O_v.\textit{isGlobal}\Big) \Rightarrow$$

$$\begin{array}{lll} (\texttt{b}, (l_2, h_2, mod), (\textit{m}_q, pc_q)) & = & \textit{absFromShort}\big((\textit{t}_v, W, (\textit{m}_v, pc_v)), \texttt{b}\big) \\ \{(\texttt{b}, (l_2, h_2, mod), (\textit{m}_n, pc_n))\} & \sqsubseteq_{JH} & \widehat{JH}(\textit{m}_n, \textit{m}_n.\textit{nextAddress}(pc_n))(\pi_2)(O_q)(\textit{values})((start, end)) \\ \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\textit{m}_n, \textit{m}_n.\textit{nextAddress}(pc_n))(\pi_2) \end{array}$$

Utilising B.5.9[8] on page 430: $\beta_{Num}((\texttt{b}, w, (\textit{m}, apc_3))) \sqsubseteq_{Val} \{(\texttt{b}, (l_2, h_2, mod), (\textit{m}_q, pc_q))$. From the assumptions of the operational semantics and by definition of $\beta_{Context}^{R,H,JH,k}, \gamma_n = ctd_n$, we may conclude:

[8] since *absFromShort* calls $absApplyUnary$

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))$	$=$	$\mathit{absFromShort}\big((t_v, W, (m_v, pc_v)), \mathtt{b}\big)$
$\{(\mathtt{b}, (l_2, h_2, mod), (m_n, pc_n))\}$	$\sqsubseteq_{Val}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Combining the above:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\beta^{R,H,JH}_{Stack}(S) \sqsubseteq_S M$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\beta_{Num}((\mathtt{b}, w, (m, apc_3))) \sqsubseteq_{Val} \{(\mathtt{b}, (l_2, h_2, mod), (m_q, pc_q))\}$	$\sqsubseteq_{Val}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((i, i))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

and the result follows.

B.4.2.15.3 Possible transition 3 of 10

By assumption:

$m_n.\mathit{instructionAt}(pc_n) = \texttt{arraystore}\ \ t$
$S_n = S'::(\texttt{r}, \mathit{loc}, (m_p, pc_p))::(\texttt{s}, i, (m_q, pc_q))::(t_1, v, (m, apc_3))$
$((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true}))$
$(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I))$
$\mathit{checkArrayStore}\Big(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ \Big)$
$(0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length})$
$(t \in \{\texttt{s}, \texttt{i}\})$
$((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal}))$
$H' = H[\mathit{loc}.\mathit{values}(i) \ \mapsto\ (t_1, v, (m_n, pc_n))]$
$JH' = JH$
$SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle$

$P\Big|\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle ::\ldots::\langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \Rightarrow$
$\langle\!\langle R, K, H', I, HID, CHN, SF' \rangle\!\rangle$

The form of C and C' meet the preconditions for Lemma B.5.6 on page 425 and so to conclude for this case, we have only to show:

	π_1	$=$	$\beta^{R,H,JH,k}_{Context}(SF)$
$\wedge$	π_2	$=$	$\beta^{R,H,JH,k}_{Context}(SF')$
$\wedge$	$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\wedge$	$\beta^{R,H,JH}_{Stack}(S)$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\wedge$	$\beta^{R,H,JH}_{Val}(t_1, v, (m, apc_3))$	$\sqsubseteq_{Val}$	$\widehat{H}\big(\beta^{R,H,JH}_{Ref}(\texttt{r}, \mathit{loc}, (m_p, pc_p))\big).\mathit{values}((i, i))$
$\wedge$	$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\wedge$	$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\wedge$	$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\wedge$	$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1$	$=$	$\beta^{R,H,JH,k}_{Context}(SF)$
$\wedge$ π_2	$=$	$\beta^{R,H,JH,k}_{Context}(SF')$
$\wedge$ $\beta^{R,H,JH}_{Stack}(S_n)$	$\sqsubseteq_S$	$\widehat{S}(m_n, pc_n)(\pi_1)$

Table B.9 gives the relevant excerpt from the flow-logic clause for this instruction. It is worth noting that sections two and three may be expressed as:

$$\forall(start, end) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\textsf{\textit{values}})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)\textsf{\textit{values}})\big) \end{array} \right) . (start \leq j \leq end) : \langle BLOCK \rangle$$

and:

$$\not\exists(D_1, D_2) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\textsf{\textit{values}})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)(\textsf{\textit{values}})\big) \end{array} \right) . (D_1 \leq j \leq D_2) : \langle BLOCK \rangle$$

and:

- the two conditions are mutually exclusive;
- one of the two conditions must be true;
- the same set of constraints ($\langle BLOCK \rangle$) are to be satisfied regardless of which condition is triggered;

in line with the explanation in Section 4.2.2.2 on how the `arraystore` instruction functions in the abstract analysis i.e. array index i in the concrete semantics is represented as an abstract number (l, h, mod) . $l \leq i \leq h$ storing the value against an existing exactly-matching array index interval is preferred to adding either a fresh array index interval or storing the value against the default all-elements array index interval, otherwise.

From the operational semantic rule, it must be *t* $= t_1 \in \{\texttt{i}, \texttt{s}\}$.

$$
\begin{array}{llll}
\beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1 \;\Rightarrow \\
\beta^{R,H,JH}_{Stack}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & & \sqsubseteq_S & A_1 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & & \sqsubseteq_S & A_2 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M \;\Rightarrow \\
\exists\, \{O_q\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & \sqsubseteq_{Val} & X \wedge \\
\exists\, \{(\mathtt{s}, (l, h, mod), (\mathit{m}_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\
\exists\, \{O_v\} = \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & \sqsubseteq_{Val} & A_2
\end{array}
$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(\mathit{loc}_n)$ and $O_3 = \mathtt{getJHorH}(\mathit{loc}_3)$. Combined with the above:

$$
\mathit{checkArrayStore}\left(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p)), (\mathit{t}_1, v, (\mathit{m}, apc_3)), (t = \mathtt{r})\ \right)
$$

=

$$
\mathit{checkArrayStore}\left(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ \right)
$$

From the op. sem. we have $(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length})$.

Since $\{O_q\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) \wedge O_q.\mathit{length} = (min_array_length, max_array_length) \;\Rightarrow$

$$
\begin{array}{lllll}
min_array_length \leq & 0 & \leq & i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \leq max_array_length \wedge \\
& l & \leq & i < h \;\Rightarrow \\
& & & i \in \{max(0,l), \ldots, min(max_array_length, h)\} \;\Rightarrow \\
& & \exists & j \in \{max(0,l), \ldots, min(max_array_length, h)\} \,.\, i = j
\end{array}
$$

Combining all of the above we may conclude:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
M & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

$$
\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(t = \mathtt{r}) & \Rightarrow \{(t_v, W, (m_v, pc_v))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
(t \neq \mathtt{r}) & \Rightarrow \{(t_v, W, (m_n, pc_n))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

$$
\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(t = \mathtt{r}) & \Rightarrow \{(t_v, W, (m_v, pc_v))\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
(t \neq \mathtt{r}) & \Rightarrow \{(t_v, W, (m_n, pc_n))\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

From the assumptions of the operational semantics including *t* and by definition of $\beta^{R,H,JH,k}_{Context}$,$\gamma_n = ctd_n$, we may conclude:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
M & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
\{(t_v, W, (m_n, pc_n))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

Combining the above:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\beta^{R,H,JH}_{Stack}(S) \sqsubseteq_S M$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\beta_{Num}((t_1, v, (m, apc_3))) \sqsubseteq_{Val} \{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_{Val}$	$\widehat{H}(O_q = \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (m_p, pc_p))))(\mathit{values})((i,i))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

and the result follows.

B.4.2.15.4 Possible transition 4 of 10

By assumption:

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \mathtt{arraystore}\ \ t \\
S_n = S::(\mathtt{r}, \mathit{loc}, (m_p, pc_p))::(\mathtt{s}, i, (m_q, pc_q))::(t_1, v, (m, apc_3)) \\
((\mathit{loc} = \mathtt{null}) \vee (\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray} = \mathtt{true})) \\
(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\
\mathit{checkArrayStore}\Big(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \mathtt{r})\ \Big) \\
(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length}) \\
(t \in \{\mathtt{s}, \mathtt{i}\}) \\
((ctd_n = 1 \wedge \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \\
H' = H \\
JH' = JH[\mathit{loc}.\mathit{values}(i) \mapsto (t_1, v, (m_n, pc_n))] \\
SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle\!\rangle \\
\hline
P \Big| \ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \Rightarrow \\
\qquad \langle\!\langle R, K, H', I, HID, CHN, SF' \rangle\!\rangle
\end{array}
$$

The form of C and C' meet the preconditions for Lemma B.5.6 on page 425 and so to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{Val}(t_1, v, (m_n, pc_n)) & \sqsubseteq_{Val} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_p, pc_p))\big).\mathit{values}((i,i)) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lll}
\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

Table B.9 gives the relevant excerpt from the flow-logic clause for this instruction. It is worth noting that sections two and three may be expressed as:

$$
\forall (start, end) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)\mathit{values}\big) \end{array} \right) .\ (start \leq j \leq end) : \langle BLOCK \rangle
$$

and:

$$
\nexists (D_1, D_2) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)(\mathit{values})\big) \end{array} \right) .\ (D_1 \leq j \leq D_2) : \langle BLOCK \rangle
$$

and:

- the two conditions are mutually exclusive;
- one of the two conditions must be true;
- the same set of constraints ($\langle BLOCK \rangle$) are to be satisfied regardless of which condition is triggered;

in line with the explanation in Section 4.2.2.2 on how the `arraystore` instruction functions in the abstract analysis i.e. array index i in the concrete semantics is represented as an abstract number (l, h, mod) . $l \leq i \leq h$ storing the value against an existing exactly-matching array index interval is preferred to adding either a fresh array index interval or storing the value against the default all-elements array index interval, otherwise.

From the operational semantic rule, it must be $t = t_1 \in \{\mathtt{i}, \mathtt{s}\}$.

$$
\begin{array}{llll}
\beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1 \ \Rightarrow \\
\beta^{R,H,JH}_{Stack}((t_1, v, (m, apc_3))) & & \sqsubseteq_S & A_1 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (m_q, pc_q))) & & \sqsubseteq_S & A_2 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{r}, loc, (m_p, pc_p))) & & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M \ \Rightarrow \\
\exists\, \{O_q\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, loc, (m_p, pc_p))) & \sqsubseteq_{Val} & X \wedge \\
\exists\, \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (m_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\
\exists\, \{O_v\} = \{(t_v, W, (m_v, pc_v))\} & \sqsupseteq_{Val}\ \beta_{Num}((t_1, v, (m, apc_3))) & \sqsubseteq_{Val} & A_2
\end{array}
$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(loc_n)$ and $O_3 = \mathtt{getJHorH}(loc_3)$. Combined with the above:

$$
checkArrayStore\ \Big(\ \mathtt{getJHorH}(loc_n).owner, \mathtt{getJHorH}(loc_3).owner, \mathtt{getJHorH}(loc).owner, (\mathtt{r}, loc, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \mathtt{r})\ \Big)
$$

=

$$
checkArrayStore\ \Big(\ O_n.owner, O_3.owner, O_q.owner, O_q, O_v, (t = \mathtt{r})\ \Big)
$$

From the op. sem. we have $(0 \leq i < \mathtt{getJHorH}(loc).length)$.

Since $\{O_q\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) \wedge O_q.\mathit{length} = (min_array_length, max_array_length) \Rightarrow$

$$
\begin{array}{rcll}
min_array_length \leq & 0 & \leq & i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \leq max_array_length \wedge \\
& l & \leq & i < h \Rightarrow \\
& & & i \in \{max(0,l), \ldots, min(max_array_length, h)\} \Rightarrow \\
& & \exists & j \in \{max(0,l), \ldots, min(max_array_length, h)\} \,.\, i = j
\end{array}
$$

Combining all of the above we may conclude:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\
M & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

$$
\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(\mathit{t} = \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
(\mathit{t} \neq \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_n, pc_n))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

$$
\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(\mathit{t} = \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
(\mathit{t} \neq \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_n, pc_n))\} & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

From the assumptions of the operational semantics including *t* and by definition of $\beta^{R,H,JH,k}_{Context}, \gamma_n = ctd_n$, we may conclude:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Combining the above:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\beta^{R,H,JH}_{Stack}(S) \sqsubseteq_S M$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\beta_{Num}((t_1, v, (m_n, pc_n))) \sqsubseteq_{Val} \{(t_v, W, (m_n, pc_n))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})(i, i)$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

and the result follows.

B.4.2.15.5 Possible transition 5 of 10

By assumption:

$$
\frac{\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \texttt{arraystore}\ \ t \\
S_n = S'::(\texttt{r}, \mathit{loc}, (m_p, pc_p))::(\texttt{s}, i, (m_q, pc_q))::(t_1, v, (m, apc_3)) \\
((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true})) \\
(\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\
\mathit{checkArrayStore}\Big(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ \Big) \\
(0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length}) \\
(t = \texttt{r}) \\
\texttt{getJHorH}(v).\mathit{isArray} = \texttt{false} \\
(\texttt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \texttt{getJHorH}(v).\mathit{refType}) \\
((ctd_n = 0) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{transient} \neq \texttt{NOT_TRANSIENT}) \vee (ctd_n = 1 \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \\
H' = H[\mathit{loc}.\mathit{values}(i) \ \mapsto\ (t_1, v, (m, apc_3))] \\
JH' = JH \\
SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle ::\ldots:: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle
\end{array}}{P\Big|\ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle ::\ldots:: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \ \Rightarrow\ \langle\!\langle R, K, H', I, HID, CHN, SF' \rangle\!\rangle}
$$

The form of C and C' meet the preconditions for Lemma B.5.6 on page 425 and so to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{Val}(t_1, v, (m, apc_3)) & \sqsubseteq_{Val} & \widehat{H}\big(\beta^{R,H,JH}_{Ref}(\texttt{r}, \mathit{loc}, (m_p, pc_p))\big).\mathit{values}((i,i)) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

Table B.9 gives the relevant excerpt from the flow-logic clause for this instruction. It is worth noting that sections two and three may be expressed as:

$$\forall(start, end) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\textit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)\textit{values})\big) \end{array} \right)\ .\ (start \leq j \leq end) : \langle BLOCK \rangle$$

and:

$$\nexists(D_1, D_2) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\textit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)(\textit{values})\big) \end{array} \right)\ .\ (D_1 \leq j \leq D_2) : \langle BLOCK \rangle$$

and:

- the two conditions are mutually exclusive;
- one of the two conditions must be true;
- the same set of constraints ($\langle BLOCK \rangle$) are to be satisfied regardless of which condition is triggered;

in line with the explanation in Section 4.2.2.2 on how the `arraystore` instruction functions in the abstract analysis i.e. array index i in the concrete semantics is represented as an abstract number $(l, h, mod)\ .\ l \leq i \leq h$ storing the value against an existing exactly-matching array index interval is preferred to adding either a fresh array index interval or storing the value against the default all-elements array index interval, otherwise.

From the operational semantic rule, it must be $t = t_1 = \mathtt{r}$.

$$
\begin{array}{llll}
\beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1 \;\Rightarrow \\
\beta^{R,H,JH}_{Stack}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & & \sqsubseteq_S & A_1 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & & \sqsubseteq_S & A_2 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M \;\Rightarrow \\
\exists\, \{O_q\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & \sqsubseteq_{Val} & X \wedge \\
\exists\, \{(\mathtt{s}, (l, h, mod), (\mathit{m}_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\
\exists\, \{O_v\} = \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Val}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & \sqsubseteq_{Val} & A_2
\end{array}
$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(\mathit{loc}_n)$ and $O_3 = \mathtt{getJHorH}(\mathit{loc}_3)$. Combined with the above:

$$
\mathit{checkArrayStore}\left(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p)), (\mathit{t}_1, v, (\mathit{m}, apc_3)), (t = \mathtt{r})\ \right)
$$

$$
=
$$

$$
\mathit{checkArrayStore}\left(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ \right)
$$

From the op. sem. we have $(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length})$.

Since $\{O_q\} \sqsupseteq_{Val} \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) \wedge O_q.\mathit{length} = (min_array_length, max_array_length) \;\Rightarrow$

$$
\begin{array}{lllll}
min_array_length \leq & 0 & \leq & i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \leq max_array_length \wedge \\
& l & \leq & i < h \;\Rightarrow \\
& & & i \in \{max(0,l), \ldots, min(max_array_length, h)\} \;\Rightarrow \\
& & \exists & j \in \{max(0,l), \ldots, min(max_array_length, h)\} \,.\, i = j
\end{array}
$$

Combining all of the above we may conclude:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
M & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

$$
\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(t = \mathtt{r}) & \Rightarrow \{(t_v, W, (m_v, pc_v))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
(t \neq \mathtt{r}) & \Rightarrow \{(t_v, W, (m_n, pc_n))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

$$
\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \texttt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(t = \mathtt{r}) & \Rightarrow \{(t_v, W, (m_v, pc_v))\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
(t \neq \mathtt{r}) & \Rightarrow \{(t_v, W, (m_n, pc_n))\} & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

From the assumptions of the operational semantics including *t* and by definition of $\beta^{R,H,JH,k}_{Context}$,$\gamma_n = ctd_n$, we may conclude:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
M & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
\{(t_v, W, (m_v, pc_v))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

Combining the above:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\beta^{R,H,JH}_{Stack}(S) \sqsubseteq_S M$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\beta^{R,H,JH}_{Val}((t_1, v, (m, apc_3))) \sqsubseteq_{Val} \{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_{Val}$	$\widehat{H}(O_q = \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (m_p, pc_p))))(\mathit{values})((i,i))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

and the result follows.

B.4.2.15.6 Possible transition 6 of 10

By assumption:

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \mathtt{arraystore}\ \ t \\
S_n = S' :: (\mathtt{r}, \mathit{loc}, (m_p, pc_p)) :: (\mathtt{s}, i, (m_q, pc_q)) :: (t_1, v, (m, apc_3)) \\
((\mathit{loc} = \mathtt{null}) \vee (\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray} = \mathtt{true})) \\
(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\
\\
\mathit{checkArrayStore}\Big(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \mathtt{r})\ \Big) \\
(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length}) \\
(t = \mathtt{r}) \\
\mathtt{getJHorH}(v).\mathit{isArray} = \mathtt{false} \\
(\mathtt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \mathtt{getJHorH}(v).\mathit{refType}) \\
((ctd_n = 1 \wedge \mathtt{getJHorH}(\mathit{loc}).\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg \mathtt{getJHorH}(\mathit{loc}).\mathit{isGlobal})) \\
H' = H \\
JH' = JH[\mathit{loc}.\mathit{values}(i) \ \mapsto\ (t_1, v, (m, apc_3))] \\
SF' = \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S' \rangle\!\rangle \\
\hline
P \Big| \ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \Rightarrow \\
\quad \langle\!\langle R, K, H', I, HID, CHN, SF' \rangle\!\rangle
\end{array}
$$

The form of C and C' meet the preconditions for Lemma B.5.6 on page 425 and so to conclude for this case, we have only to show:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{Val}(t_1, v, (m, apc_3)) & \sqsubseteq_{Val} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big(\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_p, pc_p))\big).\mathit{values}((i,i)) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$
\begin{array}{lll}
\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}
$$

Table B.9 gives the relevant excerpt from the flow-logic clause for this instruction. It is worth noting that sections two and three may be expressed as:

$$
\forall (start, end) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)\mathit{values}\big) \end{array} \right) .\ (start \leq j \leq end) : \langle BLOCK \rangle
$$

and:

$$
\nexists (D_1, D_2) \in \left(\begin{array}{c} dom\big(\widehat{JH}(m_n, pc_n)(\pi_1)(O_q)(\mathit{values})\big) \\ \cup \\ dom\big(\widehat{H}(O_q)(\mathit{values})\big) \end{array} \right) .\ (D_1 \leq j \leq D_2) : \langle BLOCK \rangle
$$

and:

- the two conditions are mutually exclusive;
- one of the two conditions must be true;
- the same set of constraints ($\langle BLOCK \rangle$) are to be satisfied regardless of which condition is triggered;

in line with the explanation in Section 4.2.2.2 on how the `arraystore` instruction functions in the abstract analysis i.e. array index i in the concrete semantics is represented as an abstract number (l, h, mod) . $l \leq i \leq h$ storing the value against an existing exactly-matching array index interval is preferred to adding either a fresh array index interval or storing the value against the default all-elements array index interval, otherwise.

From the operational semantic rule, it must be $t = t_1 = \mathtt{r}$.

$$
\begin{array}{llll}
\beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1 \ \Rightarrow \\
\beta^{R,H,JH}_{Stack}((t_1, v, (m, apc_3))) & & \sqsubseteq_S & A_1 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (m_q, pc_q))) & & \sqsubseteq_S & A_2 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{r}, loc, (m_p, pc_p))) & & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M \ \Rightarrow \\
\exists \{O_q\} & \sqsupseteq_{Val} \beta^{R,H,JH}_{Ref}((\mathtt{r}, loc, (m_p, pc_p))) & \sqsubseteq_{Val} & X \wedge \\
\exists \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} & \sqsupseteq_{Val} \beta_{Num}((\mathtt{s}, i, (m_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\
\exists \{O_v\} = \{(t_v, W, (m_v, pc_v))\} & \sqsupseteq_{Val} \beta^{R,H,JH}_{Val}((t_1, v, (m, apc_3))) & \sqsubseteq_{Val} & A_2
\end{array}
$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(loc_n)$ and $O_3 = \mathtt{getJHorH}(loc_3)$. Combined with the above:

$$
checkArrayStore\left(\ \mathtt{getJHorH}(loc_n).owner, \mathtt{getJHorH}(loc_3).owner, \mathtt{getJHorH}(loc).owner, (\mathtt{r}, loc, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \mathtt{r}) \ \right)
$$

=

$$
checkArrayStore\left(\ O_n.owner, O_3.owner, O_q.owner, O_q, O_v, (t = \mathtt{r}) \ \right)
$$

From the op. sem. we have $(0 \leq i < \mathtt{getJHorH}(loc).length)$.

Since $\{O_q\} \sqsupseteq_{Val} \beta_{Ref}^{R,H,JH}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) \wedge O_q.\mathit{length} = (min_array_length, max_array_length) \Rightarrow$

$$
\begin{array}{rcll}
min_array_length \leq & 0 & \leq & i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length} \leq max_array_length \wedge \\
& l & \leq & i < h \Rightarrow \\
& & & i \in \{max(0,l), \ldots, min(max_array_length, h)\} \Rightarrow \\
& & \exists & j \in \{max(0,l), \ldots, min(max_array_length, h)\} \,.\, i = j
\end{array}
$$

Combining all of the above we may conclude:

$$
\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\
M & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

$$
\neg\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(\mathit{t} = \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
(\mathit{t} \neq \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_n, pc_n))\} & \sqsubseteq_H & \widehat{H}(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

$$
\Big(\gamma_n = 1 \wedge O_q.\mathit{transient} = \mathtt{NOT_TRANSIENT} \wedge \neg O_v.\mathit{isGlobal}\Big) \Rightarrow
$$

$$
\begin{array}{llll}
(\mathit{t} = \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
(\mathit{t} \neq \mathtt{r}) \Rightarrow \{(\mathit{t}_v, W, (\mathit{m}_n, pc_n))\} & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end)) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}
$$

From the assumptions of the operational semantics including *t* and by definition of $\beta_{Context}^{R,H,JH,k}, \gamma_n = ctd_n$, we may conclude:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
M	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)(O_q)(\mathit{values})((start, end))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

Combining the above:

$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n))$
$\beta^{R,H,JH}_{Stack}(S) \sqsubseteq_S M$	$\sqsubseteq_S$	$\widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$
$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\beta^{R,H,JH}_{Val}((t_1, v, (m, apc_3))) \sqsubseteq_{Val} \{(t_v, W, (m_v, pc_v))\}$	$\sqsubseteq_{Val}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)\big(O_q = \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_p, pc_p))\big).\mathit{values}((i, i))$
$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)$

and the result follows.

B.4.2.15.7 Possible transition 7 of 10

By assumption:

$$
\begin{array}{l}
m_n.\mathit{instructionAt}(pc_n) = \mathtt{arraystore}\ \ t \\
S_n = S' :: (\mathtt{r}, \mathit{loc}, (m_p, pc_p)) :: (\mathtt{s}, i, (m_q, pc_q)) :: (t_1, v, (m, apc_3)) \\
((\mathit{loc} = \mathtt{null}) \vee (\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \mathtt{getJHorH}(\mathit{loc}).\mathit{isArray} = \mathtt{true})) \\
(\mathit{loc} \neq \mathtt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\
\mathit{checkArrayStore}\Big(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \mathtt{r})\ \Big) \\
(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length}) \\
(t = \mathtt{r}) \\
\mathtt{getJHorH}(v).\mathit{isArray} = \mathtt{false} \\
\neg(\mathtt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \mathtt{getJHorH}(v).\mathit{refType}) \\
\mathit{SF}' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ \\ (\mathtt{r}, \mathit{loc}_{\mathtt{ArrayStoreException}}, (1, 1)), \mathtt{java.lang.ArrayStoreException}, ctd_n, io_n \end{array}\right) \\
\hline
P \Big| \ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle\!\rangle \ \Rightarrow \\
\quad \langle\!\langle R, K, H, I, HID, JH, CHN, \mathit{SF}' \rangle\!\rangle
\end{array}
$$

$\widehat{S}(m_n, pc_n)(\pi_1) = M::X::A_2::A_1$
$\big(\{\widehat{\sigma}_{\texttt{Null}}\} \sqsubseteq_S X\big) \Rightarrow$
$\quad \mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{NullPointerException}})$

$\forall\, \{O_q\} = \{(\mathtt{r}, Y, (\mathit{m}_r, pc_r))\} \sqsubseteq_S X :$

$\big(O_q.\mathit{isArray}\big) \Rightarrow$

$\quad \forall\, \{(\mathtt{s}, (l, h, mod), (\mathit{m}_q, pc_q))\} \sqsubseteq_S A_2 :$

$\quad\quad \forall\, \{O_v\} = \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} \sqsubseteq_S A_1 :$

$\quad\quad \neg \mathit{checkArrayStore}\big(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ \big) \Rightarrow$

$\quad\quad\quad \mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$

$\quad\quad\quad \mathit{checkArrayStore}\big(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ \big) \Rightarrow$

$\quad\quad\quad\quad \Big((\mathit{t} = t_v = \mathtt{r}) \wedge \neg O_v.\mathit{isArray} \wedge (O_v.\mathit{type} \not\preceq O_q.\mathit{type})\Big) \Rightarrow$
$\quad\quad\quad\quad\quad \mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{ArrayStoreException}})$

$\quad\quad\quad\quad ((l < 0) \vee (h \geq min_array_length)) \Rightarrow$
$\quad\quad\quad\quad\quad \mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{ArrayIndexOutOfBoundsException}})$

Table B.10: arraystore t seventh until tenth cases

The form of C and C' meet the preconditions for Lemma B.5.1 on page 407 and so to conclude for this case, we have only to show:

$$\mathsf{HANDLE}(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}_{\mathtt{ArrayStoreException}}, (1,1)), \emptyset)$$

From $SF\ \mathcal{R}^{R,H,JH}_{\mathrm{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(\mathit{m}_n, pc_n)(\pi_1) \end{array}$$

and from the flowlogic clause in Table B.10 we may reason:

$$\begin{array}{llll} \beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & & \sqsubseteq_S & A_1 \wedge \\ \beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & & \sqsubseteq_S & A_2 \wedge \\ \beta^{R,H,JH}_{Stack}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & & \sqsubseteq_S & X \wedge \\ \beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M\ \Rightarrow \\ \exists\ \{O_q\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & \sqsubseteq_{Val} & X \wedge \\ \exists\ \{(\mathtt{s}, (l, h, mod), (\mathit{m}_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\ \exists\ \{O_v\} = \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Val}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & \sqsubseteq_{Val} & A_2\ \Rightarrow \end{array}$$

$$\begin{array}{lll} \mathit{t}_1 = \mathit{t}_1 = \mathtt{r} & & \wedge \\ \neg \mathtt{getJHorH}(v).\mathit{isArray} \wedge \neg\big(\mathtt{getJHorH}(\mathit{loc}).\mathit{refType} \preceq \mathtt{getJHorH}(v).\mathit{refType}\big) & \Rightarrow & \neg O_v.\mathit{isArray} \wedge (O_v.\mathit{type} \not\preceq O_q.\mathit{type}) \end{array}$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(\mathit{loc}_n)$ and $O_3 = \mathtt{getJHorH}(\mathit{loc}_3)$. Combined with the above:

$$\mathit{checkArrayStore}\ \big(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p)), (\mathit{t}_1, v, (\mathit{m}, apc_3)), (t = \mathtt{r})\ \big)$$

=

$$\mathit{checkArrayStore}\ \big(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ \big)$$

Combining all of the above we may conclude:

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{ArrayStoreException}})$

but, from Appendix D.7, we have $\widehat{\sigma}_{\texttt{ArrayStoreException}} = \beta^{R,H,JH}_{Ref}(\mathit{loc}_{\texttt{ArrayStoreException}})$ and $\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 = \beta^{R,H,JH,k}_{Context}(SF)$ and the result follows.

B.4.2.15.8 Possible transition 8 of 10

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{arraystore}\ \ t \\ S_n = S' :: (\texttt{r}, \mathit{loc}, (m_p, pc_p)) :: (\texttt{s}, i, (m_q, pc_q)) :: (t_1, v, (m, apc_3)) \\ ((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true})) \\ (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\ \mathit{checkArrayStore}\Big(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ \Big) \\ \neg(0 \leq i < \texttt{getJHorH}(\mathit{loc}).\mathit{length}) \\ SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{ArrayIndexOutOfBoundsException}}, (1,1)), \texttt{java.lang.ArrayIndexOutOfBoundsException}, ctd_n, io_n \end{array}\right) \end{array}}{P \Big| \begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle \rangle\!\rangle \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, SF' \rangle\!\rangle \end{array}}$$

The form of C and C' meet the preconditions for Lemma B.5.1 on page 407 and so to conclude for this case, we have only to show:

$$\mathsf{HANDLE}(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH}_{Ref}(\texttt{r}, \mathit{loc}_{\texttt{ArrayIndexOutOfBoundsException}}, (1,1)), \emptyset)$$

From $SF\ \mathcal{R}^{R,H,JH}_{\mathrm{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lccl} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from the flowlogic clause in Table B.10 we may reason:

$$
\begin{array}{llll}
\beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(\mathit{m}_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1 \;\Rightarrow \\
\beta^{R,H,JH}_{Stack}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & & \sqsubseteq_S & A_1 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & & \sqsubseteq_S & A_2 \wedge \\
\beta^{R,H,JH}_{Stack}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & & \sqsubseteq_S & X \wedge \\
\beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M \;\Rightarrow \\
\exists\, \{O_q\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p))) & \sqsubseteq_{Val} & X \wedge \\
\exists\, \{(\mathtt{s}, (l, h, mod), (\mathit{m}_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (\mathit{m}_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\
\exists\, \{O_v\} = \{(\mathit{t}_v, W, (\mathit{m}_v, pc_v))\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Val}((\mathit{t}_1, v, (\mathit{m}, apc_3))) & \sqsubseteq_{Val} & A_2 \;\Rightarrow \\
\neg(0 \leq i < \mathtt{getJHorH}(\mathit{loc}).\mathit{length}) & & \Rightarrow & ((l < 0) \vee (h \geq min_array_length))
\end{array}
$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \mathtt{getJHorH}(\mathit{loc}_n)$ and $O_3 = \mathtt{getJHorH}(\mathit{loc}_3)$. Combined with the above:

$$
\mathit{checkArrayStore}\left(\ \mathtt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \mathtt{getJHorH}(\mathit{loc}).\mathit{owner}, (\mathtt{r}, \mathit{loc}, (\mathit{m}_p, pc_p)), (\mathit{t}_1, v, (\mathit{m}, apc_3)), (t = \mathtt{r})\ \right)
$$

=

$$
\mathit{checkArrayStore}\left(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \mathtt{r})\ \right)
$$

Combining all of the above we may conclude:

$$
\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\mathtt{ArrayIndexOutOfBoundsException}})
$$

but, from Appendix D.7, we have $\widehat{\sigma}_{\mathtt{ArrayIndexOutOfBoundsException}} = \beta^{R,H,JH}_{Ref}(\mathit{loc}_{\mathtt{ArrayIndexOutOfBoundsException}})$ and $\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 = \beta^{R,H,JH,k}_{Context}(SF)$ and the result follows.

B.4.2.15.9 Possible transition 9 of 10

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{arraystore}\ \ t \\ S_n = S' :: (\texttt{r}, \mathit{loc}, (m_p, pc_p)) :: (\texttt{s}, i, (m_q, pc_q)) :: (t_1, v, (m, apc_3)) \\ ((\mathit{loc} = \texttt{null}) \vee (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \in (\mathit{dom}(H) \cup \mathit{dom}(JH)) \wedge \texttt{getJHorH}(\mathit{loc}).\mathit{isArray} = \texttt{true})) \\ (\mathit{loc} \neq \texttt{null} \wedge \mathit{loc} \notin \mathit{dom}(I)) \\ \neg \mathit{checkArrayStore}\Big(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ \Big) \\ SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \texttt{java.lang.SecurityException}, ctd_n, io_n \end{array} \right) \end{array}}{{}_P| \ \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle\!\rangle \ \Rightarrow \ \langle\!\langle R, K, H, I, HID, JH, CHN, SF' \rangle\!\rangle}$$

The form of C and C' meet the preconditions for Lemma B.5.1 on page 407 and so to conclude for this case, we have only to show:

$$\mathsf{HANDLE}(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH}_{Ref}(\texttt{r}, \mathit{loc}_{\texttt{SecurityException}}, (1,1)), \emptyset)$$

From $SF\ \mathcal{R}^{R,H,JH}_{\mathrm{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lccl} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from the flowlogic clause in Table B.10 we may reason:

$$\begin{array}{lllll} \beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M :: X :: A_2 :: A_1\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}((t_1, v, (m, apc_3))) & & \sqsubseteq_S & A_1\ \wedge \\ \beta^{R,H,JH}_{Stack}((\texttt{s}, i, (m_q, pc_q))) & & \sqsubseteq_S & A_2\ \wedge \\ \beta^{R,H,JH}_{Stack}((\texttt{r}, \mathit{loc}, (m_p, pc_p))) & & \sqsubseteq_S & X\ \wedge \\ \beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M\ \Rightarrow \\ \exists\ \{O_q\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\texttt{r}, \mathit{loc}, (m_p, pc_p))) & \sqsubseteq_{Val} & X\ \wedge \\ \exists\ \{(\texttt{s}, (l, h, mod), (m_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\texttt{s}, i, (m_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\ \exists\ \{O_v\} = \{(t_v, W, (m_v, pc_v))\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Val}((t_1, v, (m, apc_3))) & \sqsubseteq_{Val} & A_2 \end{array}$$

By definition of $\beta^{R,H,JH,k}_{Context}$, $O_n = \texttt{getJHorH}(\mathit{loc}_n)$ and $O_3 = \texttt{getJHorH}(\mathit{loc}_3)$. Combined with the above:

$$\neg \mathit{checkArrayStore}\left(\ \texttt{getJHorH}(\mathit{loc}_n).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}_3).\mathit{owner}, \texttt{getJHorH}(\mathit{loc}).\mathit{owner}, (\texttt{r}, \mathit{loc}, (m_p, pc_p)), (t_1, v, (m, apc_3)), (t = \texttt{r})\ \right)$$

=

$$\neg \mathit{checkArrayStore}\left(\ O_n.\mathit{owner}, O_3.\mathit{owner}, O_q.\mathit{owner}, O_q, O_v, (t = \texttt{r})\ \right)$$

Combining all of the above we may conclude:

$$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\texttt{SecurityException}})$$

but, from Appendix D.7, we have $\widehat{\sigma}_{\texttt{SecurityException}} = \beta^{R,H,JH}_{Ref}(\mathit{loc}_{\texttt{SecurityException}})$ and $\exists\, \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\,.\,\pi_1 = \beta^{R,H,JH,k}_{Context}(SF)$ and the result follows.

B.4.2.15.10 Possible transition 10 of 10

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{arraystore}\ \ t \\ S_n = S' :: (\texttt{r}, \mathit{loc}, (m_p, pc_p)) :: (\texttt{s}, i, (m_q, pc_q)) :: (t_1, v, (m, apc_3)) \\ (\mathit{loc} = \texttt{null} \vee \mathit{loc} \in \mathit{dom}(I)) \ \Rightarrow \\ \\ \quad H' = H, JH' = JH \\ \quad SF' = \mathit{catchException}\left(\begin{array}{l} \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle, \\ \\ (\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1, 1)), \texttt{java.lang.NullPointerException}, ctd_n, io_n \end{array}\right) \end{array}}{P \left| \begin{array}{l} \langle\!\langle R, K, H, I, HID, JH, CHN, \langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle\!\rangle \ \Rightarrow \\ \langle\!\langle R, K, H, I, HID, JH, CHN, SF' \rangle\!\rangle \end{array} \right.}$$

The form of C and C' meet the preconditions for Lemma B.5.1 on page 407 and so to conclude for this case, we have only to show:

$$\mathsf{HANDLE}(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH}_{Ref}(\texttt{r}, \mathit{loc}_{\texttt{NullPointerException}}, (1, 1)), \emptyset)$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from the flowlogic clause in Table B.10 we may reason:

$$\begin{array}{llll} \beta^{R,H,JH}_{Stack}(S_n) & & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \wedge \widehat{S}(m_n, pc_n)(\pi_1) = M{::}X{::}A_2{::}A_1\ \Rightarrow \\ \beta^{R,H,JH}_{Stack}((t_1, v, (m, apc_3))) & & \sqsubseteq_S & A_1\ \wedge \\ \beta^{R,H,JH}_{Stack}((\mathtt{s}, i, (m_q, pc_q))) & & \sqsubseteq_S & A_2\ \wedge \\ \beta^{R,H,JH}_{Stack}((\mathtt{r}, loc, (m_p, pc_p))) & & \sqsubseteq_S & X\ \wedge \\ \beta^{R,H,JH}_{Stack}(S) & & \sqsubseteq_S & M\ \Rightarrow \\ \exists\ \{O_q\} \sqsupseteq_{Val} \{\widehat{\sigma}_{\mathtt{Null}}\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Ref}((\mathtt{r}, loc, (m_p, pc_p))) & \sqsubseteq_{Val} & X\ \wedge \\ \exists\ \{(\mathtt{s}, (l, h, mod), (m_q, pc_q))\} & \sqsupseteq_{Val}\ \beta_{Num}((\mathtt{s}, i, (m_q, pc_q))) & \sqsubseteq_{Val} & A_1 \\ \exists\ \{O_v\} = \{(t_v, W, (m_v, pc_v))\} & \sqsupseteq_{Val}\ \beta^{R,H,JH}_{Val}((t_1, v, (m, apc_3))) & \sqsubseteq_{Val} & A_2 \end{array}$$

Combining all of the above we may conclude:

$\mathsf{HANDLE}(\pi_1, \pi_1, \widehat{\sigma}_{\mathtt{NullPointerException}})$

but, from Appendix D.7, we have $\widehat{\sigma}_{\mathtt{NullPointerException}} = \beta^{R,H,JH}_{Ref}(loc_{\mathtt{NullPointerException}})$ and $\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 = \beta^{R,H,JH,k}_{Context}(SF)$ and the result follows.

}

B.4.2.16 Case `jsr` *addr*:

By assumption:

$$\frac{\mathsf{m}_n.\mathit{instructionAt}(pc_n) = \texttt{jsr}\ \ \mathit{addr}}{P \left| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle \ \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF'\rangle \end{array} \right.}$$

whence:

$$SF = \text{《}\mathsf{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathsf{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathsf{loc}_n, itd_n, ctd_n, io_n, (\mathsf{m}_n, pc_n), \mathsf{L}_n, \mathsf{s}_n\rangle$$

$$SF' = \text{《}\mathsf{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathsf{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathsf{loc}_n, itd_n, ctd_n, io_n, (\mathsf{m}_n, addr), \mathsf{L}_n, S\rangle$$

$$S = \mathsf{S}_n :: (\texttt{ra}, \mathsf{m}_n.\mathit{nextAddress}(pc_n), (\mathsf{m}_n, pc_n))$$

The form of C and C' meet the preconditions for Lemma B.5.4 on page 419 and so to conclude for this case, we have only to show:

	π_1	$=$	$\beta^{R,H,JH,k}_{Context}(SF)$
$\wedge$	π_2	$=$	$\beta^{R,H,JH,k}_{Context}(SF')$
$\wedge$	$\{\pi_2\}$	$\sqsubseteq_C$	$\widehat{C}(\mathsf{m}_n, addr)$
$\wedge$	$\beta^{R,H,JH}_{Stack}(S)$	$\sqsubseteq_S$	$\widehat{S}(\mathsf{m}_n, addr)(\pi_2)$
$\wedge$	$\widehat{L}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_L$	$\widehat{L}(\mathsf{m}_n, addr)(\pi_2)$
$\wedge$	$\widehat{JH}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{JH}$	$\widehat{JH}(\mathsf{m}_n, addr)(\pi_2)$
$\wedge$	$\widehat{MNAMES}(m_n, pc_n)(\pi_1)$	$\sqsubseteq_{MNAMES}$	$\widehat{MNAMES}(\mathsf{m}_n, addr)(\pi_2)$
$\wedge$	$\{(\pi_1, \pi_2)\}$	$\sqsubseteq_{CG}$	$\widehat{CG}$
$\Rightarrow$	C'	$\mathcal{R}^{R,H,JH,k}_{\text{Config}}$	$(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lcll} & \exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\ \wedge & \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\ \wedge & \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \end{array}$$

and from $(\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \models_{\texttt{CFA}} P$ and from inspection of the flow-logic rule for `goto` $addr$ the proof obligation reduces to:

$$\beta^{R,H,JH}_{Stack}(S) = \beta^{R,H,JH}_{Stack}(S_n{::}(\texttt{ra}, m_n.\mathit{nextAddress}(pc_n), (m_n, pc_n))) \sqsubseteq_S \widehat{S}(m_n, addr)(\pi_2)$$

From the flow-logic rule, we have:

$$\widehat{S}(m_n, pc_n)(\pi_1){::}\{(\text{ra},\ m_n.\mathit{nextAddress}(pc_n), (m_n, pc_n))\} \quad \sqsubseteq_S \quad \widehat{S}(m_n, \mathit{addr})(\pi_2)$$

Combining terms:

$$\begin{array}{rcl} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n{::}(\texttt{ra}, m_n.\mathit{nextAddress}(pc_n), (m_n, pc_n))) \\ & \sqsubseteq_S & \beta^{R,H,JH}_{Stack}(S_n){::}\beta_{ReturnAddress}(\texttt{ra}, m_n.\mathit{nextAddress}(pc_n), (m_n, pc_n)) \\ & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1){::}\{(\text{ra},\ m_n.\mathit{nextAddress}(pc_n), (m_n, pc_n))\} \\ & \sqsubseteq_S & \widehat{S}(m_n, \mathit{addr})(\pi_2) \end{array}$$

and the result follows.

B.4.2.17 Case `ret` *i*:

By assumption:

$$\frac{\begin{array}{l} m_n.\mathit{instructionAt}(pc_n) = \texttt{ret}\ \ i \\ (\texttt{ra}, \mathit{addr}, (m_n, apc)) = L(i) \end{array}}{P \left| \begin{array}{l} \langle R, K, H, HID, JH, CHN, SF\rangle\ \Rightarrow \\ \langle R, K, H, HID, JH, CHN, SF'\rangle \end{array} \right.}$$

whence:

$$SF = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$SF' = \text{《}\mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1\text{》} :: \text{《}\mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2\text{》} :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, addr), L_n, S \rangle$$

$$S = S_n$$

The form of C and C' meet the preconditions for Lemma B.5.4 on page 419 and so to conclude for this case, we have only to show:

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, addr) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, addr)(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, addr)(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, addr)(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, addr)(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})
\end{array}$$

From $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$ we must have:

$$\begin{array}{lll}
\exists\ \pi_1 \sqsubseteq_C \widehat{C}(m_n, pc_n)\ .\ \pi_1 & = \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge \qquad \pi_2 & = \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge \qquad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1)
\end{array}$$

By assumption:

$$\begin{array}{lll}
\beta^{R,H,JH}_{LocalVar}(\mathsf{L}_n) = M\ . & & \\
\quad \forall\, j \in \mathsf{dom}(\mathsf{L}_n) : & & \\
\qquad \beta^{R,H,JH}_{Val}(\mathsf{L}_n(j)) & \sqsubseteq_{Val} M(i) & \\
 & \sqsubseteq_{Val} \widehat{L}(\mathsf{m}_n, pc_n)(\pi_1)(i) & \\
\Rightarrow \quad \beta^{R,H,JH}_{Val}(\mathsf{L}_n(i)) & \sqsubseteq_{Val} \widehat{L}(\mathsf{m}_n, pc_n)(\pi_1)(i) & \\
\Rightarrow \quad \beta^{R,H,JH}_{Val}((\mathtt{ra}, \mathsf{addr}, (\mathsf{m}_n, apc))) & \sqsubseteq_{Val} \widehat{L}(\mathsf{m}_n, pc_n)(\pi_1)(i) &
\end{array}$$

From the flow-logic rule, we have:

$$\forall\, \{(\mathrm{ra},\ \mathsf{addr}, (\mathsf{m}_r, pc_r))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(i) :$$

$$\pi_2 = ((O_1, \psi_1, \gamma_1, \xi_1, (\mathsf{m}_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (\mathsf{m}_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (\mathsf{m}_n, \mathsf{addr})))$$

$$\begin{array}{lll}
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathsf{m}_n, \mathsf{addr}) \\
\widehat{S}(m_n, pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

Combining the above:

$$\begin{array}{lll}
\exists\, \{(\mathrm{ra},\ \mathsf{addr}, (\mathsf{m}_r, pc_r)))\} \sqsubseteq_L \widehat{L}(m_n, pc_n)(\pi_1)(j)\ .\ (\mathrm{ra},\ \mathsf{addr}, (\mathsf{m}_r, pc_r)) & \sqsupseteq_{Val} & \beta^{R,H,JH}_{Val}(\mathtt{ra}, \mathsf{addr}, (\mathsf{m}_n, apc)) \Rightarrow \\
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathsf{m}_n, \mathsf{addr}) \\
\widehat{S}(m_n, pc_n)(\pi_1) & \sqsubseteq_S & \widehat{S}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathsf{m}_n, \mathsf{addr})(\pi_2) \\
\{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

combined with:

$$\begin{array}{rcl} \beta^{R,H,JH}_{Stack}(S) & = & \beta^{R,H,JH}_{Stack}(S_n) \\ & \sqsubseteq_S & \widehat{S}(m_n, pc_n)(\pi_1) \\ & \sqsubseteq_S & \widehat{S}(m_n, addr)(\pi_2) \end{array}$$

and the result follows.

B.5 Auxiliary Functions and Helper Proofs

B.5.1 Definition of SINGLE_VALUE_STACKS() and *absNbWords*

These are used in the flow-logic equivalent of the core instructions of Section 3.9.1 for pattern-matching and binding subsequences of the operand stack. In particular, SINGLE_VALUE_STACKS() expands out a sequence of length n sets of variable sizes to the equivalent "cross-product" of all possible combinations of sequence of sets of singleton size, where each possible combination of length n and the ordered of the elements of the set is retained. By producing all possible (ordered) permutations, we are assured one of them will correspond to the abstract representation of the operand stack i.e.

$$\begin{aligned}\beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \widehat{S}(m_n, pc_n)(\pi_1) \;\Rightarrow\; & \exists \quad Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big) \;.\; \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \; Z \\ & \wedge \quad |Z \in \texttt{SINGLE_VALUE_STACKS}\Big(\widehat{S}(m_n, pc_n)(\pi_1)\Big) \;.\; \beta^{R,H,JH}_{Stack}(S_n) \sqsubseteq_S \; Z| \geq 1\end{aligned}$$

While SINGLE_VALUE_STACKS() generates combinations with the same length and singleton set values, different combinations may have different abstract depth, since different types (i.e. `int` and `short`) may share the same stack location, as per the abstract *absNbWords* size calculator, and so this expansion to single value sets allows all possible abstract stacks to be generated and pattern-matched against a particular depth, with the appropriate subsequence copied forward to the next address.

$$\begin{aligned}\texttt{SINGLE_VALUE_STACKS}\Big(\epsilon\Big) &= \epsilon \\ \texttt{SINGLE_VALUE_STACKS}\Big(\{(t_1, Y_1, (m_1, pc_1)), \ldots, (t_1, Y_k, (m_k, pc_k))\}\Big) &= \{(t_1, Y_1, (m_1, pc_1))\} :: \ldots :: \{(t_k, Y_k, (m_k, pc_k))\} \\ \texttt{SINGLE_VALUE_STACKS}\Big(\{A\} :: \ldots :: \{X\}\Big) &= \{\{a\} :: \ldots :: \{x\} | a \in A, \ldots, x \in X\}\end{aligned}$$

$$
\begin{array}{lcll}
\mathit{absNbWords} & : & (\overline{\mathsf{Val}})^* & \to \mathbb{N}_0 \\
\mathit{absNbWords} & : & \epsilon & = 0 \\
\mathit{absNbWords} & : & \{(t, Y, (m_n, pc_n))\} & = \mathit{nbWords}((t, Y, (m_n, pc_n))) \\
\mathit{absNbWords} & : & \{(t_1, Y_1, (m_1, pc_1))\} :: \ldots :: \{(t_p, Y_p, (m_p, pc_p))\} & = \sum_{i=1}^{p} \mathit{nbWords}(\ (t_i, Y_i, (m_i, pc_i))\ )
\end{array}
$$

B.5.2 Abstract exception handler

HANDLE : (`Context` : $\pi_{Original}$) × (`Context` : $\pi_{Current}$) × (AbstractObject : $O_{exception}$) × ($\mathcal{P}$(`Context` × `Context`) : $AlreadySeen$) → `void`

$$\text{HANDLE}\left(\begin{array}{lcll} \pi_{Original} & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) & \wedge \\ \pi_{Current} & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),(O_3,\psi_3,\gamma_3,\xi_3,(m_3,pc_3))) & \wedge \\ \mathit{findHandler}(m_3,pc_3,O_{exception}.\mathit{refType}) & = & \bot & \end{array}\right) \Rightarrow$$

$$\begin{array}{lll} \pi_2 & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_n,\xi_n,(m_2,10))) \\ \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_2,10) \\ \{O_{exception}\} & \sqsubseteq_S & \widehat{S}(m_2,10)(\pi_2) \\ \{[]\} & \sqsubseteq_L & \widehat{L}(m_2,10)(\pi_2) \\ \widehat{JH}(m_n,pc_n)(\pi_{Original}) & \sqsubseteq_{JH} & \widehat{JH}(m_2,10)(\pi_2) \\ \widehat{MNAMES}(m_n,pc_n)(\pi_{Original}) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_i,pc')(\pi_i) \\ \{(O_{exception},\pi_{Original},\pi_{Current})\} & \sqsubseteq_E & \widehat{E} \\ \{(\pi_{Original},\pi_{Current})\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

$$\text{HANDLE}\left(\begin{array}{lcll} \pi_{Original} & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) & \wedge \\ \pi_{Current} & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_q,\psi_q,\gamma_q,\xi_q,(m_q,pc_q))) & \wedge \\ \mathit{findHandler}(m_q,pc_q,O_{exception}.\mathit{refType}) & = & pc' \neq \bot & \end{array}\right) \Rightarrow$$

$$\begin{array}{lll} \pi_2 & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_q,pc_2)),\ldots,(O_q,\psi_q,\gamma_n,\xi_n,(m_q,pc')) \\ \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_q,pc') \\ \{O_{exception}\} & \sqsubseteq_S & \widehat{S}(m_q,pc')(\pi_2) \\ \widehat{L}(m_q,pc_q)(\pi_{Current}) & \sqsubseteq_L & \widehat{L}(m_q,pc')(\pi_2) \\ \widehat{JH}(m_n,pc_n)(\pi_{Original}) & \sqsubseteq_{JH} & \widehat{JH}(m_q,pc')(\pi_2) \\ \widehat{MNAMES}(m_q,pc_q)(\pi_{Current}) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_q,pc')(\pi_2) \\ \{(O_{exception},\pi_{Original},\pi_{Current})\} & \sqsubseteq_E & \widehat{E} \\ \{(\pi_{Original},\pi_{Current})\} & \sqsubseteq_{CG} & \widehat{CG} \end{array}$$

$$\mathsf{HANDLE}\left(\begin{array}{lcll} \pi_{Original} & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),\ldots,(O_n,\psi_n,\gamma_n,\xi_n,(m_n,pc_n))) & \wedge \\ \pi_{Current} & = & ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),(O_3,\psi_3,\gamma_3,\xi_3,(m_3,pc_3)),\ldots,(O_q,\psi_q,\gamma_q,\xi_q,(m_q,pc_q))) & \wedge \\ \mathit{findHandler}(m_q,pc_q,O_{exception}.\mathit{refType}) & = & \bot & \end{array}\right) \Rightarrow$$

$$\forall\ \pi_{Invoking} = ((O_1,\psi_1,\gamma_1,\xi_1,(m_1,pc_1)),(O_2,\psi_2,\gamma_2,\xi_2,(m_2,pc_2)),(O_3,\psi_3,\gamma_3,\xi_3,(m_3,pc_3)),X)) \sqsubseteq_C \widehat{C}(m_{(q-1)},pc_{(q-1)}) :$$

$$\Big(\{(\pi_{Invoking},\pi_{Current})\}\} \not\subseteq AlreadySeen\Big) \Rightarrow \quad \mathsf{HANDLE}(\pi_{Original},\pi_{Invoking},O_{exception},AlreadySeen \cup \{(\pi_{Invoking},\pi_{Current})\})$$

From the assumptions of this theorem we will be using the below:

$$\langle\!\langle R, K, H, I, HID, JH, CHN, SF \rangle\!\rangle \; \mathcal{R}^{R,H,JH}_{\text{Config}} \; (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$$

$$\Rightarrow \quad SF \; \mathcal{R}^{R,H,JH}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$$

$$\wedge \quad \beta^{R,H,JH}_{Registry}(R) \sqsubseteq_R \widehat{R}$$

$$\wedge \quad \beta^{H}_{StaticHeap}(K) \sqsubseteq_K \widehat{K}$$

$$\wedge \quad \beta^{H,JH}_{DynamicHeap}(H) \sqsubseteq_H \widehat{H}$$

$$\wedge \quad \beta^{R,H,JH}_{Invalidated}(I) \sqsubseteq_I \widehat{I}$$

$$SF \; \mathcal{R}^{R,H,JH}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$$

$$\Rightarrow \quad SF = \langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \ldots :: \langle\!\langle loc_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle\!\rangle$$

$$\wedge \quad \forall i \in \{3, \ldots, n\} :$$

$$\begin{array}{lll} \pi_i & = & \beta^{R,H,JH,k}_{Context}\big(\langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle loc_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle\big) \\ \wedge \; \{\pi_i\} & \sqsubseteq_C & \widehat{C}(m_i, pc_i) \\ \wedge \; \{m_1, \ldots, m_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_i, pc_i)(\pi_i) \\ \wedge \; \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(m_i, pc_i)(\pi_i) \\ \wedge \; \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(m_i, pc_i)(\pi_i) \end{array}$$

B.5.3 Proof of abstract exception handling

Lemma B.5.1 (Abstract exception handler)**.** Let:

- $P \in$ Program
- $C = \langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P \Big| \begin{array}{l} C \Rightarrow \\ C' \end{array}$
- $SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$
- $C' = \langle R, K, H, I, HID, JH, CHN, SF' \rangle$. $SF' = \mathit{catchException}(SF, (\mathtt{r}, \mathit{loc}, (m_r, pc_r)), \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType}), ctd_n, io_n)$

Then:

$$
\begin{array}{lll}
C & \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})\ \wedge \\
 & \mathsf{HANDLE}(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_r, pc_r)), \emptyset)\ \Rightarrow \\
C' & \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})
\end{array}
$$

Proof:

From the assumptions of this theorem we will be using the below:

$$
\begin{array}{ll}
 & \langle R, K, H, I, HID, JH, CHN, SF \rangle\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \\
\Rightarrow & SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) \\
\wedge & \beta^{R,H,JH}_{Registry}(R) \sqsubseteq_R \widehat{R} \\
\wedge & \beta^{H}_{StaticHeap}(K) \sqsubseteq_K \widehat{K} \\
\wedge & \beta^{H,JH}_{DynamicHeap}(H) \sqsubseteq_H \widehat{H} \\
\wedge & \beta^{R,H,JH}_{Invalidated}(I) \sqsubseteq_I \widehat{I}
\end{array}
$$

$$SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$$

$$\Rightarrow \quad SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$$

$$\wedge \quad \forall i \in \{3, \ldots, n\} :$$

$$\begin{array}{lllll}
 & \pi_i & = & \beta^{R,H,JH,k}_{Context}\big(\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle\big) \\
\wedge & \{\pi_i\} & \sqsubseteq_C & \widehat{C}(m_i, pc_i) \\
\wedge & \{m_1, \ldots, m_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(m_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(m_i, pc_i)(\pi_i)
\end{array}$$

and the definition of $\beta^{R,H,JH,k}_{Context}$ in Table 4.2.

By construction of $\beta^{R,H,JH,k}_{Context}$ our choice of representation for a callstack includes the addresses of at least the first three stack frames (including the applet lifecycle method invoked on the third stack frame) and the last two stack frames of the callstack passed to the function, so it is *always* possible from a context to determine both the address of the current method and the address of the method calling the current method. Further, from $C\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$ we have a sequence of such contexts representing each method invocation in the callstack SF. Since for every method invocation in the callstack SF we have a context containing $(invoking_{method}, invoked_{method})$ we can always calculate (a possible over-approximation[9] of) the set of its calling contexts, we have sufficient information to emulate unwinding the callstack one frame at a time.

Both *catchException* and HANDLE use the *findHandler* function to determine whether a particular address can handle a particular exception type. From the assumptions of this theorem, we have:

$$\begin{array}{rcl}
\beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType} & = & \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType}) \Rightarrow \\
\mathit{findHandler}(m_i, pc_i, \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType}) & \equiv & \mathit{findHandler}(m_i, pc_i, \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType})) \Rightarrow \\
\mathit{findHandler}(m_i, pc_i, \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType}) \neq \bot & \iff & \mathit{findHandler}(m_i, pc_i, \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType})) \neq \bot \wedge \\
\mathit{findHandler}(m_i, pc_i, \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType}) = \bot & \iff & \mathit{findHandler}(m_i, pc_i, \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType})) = \bot
\end{array}$$

and so *catchException* and HANDLE will action their respective clauses at the same address. This implies when *catchException* matches its third clause and chooses to unwind the callstack of a callstack with a current length of at least four by a stack frame and recursively check the next frame when

$\mathit{findHandler}(m_k, pc_k, \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType})) = \bot$, then

HANDLE must also conclude $\mathit{findHandler}(m_k, pc_k, \beta^{R,H,JH}_{Ref}(\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType}) = \bot$ and in the case of an

[9] Note this is a potentially a set of calling context where $n > k$ in the k-CFA.

abstract callstack of at least four stack frames will check its calling method address for the set of *possible* caller methods and recursively check those addresses using HANDLE.

$$\forall\, \pi_{Invoking} \quad = \quad ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), (O_3, \psi_3, \gamma_3, \xi_3, (m_3, pc_3)), X)) \sqsubseteq_C \hat{C}(m_{(k-1)}, pc_{(k-1)}) :$$
$$\left(\{(\pi_{Invoking}, \pi_{Current})\}) \not\subseteq AlreadySeen\right) \Rightarrow \quad \text{HANDLE}(\pi_{Original}, \pi_{Invoking}, O_{exception}, AlreadySeen \cup \{(\pi_{Invoking}, \pi_{Current})\})$$

Since pattern-matching of current and previous method is used to identify the set of possible caller contexts, we are assured one of them must be the correct context as argued above, but we may also propagate exception information to some extra addresses, thus hindering precision but not correctness.

Given $SF' = \textit{catchException}(SF, (\mathtt{r}, \textit{loc}, (m_r, pc_r)), \mathtt{getJHorH}((\mathtt{r}, \textit{loc}, (m_r, pc_r)).\textit{refType}), ctd_n, io_n)$, then SF' must have one of two forms:

Case 1: exception programatically handled by applet

In this case, we match the second clause of *catchException*, and for some $i \in \mathbb{N},\ 3 \leq i \leq n$:

$$SF' = \langle \textit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \textit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \textit{loc}_i, itd_i, ctd_n, io_n, (m_i, pc'), L_i, (\mathtt{r}, \textit{loc}_{exc}, (m, apc)) \rangle$$

where $\textit{findHandler}(m_i, pc_i, \mathtt{getJHorH}((\mathtt{r}, \textit{loc}, (m_r, pc_r)).\textit{refType})) = pc' \neq \bot$.

Now either $n = i$ and the *catchException* and HANDLE functions have identified a handler for the current method/stack frame directly, since as argued above:

$$\textit{findHandler}(m_i, pc_i, \beta^{R,H,JH}_{Ref}(\mathtt{r}, \textit{loc}, (m_r, pc_r)).\textit{refType}) \neq \bot \iff \textit{findHandler}(m_i, pc_i, \mathtt{getJHorH}((\mathtt{r}, \textit{loc}, (m_r, pc_r)).\textit{refType})) \neq \bot$$

or both functions have recursively checked their parent method/stack frame and found a handler at m_i.

To conclude the proof for this case, since only the callstack has changed, we need to show:

$$
\begin{array}{lll}
& SF' \; \mathcal{R}^{R,H,JH}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) & \\
\Rightarrow & \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \sqsubseteq_{JH} \widehat{JH}(m_i, pc')(\beta^{R,H,JH,k}_{Context}(SF')) & \\
\wedge & SF' = \langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \langle\!\langle loc_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle\!\rangle :: \ldots :: \langle\!\langle loc_i, itd_i, ctd_n, io_n, (m_i, pc'), L_i, (\mathtt{r}, loc_{exc}, (m, apc)) \rangle\!\rangle & \\
& \wedge \quad \pi_i & = \quad \beta^{R,H,JH,k}_{Context}(SF') \\
& \wedge \quad \{\pi_i\} & \sqsubseteq_C \quad \widehat{C}(m_i, pc') \\
& \wedge \quad \widehat{MNAMES}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) & \sqsubseteq_{MNAMES} \quad \widehat{MNAMES}(m_i, pc')(\pi_i) \\
& \wedge \quad \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L \quad \widehat{L}(m_i, pc')(\pi_i) \\
& \wedge \quad \beta^{R,H,JH}_{Ref}(\mathtt{r}, loc_{exc}, (m, apc)) & \sqsubseteq_S \quad \widehat{S}(m_i, pc')(\pi_i)
\end{array}
$$

But we have: $\text{HANDLE}(\beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH,k}_{Context}(SF), \beta^{R,H,JH}_{Ref}(\mathtt{r}, loc, (m_r, pc_r)), \emptyset) \Rightarrow$

$$
\text{HANDLE}\left(\begin{array}{lll}
\pi_{Original} & = & ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (m_n, pc_n))) \quad \wedge \\
\pi_{Current} & = & ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_2, pc_2)), \ldots, (O_q, \psi_q, \gamma_q, \xi_q, (m_q, pc_q))) \quad \wedge \\
findHandler(m_q, pc_q, O_{exception}.refType) & = & pc' \neq \bot
\end{array}\right) \Rightarrow
$$

$$
\begin{array}{lll}
\pi_2 & = & ((O_1, \psi_1, \gamma_1, \xi_1, (m_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (m_q, pc_2)), \ldots, (O_q, \psi_q, \gamma_n, \xi_n, (m_q, pc')) \\
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_q, pc') \\
\{O_{exception}\} & \sqsubseteq_S & \widehat{S}(m_q, pc')(\pi_2) \\
\widehat{L}(m_q, pc_q)(\pi_{Current}) & \sqsubseteq_L & \widehat{L}(m_q, pc')(\pi_2) \\
\widehat{JH}(m_n, pc_n)(\pi_{Original}) & \sqsubseteq_{JH} & \widehat{JH}(m_q, pc')(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_{Original}) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_q, pc')(\pi_2) \\
\{(O_{exception}, \pi_{Original}, \pi_{Current})\} & \sqsubseteq_E & \widehat{E} \\
\{(\pi_{Original}, \pi_{Current})\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

and since:

- $\pi_{Original} = \beta^{R,H,JH,k}_{Context}(SF)$
- $\xi_n = itd_n$
- $\gamma_n = io_n$
- $(m_i, pc_i) = (m_q, pc_q)$
- $\pi_2 = \pi_i = \beta^{R,H,JH,k}_{Context}(SF')$
- $O_{exception} = \beta^{R,H,JH}_{Ref}(\mathtt{r}, loc, (m_r, pc_r))$

the result follows.

Case 2: exception reaches top-level JCRE

In this case, we match the first clause of *catchException* since no exception handler has been found in any of the stack frames from n to 3:

$$SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (\mathit{m}_1, pc_1), L_1, \mathit{S}_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_n, io_n, (\mathit{m}_2, 10), [\,], (\mathtt{r}, \mathit{loc}_{exc}, (\mathit{m}, apc)) \rangle$$

To conclude the proof for this case, since only the callstack has changed, we need to show:

$$\begin{array}{lll}
 & SF' \; \mathcal{R}^{\mathit{R},\mathit{H},\mathit{JH}}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) & \\
\Rightarrow & \widehat{JH}(m_n, pc_n)(\beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF)) \sqsubseteq_{JH} \widehat{JH}(\mathit{m}_2, 10)(\beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF')) & \\
\wedge & SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (\mathit{m}_1, pc_1), L_1, \mathit{S}_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_n, io_n, (\mathit{m}_2, 10), [\,], (\mathtt{r}, \mathit{loc}_{exc}, (\mathit{m}, apc)) \rangle & \\
 & \wedge \quad \pi_i \quad = \quad \beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF') & \\
 & \wedge \quad \{\pi_i\} \quad \sqsubseteq_C \quad \widehat{C}(\mathit{m}_2, 10) & \\
 & \wedge \quad \widehat{MNAMES}(m_n, pc_n)(\beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF)) \quad \sqsubseteq_{MNAMES} \quad \widehat{MNAMES}(\mathit{m}_2, 10)(\pi_i) & \\
 & \wedge \quad \beta^{\mathit{R},\mathit{H},\mathit{JH}}_{LocalVar}([\,]) \quad \sqsubseteq_L \quad \widehat{L}(\mathit{m}_2, 10)(\pi_i) & \\
 & \wedge \quad \beta^{\mathit{R},\mathit{H},\mathit{JH}}_{Ref}(\mathtt{r}, \mathit{loc}_{exc}, (\mathit{m}, apc)) \quad \sqsubseteq_S \quad \widehat{S}(\mathit{m}_2, 10)(\pi_i) &
\end{array}$$

Since no exception handler has been found in any of the stack frames from n to 3, as argued previously, we may conclude:

$$\text{HANDLE}(\beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF), \beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF), \beta^{\mathit{R},\mathit{H},\mathit{JH}}_{Ref}(\mathtt{r}, \mathit{loc}, (\mathit{m}_r, pc_r)), \emptyset) \;\Rightarrow$$

$$\text{HANDLE}\left(\begin{array}{lcll}
\pi_{Original} & = & ((O_1, \psi_1, \gamma_1, \xi_1, (\mathit{m}_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (\mathit{m}_2, pc_2)), \ldots, (O_n, \psi_n, \gamma_n, \xi_n, (\mathit{m}_n, pc_n))) & \wedge \\
\pi_{Current} & = & ((O_1, \psi_1, \gamma_1, \xi_1, (\mathit{m}_1, pc_1)), (O_2, \psi_2, \gamma_2, \xi_2, (\mathit{m}_2, pc_2)), (O_3, \psi_3, \gamma_3, \xi_3, (\mathit{m}_3, pc_3))) & \wedge \\
\mathit{findHandler}(\mathit{m}_3, pc_3, O_{exception}.\mathit{refType}) & = & \bot &
\end{array}\right) \Rightarrow$$

$$\begin{array}{lll}
\pi_2 & = & ((O_1, \psi_1, \gamma_1, \xi_1, (\mathit{m}_1, pc_1)), (O_2, \psi_2, \gamma_n, \xi_n, (\mathit{m}_2, 10))) \\
\{\pi_2\} & \sqsubseteq_C & \widehat{C}(\mathit{m}_2, 10) \\
\{O_{exception}\} & \sqsubseteq_S & \widehat{S}(\mathit{m}_2, 10)(\pi_2) \\
\{[]\} & \sqsubseteq_L & \widehat{L}(\mathit{m}_2, 10)(\pi_2) \\
\widehat{JH}(m_n, pc_n)(\pi_{Original}) & \sqsubseteq_{JH} & \widehat{JH}(\mathit{m}_2, 10)(\pi_2) \\
\widehat{MNAMES}(m_n, pc_n)(\pi_{Original}) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_2, 10)(\pi_2) \\
\{(O_{exception}, \pi_{Original}, \pi_{Current})\} & \sqsubseteq_E & \widehat{E} \\
\{(\pi_{Original}, \pi_{Current})\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}$$

since:

- $\pi_{Original} = \beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF)$
- $\xi_n = itd_n$
- $\gamma_n = io_n$
- $\pi_2 = \beta^{\mathit{R},\mathit{H},\mathit{JH},k}_{Context}(SF') = \pi_i$
- $O_{exception} = \beta^{\mathit{R},\mathit{H},\mathit{JH}}_{Ref}(\mathtt{r}, \mathit{loc}, (\mathit{m}_r, pc_r))$

the result follows.

■

B.5.4 Second abstract exception handling proof

Lemma B.5.2 (Abstract exception handler, JCRE owned exception with an exception reason field changed)**.** Let:

- $P \in$ Program
- $C = \langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P \vdash C \Rightarrow C'$
- $SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$
- $\mathit{loc} \in \mathit{dom}(H)$
- $C' = \langle R, K, H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \mapsto (t, v, (m_n, pc_n))], I, HID, JH, CHN, SF' \rangle$.
  $SF' = \mathit{catchException}(SF, (\mathtt{r}, \mathit{loc}, (m_r, pc_r)), \mathtt{getJHorH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)).\mathit{refType}), ctd_n, io_n)$

Then:

$$
\begin{array}{rl}
 & \beta_{Val}^{R,H,JH}(t, v, (m_n, pc_n)) \sqsubseteq_{Val} \widehat{H}\big(\beta_{Ref}^{R,H,JH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)))\big).\mathit{values}(f.\mathit{id}) \ \wedge \\
C \quad & \mathcal{R}_{\text{Config}}^{R,H,JH}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \ \wedge \\
 & \mathsf{HANDLE}(\beta_{Context}^{R,H,JH,k}(SF), \beta_{Context}^{R,H,JH,k}(SF), \beta_{Ref}^{R,H,JH}(\mathtt{r}, \mathit{loc}, (m_r, pc_r)), \emptyset) \ \Rightarrow \\
C' \quad & \mathcal{R}_{\text{Config}}^{R,H,JH}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})
\end{array}
$$

Proof:

From the assumptions of:

- $\mathit{loc} \in \mathit{dom}(H)$
- $\beta_{Val}^{R,H,JH}(t, v, (m_n, pc_n)) \sqsubseteq_{Val} \widehat{H}\big(\beta_{Ref}^{R,H,JH}((\mathtt{r}, \mathit{loc}, (m_r, pc_r)))\big).\mathit{values}(f.\mathit{id})$
- $C\ \mathcal{R}_{\text{Config}}^{R,H,JH}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$
- $H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \mapsto (t, v, (m_n, pc_n))]$ being the only change to the heap.

we may conclude that $\beta_{DynamicHeap}^{H,JH}(H') \sqsubseteq_H \widehat{H}$, and the remainder follows from Lemma B.5.1 since changing a field of an exception object cannot change the first stack frame capable of handling the reference type of the exception.

B.5.5 Subject Reduction Simplification Lemma 1

Lemma B.5.3 (Only operand stack changed in transition to next address)**.** Let:

- $P \in$ Program
- $C = \langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P \Big| \begin{array}{l} C \Rightarrow \\ C' \end{array}$
- $C' = \langle R, K, H, I, HID, JH, CHN, SF' \rangle$
- $SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$
- $SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$
- $C \; \mathcal{R}^{R,H,JH}_{\text{Config}} \; (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$

Then:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
 & & & \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

Proof:

We have $\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$ by assumption and so proof obligation reduces to proving:

$$C' \; \mathcal{R}^{R,H,JH,k}_{\text{Config}} (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$$

From $C\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$ we have:

$$
\begin{array}{lllll}
 & \beta^{R,H,JH}_{Registry}(R) & \sqsubseteq_R & \widehat{R} \\
\wedge & \beta^{H}_{StaticHeap}(K) & \sqsubseteq_K & \widehat{K} \\
\wedge & \beta^{H,JH}_{DynamicHeap}(H) & \sqsubseteq_H & \widehat{H} \\
\wedge & \beta^{R,H,JH}_{Invalidated}(I) & \sqsubseteq_I & \widehat{I}
\end{array}
$$

Since R, K, H and I remain unchanged in C', proving $C'\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$ reduces to proving:

$$SF'\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})\ \wedge \{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$$

Now:

$$
\begin{array}{lllll}
 & SF'\ \mathcal{R}^{R,H,JH,k}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) \\
\iff & \beta^{H,JH}_{DynamicHeap}(JH) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\wedge & \forall i \in \{3, \ldots, n\} : \\
 & \quad \pi_i & = & \beta^{R,H,JH,k}_{Context}\big(\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle\big) \\
 & \wedge\ \{\pi_i\} & \sqsubseteq_C & \widehat{C}(m_i, pc_i) \\
 & \wedge\ \{m_1, \ldots, m_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_i, pc_i)(\pi_i) \\
 & \wedge\ \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(m_i, pc_i)(\pi_i) \\
 & \wedge\ \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(m_i, pc_i)(\pi_i)
\end{array}
$$

Since SF and SF' are identical below the top stack frame, we have from $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$:

$\forall i \in \{3, \ldots, (n-1)\}$:

$$
\begin{array}{lllll}
 & \pi_i & = & \beta^{R,H,JH,k}_{Context}(\langle\!\langle \mathsf{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathsf{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle) \\
\wedge & \{\pi_i\} & \sqsubseteq_C & \widehat{C}(m_i, pc_i) \\
\wedge & \{m_1, \ldots, m_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(m_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(m_i, pc_i)(\pi_i)
\end{array}
$$

reducing the proof obligation to:

$$
SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})
$$

$$
\iff \beta^{H,JH}_{DynamicHeap}(JH) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.nextAddress(pc_n))(\beta^{R,H,JH,k}_{Context}(SF'))
$$

$$
\begin{array}{lllll}
\wedge & \pi_n & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_n\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.nextAddress(pc_n)) \\
\wedge & \{m_1, \ldots, m_n\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.nextAddress(pc_n))(\pi_n) \\
\wedge & \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_L & \widehat{L}(m_n, m_n.nextAddress(pc_n))(\pi_n) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.nextAddress(pc_n))(\pi_n)
\end{array}
$$

Again from $SF \; \mathcal{R}^{R,H,JH}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$, we have:

$$
\begin{array}{lll}
\beta^{R,H,JH}_{DynamicHeap}(\mathsf{JH}) & \sqsubseteq_{JH} & \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \{\beta^{R,H,JH,k}_{Context}(SF)\} & \sqsubseteq_{C} & \widehat{C}(m_n, pc_n) \\
\wedge \quad \{\mathsf{m}_1, \ldots, \mathsf{m}_n\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_{L} & \widehat{L}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_{S} & \widehat{S}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF))
\end{array}
$$

Substituting in:

$$
SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})
$$

$$
\begin{array}{lll}
\iff & \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \sqsubseteq_{JH} \widehat{JH}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) & \\
\quad \wedge \quad \{\beta^{R,H,JH,k}_{Context}(SF')\} & \sqsubseteq_{C} & \widehat{C}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n)) \\
\quad \wedge \quad \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_{S} & \widehat{S}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\quad \wedge \quad \widehat{L}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) & \sqsubseteq_{L} & \widehat{L}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\quad \wedge \quad \widehat{MNAMES}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF'))
\end{array}
$$

and by change of bound variable:

$$SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) \iff$$

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}$$

the result follows.

■

B.5.6 Corollary to Subject Reduction Simplification Lemma 1

Corollary B.5.4 (Only operand stack changed in intra-method transition)**.** Let:

- $P \in$ Program
- $C = \langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P \left| \begin{array}{l} C \Rightarrow \\ C' \end{array} \right.$
- $C' = \langle R, K, H, I, HID, JH, CHN, SF' \rangle$
- $SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$
- $SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, addr), L_n, S \rangle$
- $C \; \mathcal{R}^{R,H,JH}_{\text{Config}} \; (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$

Then:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, addr) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, addr)(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, addr)(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, addr)(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, addr)(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})
\end{array}
$$

Proof:

Lemma B.5.3 with SF' top stack frame intra-method transition generalised to program counter $addr$ throughout proof.

■

B.5.7 Subject Reduction Simplification Lemma 2

Lemma B.5.5 (Only operand stack and local variable array change in transition to next address)**.** Let:

- $P \in$ Program
- $C = \langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P \Big| \begin{array}{l} C \Rightarrow \\ C' \end{array}$
- $C' = \langle R, K, H, I, HID, JH, CHN, SF' \rangle$
- $SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$
- $SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L, S \rangle$
- $C \; \mathcal{R}^{R,H,JH}_{\text{Config}} \; (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$

Then:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
 & & & \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

Proof:

We have $\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$ by assumption and so proof obligation reduces to proving:

$$C' \; \mathcal{R}^{R,H,JH,k}_{\text{Config}} (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$$

From $C\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$ we have:

$$
\begin{array}{lllll}
 & \beta^{R,H,JH}_{Registry}(R) & \sqsubseteq_R & \widehat{R} \\
\wedge & \beta^{H}_{StaticHeap}(K) & \sqsubseteq_K & \widehat{K} \\
\wedge & \beta^{H,JH}_{DynamicHeap}(H) & \sqsubseteq_H & \widehat{H} \\
\wedge & \beta^{R,H,JH}_{Invalidated}(I) & \sqsubseteq_I & \widehat{I}
\end{array}
$$

Since R, K, H and I remain unchanged in C', proving $C'\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$ reduces to proving:

$$SF'\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})\ \wedge \{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$$

Now:

$$
\begin{array}{lllll}
 & SF'\ \mathcal{R}^{R,H,JH,k}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) \\
\iff & \beta^{H,JH}_{DynamicHeap}(JH) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\wedge & \forall i \in \{3, \ldots, n\} : \\
 & \quad \pi_i & = & \beta^{R,H,JH,k}_{Context}(\langle\!\langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathit{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle) \\
 & \wedge\ \{\pi_i\} & \sqsubseteq_C & \widehat{C}(m_i, pc_i) \\
 & \wedge\ \{m_1, \ldots, m_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_i, pc_i)(\pi_i) \\
 & \wedge\ \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(m_i, pc_i)(\pi_i) \\
 & \wedge\ \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(m_i, pc_i)(\pi_i)
\end{array}
$$

Since SF and SF' are identical below the top stack frame, we have from $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$:

$\forall i \in \{3, \ldots, (n-1)\}$:

$$
\begin{array}{lllll}
 & \pi_i & = & \beta^{R,H,JH,k}_{Context}(\langle\!\langle \mathsf{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathsf{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle) \\
\wedge & \{\pi_i\} & \sqsubseteq_C & \widehat{C}(\mathsf{m}_i, pc_i) \\
\wedge & \{\mathsf{m}_1, \ldots, \mathsf{m}_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathsf{m}_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(\mathsf{m}_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(\mathsf{m}_i, pc_i)(\pi_i)
\end{array}
$$

reducing the proof obligation to:

$$
\begin{array}{lllll}
 & SF' \; \mathcal{R}^{R,H,JH,k}_{\mathrm{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) & & & \\
 & & & & \\
\iff & \beta^{H,JH}_{DynamicHeap}(\mathsf{JH}) \sqsubseteq_{JH} \widehat{JH}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) & & & \\
 & \wedge & \pi_n & = & \beta^{R,H,JH,k}_{Context}(SF') \\
 & \wedge & \{\pi_n\} & \sqsubseteq_C & \widehat{C}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n)) \\
 & \wedge & \{\mathsf{m}_1, \ldots, \mathsf{m}_n\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_n) \\
 & \wedge & \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_n) \\
 & \wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\pi_n)
\end{array}
$$

Again from $SF \; \mathcal{R}^{R,H,JH}_{\mathrm{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$, we have:

$$
\begin{array}{lll}
\beta^{R,H,JH}_{DynamicHeap}(\mathit{JH}) & \sqsubseteq_{JH} & \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \{\beta^{R,H,JH,k}_{Context}(SF)\} & \sqsubseteq_{C} & \widehat{C}(m_n, pc_n) \\
\wedge \quad \{\mathit{m}_1, \ldots, \mathit{m}_n\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_{L} & \widehat{L}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_{S} & \widehat{S}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF))
\end{array}
$$

Substituting in:

$$
SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})
$$

$$
\begin{array}{lll}
\iff & \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \sqsubseteq_{JH} \widehat{JH}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) & \\
\quad \wedge \quad \{\beta^{R,H,JH,k}_{Context}(SF')\} & \sqsubseteq_{C} & \widehat{C}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n)) \\
\quad \wedge \quad \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_{S} & \widehat{S}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\quad \wedge \quad \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_{L} & \widehat{L}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\quad \wedge \quad \widehat{MNAMES}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathit{m}_n, \mathit{m}_n.\mathit{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF'))
\end{array}
$$

and by change of bound variable:

$$SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) \iff$$

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{LocalVar}(L) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}$$

the result follows.

■

B.5.8 Subject Reduction Simplification Lemma 3

Lemma B.5.6 (Only operand stack and existing object field changed in transition to next address)**.** Let:

- $P \in$ Program
- $C = \langle R, K, H, I, HID, JH, CHN, SF \rangle$ be a well-formed semantic configuration such that $P \Big| \begin{array}{l} C \Rightarrow \\ C' \end{array}$
- $(\mathtt{r}, \mathit{loc}, (m_a, pc_a)) \in \mathit{dom}(H)$
- $C' = \langle R, K, H[\mathit{loc}.\mathit{values}(f.\mathit{id}) \mapsto (t, v, (m_b, pc_b))], I, HID, JH, CHN, SF' \rangle$
- $SF = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, pc_n), L_n, S_n \rangle$
- $SF' = \langle \mathit{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle :: \langle \mathit{loc}_2, itd_2, ctd_2, io_2, (m_2, pc_2), L_2, S_2 \rangle :: \ldots :: \langle \mathit{loc}_n, itd_n, ctd_n, io_n, (m_n, m_n.\mathit{nextAddress}(pc_n)), L_n, S \rangle$
- $C \; \mathcal{R}^{R,H,JH}_{\text{Config}} \; (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$

Then:

$$
\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \beta^{R,H,JH}_{Val}(t, v, (m_b, pc_b)) & \sqsubseteq_{Val} & \widehat{H}\big(\beta^{R,H,JH}_{Ref}((\mathtt{r}, \mathit{loc}, (m_a, pc_a)))\big).\mathit{values}(f.\mathit{id}) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG} \\
 & & & \\
\Rightarrow & C' & \mathcal{R}^{R,H,JH,k}_{\text{Config}} & (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG}) \\
\wedge & \{(\pi_1, \pi_2)\} & \sqsubseteq_{CG} & \widehat{CG}
\end{array}
$$

Proof:

We have $\{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$ by assumption and so proof obligation reduces to proving:

$$C' \; \mathcal{R}^{R,H,JH,k}_{\text{Config}} (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$$

From $C\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$ we have:

$$
\begin{array}{lll}
\beta^{R,H,JH}_{Registry}(R) & \sqsubseteq_R & \widehat{R} \\
\wedge\ \beta^{H}_{StaticHeap}(K) & \sqsubseteq_K & \widehat{K} \\
\wedge\ \beta^{H,JH}_{DynamicHeap}(H) & \sqsubseteq_H & \widehat{H} \\
\wedge\ \beta^{R,H,JH}_{Invalidated}(I) & \sqsubseteq_I & \widehat{I}
\end{array}
$$

and since $H' = H[loc.values(f.id) \mapsto (t, v, (m_b, pc_b))]$ we require only:

$$\beta^{R,H,JH}_{Val}(t, v, (m_b, pc_b)) \sqsubseteq_{Val} \widehat{H}\big(\beta^{R,H,JH}_{Ref}((\mathtt{r}, loc, (m_a, pc_a)))\big).values(f.id)$$

to conclude $\beta^{H,JH}_{DynamicHeap}(H') \sqsubseteq_H \widehat{H}$. Since we have this by one of the assumptions and since R, K and I remain unchanged in C', proving $C'\ \mathcal{R}^{R,H,JH}_{\text{Config}}\ (\widehat{R}, \widehat{K}, \widehat{H}, \widehat{I}, \widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{E}, \widehat{MNAMES}, \widehat{REC}, \widehat{CG})$ reduces to proving:

$$SF'\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})\ \wedge \{(\pi_1, \pi_2)\} \sqsubseteq_{CG} \widehat{CG}$$

Now:

$$
\begin{array}{lll}
& SF'\ \mathcal{R}^{R,H,JH,k}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) & \\
\iff & \beta^{H,JH}_{DynamicHeap}(JH) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.nextAddress(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) & \\
\wedge & \forall i \in \{3, \ldots, n\} : & \\
& \quad \pi_i = \beta^{R,H,JH,k}_{Context}\big(\langle\!\langle loc_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle loc_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle\big) & \\
& \wedge\ \{\pi_i\} \sqsubseteq_C \widehat{C}(m_i, pc_i) & \\
& \wedge\ \{m_1, \ldots, m_i\} \sqsubseteq_{MNAMES} \widehat{MNAMES}(m_i, pc_i)(\pi_i) & \\
& \wedge\ \beta^{R,H,JH}_{LocalVar}(L_i) \sqsubseteq_L \widehat{L}(m_i, pc_i)(\pi_i) & \\
& \wedge\ \beta^{R,H,JH}_{Stack}(S_i) \sqsubseteq_S \widehat{S}(m_i, pc_i)(\pi_i) &
\end{array}
$$

Since SF and SF' are identical below the top stack frame, we have from $SF\ \mathcal{R}^{R,H,JH}_{\text{CallStack}}\ (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$:

$\forall i \in \{3, \ldots, (n-1)\}$:

$$
\begin{array}{lllll}
 & \pi_i & = & \beta^{R,H,JH,k}_{Context}(\langle\!\langle \mathsf{loc}_1, itd_1, ctd_1, io_1, (m_1, pc_1), L_1, S_1 \rangle\!\rangle :: \ldots :: \langle\!\langle \mathsf{loc}_i, itd_i, ctd_i, io_i, (m_i, pc_i), L_i, S_i \rangle\!\rangle) \\
\wedge & \{\pi_i\} & \sqsubseteq_C & \widehat{C}(m_i, pc_i) \\
\wedge & \{m_1, \ldots, m_i\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{LocalVar}(L_i) & \sqsubseteq_L & \widehat{L}(m_i, pc_i)(\pi_i) \\
\wedge & \beta^{R,H,JH}_{Stack}(S_i) & \sqsubseteq_S & \widehat{S}(m_i, pc_i)(\pi_i)
\end{array}
$$

reducing the proof obligation to:

$$
\begin{array}{lllll}
 & SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) & & & \\
 & & & & \\
\iff & \beta^{H,JH}_{DynamicHeap}(JH) \sqsubseteq_{JH} \widehat{JH}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) & & \\
 & \wedge \; \pi_n & = & \beta^{R,H,JH,k}_{Context}(SF') & \\
 & \wedge \; \{\pi_n\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathsf{nextAddress}(pc_n)) & \\
 & \wedge \; \{m_1, \ldots, m_n\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_n) & \\
 & \wedge \; \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_n) & \\
 & \wedge \; \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathsf{nextAddress}(pc_n))(\pi_n) &
\end{array}
$$

Again from $SF \; \mathcal{R}^{R,H,JH}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG})$, we have:

$$
\begin{array}{lll}
\beta^{H,JH}_{DynamicHeap}(\mathsf{JH}) & \sqsubseteq_{JH} & \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \{\beta^{R,H,JH,k}_{Context}(SF)\} & \sqsubseteq_{C} & \widehat{C}(m_n, pc_n) \\
\wedge \quad \{\mathsf{m}_1, \ldots, \mathsf{m}_n\} & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \beta^{R,H,JH}_{LocalVar}(L_n) & \sqsubseteq_{L} & \widehat{L}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \\
\wedge \quad \beta^{R,H,JH}_{Stack}(S_n) & \sqsubseteq_{S} & \widehat{S}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF))
\end{array}
$$

Substituting in:

$$
\begin{array}{lll}
SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) & & \\
\iff \quad \widehat{JH}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) \sqsubseteq_{JH} \widehat{JH}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) & & \\
\wedge \quad \{\beta^{R,H,JH,k}_{Context}(SF')\} & \sqsubseteq_{C} & \widehat{C}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n)) \\
\wedge \quad \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_{S} & \widehat{S}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\wedge \quad \widehat{L}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) & \sqsubseteq_{L} & \widehat{L}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF')) \\
\wedge \quad \widehat{MNAMES}(m_n, pc_n)(\beta^{R,H,JH,k}_{Context}(SF)) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(\mathsf{m}_n, \mathsf{m}_n.\mathsf{nextAddress}(pc_n))(\beta^{R,H,JH,k}_{Context}(SF'))
\end{array}
$$

and by change of bound variable:

$$SF' \; \mathcal{R}^{R,H,JH,k}_{\text{CallStack}} \; (\widehat{JH}, \widehat{C}, \widehat{L}, \widehat{S}, \widehat{MNAMES}, \widehat{CG}) \iff$$

$$\begin{array}{lllll}
 & \pi_1 & = & \beta^{R,H,JH,k}_{Context}(SF) \\
\wedge & \pi_2 & = & \beta^{R,H,JH,k}_{Context}(SF') \\
\wedge & \{\pi_2\} & \sqsubseteq_C & \widehat{C}(m_n, m_n.\mathit{nextAddress}(pc_n)) \\
\wedge & \beta^{R,H,JH}_{Stack}(S) & \sqsubseteq_S & \widehat{S}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{L}(m_n, pc_n)(\pi_1) & \sqsubseteq_L & \widehat{L}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{JH}(m_n, pc_n)(\pi_1) & \sqsubseteq_{JH} & \widehat{JH}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2) \\
\wedge & \widehat{MNAMES}(m_n, pc_n)(\pi_1) & \sqsubseteq_{MNAMES} & \widehat{MNAMES}(m_n, m_n.\mathit{nextAddress}(pc_n))(\pi_2)
\end{array}$$

the result follows.

■

B.5.9 Abstract Numerical Operations

B.5.9.1 Abstract Binary Numerical Operators

Lemma B.5.7 (Abstract number produced by result of abstract binary numerical operators contains the concrete number produced in the op. sem.)**.** Let:

- $t \in \{\mathtt{b}, \mathtt{s}, \mathtt{i}\}$
- $binop \in$ BinaryNumericOperator
- $(t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q))$ be valid numeric values i.e. the constants c_1, c_2 are in range of their type t
- $(t, r, (m_n, pc_n)) = \mathit{applyBinary}(binop, (t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q)), (m_n, pc_n))$
- $\beta_{Num}(t, c_1, (m_p, pc_p)) \sqsubseteq_{Val} \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\}$
- $\beta_{Num}(t, c_2, (m_q, pc_q)) \sqsubseteq_{Val} \{(t, (l_2, h_2, mod_2), (m_q, pc_q))\}$
- $(t, (l_3, h_3, mod_3), (m_n, pc_n)) = absApplyBinary\left(binop, \begin{array}{l}(t, (l_1, h_1, mod_1), (m_p, pc_p)),\\ (t, (l_2, h_2, mod_2), (m_q, pc_q)), (m_n, pc_n)\end{array}\right)$

Then:

$$\beta_{Num}(t, r, (m_n, pc_n)) \ \sqsubseteq_{Val} \ \{(t, (l_3, h_3, mod_3), (m_n, pc_n))\}$$

Proof:

By definition:

$$\beta_{Num}(t, c_1, (m_p, pc_p)) \sqsubseteq_{Val} \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \iff l_1 \leq c_1 \leq h_1$$
$$\beta_{Num}(t, c_2, (m_q, pc_q)) \sqsubseteq_{Val} \{(t, (l_2, h_2, mod_2), (m_q, pc_q))\} \iff l_2 \leq c_2 \leq h_2$$

Define:

$$\begin{array}{lll}(t, (l_3, h_3, mod_3), (m_n, pc_n)) & = & absApplyBinary\left(binop, \begin{array}{l}(t, (l_1, h_1, mod_1), (m_p, pc_p)),\\ (t, (l_2, h_2, mod_2), (m_q, pc_q)), (m_n, pc_n)\end{array}\right)\\ & \iff & l_3 = \mathtt{min}(result_{i,j}) \wedge h_3 = \mathtt{max}(result_{i,j}) \wedge mod_3 = \mathtt{max}(mod_1, mod_2)\\ & \wedge & \forall\, i \in \{l_1, \ldots, h_1\}\, \forall\, j \in \{l_2, \ldots, h_2\} :\\ & & (t, result_{i,j}, (m_n, pc_n)) = \mathit{applyBinary}(binop, (t, i, (m_p, pc_p)), (t, j, (m_q, pc_q)), (m_n, pc_n))\end{array}$$

In particular:

$$\begin{array}{rcl}(t, result_{c_1,c_2}, (m_n, pc_n)) & = & \mathit{applyBinary}(binop, (t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q)), (m_n, pc_n))\\ & = & (t, r, (m_n, pc_n))\end{array}$$

and so:

$$\begin{array}{lll} \texttt{min}(result_{i,j}) & \leq result_{c_1,c_2} & \leq \texttt{max}(result_{i,j}) \Rightarrow \\ l_3 & \leq r & \leq h_3 \end{array}$$

and the result follows.

■

B.5.9.2 Abstract Unary Numerical Operators

Lemma B.5.8 (Abstract number produced by result of abstract unary numerical operators contains the concrete number produced in the op. sem. First $unop$ form)**.** Let:

- $t \in \{\texttt{b}, \texttt{s}, \texttt{i}\}$
- $unop \in$ UnaryNumericOperator
- $(t, c_1, (m_p, pc_p))$ be valid numeric values i.e. the constant c_1 is in range of its type t
- $(t, r, (m_n, pc_n)) = \mathit{applyUnary}(unop, (t, c_1, (m_p, pc_p)), (m_n, pc_n))$
- $\beta_{Num}(t, c_1, (m_p, pc_p)) \sqsubseteq_{Val} \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\}$
- $(t, (l_3, h_3, mod_3), (m_n, pc_n)) = absApplyUnary(unop, (t, (l_1, h_1, mod_1), (m_p, pc_p)), (m_n, pc_n))$

Then:

$$\beta_{Num}(t, r, (m_n, pc_n)) \ \sqsubseteq_{Val} \ \{(t, (l_3, h_3, mod_3), (m_n, pc_n))\}$$

Proof:

By definition:

$$\beta_{Num}(t, c_1, (m_p, pc_p)) \sqsubseteq_{Val} \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \quad \iff \quad l_1 \leq c_1 \leq h_1$$

Define:

$$\begin{array}{rcl} (t, (l_3, h_3, mod_3), (m_n, pc_n)) & = & absApplyUnary\left(unop, \ \ (t, (l_1, h_1, mod_1), (m_p, pc_p)), (m_n, pc_n)\right) \\ & \iff & l_3 = \texttt{min}(result_i) \wedge h_3 = \texttt{max}(result_i) \wedge mod_3 = mod_1 \\ & \wedge & \forall\, i \in \{l_1, \ldots, h_1\} : \\ & & (t, result_i, (m_n, pc_n)) = \mathit{applyUnary}(unop, (t, i, (m_p, pc_p)), (m_n, pc_n)) \end{array}$$

In particular:

$$\begin{array}{rcl} (t, result_{c_1}, (m_n, pc_n)) & = & \mathit{applyUnary}(unop, (t, c_1, (m_p, pc_p)), (m_n, pc_n)) \\ & = & (t, r, (m_n, pc_n)) \end{array}$$

and so:

$$\begin{array}{lll} \texttt{min}(result_i) & \leq result_{c_1} & \leq \texttt{max}(result_i) \ \Rightarrow \\ l_3 & \leq r & \leq h_3 \end{array}$$

and the result follows.

■

Lemma B.5.9 (Abstract number produced by result of abstract unary numerical operators contains the concrete number produced in the op. sem. Second $unop$ form)**.** Let:

- $t \in \{\mathtt{b}, \mathtt{s}, \mathtt{i}\}$
- $unop \in$ UnaryNumericOperator
- $(t, c_1, (m_p, pc_p))$ be valid numeric values i.e. the constant c_1 is in range of its type t
- $(t, r, (m_n, pc_n)) = \mathit{applyUnary}(unop, opt_t, (t, c_1, (m_p, pc_p)), (m_n, pc_n))$
- $\beta_{Num}(t, c_1, (m_p, pc_p)) \sqsubseteq_{Val} \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\}$
- $(t, (l_3, h_3, mod_3), (m_n, pc_n)) = absApplyUnary(unop, opt_t, (t, (l_1, h_1, mod_1), (m_p, pc_p)), (m_n, pc_n))$

Then:

$$\beta_{Num}(t, r, (m_n, pc_n)) \ \sqsubseteq_{Val} \ \{(t, (l_3, h_3, mod_3), (m_n, pc_n))\}$$

Proof:

By definition:

$$\beta_{Num}(t, c_1, (m_p, pc_p)) \sqsubseteq_{Val} \{(t, (l_1, h_1, mod_1), (m_p, pc_p))\} \quad \iff \quad l_1 \leq c_1 \leq h_1$$

Define:

$$\begin{array}{rcl} (t, (l_3, h_3, mod_3), (m_n, pc_n)) & = & absApplyUnary\left(unop, opt_t, \ \ (t, (l_1, h_1, mod_1), (m_p, pc_p)), (m_n, pc_n)\ \right) \\ & \iff & l_3 = \mathtt{min}(result_i) \wedge h_3 = \mathtt{max}(result_i) \wedge mod_3 = mod_1 \\ & \wedge & \forall\, i \in \{l_1, \ldots, h_1\} : \\ & & (t, result_i, (m_n, pc_n)) = \mathit{applyUnary}(unop, opt_t, (t, i, (m_p, pc_p)), (m_n, pc_n)) \end{array}$$

In particular:

$$\begin{array}{rcl} (t, result_{c_1}, (m_n, pc_n)) & = & \mathit{applyUnary}(unop, opt_t, (t, c_1, (m_p, pc_p)), (m_n, pc_n)) \\ & = & (t, r, (m_n, pc_n)) \end{array}$$

and so:

$$\begin{array}{llll} \mathtt{min}(result_i) & \leq result_{c_1} & \leq \mathtt{max}(result_i) & \Rightarrow \\ l_3 & \leq r & \leq h_3 & \end{array}$$

and the result follows.

■

B.5.10 Abstract Logical Operations

B.5.10.1 Abstract Binary Logical Operators

Lemma B.5.10 (Abstract boolean produced by result of abstract binary logical operators contains the concrete boolean produced in the op. sem.)**.** Let:

- $t \in \{\texttt{r}, \texttt{s}\}$
- $binop \in$ ComparisonOperator
- $(t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q))$ be valid operand values according to their type t
- $r \in \{\texttt{true}, \texttt{false}\} = \textit{applyBinary}(binop, (t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q)), (m_n, pc_n))$
- $\beta_{Val}^{R,H,JH}(t, c_1, (m_p, pc_p)) \sqsubseteq_{Val} \{(t, Y_1, (m_p, pc_p))\}$
- $\beta_{Val}^{R,H,JH}(t, c_2, (m_q, pc_q)) \sqsubseteq_{Val} \{(t, Y_2, (m_q, pc_q))\}$
- $S \subseteq \mathcal{P}(\texttt{true}, \texttt{false}).S \neq \emptyset = absApplyBinary \left(binop, \begin{array}{l} (t, Y_1, (m_p, pc_p)), \\ (t, Y_2, (m_q, pc_q)), (m_n, pc_n) \end{array} \right)$

Then:

$$\textit{applyBinary}(binop, (t, c_1, (m_p, pc_p)), (t, c_2, (m_q, pc_q)), (m_n, pc_n)) \in absApplyBinary \left(binop, \begin{array}{l} (t, (l_1, h_1, mod_1), (m_p, pc_p)), \\ (t, (l_2, h_2, mod_2), (m_q, pc_q)), (m_n, pc_n) \end{array} \right)$$

Proof:

Define:

$$absApplyBinary \left(binop, \begin{array}{l} (t, Y_1, (m_p, pc_p)), \\ (t, Y_2, (m_q, pc_q)), (m_n, pc_n) \end{array} \right) = \begin{cases} absApplyBinaryNum \left(binop, \begin{array}{l} (t, Y_1, (m_p, pc_p)), \\ (t, Y_2, (m_q, pc_q)), (m_n, pc_n) \end{array} \right) & t = \texttt{s} \\ absApplyBinaryRef \left(binop, \begin{array}{l} (t, Y_1, (m_p, pc_p)), \\ (t, Y_2, (m_q, pc_q)), (m_n, pc_n) \end{array} \right) & t = \texttt{r} \end{cases}$$

where:

$$absApplyBinaryNum \left(binop, \begin{array}{l} (t, (l_1, h_1, mod_1), (m_p, pc_p)), \\ (t, (l_2, h_2, mod_2), (m_q, pc_q)), (m_n, pc_n) \end{array} \right)$$
$$= \bigcup_{i \in \{l_1, \ldots, h_1\}} \bigcup_{j \in \{l_2, \ldots, h_2\}} \textit{applyBinary}(binop, (t, i, (m_p, pc_p)), (t, j, (m_q, pc_q)), (m_n, pc_n))$$

and:

$$absApplyBinaryRef \quad \left(\begin{array}{l} binop, \\ \\ \end{array} \begin{array}{l} (\mathit{t}, Y_1, (\mathit{m}_p, \mathit{pc}_p)), \\ (\mathit{t}, Y_2, (\mathit{m}_q, \mathit{pc}_q)), (\mathit{m}_n, \mathit{pc}_n) \end{array} \right)$$

$$= \begin{cases} \{\texttt{true}, \texttt{false}\} \iff \left(\begin{array}{lcl} Y_1.\mathit{type} & = & Y_2.\mathit{type} \wedge \\ Y_1.\mathit{refType} & = & Y_2.\mathit{refType} \wedge \\ Y_1.\mathit{isArray} & = & Y_2.\mathit{isArray} \wedge \\ Y_1.\mathit{owner} & = & Y_2.\mathit{owner} \wedge \\ Y_1.\mathit{entryPoint} & = & Y_2.\mathit{entryPoint} \wedge \\ Y_1.\mathit{isGlobal} & = & Y_2.\mathit{isGlobal} \wedge \\ Y_1.\mathit{transient} & = & Y_2.\mathit{transient} \wedge \\ Y_1.\mathit{creationPoint} & = & Y_2.\mathit{creationPoint} \wedge \\ Y_1.\mathit{length} & = & Y_2.\mathit{length} \end{array} \right) \\ \{\texttt{false}\} \qquad \text{otherwise} \end{cases}$$

The proof is immediate when $(t, c_1, (\mathit{m}_p, \mathit{pc}_p)), (t, c_2, (\mathit{m}_q, \mathit{pc}_q))$ are abstract numbers since by definition:

$$\beta_{Num}(t, c_1, (\mathit{m}_p, \mathit{pc}_p)) \sqsubseteq_{Val} \{(\mathit{t}, (l_1, h_1, mod_1), (\mathit{m}_p, \mathit{pc}_p))\} \iff l_1 \leq c_1 \leq h_1$$

$$\beta_{Num}(t, c_2, (\mathit{m}_q, \mathit{pc}_q)) \sqsubseteq_{Val} \{(\mathit{t}, (l_2, h_2, mod_2), (\mathit{m}_q, \mathit{pc}_q))\} \iff l_2 \leq c_2 \leq h_2$$

and so when $c_1 = i, c_2 = j,$ $\mathit{applyBinary}(binop, (t, c_1, (\mathit{m}_p, \mathit{pc}_p)), (t, c_2, (\mathit{m}_q, \mathit{pc}_q)), (\mathit{m}_n, \mathit{pc}_n)) \in$

$$absApplyBinaryNum \quad \left(\begin{array}{l} binop, \\ \\ \end{array} \begin{array}{l} (\mathit{t}, (l_1, h_1, mod_1), (\mathit{m}_p, \mathit{pc}_p)), \\ (\mathit{t}, (l_2, h_2, mod_2), (\mathit{m}_q, \mathit{pc}_q)), (\mathit{m}_n, \mathit{pc}_n) \end{array} \right)$$

$$= \bigcup_{i \in \{l_1, \ldots, h_1\}} \bigcup_{j \in \{l_2, \ldots, h_2\}} \mathit{applyBinary}(binop, (t, i, (\mathit{m}_p, \mathit{pc}_p)), (t, j, (\mathit{m}_q, \mathit{pc}_q)), (\mathit{m}_n, \mathit{pc}_n))$$

When concrete object references are compared, the *heapID* may be used to conclusively answer the question of reference/pointer equality. When *heapID*s are identical, it must be the case that all the attributes of the object reference are identical. The converse is not true, when all attributes of two object references match, this does mean they refer to the same object. When attributes of two object references differ, they definitely do not refer to the same object. This logic is encoded in the $absApplyBinaryRef$, and so the result follows.

■

Appendix C

A Reaching Definitions for Local Variables Analysis

C.1 Introduction

In this Chapter, we recast the classical Reaching Definitions analysis in a Carmel setting and define a Reaching Definitions for Local Variable analysis capable of determining for each address in a Carmel program P the set of possible pairs of (local variable $\times$ program counter) where each local variable may have received its definition (assignment). The material in this chapter reproduces and reuses the monotone framework presented in Chapter 2 of [NNH10]. In particular, we present our reaching definitions for local variables analysis as an instance of a monotone framework based on the reaching definitions analysis monotone framework instance, detail the corresponding $flow$, $kill$ and gen functions, and reproduce the general MFP solver given in [NNH10] capable of efficiently solving the dataflow equations generated from the analysis.

Note that in Carmel every method m has a unique entry point m.*firstAddress* and the set of local variables at that address corresponds to the binding of the formal parameters of the method with their runtime values, and is populated by the calling method. We include the formal parameters as the initial values for m.*firstAddress*. Additionally, our analysis considers the local variables that are actually referenced in a method i.e. the local variables that are referenced in any of the 3 following instructions:

`load` *t i*

`store` *t i*

`inc` *t j c*

of which only instructions:

`store` *t i*

`inc` *t j c*

assign new values to a local variable[1].

C.2 Scope and Definition of Analysis

The scope of our analysis is per Carmel method m in P and we redefine:

$$\begin{aligned}\textbf{Var}_* &= \{i \mid m.\textit{instructionAt}(pc) \in \{\texttt{load } t\ i, \texttt{store } t\ i, \texttt{inc } t\ i\ c\},\ (m, pc) \in m.\textit{addresses}\} \cup \\ &\quad \{j \mid j \in \{0, \ldots, n\}, (\tau_i)_0^n \to \tau_r = m.\textit{type}\} \\ \textbf{Lab}_* &= \{pc \mid (m, pc) \in m.\textit{addresses}\}\end{aligned}$$

and note that since P has passed bytecode verification, both of these sets must be finite, and therefore the powerset of their cross-product must be finite too.

Table C.1 presents the Reaching Definitons Local Variable analysis as an instance of a monotone framework as presented in [NNH10].
Table C.2 defines the corresponding $kill$ and gen functions.
Table C.3 defines the $flow$ relation to generate the intra-method control-flow.
Table C.4 reproduces the MFP solver that is capable of solving the constraints generated by the analysis.

[1] The `load` *t i* instruction simply $reads$ and pushes on the top of the operand stack the value of the local variable *i* of type *t*.

Table C.1: Monotone framework definition for Reaching Definitions for Local Variables Analysis

	Reaching Definitions for Local Variables
L	$\mathcal{P}(\mathbf{Var}_* \times \mathbf{Lab}_*)$
$\sqsubseteq$	$\subseteq$
$\bigsqcup$	$\cup$
$\bot$	$\emptyset$
ι	$\{j \mid j \in \{0,\ldots,n\}, (\tau_i)_0^n \rightarrow \tau_r = \textit{m.type}\}$
E	$\{m.\textit{firstAddress}\}$
F	$flow(m, m.\textit{firstAddress})$
$\mathcal{F}$	$\{f : L \mapsto L \mid \exists l_k, l_g : f(l) = (l \backslash l_k) \cup l_g\}$
f_ℓ	$f_\ell(l) = (l \backslash kill_{RDLV}(m, m.\textit{instructionAt}(\ell)) \cup gen_{RDLV}(m, m.\textit{instructionAt}(\ell))$

Table C.2: Kill- and gen- function definition for Reaching Definitions for Local Variables Analysis

$(m, m.\textit{instructionAt}(\ell))$	$kill_{RDLV}(m, m.\textit{instructionAt}(\ell))$	$gen_{RDLV}(m, m.\textit{instructionAt}(\ell))$
$\in \{$`store` *t i*, `inc` *t i c*$\}$	$\{(i, m.\textit{firstAddress})\} \cup$ $\{(i, pc) \mid m.\textit{instructionAt}(pc) \in \{$`store` *t i*, `inc` *t i c*$\}, pc \in m.\textit{addresses}\}$	$\{(i, \ell)\}$
$\notin \{$`store` *t i*, `inc` *t i c*$\}$	$\emptyset$	$\emptyset$

Table C.3: Flow relation $flow$ for Reaching Definitions for Local Variables Analysis

$(m, m.\textit{instructionAt}(\ell))$	$flow(m, m.\textit{instructionAt}(\ell))$
`return` `return` *t*	$\emptyset$
`goto` *addr*	$\{(\ell, \textit{addr})\} \cup flow(m, \textit{addr})$
`if` *t op* `goto` *addr* `if` *t op* `null goto` *addr*	$\{(\ell, \textit{addr})\} \cup \{(\ell, m.\textit{nextAddress}(\ell))\} \cup flow(m, \textit{addr}) \cup flow(m, m.\textit{nextAddress}(\ell))$
`lookupswitch` *t* $(k_i$ => $apc_i)_1^r$, `default` => $apc_{default}$ `tableswitch` *t low* => $(apc_i)_1^r$, `default` => $apc_{default}$	$\bigcup_{i \in \{1,..,r\} \cup \{default\}} \{(\ell, apc_i)\} \cup \bigcup_{i \in \{1,..,r\} \cup \{default\}} flow(m, apc_i)$
$\notin \{$ `return` `return` *t* `goto` *addr* `if` *t op* `goto` *addr* `if` *t op* `null goto` *addr* `lookupswitch` *t* $(k_i$ => $apc_i)_1^r$, `default` => $apc_{default}$ `tableswitch` *t low* => $(apc_i)_1^r$, `default` => $apc_{default}$ $\}$	$\{(\ell, m.\textit{nextAddress}(\ell))\} \cup flow(m, m.\textit{nextAddress}(\ell))$

Table C.4: Algorithm for solving dataflow equations arising from Reaching Definitions for Local Variables Analysis

INPUT: An instance of a Monotone Framework:
$(L, \mathcal{F}, F, E, \iota, f)$

OUTPUT: MFP$_\circ$, $MFP_\bullet$

Step 1: Initialisation (of W and Analysis)

```
W := nil;
for all (ℓ, ℓ') in F do
    W := cons((ℓ, ℓ'), W);
for all ℓ in F or E do
    if ℓ ∈ E then Analysis[ℓ] := ι
             else Analysis[ℓ] := ⊥_L;
```

Step 2: Iteration (updating W and Analysis)

```
while W ≠ nil do
    ℓ := fst(head(W));
    ℓ' = snd(head(W));
    W := tail(W);
    if f_ℓ(Analysis[ℓ]) ⋢ Analysis[ℓ'] then
        Analysis[ℓ'] := Analysis[ℓ'] ⊔ f_ℓ(Analysis[ℓ]);
        for all (ℓ', ℓ'') in F do
            W := cons((ℓ', ℓ''), W);
```

Step 3: Presenting the result ($MFP_\circ$ and $MFP_\bullet$)

```
for all ℓ in F or E do
    MFP_∘(ℓ) := Analysis[ℓ];
    MFP_•(ℓ) := f_ℓ(Analysis[ℓ]);
```

Appendix D

Initial States for Object Heap and Static Heap

D.1 Introduction

There are certain objects the JCRE is required to create when the *Java Card* smartcard is first initialised, which the JCVM assumes exist as part of the *Java Card* platform specification, and which are created prior to, and exist independently from, the loading of any user applets and libraries. These framework objects created at initialisation are referenced in the operational semantics of Chapter 3 including the initial configurations of Section 3.7.1.

One such set of objects is the set of JCRE-owned exception objects, as defined in Section D.2.

A second set of objects contain the two global arrays, as defined in Section D.3.

The final set of miscellaneous JCRE-owned objects are defined in Section D.4.

After presenting these three sets of objects, we are able to define H_{Init}[1] and K_{Init}[2] in Sections D.5 and D.6. We then define their abstract equivalents $\widehat{H_{Init}}$ and $\widehat{K_{Init}}$ in Section D.7 and D.8, respectively.

To simplify the specification of these objects, we define in Table D.1 the following operator *loc* $\overset{\sim}{\rightarrow}$ *H* as a shorthand for "*loc* is defined in the object heap with the following properties" i.e. *loc* $\in dom(H)$ and each field of the associated

[1] H_{Init} being the initial object heap of a freshly initialised JCVM referenced in the initial configurations of Section 3.7.1
[2] K_{Init} being the initial static heap of a freshly initialised JCVM referenced in the initial configurations of Section 3.7.1

object is explicitly specified.

$$
\mathit{loc} \quad \overset{\rightsquigarrow}{H} \quad \left\{ \left(\begin{array}{l} \mathtt{r}, \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left(\begin{array}{l} \mathit{type} : \tau_1, \\ \mathit{refType} : \tau_2, \\ \mathit{isArray} : is, \\ \mathit{owner} : own, \\ \mathit{entryPoint} : ep, \\ \mathit{isGlobal} : global, \\ \mathit{transient} : trans, \\ \mathit{creationPoint} : (\mathit{m}, pc), \\ \mathit{length} : l \\ \mathit{heapID} : hId \\ \mathit{values} : vals \end{array} \right) , (\mathit{m}, pc) \right) \middle| \begin{array}{lcl} \tau_1 & = & \mathit{H}(\mathit{loc}).\mathit{type} \\ \tau_2 & = & \mathit{H}(\mathit{loc}).\mathit{refType} \\ is & = & \mathit{H}(\mathit{loc}).\mathit{isArray} \\ own & = & \mathit{H}(\mathit{loc}).\mathit{owner} \\ ep & = & \mathit{H}(\mathit{loc}).\mathit{entryPoint} \\ global & = & \mathit{H}(\mathit{loc}).\mathit{isGlobal} \\ trans & = & \mathit{H}(\mathit{loc}).\mathit{transient} \\ (\mathit{m}, pc) & = & \mathit{H}(\mathit{loc}).\mathit{creationPoint} \\ l & = & \mathit{H}(\mathit{loc}).\mathit{length} \\ hId & = & \mathit{H}(\mathit{loc}).\mathit{heapID} \\ vals & = & \mathit{H}(\mathit{loc}).\mathit{values} \end{array} \right\}
$$

Table D.1: A Succinct Notation for Defining Objects in the domain of object heap *H*

D.2 JCRE–owned exceptions

As per [Ora11a], to encourage resource savings via re-use of exception objects, the JCVM is required to create, hold a reference to, and designate as a JCRE temporary entry point object, at least one instance of each of the following exception objects, which are thrown either automatically by the JCVM during execution of Applet code, or explicitly by Applet code via invocation of static methods.

$$
\mathit{loc}_{\mathtt{ArithmeticException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\begin{array}{l} \mathtt{r}, \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \left(\begin{array}{ll} \mathit{type} : & \mathtt{java.lang.ArithmeticException}, \\ \mathit{refType} : & \mathtt{java.lang.ArithmeticException}, \\ \mathit{isArray} : & \mathtt{false}, \\ \mathit{owner} : & (\mathtt{JCRE}, \mathtt{JCRE}), \\ \mathit{entryPoint} : & \mathtt{temporary}\ , \\ \mathit{isGlobal} : & \mathtt{false}, \\ \mathit{transient} : & \mathtt{NOT_TRANSIENT}, \\ \mathit{creationPoint} : & (1,1), \\ \mathit{length} : & 0 \\ \mathit{heapID} : & -2 \\ \mathit{values} : & [] \end{array} \right) , (1,1) \right)
$$

$$loc_{\texttt{ArrayIndexOutOfBoundsException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.lang.ArrayIndexOutOfBoundsException}, \\ \textit{refType}: & \texttt{java.lang.ArrayIndexOutOfBoundsException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -3 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{ArrayStoreException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.lang.ArrayStoreException}, \\ \textit{refType}: & \texttt{java.lang.ArrayStoreException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -4 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{ClassCastException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.lang.ClassCastException}, \\ \textit{refType}: & \texttt{java.lang.ClassCastException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -5 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{IndexOutOfBoundsException}} \quad \overset{\rightarrow}{\vec{H}} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.lang.IndexOutOfBoundsException}, \\ \textit{refType}: & \texttt{java.lang.IndexOutOfBoundsException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -6 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{NegativeArraySizeException}} \quad \overset{\rightarrow}{\vec{H}} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.lang.NegativeArraySizeException}, \\ \textit{refType}: & \texttt{java.lang.NegativeArraySizeException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -7 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{NullPointerException}} \quad \overset{\rightarrow}{\vec{H}} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.lang.NullPointerException}, \\ \textit{refType}: & \texttt{java.lang.NullPointerException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -8 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{SecurityException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.lang.SecurityException}, \\ \textit{refType}: & \texttt{java.lang.SecurityException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -9 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{CardException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.CardException}, \\ \textit{refType}: & \texttt{java.framework.CardException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -10 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{UserException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.UserException}, \\ \textit{refType}: & \texttt{java.framework.UserException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -11 \\ \textit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$loc_{\texttt{CardRuntimeException}} \overset{\leftrightarrow}{H} \left(\texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.CardRuntimeException}, \\ \textit{refType}: & \texttt{java.framework.CardRuntimeException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -12 \\ \textit{values}: & [] \end{array} \right), (1,1) \right)$$

$$loc_{\texttt{APDUException}} \overset{\leftrightarrow}{H} \left(\texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.APDUException}, \\ \textit{refType}: & \texttt{java.framework.APDUException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -13 \\ \textit{values}: & [] \end{array} \right), (1,1) \right)$$

$$loc_{\texttt{ISOException}} \overset{\leftrightarrow}{H} \left(\texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.ISOException}, \\ \textit{refType}: & \texttt{java.framework.ISOException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -14 \\ \textit{values}: & [] \end{array} \right), (1,1) \right)$$

$$loc_{\texttt{PINException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.PINException}, \\ \textit{refType}: & \texttt{java.framework.PINException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -15 \\ \textit{values}: & [\texttt{java.framework.CardRuntimeException.reason}.\textit{id} \mapsto (\texttt{s}, 0, (1,1))] \end{array} \right), (1,1) \right)$$

$$loc_{\texttt{SystemException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.SystemException}, \\ \textit{refType}: & \texttt{java.framework.SystemException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -16 \\ \textit{values}: & [\texttt{java.framework.CardRuntimeException.reason}.\textit{id} \mapsto (\texttt{s}, 0, (1,1))] \end{array} \right), (1,1) \right)$$

$$loc_{\texttt{TransactionException}} \quad \overset{\rightsquigarrow}{H} \quad \left(\texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{java.framework.TransactionException}, \\ \textit{refType}: & \texttt{java.framework.TransactionException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -17 \\ \textit{values}: & [\texttt{java.framework.CardRuntimeException.reason}.\textit{id} \mapsto (\texttt{s}, 0, (1,1))] \end{array} \right), (1,1) \right)$$

$$loc_{\texttt{CryptoException}} \quad \tilde{H} \quad \left(\texttt{r}, \left(\begin{array}{ll} \textit{type}: & \texttt{javacard.security.CryptoException}, \\ \textit{refType}: & \texttt{javacard.security.CryptoException}, \\ \textit{isArray}: & \texttt{false}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{false}, \\ \textit{transient}: & \texttt{NOT_TRANSIENT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & 0 \\ \textit{heapID}: & -18 \\ \textit{values}: & [\texttt{java.framework.CardRuntimeException.reason}.\textit{id} \mapsto (\texttt{s}, 0, (1,1))] \end{array} \right), (1,1) \right)$$

D.3 JCRE–owned global arrays

The JCRE specification mandates[3] two global arrays:

- the APDU buffer held by the APDU object[4] and associated with the Applet lifecycle method `process(APDU apdu)`
- the installation buffer `bArray` associated with the Applet lifecycle method `install(byte[] bArray, short bOffset, byte bLength)`

Due to the variation in hardware resources available to different smartcards, the JCRE specification [Ora11b] dictates a maximum size for the install buffer, and a minimum size and a maximum size for the APDU buffer supporting extended APDU lengths. We abstract these here for the concrete objects; this poses no problem for our program analysis as we allow the *length* attribute to be an interval of possible lengths for abstract array objects.

$$\begin{array}{rcll} 37 & \leq & \texttt{CARD_SPECIFIC_APDU_BUFFER_LENGTH} & \leq \texttt{Short.MAX_VALUE} \\ 0 & \leq & \texttt{CARD_SPECIFIC_INSTALL_BUFFER_LENGTH} & \leq 127 \end{array}$$

[3] Section 6.2.2 of [Ora11b]
[4] and retrieved via the APDU `getBuffer()` method

$$loc_{\texttt{APDUBuffer}} \quad \tilde{H} \quad \left(\begin{array}{l} \texttt{r}, \\ \left(\begin{array}{ll} \textit{type}: & \texttt{byte}, \\ \textit{refType}: & \texttt{byte[ ]}, \\ \textit{isArray}: & \texttt{true}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{true}, \\ \textit{transient}: & \texttt{CLEAR_ON_DESELECT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & \texttt{CARD_SPECIFIC_APDU_BUFFER_LENGTH} \\ \textit{heapID}: & -19 \\ \textit{values}: & \left[\begin{array}{l} 0 \mapsto (\texttt{b}, 0, (1,1)), \\ \ldots, \\ (\texttt{CARD_SPECIFIC_APDU_BUFFER_LENGTH} - 1) \mapsto (\texttt{b}, 0, (1,1)) \end{array} \right] \end{array} \right) \end{array} , (1,1) \right)$$

$$loc_{\texttt{AppletInstallBuffer}} \quad \tilde{H} \quad \left(\begin{array}{l} \texttt{r}, \\ \left(\begin{array}{ll} \textit{type}: & \texttt{byte}, \\ \textit{refType}: & \texttt{byte[ ]}, \\ \textit{isArray}: & \texttt{true}, \\ \textit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \textit{entryPoint}: & \texttt{temporary}, \\ \textit{isGlobal}: & \texttt{true}, \\ \textit{transient}: & \texttt{CLEAR_ON_DESELECT}, \\ \textit{creationPoint}: & (1,1), \\ \textit{length}: & \texttt{CARD_SPECIFIC_INSTALL_BUFFER_LENGTH} \\ \textit{heapID}: & -20 \\ \textit{values}: & \left[\begin{array}{l} 0 \mapsto (\texttt{b}, 0, (1,1)), \\ \ldots, \\ (\texttt{CARD_SPECIFIC_INSTALL_BUFFER_LENGTH} - 1) \mapsto (\texttt{b}, 0, (1,1)) \end{array} \right] \end{array} \right) \end{array} , (1,1) \right)$$

D.4 Miscellaneous JCRE-owned objects

$$\mathit{loc}_{\texttt{JCRE}} \quad \overset{\leftrightarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \mathit{type}: & \texttt{null}, \\ \mathit{refType}: & \texttt{null}, \\ \mathit{isArray}: & \texttt{false}, \\ \mathit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \mathit{entryPoint}: & \texttt{no}\ , \\ \mathit{isGlobal}: & \texttt{false}, \\ \mathit{transient}: & \texttt{NOT_TRANSIENT}, \\ \mathit{creationPoint}: & (1,1), \\ \mathit{length}: & 0 \\ \mathit{heapID}: & -1 \\ \mathit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

$$\mathit{loc}_{\texttt{APDUObject}} \quad \overset{\leftrightarrow}{H} \quad \left(\begin{array}{l} \texttt{r}, \left(\begin{array}{ll} \mathit{type}: & \texttt{javacard.framework.APDU}, \\ \mathit{refType}: & \texttt{javacard.framework.APDU}, \\ \mathit{isArray}: & \texttt{false}, \\ \mathit{owner}: & (\texttt{JCRE}, \texttt{JCRE}), \\ \mathit{entryPoint}: & \texttt{temporary}\ , \\ \mathit{isGlobal}: & \texttt{false}, \\ \mathit{transient}: & \texttt{NOT_TRANSIENT}, \\ \mathit{creationPoint}: & (1,1), \\ \mathit{length}: & 0 \\ \mathit{heapID}: & -21 \\ \mathit{values}: & [] \end{array} \right), (1,1) \end{array} \right)$$

Note that we do not fill in the fields for $\mathit{loc}_{\texttt{APDUObject}}$ as we intercept method calls to e.g. `getBuffer()` to ensure the $\mathit{loc}_{\texttt{APDUBuffer}}$ is returned.

D.5 Initial State for Object Heap - $\boldsymbol{H}_{Init}$

We may now define the following framework objects created at initialisation and which are referenced in the operational semantics of Chapter 3 including the initial configurations of Section 3.7.1.

$$H_{Init} = \quad H_{init_exceptions} \cup H_{init_global_arrays} \cup H_{init_miscellaneous_objects}$$

$$H_{init_exceptions} = \left\{ \begin{array}{l} loc_{\texttt{ArithmeticException}}, loc_{\texttt{ArrayIndexOutOfBoundsException}}, \\ loc_{\texttt{ArrayStoreException}}, loc_{\texttt{ClassCastException}}, \\ loc_{\texttt{IndexOutOfBoundsException}}, loc_{\texttt{NegativeArraySizeException}}, \\ loc_{\texttt{NullPointerException}}, loc_{\texttt{SecurityException}}, \\ loc_{\texttt{CardException}}, loc_{\texttt{UserException}}, \\ loc_{\texttt{CardRuntimeException}}, loc_{\texttt{APDUException}}, \\ loc_{\texttt{ISOException}}, loc_{\texttt{PINException}}, \\ loc_{\texttt{SystemException}}, loc_{\texttt{TransactionException}}, \\ loc_{\texttt{CryptoException}}, \end{array} \right\}$$

$$H_{init_global_arrays} = \left\{ loc_{\texttt{APDUBuffer}}, loc_{\texttt{AppletInstallBuffer}} \right\}$$

$$H_{init_miscellaneous_objects} = \left\{ loc_{\texttt{JCRE}}, loc_{\texttt{APDUObject}} \right\}$$

D.6 Initial State for Static Heap - K_{Init}

In contrast to H_{Init}, which has remained unchanged for many releases, each release of the *Java Card* specification defines more static constants and so in the interest of brevity it is best to generate K_{Init} from the appropriate constants `javadoc` files e.g. in version 3.0.1:

`javacard_specifications-3_0_1-RR/classic/api_classic/constant-values.html`

there are over 300 static constants, a sample of which are reproduced here for example:

```
public static final byte   javacard.framework.APDU.PROTOCOL_MEDIA_CONTACTLESS_TYPE_A = -128
public static final byte   javacard.framework.APDU.PROTOCOL_MEDIA_CONTACTLESS_TYPE_B = -112
public static final byte   javacard.framework.APDU.PROTOCOL_MEDIA_DEFAULT = 0
public static final byte   javacard.framework.APDU.PROTOCOL_MEDIA_MASK = -16
public static final byte   javacard.framework.APDU.PROTOCOL_MEDIA_USB = -96
public static final byte   javacard.framework.APDU.PROTOCOL_T0 = 0
```

```
public static final byte   javacard.framework.APDU.PROTOCOL_T1 = 1
public static final byte   javacard.framework.APDU.PROTOCOL_TYPE_MASK = 15
public static final byte   javacard.framework.APDU.STATE_ERROR_IO = -3
public static final byte   javacard.framework.APDU.STATE_ERROR_NO_T0_GETRESPONSE = -1
public static final byte   javacard.framework.APDU.STATE_ERROR_NO_T0_REISSUE = -4
public static final byte   javacard.framework.APDU.STATE_ERROR_T1_IFD_ABORT = -2
public static final byte   javacard.framework.APDU.STATE_FULL_INCOMING = 2
public static final byte   javacard.framework.APDU.STATE_FULL_OUTGOING = 6
public static final byte   javacard.framework.APDU.STATE_INITIAL = 0
public static final byte   javacard.framework.APDU.STATE_OUTGOING = 3
public static final byte   javacard.framework.APDU.STATE_OUTGOING_LENGTH_KNOWN = 4
public static final byte   javacard.framework.APDU.STATE_PARTIAL_INCOMING = 1
public static final byte   javacard.framework.APDU.STATE_PARTIAL_OUTGOING = 5
public static final short   javacard.framework.APDUException.BAD_LENGTH = 3
public static final short   javacard.framework.APDUException.BUFFER_BOUNDS = 2
public static final short   javacard.framework.APDUException.ILLEGAL_USE = 1
public static final short   javacard.framework.APDUException.IO_ERROR = 4
public static final short   javacard.framework.APDUException.NO_T0_GETRESPONSE = 170
public static final short   javacard.framework.APDUException.NO_T0_REISSUE = 172
public static final short   javacard.framework.APDUException.T1_IFD_ABORT = 171
public static final byte   javacard.framework.ISO7816.CLA_ISO7816 = 0
public static final byte   javacard.framework.ISO7816.INS_EXTERNAL_AUTHENTICATE = -126
public static final byte   javacard.framework.ISO7816.INS_SELECT = -92
public static final byte   javacard.framework.ISO7816.OFFSET_CDATA = 5
public static final byte   javacard.framework.ISO7816.OFFSET_CLA = 0
public static final byte   javacard.framework.ISO7816.OFFSET_EXT_CDATA = 7
public static final byte   javacard.framework.ISO7816.OFFSET_INS = 1
public static final byte   javacard.framework.ISO7816.OFFSET_LC = 4
public static final byte   javacard.framework.ISO7816.OFFSET_P1 = 2
public static final byte   javacard.framework.ISO7816.OFFSET_P2 = 3
public static final short   javacard.framework.ISO7816.SW_APPLET_SELECT_FAILED = 27033
public static final short   javacard.framework.ISO7816.SW_BYTES_REMAINING_00 = 24832
public static final short   javacard.framework.ISO7816.SW_CLA_NOT_SUPPORTED = 28160
```

```
public static final short   javacard.framework.ISO7816.SW_COMMAND_CHAINING_NOT_SUPPORTED = 26756
public static final short   javacard.framework.ISO7816.SW_COMMAND_NOT_ALLOWED = 27014
public static final short   javacard.framework.ISO7816.SW_CONDITIONS_NOT_SATISFIED = 27013
public static final short   javacard.framework.ISO7816.SW_CORRECT_LENGTH_00 = 27648
public static final short   javacard.framework.ISO7816.SW_DATA_INVALID = 27012
public static final short   javacard.framework.ISO7816.SW_FILE_FULL = 27268
public static final short   javacard.framework.ISO7816.SW_FILE_INVALID = 27011
public static final short   javacard.framework.ISO7816.SW_FILE_NOT_FOUND = 27266
public static final short   javacard.framework.ISO7816.SW_FUNC_NOT_SUPPORTED = 27265
public static final short   javacard.framework.ISO7816.SW_INCORRECT_P1P2 = 27270
public static final short   javacard.framework.ISO7816.SW_INS_NOT_SUPPORTED = 27904
public static final short   javacard.framework.ISO7816.SW_LAST_COMMAND_EXPECTED = 26755
public static final short   javacard.framework.ISO7816.SW_LOGICAL_CHANNEL_NOT_SUPPORTED = 26753
public static final short   javacard.framework.ISO7816.SW_NO_ERROR = -28672
public static final short   javacard.framework.ISO7816.SW_RECORD_NOT_FOUND = 27267
public static final short   javacard.framework.ISO7816.SW_SECURE_MESSAGING_NOT_SUPPORTED = 26754
public static final short   javacard.framework.ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED = 27010
public static final short   javacard.framework.ISO7816.SW_UNKNOWN = 28416
public static final short   javacard.framework.ISO7816.SW_WARNING_STATE_UNCHANGED = 25088
public static final short   javacard.framework.ISO7816.SW_WRONG_DATA = 27264
public static final short   javacard.framework.ISO7816.SW_WRONG_LENGTH = 26368
public static final short   javacard.framework.ISO7816.SW_WRONG_P1P2 = 27392
public static final byte   javacard.framework.JCSystem.ARRAY_TYPE_BOOLEAN = 1
public static final byte   javacard.framework.JCSystem.ARRAY_TYPE_BYTE = 2
public static final byte   javacard.framework.JCSystem.ARRAY_TYPE_INT = 4
public static final byte   javacard.framework.JCSystem.ARRAY_TYPE_OBJECT = 5
public static final byte   javacard.framework.JCSystem.ARRAY_TYPE_SHORT = 3
public static final byte   javacard.framework.JCSystem.CLEAR_ON_DESELECT = 2
public static final byte   javacard.framework.JCSystem.CLEAR_ON_RESET = 1
public static final byte   javacard.framework.JCSystem.MEMORY_TYPE_PERSISTENT = 0
public static final byte   javacard.framework.JCSystem.MEMORY_TYPE_TRANSIENT_DESELECT = 2
public static final byte   javacard.framework.JCSystem.MEMORY_TYPE_TRANSIENT_RESET = 1
public static final byte   javacard.framework.JCSystem.NOT_A_TRANSIENT_OBJECT = 0
```

```
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_AES = 2
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_DES = 1
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_DSA_PRIVATE = 4
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_DSA_PUBLIC = 3
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_EC_F2M_PRIVATE = 6
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_EC_F2M_PUBLIC = 5
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_EC_FP_PRIVATE = 8
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_EC_FP_PUBLIC = 7
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_HMAC = 9
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_KOREAN_SEED = 10
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_RSA_CRT_PRIVATE = 13
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_RSA_PRIVATE = 12
public static final byte   javacard.security.KeyBuilder.ALG_TYPE_RSA_PUBLIC = 11
public static final short   javacard.security.KeyBuilder.LENGTH_AES_128 = 128
```

D.7 Abstract Initial State for Object Heap - $\widehat{H_{Init}}$

The abstract equivalent of the object heap of the newly initial JCVM H_{Init} is $\widehat{H_{Init}}$ and it is defined:

$$
\begin{array}{lcl}
\beta^{H,JH}_{DynamicHeap}(H_{Init}) & \sqsubseteq_H & \widehat{H_{Init}} \\
\widehat{\sigma}_{\texttt{JCRE}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{JCRE}}) \\
\widehat{\sigma}_{\texttt{ArithmeticException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{ArithmeticException}}) \\
\widehat{\sigma}_{\texttt{NegativeArraySizeException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{NegativeArraySizeException}}) \\
\widehat{\sigma}_{\texttt{NullPointerException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{NullPointerException}}) \\
\widehat{\sigma}_{\texttt{ArrayIndexOutOfBoundsException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{ArrayIndexOutOfBoundsException}}) \\
\widehat{\sigma}_{\texttt{SecurityException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{SecurityException}}) \\
\widehat{\sigma}_{\texttt{ClassCastException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{ClassCastException}}) \\
\widehat{\sigma}_{\texttt{ArrayStoreException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{ArrayStoreException}}) \\
\widehat{\sigma}_{\texttt{IndexOutOfBoundsException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{IndexOutOfBoundsException}}) \\
\widehat{\sigma}_{\texttt{CardException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{CardException}}) \\
\widehat{\sigma}_{\texttt{UserException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{UserException}}) \\
\widehat{\sigma}_{\texttt{CardRuntimeException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{CardRuntimeException}}) \\
\widehat{\sigma}_{\texttt{APDUException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{APDUException}}) \\
\widehat{\sigma}_{\texttt{ISOException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{ISOException}}) \\
\widehat{\sigma}_{\texttt{PINException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{PINException}}) \\
\widehat{\sigma}_{\texttt{SystemException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{SystemException}}) \\
\widehat{\sigma}_{\texttt{TransactionException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{TransactionException}}) \\
\widehat{\sigma}_{\texttt{ServiceException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{ServiceException}}) \\
\widehat{\sigma}_{\texttt{CryptoException}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{CryptoException}}) \\
\widehat{\sigma}_{\texttt{AppletInstallBuffer}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{AppletInstallBuffer}}) \\
\widehat{\sigma}_{\texttt{APDUBuffer}} & = & \beta^{R,H,JH}_{Ref}(loc_{\texttt{APDUBuffer}})
\end{array}
$$

D.8 Abstract Initial State for Object Heap - $\widehat{K_{Init}}$

The abstract equivalent of the static heap of the newly initial JCVM K_{Init} is $\widehat{K_{Init}}$ and it is defined:

$$\beta^{H}_{StaticHeap}(K_{Init}) \quad \sqsubseteq_{K} \quad \widehat{K_{Init}}$$

Appendix E

Design and implementation of the Fulgurite iterative worklist solver

E.1 Overview

In this Chapter we present a high–level view of the design and implementation of the Fulgurite iterative worklist solver, which we have developed to compute the least solution to the constraints generated by the extended control–flow analysis of Chapter 5 for any Carmel program P meeting the requirements of Section 1.3. As detailed in Chapter 6, from the least solution to the extended control–flow analysis for P, we are able to generate a family of integer linear programming problems, one for each applet lifecycle method in P per registered applet whose solution yields the worst-case resource-usage for that applet lifecycle method[1].

We have chosen to implement Fulgurite as a Java application for a variety of reasons, including:

- each abstract value can be represented as an object, and each finite set of states into which an attribute may be partitioned can be represented as a typesafe enum, facilitating future extensibility and maintenance;
- Java's built-in Collections framework of discrete mathematical structures including sets, sequences and maps matches the core requirements of the worklist solver;
- adding support for new *Java Card* API methods is straight-forward, facilitating upgrades of the solver to handle new editions of *Java Card* Classic;

[1] In Section 6.4, we specify the loop validation conditions for being able to determine for each applet lifecycle method whether we can determine the worst-case resource-usage analysis for that applet lifecycle method from the least solution to the constraints generated by the extended control–flow analysis. Where the validation conditions are not met for a particular registered applet and applet lifecycle method combination, the worst-case resource-usage for that combination is *unknown*.

- greatly simplifies the implementation of abstract numeric operations, since the same numeric data types in *Java Card* are available in Java, with the same semantics;
- modifying the implementation of the program analysis clause for a particular bytecode instruction is straight-forward. In particular, handling of each bytecode instruction is delegated to a particular method and abstract values are implemented as immutable objects to isolate, as far as possible, the effect of each clause;
- existence of mature third party libraries such as SableCC[2] (for parser generation and tree-walking facilities) and WALA libraries[3] (which we use for graphing functions including computation of dominator relations for natural loop identification);
- excellent debugging and unit testing tools available;
- should *Java Card* applications ever get sufficiently large that computation of the extended control–flow analysis would benefit from concurrent processing, it would be straight-forward to modify the solver to employ standard techniques to allow this e.g. an appropriate source of splitting[4] the workload would be by package i.e. P is partitioned into a number of packages and each thread computes the least solution for its partition, then a final single-threaded pass over the packages ensures least-solution is reached (assuring inter-package inter-applet communications are handled).

E.2 Implementation of Abstract Values in Fulgurite

Since the program analyses of Chapters 4 and 5 are both first-order analyses, it seems appropriate to begin with how we implement abstract values in Fulgurite.

E.2.1 Individual Values

A UML diagram of the hierarchy of the implementation of abstract values in Fulgurite is given in Figure E.1. The root node is `AbstractValue`. Figures E.2, E.3 and E.4 provide more granular information of the UML hierarchy, particularly the input- parameters and return types of methods.

At a high-level, the correspondence between abstract values in the program analysis and implementation types in Fulgurite is as follows:

- an abstract value corresponds to an instance of `AbstractValue`;

[2] http://www.sablecc.org/
[3] http://wala.sourceforge.net/wiki/index.php/Main_Page
[4] Using the streams, `Spliterator` and fork-join threading patterns of Java 7 and 8

- an abstract return address corresponds to an instance of `AbstractReturnAddress`;
- an abstract object reference corresponds to an instance of `AbstractObjectReference`;
- an abstract number corresponds to an instance of `AbstractCarmelNumber`, as well as being an instance of `AbstractValue`:
  - an abstract byte number corresponds to an instance of `AbstractByteNumber`;
  - an abstract short number corresponds to an instance of `AbstractShortNumber`;
  - an abstract integer number corresponds to an instance of `AbstractIntegerNumber`;

Recapping and summarising Sections 4.2.1 *ff*:

$$\overline{\mathsf{Val}} \;=\; \overline{\mathsf{Num}} + \overline{\mathsf{Ref}} + \overline{\mathsf{RetAddr}}$$

$$\beta^{R,H,JH}_{Val}(t, Y, (m_n, pc_n)) \;=\; \begin{cases} \beta^{R,H,JH}_{Ref}(t, Y, (m_n, pc_n)) = (t, Z_1, (m_n, pc_n)) & t = \mathtt{r} \\ \beta_{ReturnAddress}(t, Y, (m_n, pc_n)) = (t, Z_2, (m_n, pc_n)) & t = \mathtt{ra} \\ \beta_{Num}(t, Y, (m_n, pc_n)) = (t, Z_3, (m_n, pc_n)) & t \in \{\mathtt{b}, \mathtt{s}, \mathtt{i}\} \end{cases}$$

for some Z_1, Z_2, Z_3, and noting from Section 3.4:

$$\textit{nbWords} \;:\; \mathsf{Value}^* \;\rightarrow\; \mathbb{N}_0$$

$$\textit{nbWords} \;:\; \epsilon \;=\; 0$$

$$\textit{nbWords} \;:\; (t, Y, (m_n, pc_n)) \;=\; \begin{cases} 2, & t = \mathtt{i} \\ 1, & otherwise \end{cases}$$

$$\textit{nbWords} \;:\; (t_1, Y_1, (m_1, pc_1))::\ldots::(t_p, Y_p, (m_p, pc_p)) \;=\; \textstyle\sum_{i=1}^{p} \textit{nbWords}\big(\ (t_i, Y_i, (m_i, pc_i))\ \big)$$

and reproducing for convenience the least-upper bound operator from page 103 as Table E.1:

$$
(t_1, Y_1, (m_1, pc_1)) \sqcup_{Val} (t_2, Y_2, (m_2, pc_2)) \quad = \quad
\begin{cases}
\{(t_1, (min, max, mod), (m_1, pc_1))\} & t_1 = t_2 \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \\
& m_1 = m_2 \\
& pc_1 = pc_2 \\
& Y_1 = (l_1, h_1, mod_1) \\
& Y_2 = (l_2, h_2, mod_2) \\
& min = \texttt{minimum}(l_1, l_2) \\
& max = \texttt{maximum}(h_1, h_2) \\
& mod = \texttt{maximum}(mod_1, mod_2)) \\
& mod < \texttt{MAX_MOD_COUNT} \\
\{(t_1, (\bot_t, \top_t, \texttt{MAX_MOD_COUNT}), (m_1, pc_1))\} & t_1 = t_2 \in \{\texttt{b}, \texttt{s}, \texttt{i}\} \\
& m_1 = m_2 \\
& pc_1 = pc_2 \\
& Y_1 = (l_1, h_1, mod_1) \\
& Y_2 = (l_2, h_2, mod_2) \\
& min = \texttt{minimum}(l_1, l_2) \\
& max = \texttt{maximum}(h_1, h_2) \\
& mod = \texttt{maximum}(mod_1, mod_2)) \\
& mod \geq \texttt{MAX_MOD_COUNT} \\
\{(t_1, Y_1, (m_1, pc_1)), (t_2, Y_2, (m_2, pc_2))\} & \text{otherwise}
\end{cases}
$$

Table E.1: Binary least upper-bounds operator $\bigsqcup_{Val}$ over the complete lattice $\langle \widehat{\mathsf{Val}}, \ \sqsubseteq_{Val} \rangle$

We are reminded of the following:

- both concrete- and abstract- values have an operand type t and an associated runtime address label (m_n, pc_n), both of which are preserved in each of the representation functions;
- the size of the concrete- and abstract- values in terms of stack words is available via the *nbWords* function and is determined solely by the value's operand type t, which as noted above is preserved by all the representation functions;
- the least-upper-bound (LUB) of two abstract values is well-defined and is typically the union of the two values, except for numeric types with the same operand type and address, in which case the LUB produces a single abstract value where the lower end of the resulting numeric interval contained within the abstract value is set to the minimum of the lower intervals of the LUB operands, the upper end of the resulting numeric interval is set to the maximum of the upper intervals of the LUB operands, and the modification count of the resulting abstract val is set to the maximum of the modification counts of the LUB operands;
- the form of Z_1, Z_2, Z_3 is dependent solely on the value's operand type t, which as noted above is preserved by all the representation functions, which determines the representation function applied.

These properties are enshrined in the root node `AbstractValue` of the implementation tree of abstract values in Fulgurite, as shown in Figure E.1. `AbstractValue` defines a set of methods to answer the question of what the operand type of an abstract value is:

```
int haveSameAbstractValueTypes(AbstractValue av1, AbstractValue av2);
boolean isByteAbstractValue();
boolean isShortAbstractValue();
boolean isIntegerAbstractValue();
boolean isObjectAbstractValue();
boolean isReturnAddressAbstractValue();
boolean isNumericAbstractValue();
```

and defines another pair of methods which ensure, respectively, the LUB operation is defined for all pairs of abstract values, and that each abstract value knows the number of stack words it would (or does) occupy:

```
SortedSet<AbstractValue> LUB(AbstractValue otherValue);
byte getNumberOfWords();
```

In accordance with the third point above, in `AbstractValueBaseImpl` we provide a default implementation of the LUB operator to be the union of the two operands where the two operands are from different operand types[5] and delegate to a new method in the same class:

```
protected SortedSet<AbstractValue> LUBSameClass(AbstractValue otherValue)
```

the responsibility for appropriate handling of the LUB for two abstract values of the same type. In `AbstractValueBaseImpl`, we define `LUBSameClass(AbstractValue otherValue)` to be the union of the two operands and this behaviour is inherited (and is the correct implementation) for abstract return addresses (modelled by `AbstractReturnAddress`) and abstract object references (modelled by `AbstractObjectReference`). In contrast, the implementation of the `LUBSameClass` operator in the abstract numeric types `AbstractByteNumber`, `AbstractShortNumber` and `AbstractIntegerNumber` produces a single-value where both operands have the same address and are of the same class, and otherwise produces the union of the operands, as expected.

E.2.2 Implementation of sets of abstract values in the complete lattice $\langle \widehat{\mathsf{Val}},\ \sqsubseteq_{Val} \rangle$

Utilising the pair-wise LUB operator defined for abstract values, we implement sets of abstract values $\in \langle \widehat{\mathsf{Val}},\ \sqsubseteq_{Val} \rangle$ in Fulgurite as instances of the Java class `CarmelLUBTreeSet` whose method `isLessThan(CarmelLUBTreeSet)` corresponds to the strict partial order:

$$X.\texttt{isLessThan(Y)} \iff X \sqsubseteq_{Val} Y \wedge X \neq Y$$

and whose methods `add` and `addAll` correspond to $\bigsqcup_{Val}$. In particular, once abstract value(s) are requested to be added to a set of abstract values, via the `add` and `addAll` methods, the LUB is recalculated. Appropriate thread–synchronization[6] ensures reads from, and writes to, this critical collection class are sequenced/coherent.

We are interested in the strict partial order because in our worklist algorithm it is always the case that $Y = F_j(X)$ for some monotone function F_j, and our real interest is whether or not some abstract entity has increased or not.

Some maps use `CarmelLUBTreeSet` directly e.g. the abstract static heap:

```
protected final Map<CarmelField, CarmelLUBTreeSet> K = new TreeMap<CarmelField, CarmelLUBTreeSet>();
```

Other classes use `CarmelLUBTreeSet` indirectly e.g.

[5] implementation class $\equiv$ operand type
[6] Every `public` method of `CarmelLUBTreeSet` is `synchronized`.

- the implementation of the abstract operand stack `CarmelOperandStack` is as a sequence of `CarmelLUBTreeSet`s.

  Further the analysis entity $\widehat{S}$ is implemented as a map:

```
Map<CarmelAddress, Map<AbstractContextSequenceInterface, CarmelOperandStack>> S =
    new TreeMap<CarmelAddress, Map<AbstractContextSequenceInterface, CarmelOperandStack>>();
```

- the implementation of the abstract local variable array `CarmelLocalVarArray` is as a map

  from key to `CarmelLUBTreeSet`

  Further the analysis entity $\widehat{L}$ is implemented as a map:

```
Map<CarmelAddress, Map<AbstractContextSequenceInterface, CarmelLocalVarArray>> L =
    new TreeMap<CarmelAddress, Map<AbstractContextSequenceInterface, CarmelLocalVarArray>>();
```

E.3 Fulgurite worklist algorithm

The Fulgurite worklist algorithm is principally coded in the `Fulgurite` class. The `Main` class is the entry-point class and passes to the `Fulgurite` class all the information required from parsing the input files, and access to the utility classes to be able to make all the decisions required of the solver e.g.

- to enumerate all the types of a class instance to see e.g. whether a `checkcast` instruction should succeed;
- to determine the bytecode instructions that are "leaders" [Zha] in a Carmel method, so that methods may be decomposed into basic blocks;
- to pre-calculate the reaching definitions for local variable analysis and retain the reaching definitions for all `if` statements;
- to be able to resolve dynamic method lookup;
- to represent in the analysis results the initial configurations via the `loadInitialAppletContexts()` method.

prior to the worklist iteration.

Tables E.2 to E.4 show the heart of the worklist algorithm, encoded in the `Fulgurite` method `performLFPCalculation()`. Note how closely it mirrors the algorithm presented in Section 4.7. As can be seen from the cited tables, the worklist algorithm delegates the handling of each bytecode instruction, and the high-level objects it needs, to its own `abstract` method. Table E.4 details an excerpt of some of these `abstract` methods. This delegation is a deliberate design pattern to aid clarity, and to keep the size of `Fulgurite` manageable. Other classes flesh out the `abstract`

methods e.g. `CommonCodeFulgurite` contains the handling of most bytecode instructions, and `ArraysFulgurite` handles the array bytecode instructions and `APIContextFulgurite` handles the *Java Card* API methods.

As described above, Fulgurite delegates the handling of each program analysis clause to its own method. As an example of the implementation of one of the program analysis clause, Table E.5 details the implementation of binary numeric operations i.e. bytecode instructions of the form `numop` t $binop$. Tables E.6 and E.7 show the implementation of binary numerical operators for `int`eger addition, and `short` subtraction, respectively. The other binary numerical operators' implementation follow a similar pattern i.e. to iterate over all combinations from least to greatest over each of the binary operators operands, perform the primitive arithmetic over the appropriate data type for each operand combination, and record the least and greatest of the primitive arithmetic results, across all the combinations.

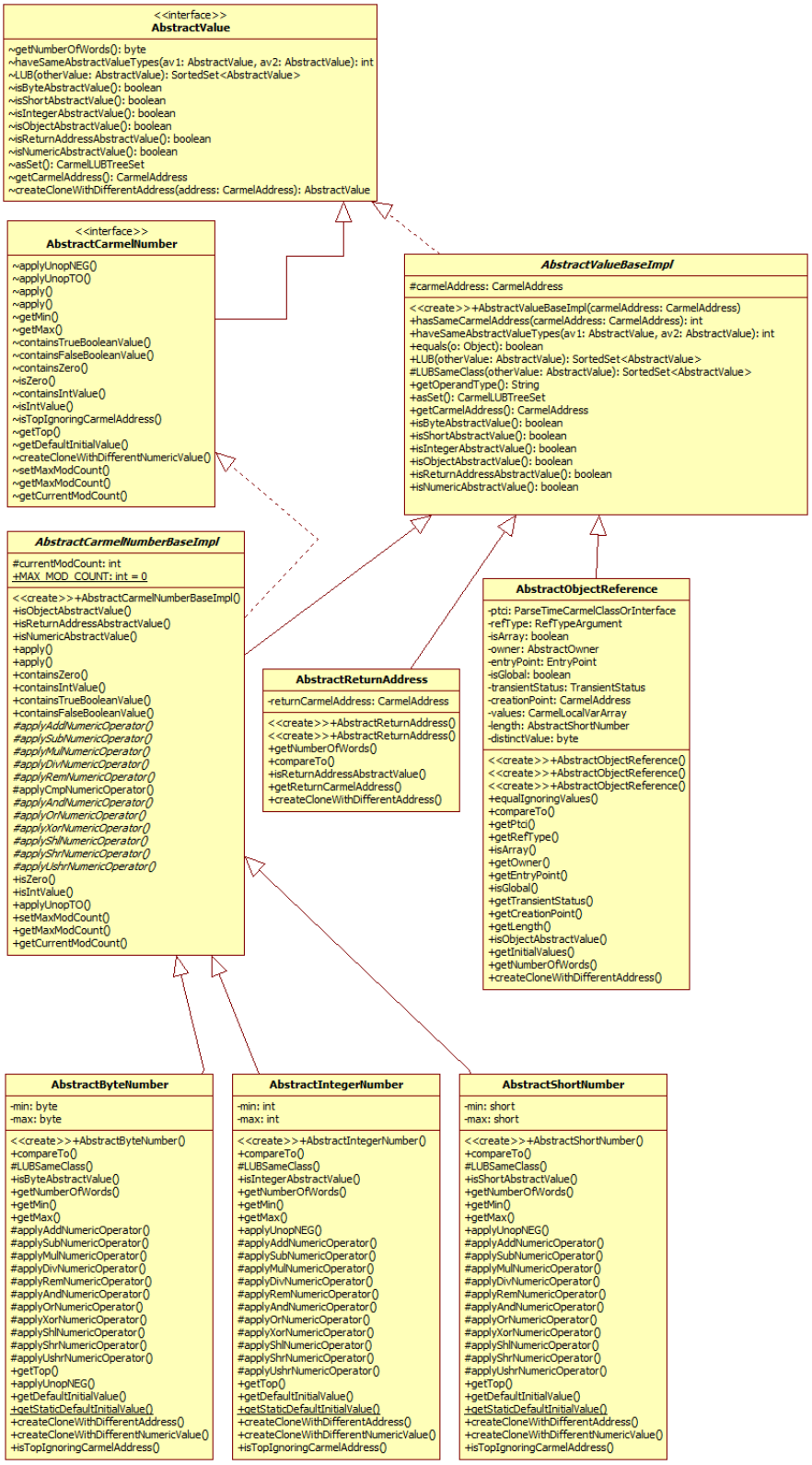


Figure E.1: UML diagram showing the implementation hierarchy of abstract values in Fulgurite. Method parameters have been omitted due to space consideration. Fuller details given in Figures E.2, E.3 and E.4

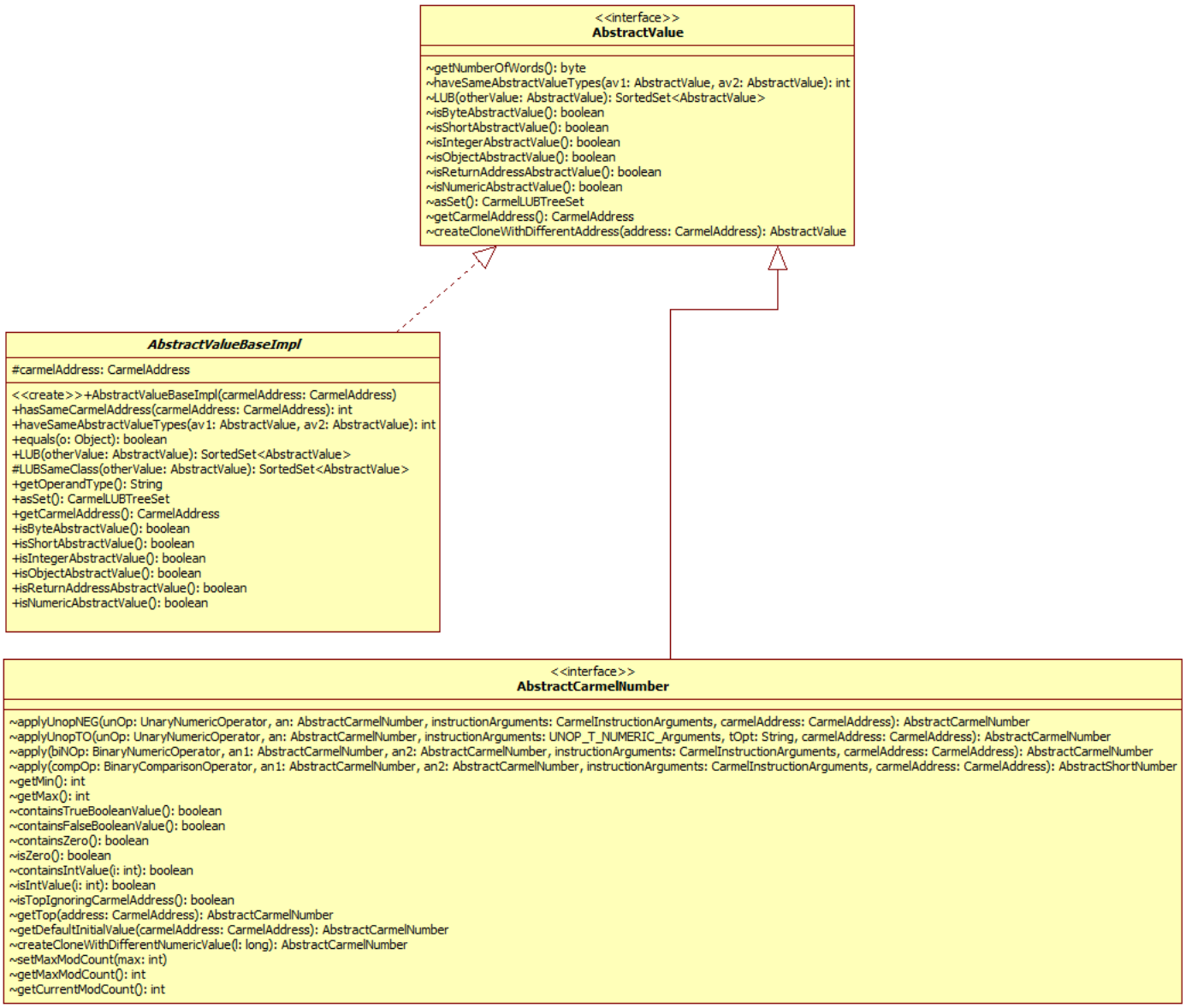


Figure E.2: UML diagram showing the high-level interfaces and base abstract class in Fulgurite

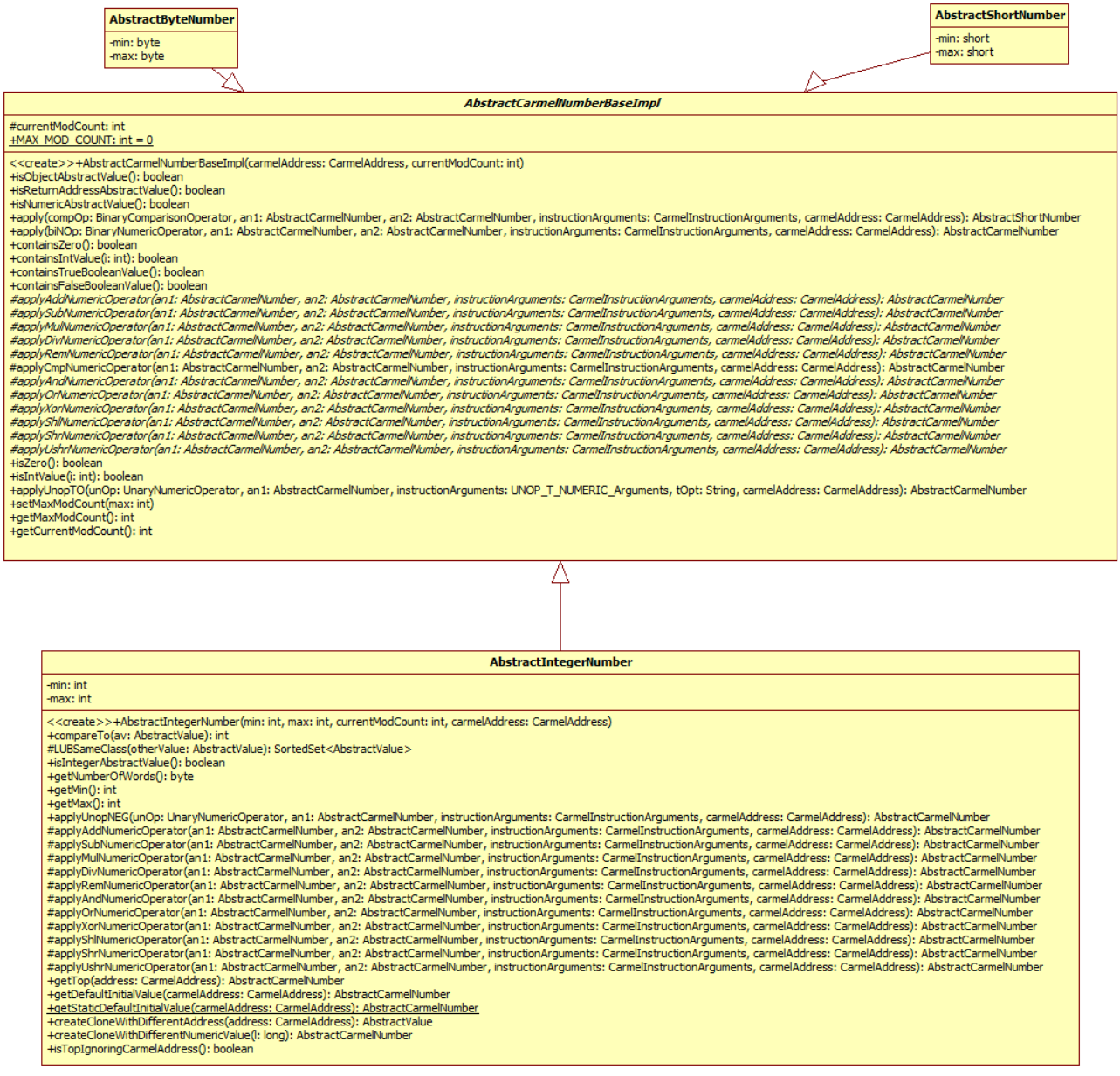


Figure E.3: UML diagram showing the abstract number values in Fulgurite

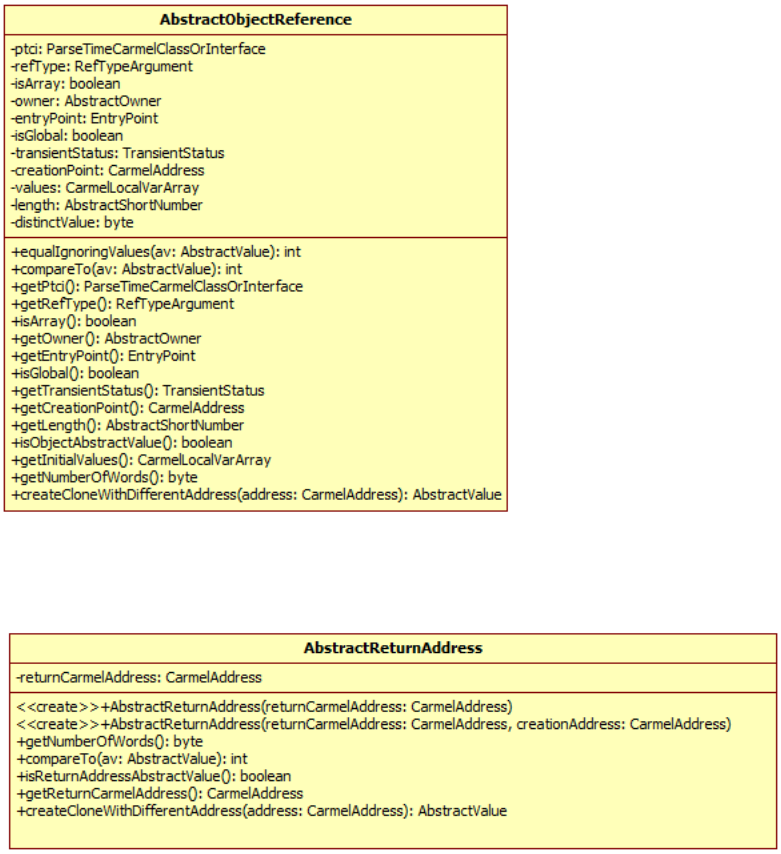


Figure E.4: UML diagram showing the abstract object reference and abstract return address values in Fulgurite

Table E.2: Excerpt from the core of the Fulgurite worklist algorithm, Part 1

```
while (shouldIterateOverWorklist) {

    loadAllCarmelAddressesOntoWorklist();

    outerLoopIterationCount++;

    shouldIterateOverWorklist = false;

    while (!(W.isEmpty())) {

     innerLoopIterationCount++;

     carmelAddress = W.pop();
     carmelInstruction = carmelAddress.getInstruction();
     contexts = C.get(carmelAddress);

     if (contexts != null) {

         if (DEBUG_MODE) {

          System.out.printf("*******************************%n");

          System.out.printf(
                "[outer loop=%d, inner loop=%d, Workload size=%d, shouldIterateOverWorklist=%s, addr =(%s,%s)]%n",
                outerLoopIterationCount, innerLoopIterationCount, W.size(), shouldIterateOverWorklist,
                (carmelAddress.getMethod() == null) ? "" : carmelAddress.getMethod().getFqMethodName(),
                (carmelAddress.getPc() == null) ? "" : carmelAddress.getPc().getPcLabel());

         }

         if (carmelInstruction == null) {
          nextAddress = null;

         }

         if (carmelInstruction != null && carmelInstruction.isIntraMethodInvocationInstruction()) {

          nextAddress = CarmelAddress.nextAddress(carmelAddress,
                this.methodToCarmelAddressMap.get(carmelAddress.getMethod()));


         } else {
          nextAddress = null;

         }

         CarmelInstructionArguments instructionArguments = carmelAddress.getInstructionArguments();

         for (AbstractContextSequenceInterface context : contexts) {

          if (context.getCurrentContextDepth() <= 2) {
              continue;
          }

          nextAddressContextSequence = (nextAddress != null) ? context.getContext(context, nextAddress)
                : null;

          if (nextAddressContextSequence == null
                && carmelInstruction.isIntraMethodInvocationInstruction()) {

              throw new RuntimeException("Could not find a next address for " + carmelAddress);

          }
```

Table E.3: Excerpt from the core of the Fulgurite worklist algorithm, Part 2

```
switch (carmelInstruction) {

case NOP:
    nop(context, carmelAddress, (NoArguments) instructionArguments);
    break;

case PUSH_T_C:
    push_t_c(context, carmelAddress, (PushTCArguments) instructionArguments);
    break;

case POP_P:
    pop_p(context, carmelAddress, (PopPArguments) instructionArguments);
    break;

case DUP_P_D:
    dup_p_d(context, carmelAddress, (DupPDArguments) instructionArguments);
    break;

case SWAP_P1_P2:
    swap_p1_p2(context, carmelAddress, (SwapP1P2Arguments) instructionArguments);
             break;

case NEW_T_CLASS:

    className = ((NewClassInstanceArgument) instructionArguments).getClassName();
    refType = new RefTypeArgument(className, false);

    if (!(refType.isRefType())) {

     throw new RuntimeException("new <class instance> not reference type "
          + instructionArguments + "  at  " + carmelAddress);
    }

    ptci = getCarmelClass(className, refType.isArray(), carmelAddress);

    if (ptci == null) {

     throw new RuntimeException(
          "parsed class null " + instructionArguments + "  at  " + carmelAddress);

    }

    new_t_class(context, carmelAddress, ptci, refType);
    break;

case CHECKCAST_T:
    checkcast_t(context, carmelAddress, (RefTypeArgument) instructionArguments);
    break;

case INSTANCEOF_T:
    instanceof_t(context, carmelAddress, (RefTypeArgument) instructionArguments);
    break;

case GETSTATIC_F:

    if (this.addressFieldCache.get(carmelAddress) == null) {

     CarmelField findCarmelField = null;
     String requiredFieldName = ((FieldArguments) instructionArguments).getFieldName();

     findCarmelField = getCarmelField(requiredFieldName, true);

     if (findCarmelField == null) {

         throw new RuntimeException(
              "Unable to find static field with name " + requiredFieldName);
     } else {

         this.addressFieldCache.put(carmelAddress, findCarmelField);
     }

    }

    getstatic_f(context, carmelAddress, this.addressFieldCache.get(carmelAddress));

    break;
```

Table E.4: Excerpt from the core of the Fulgurite worklist algorithm, Part 3

```
abstract protected void nop(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     NoArguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void push_t_c(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     PushTCArguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void pop_p(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     PopPArguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void dup_p_d(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     DupPDArguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void swap_p1_p2(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     SwapP1P2Arguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void unop_no_opt_t_numeric(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     UNOP_NO_OPT_T_NUMERIC_Arguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void unop_t_numeric(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     UNOP_T_NUMERIC_Arguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void binop_no_opt_t_numeric(AbstractContextSequenceInterface context,
     CarmelAddress carmelAddress, BINOP_NO_OPT_T_NUMERIC_Arguments instructionArguments)
          throws IllFormedCarmelOperandStackException;

abstract protected void binop_t_numeric(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     BINOP_T_NUMERIC_Arguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void load_t_i(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     LoadTIArguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void store_t_i(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     StoreTIArguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void inc_t_i_c(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     IncTICArguments instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void goto_addr(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     CarmelAddress gotoAddr) throws IllFormedCarmelOperandStackException;

abstract protected void if_t_op_goto_addr(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     IfTOpGotoAddrArguments instructionArguments, CarmelAddress gotoAddr)
          throws IllFormedCarmelOperandStackException;

abstract protected void if_t_op_null_goto_addr(AbstractContextSequenceInterface context,
     CarmelAddress carmelAddress, IfTOpNullGotoAddrArguments instructionArguments, CarmelAddress gotoAddr)
          throws IllFormedCarmelOperandStackException;

abstract protected void lookupswitch(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     Map<Integer, CarmelAddress> instructionArguments, String t, CarmelAddress defaultPC)
          throws IllFormedCarmelOperandStackException;

abstract protected void tableswitch(AbstractContextSequenceInterface context, CarmelAddress carmelAddress, String t,
     CarmelAddress[] orderedOffsetsAddresses, CarmelAddress defaultOffsetAddress, int low)
          throws IllFormedCarmelOperandStackException;

abstract protected void new_t_array(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     ParseTimeCarmelClassOrInterface ptci, RefTypeArgument refType) throws IllFormedCarmelOperandStackException;

abstract protected void new_t_class(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     ParseTimeCarmelClassOrInterface ptci, RefTypeArgument refType) throws IllFormedCarmelOperandStackException;

abstract protected void checkcast_t(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     RefTypeArgument instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void instanceof_t(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     RefTypeArgument instructionArguments) throws IllFormedCarmelOperandStackException;

abstract protected void getstatic_f(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     CarmelField getStaticField) throws IllFormedCarmelOperandStackException;

abstract protected void putstatic_f(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     CarmelField findCarmelField) throws IllFormedCarmelOperandStackException;

abstract protected void getfield_f(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     CarmelField findCarmelField) throws IllFormedCarmelOperandStackException;

abstract protected void getfield_this_f(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
     CarmelField findCarmelField) throws IllFormedCarmelOperandStackException;
```

```
@Override
protected void binop_no_opt_t_numeric(AbstractContextSequenceInterface context, CarmelAddress carmelAddress,
  BINOP_NO_OPT_T_NUMERIC_Arguments instructionArguments) throws IllFormedCarmelOperandStackException {

 this.initialiseVariables(carmelAddress, context);

 String t = instructionArguments.getT();

 Set<AbstractCarmelNumber> s2Set = null;
 Set<AbstractCarmelNumber> s1Set = null;

 AbstractCarmelNumber s3 = null;

 if (topElementOfStack != null && opStack.depth() >= 2) {

  s2Set = topElementOfStack.getAbstractCarmelShortOrIntegerNumber(t);

  if (s2Set != null) {

   // Throw away values off operand stack
   opStack = opStack.popTopElement();

   s1Set = opStack.peek().getAbstractCarmelShortOrIntegerNumber(t);

   BinaryNumericOperator binop = instructionArguments.getBinop();
   final boolean binopEqualsDivOrRem = (binop.equals(BinaryNumericOperator.DIV)
     || binop.equals(BinaryNumericOperator.REM));

   if (s1Set != null) {

    opStack = opStack.popTopElement();

    for (AbstractCarmelNumber s1 : s1Set) {

     for (AbstractCarmelNumber s2 : s2Set) {

      if (s2.containsZero() && binopEqualsDivOrRem) {

       updateE(carmelAddress, context, localVarArray,
         JCREOwnedExceptions.get(JCREOwnedTemporaryExceptionNames.ArithmeticException),
         jHMap);

      }

      if (!(s2.isZero() && binopEqualsDivOrRem)) {

       s3 = s2.apply(binop, s1, s2, instructionArguments, carmelAddress);

       addCarmelOpStackAndLocalVarArray(opStack.push(s3), localVarArray,
         nextAddressContextSequence, nextAddress, jHMap, carmelAddress, context);
      }
     }
    }
   }
  }
 }
}
```

Figure E.5: Implementation of `numop` t $binop$

```
@Override
protected AbstractCarmelNumber applySubNumericOperator(
      AbstractCarmelNumber an1, AbstractCarmelNumber an2,
      CarmelInstructionArguments instructionArguments,
      CarmelAddress carmelAddress) {

 final long an1Low = an1.getMin();
 final long an1High = an1.getMax();

 final long an2Low = an2.getMin();
 final int an2High = an2.getMax();

 final int currentModCount = 1 + Math.max(an1.getCurrentModCount(),
       an2.getCurrentModCount());

 AbstractShortNumber result = null;

 TreeSet<Short> ofShorts = new TreeSet<Short>();
 Short f;
 Short l;
 for (long i = an1Low; i <= an1High; i++) {
      for (long j = an2Low; j <= an2High; j++) {

       ofShorts.add((short) (((short) i) - ((short) j)));

       if (ofShorts.size() >= 3) {

            f = ofShorts.first();
            l = ofShorts.last();

            ofShorts.clear();

            ofShorts.add(f);
            ofShorts.add(l);
       }
      }
 }

 result = new AbstractShortNumber(ofShorts.first(), ofShorts.last(),
       currentModCount, carmelAddress);

 return result;
}
```

Figure E.6: Implementation of subtraction method for abstract short numbers

```
@Override
protected AbstractCarmelNumber applyAddNumericOperator(
      AbstractCarmelNumber an1, AbstractCarmelNumber an2,
      CarmelInstructionArguments instructionArguments,
      CarmelAddress carmelAddress) {

 final long an1Low = an1.getMin();
 final long an1High = an1.getMax();

 final long an2Low = an2.getMin();
 final long an2High = an2.getMax();

 final int currentModCount = 1 + Math.max(an1.getCurrentModCount(),
       an2.getCurrentModCount());

 AbstractIntegerNumber result = null;

 TreeSet<Integer> ofIntegers = new TreeSet<Integer>();
 Integer f = null;
 Integer l = null;
 int k = 0;
 for (long i = an1Low; i <= an1High; i++) {
      for (long j = an2Low; j <= an2High; j++) {

       k = (int) ((int) i + (int) j);

       ofIntegers.add(k);

       if (ofIntegers.size() >= 3) {

            f = ofIntegers.first();
            l = ofIntegers.last();

            ofIntegers.clear();

            ofIntegers.add(f);
            ofIntegers.add(l);
       }

      }

 }

 result = new AbstractIntegerNumber(ofIntegers.first(),
       ofIntegers.last(), currentModCount, carmelAddress);

 return result;
}
```

Figure E.7: Implementation of add method for abstract integer numbers