Imperial College London
Department of Computing

Heuristics for the Refinement of Assumptions in Generalized Reactivity Formulae

Davide Giacomo Cavezza

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London and
the Diploma of Imperial College London, September 2020

Abstract

Reactive synthesis is concerned with automatically generating implementations from formal specifications. These specifications are typically written in the language of generalized reactivity (GR(1)), a subset of linear temporal logic capable of expressing the most common industrial specification patterns, and describe the requirements about the behavior of a system under assumptions about the environment where the system is to be deployed. Oftentimes no implementation exists which guarantees the required behavior under all possible environments, typically due to missing assumptions (this is usually referred to as unrealizability). To address this issue, new assumptions need to be added to complete the specification, a problem known as assumptions refinement. Since the space of candidate assumptions is intractably large, searching for the best solutions is inherently hard. In particular, new methods are needed to (i) increase the effectiveness of the search procedures, measured as the ratio between the number of solutions found and of refinements explored; and (ii) improve the results' quality, defined as the weakness of the solutions. In this thesis we propose a set of heuristics to meet these goals, and a methodology to assess and compare assumptions refinement methods based on quantitative metrics. The heuristics are in the form of algorithms to generate candidate refinements during the search, and quantitative measures to assess the quality of the candidates.

We first discuss a heuristic method to generate assumptions that target the cause of unrealizability. This is done by selecting candidate refinement formulas based on Craig's interpolation. We provide a formal underpinning of the technique and evaluate it in terms of our new metric of effectiveness, as defined above, whose value is improved with respect to the state of the art. We demonstrate this on a set of popular benchmarks of embedded software. We then provide a formal, quantitative characterization of the permissiveness of environment assumptions in the form of a weakness measure. We prove that the partial order induced by this measure is consistent with the one induced by implication. The key advantage of this measure is that it allows for prioritizing candidate solutions, as we show experimentally. Lastly, we propose a notion of minimal refinements with respect to the observed counterstrategies. We demonstrate that exploring minimal refinements produces weaker solutions, and reduces the amount of computations needed to explore each refinement. However, this may come at the cost of reducing the effectiveness of the search. To counteract this effect, we propose a hybrid search approach in which both minimal and non-minimal refinements are explored.

Statement of Originality

I declare that this thesis was composed by myself and that the work it presents is my own, except where otherwise stated. I certify that all material in this thesis which is not my own work has been properly acknowledged.

Davide Cavezza

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Acknowledgements

Of course, this work would not have been possible without the advice and guidance of my supervisors, Dalal Alrajeh and András György. In particular, I thank Dalal for her continuous support to my work, with our frequent meetings and discussions. During a PhD, it is often hard to find a sense in what you are studying, and having someone who often found some context and possible applications to my ideas helped me in figuring out the meaning of all of it. Without her help, this thesis would have stopped at chapter 2. And I thank András for his careful revision of the most theoretical aspects of my research.

In general, I would like to thank all Imperial staff and faculty members I had the pleasure to meet these four years. They were all passionate and motivating researchers. And in particular, I thank the SPIKE group, including all of my officemates, for their series of presentations. They allowed me to get in touch with areas of research different from my own and keep my mind open to other fields.

I thank EPSRC for funding my PhD, and my external examiners, Alessio Lomuscio and Nir Piterman, for taking their time to read this thesis.

Special thanks go to my colleague and friend Andrea. Our discussions, especially in the first part of our PhD, allowed me to improve my study method and acquire a deeper understanding of the concepts studied.

More personal thanks go to every person who let me reach this milestone, starting from my family and teachers, from primary school to university. For more details, I refer the reader to [MasterThesis2014].

Contents

Abstract **3**

Acknowledgements **9**

1 Introduction **23**

1.1 GR(1) Assumptions Refinement 24

1.1.1 Unrealizability of GR(1) Specifications 25

1.1.2 Counterstrategy-Guided Assumptions Refinement 28

1.2 Thesis Statement 29

1.3 Main Contributions 30

1.4 Thesis Organization 33

2 Background **35**

2.1 Linear-Time Temporal Logic 35

2.1.1 Syntax 36

2.1.2 Semantics: ω-Words 37

2.1.3 Semantics: ω-Languages and ω-Automata 39

2.2 Generalized Reactivity of Rank 1 41

2.2.1 GR(1) formulae 42

2.2.2 GR(1) Games 45

2.2.3 Unrealizability and Counterstrategies 48

2.2.4 Unrealizable Cores 52

2.3 Relationship between Strict and Non-Strict GR(1) Realizability 53

2.4 Entropy and Hausdorff Dimension 55

2.5 Summary 68

3 The Assumptions Refinement Problem **69**

3.1 Problem Definition 70

3.1.1 Refinement in Strict GR(1) Formulae 70

3.1.2 Refinement in Non-Strict GR(1) Formulae 71

3.1.3 Weakest Assumptions Refinement 72

3.2 Counterstrategy-Guided Search for Assumptions Refinement 74

3.3 Assumptions Refinement Challenges 78

3.4 Heuristics Classification 80

3.4.1 Syntax-Based Bias Heuristic 81

3.4.2 Variable-Choice Bias Heuristic 84

3.4.3 FIFO Selection Heuristic 87

3.5 Case studies 88

3.6 Summary 89

4 Interpolation-Based Assumptions Generation 91

4.1 Preliminaries: Craig's Interpolation 92

4.2 Motivation 95

4.3 Abstract Counterstrategies 97

4.4 The Interpolation Bias 104

4.4.1 Extracting Counterruns 107

4.4.2 Boolean Encoding of Counterplays and Unrealizable Cores 108

4.4.3 Interpolation and Full Separability 111

4.4.4 Interpolant Translation to GR(1) Assumptions 113

4.5 Unrolling Looping Counterruns 118

4.6 Detailed Example: The Lift Controller 121

4.7 Convergence 124

4.8 Evaluation 127

4.8.1 Experiment 1 128

4.8.2 Experiment 2 134

4.9 Related Work 140

4.10 Summary 141

5 Entropy and Hausdorff Dimension of ω-languages 144

5.1 DBA of the closure of an ω-language. 145

5.2 Hausdorff Dimension of Büchi and Generalized Büchi Automata 146

5.3 Simplification of Entropy and Hausdorff Dimension 149

5.3.1 Conjunctions of Independent Formulae 149

5.3.2 Reduced Adjacency Matrix for Large Alphabets 151

5.4 Summary 155

6 Measuring Weakness of GR(1) Assumption Refinements 157

6.1 Weakness of LTL Formulae 158

6.2 Weakness Measure of GR(1) Conjunctions 160

6.2.1 Entropy and Hausdorff Dimension as Weakness Measures 160

6.2.2 Ordering Invariants and Strongly Connected Invariants 164

6.2.3 Ordering Fairness Conditions via Fairness Complements 170

6.2.4 Ordering GR(1) Conjunctions with a Common Strongly Connected Invariant 172

6.2.5 Ordering GR(1) Conjunctions with Strongly Connected Invariants 175

6.2.6 Ordering Generic GR(1) Conjunctions 177

6.3 Computing the Hausdorff Dimension of Fairness Complements 186

6.4 Evaluation 190

6.4.1 Ordering GR(1) Conjunctions via Weakness and Implication 191

6.4.2 Sorting Heuristic for GR(1) Assumptions Refinement 193

6.4.3 Hausdorff Dimension for Quantitative Model Checking 199

6.5 Related Work 200

6.6 Summary 202

7 Minimality of GR(1) Assumption Refinements 204

7.1 Motivating Example . . . 205

7.2 Minimal Assumptions Sets . . . 207

7.3 Minimal Refinements Search . . . 207

7.3.1 The Function **MinimalRefinement** . . . 208

7.3.2 The Search Algorithm . . . 213

7.3.3 Time Complexity . . . 216

7.3.4 Computation Time of Model Checking in **MinimalRefinement** . . . 218

7.4 Hybrid Refinement Generation . . . 218

7.5 Experiments . . . 219

7.5.1 Methodology . . . 219

7.5.2 Results and Discussion . . . 221

7.5.3 Tests with Small Timeout . . . 223

7.6 Related Work . . . 225

7.7 Summary . . . 226

8 Related Work 228

8.1 Automata-Based Assumptions Refinement . . . 228

8.2 Declarative Assumptions Refinement . . . 229

8.3 Reactive Synthesis . . . 230

8.4 Specification Mining . . . 232

8.5 Counterstrategy Alternative: Justice Violations Transition Systems . . . 233

8.6 Weakening Guarantees 234

8.7 LTL Specification Revision 234

8.8 Oracle-Guided Inductive Synthesis 236

9 Conclusions and Future Work **237**

9.1 Summary of Thesis Achievements 237

9.2 Future Work 239

Bibliography **241**

List of Tables

4.1 AMBA case studies for experiment 1. 130

4.2 Original JVTS case studies for experiment 1 130

4.3 Additional JVTS case studies for experiment 1 130

4.4 Statistics on full separability of the interpolants per group of case studies 133

4.5 Case studies for experiment 2 134

4.6 Statistics of the refinement trees 139

4.7 Percentage of nodes targeting parent's unrealizable cores 139

6.1 Summary of findings 164

6.2 Results of experiment 2 of heuristic search 196

6.3 Comparison between the discriminative power of implication and weakness . . . 197

7.1 Experiments' results. The rows with * do not consider HYBRID. 221

7.2 30-minutes experiment results 224

7.3 30-minutes experiment case studies 224

List of Figures

1.1 Car following example [Li14] 26

1.2 Counterstrategy-guided refinement loop 29

2.1 Counterstrategy for Example 2.2 52

3.1 Counterexample-guided inductive synthesis 74

3.2 Counterstrategy-guided assumptions refinement. The components in the shaded area implement the heuristics that are the focus of this thesis. 75

3.3 Stump containing the input refinement Ψ and the generated refinements $\Psi_i = \Psi \cup \psi_i$ in a single iteration of the counterstrategy-guided loop 77

3.4 A refinement tree 77

4.1 Resolution tree proof of the unsatisfiability of $\beta_1 \wedge \beta_2$ 95

4.2 Counterstrategy for specification in Example 2.2. The full description is given in Figure 2.1. 96

4.3 Counterstrategy of the specification in Example 2.2 after removing the failing output valuations 101

4.4 Abstract counterstrategy of the LGS example. It consists of an initial state only. When the environment sets both input variables to true in two consecutive steps, the two guarantees cannot be satisfied simultaneously. 101

4.5 Abstract counterstrategy for the specification in Example 2.2. The non-influential output variables are removed from the edges. 103

4.6 Main steps of interpolation bias 105

4.7 Looping counterrun (top) and finite counterrun (bottom) 108

4.8 Unrolling of the looping counterrun in Figure 4.7. The unrolling degree is 1. . . 120

4.9 Lift counterstrategy 123

4.10 Counterrun extracted from the counterstrategy in Fig. 4.9 with one unrolling. The new state generated by the unrolling is marked in gray. 123

4.11 Case studies solved by each of the two approaches 131

4.12 Number of solutions, of non-redundant solutions, and effectiveness of interpolation and multivarbias in each group of case studies 132

4.13 Total number of refinement nodes explored over time 137

4.14 Number of realizable refinement nodes discovered over time (excluding vacuously realizable ones) 138

4.15 Effectiveness over time 143

5.1 Muller and reduced Muller automaton of $L(\phi)$ 153

5.2 Adjacency matrix (left) and reduced adjacency matrix (right) of $L(\phi)$ 153

6.1 DMA of $L(\phi_1)$. In this and the following DMAs, the collection of accepting sets to which a state belongs is enclosed in curly braces inside the state. 161

6.2 DBAs of ρ_a (left) and ρ_b (right) 162

6.3 Büchi automaton of $\mathbf{G}(x \rightarrow \mathbf{X}y)$. The state labels $\lambda(q)$ are shown inside the nodes. The initial state (not shown) has transitions towards all the states in the figure, since the first symbol is unconstrained. 166

6.4 DBA of a one-state invariant. 168

6.5 DBA of $a\mathcal{U}b$ 169

6.6 DBA of $L(J)$ (left) and DMA of $L(J^c)$ (right). 170

6.7 DBAs of ϕ_a (left) and ϕ_b (right) of Example 6.3. 170

6.8 Subset relations between $L(\phi_a)$, $L(\phi_b)$, $L(\phi_a^c)$ and $L(\phi_b^c)$ 173

6.9 DMAs of ϕ_1^c (left) and ϕ_2^c (right) of Example 6.4. 174

6.10 DBAs of $L(\phi_1)$ (left) and $L(\phi_2)$ (right). The grayed-out states and edges are parts of the automata of $L(\rho_1)$ and $L(\rho_2)$, but not of $L(\phi_1)$ and $L(\phi_2)$. 178

6.11 DMAs of $L(\rho_1) \setminus L(J_1)$ (left) and $L(\rho_2) \setminus L(J_2)$ (right). 178

6.12 DBAs of the ω-languages of accepted suffixes L_{S_1} (left) and L_{S_2} (right). 180

6.13 DBAs of ϕ_1 (left) and ϕ_2 (right) 184

6.14 DBA $\mathcal{B}$ of the language of $\phi = \mathbf{G}((x \rightarrow \mathbf{X}y) \wedge ((x \vee y) \rightarrow \mathbf{X}(x \vee y)) \wedge \mathbf{GF}x$ (left), and DBA $\mathcal{B}[S]$ of the language C_S (right) 188

6.15 DMA of the language $C_S \cap L(\mathbf{FG}\neg x)$ referred to the language in Figure 6.14 (left), and subgraph including the accepting states only (right). In the left-hand automaton, the edge labels are omitted to avoid cluttering: each edge's label corresponds to the valuation labelling the destination state of the edge. The singleton of the state $(\{y\}, 1)$ in gray is the only one accepting any ω-word. . . . 189

6.16 Refinement tree. The gray nodes are the leaves: ψ_{21} and ψ_{23} are solutions to the refinement problem. 193

6.17 Results for BFS and heuristic search. The results of the two approaches are ranked together by decreasing weakness. The x-axis shows the positions of the solutions in the ranking, from the weakest (leftmost) to the strongest (rightmost), with FALSE being the strongest. The y-axis is the number of solutions in each position of the ranking. The filled circles on the x-axis represent the median rankings of BFS and heuristic search solutions. 195

7.1 Abstract counterstrategy for the request-grant example (see Sections 2.2.3 and 4.3) 206

7.2 Bipartite graph of assumptions and counterstrategies 208

Chapter 1

Introduction

Nowadays computing is ubiquitous. Self-driving cars, domotics, phones, and other appliances embed microcontrollers to perform specific computational tasks. According to [Res20], over 30 billion microcontrollers were shipped in 2019, and the market is expected to further expand in the next 7 years. Therefore, producing error-free microcontrollers is of utmost importance. In principle, two approaches can be used to pursue this goal: *verification* and *synthesis*.

Verification consists in checking whether an implementation satisfies its specification. This can be achieved by testing it under carefully crafted inputs, but this cannot prove the absence of errors, as some corner cases may be missed [YP05]. Hence formal methods, based on mathematically sound proofs of correctness, have become widespread in microcontroller engineering. Techniques like model checking [CGP99], abstract interpretation [Bou13], and symbolic execution [CS13], are widely applied in industry and are the subject of intensive research.

Synthesis involves producing a correct-by-construction implementation directly from the specification [Fin16]. The language of choice for describing specifications is that of *generalized reactivity* (GR(1) for short), a subset of the well-known *linear temporal logic* (LTL) widely used in formal methods due to the efficiency of synthesis methods for this subset. Algorithms for reactive synthesis have been applied, among other fields, in robotics [BDP+13], self-driving cars [LSSS14], or on-chip communication protocols [BJP+12]. The interest of research in synthesis is testified by the success of the SYNTCOMP competition [SYN], where synthesizers

from different labs compete on a set of specifications.

The outcome of synthesis depends on the quality of the specifications. Specifically, a specification for synthesis cannot have a correct implementation satisfying it in every situation when the environment assumptions are not sufficiently specified [LDS11]. In this case we say that the specification is *unrealizable*. This may be due to missing or implicit knowledge of the environment, which needs to be elicited in order for synthesis to be successful. The difficulty of capturing a precise model of the environment is tackled by *assumptions refinement*, a search over the space of all possible assumptions in order to make a specification realizable by some controller.

One of the major challenges in assumptions refinement is the intractability of the search space. Given a finite amount of time, only a small subset of candidate assumptions can be searched. In this thesis, we propose a set of heuristics to drive the search for "good" assumptions in the context of GR(1) assumptions refinement.

In the rest of this chapter, we motivate assumptions refinement by giving examples where it is applied, and by discussing the challenges it presents through those examples (Section 1.1). Then we state the research topic explored in this thesis (Section 1.2) and list our main contributions (Section 1.3). Finally we outline the structure of the thesis (Section 1.4).

1.1 GR(1) Assumptions Refinement

The problem of identifying environment assumptions in the specification of systems arises frequently in software engineering. For instance, they are instrumental for checking the completeness of user requirements [ASB19], the derivation of software specifications from requirements [JZ93], and the verification of such specifications [GPB05]. In this work, we focus on their specific role in enabling the synthesis of implementations from given GR(1) specifications.

1.1.1 Unrealizability of GR(1) Specifications

The problem of synthesizing a controller from a specification, known as *reactive synthesis*, was first formalized by Alonso Church [Chu57]. In this work, the language for expressing specifications was monadic second-order logic (a superset of LTL). However, even for LTL specifications, the computational complexity of reactive synthesis is doubly exponential in the length of the formula . GR(1) specifications have seen a surge of interest in recent years , since it has been proven that the time for reactive synthesis in this class of specifications drops to polynomial [BJP+12].

The model underlying the problem of reactive synthesis is that of *reactive systems*. These systems consist of two interacting agents, an *environment* and a *controller*. The environment provides inputs to the controller and the controller reacts with some output back to the environment. The controller represents the part of the model to be synthesized automatically. Both inputs and outputs are modelled as sets of Boolean variables, typically denoted as $\mathcal{X}$ and $\mathcal{Y}$, respectively. In the context of self-driving cars, for instance, a controller is the circuit on the car performing decisions about speed and steering, while the environment consists of the road, other cars and pedestrians around it [LSSS14]. In communication protocols for on-chip devices (like the AMBA-AHB protocol widely used as a synthesis benchmark [BJP+12, LDS11, KMR17, CGS08]), the environment is a set of devices that can issue requests to communicate over a shared bus, and the controller is an arbiter that implements a scheduling policy of their requests.

A GR(1) specification consists of a set of *assumptions* and a set of *guarantees*. They express the requirement that the controller should satisfy the guarantees whenever the environment satisfies the assumptions. The controller can be synthesized automatically only if there exists no environment satisfying the assumptions that may force a violation of some guarantee. If this environment exists, the specification is said to be *unrealizable* [PR89, ALW89] and the assumptions have to be modified in order to rule out the pathological environment behaviours. The following examples taken from the literature show that unrealizability is a common problem in GR(1) specifications.

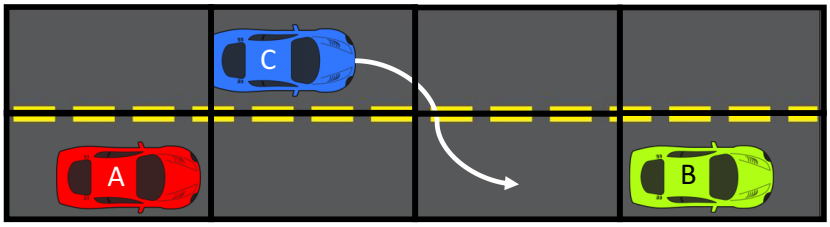


Figure 1.1: Car following example [Li14]

Example 1.1 ([Li14])**.** *Consider an autonomous car A moving in a road with two other cars B and C. The road is discretized into a grid of possible positions, and the car is equipped with sensors that look two squares ahead. The system requires car A is to follow car B using only information from its sensors, which we assume is specified as a GR(1) specification. However, this requirement makes no assumption about what obstacles may exists between cars A and B which may prevent A from fulfilling the requirement: indeed, if car C is between A and B, then A's sensors cannot detect B. Therefore, the GR(1) specification is said to be unrealizable. The example is illustrated in Figure 1.1. The problem is that there is a missing assumption which states that "there is no other car between cars A and B".*

In problems like the above example, it is of utmost importance to identify the assumptions about the world that make the specification realizable. In case those assumptions are violated, the guarantees needs to be changed or additional assumptions need to be found, thus restricting the set of environments where achieving the guarantees is required. For instance, the specification in the above example needs to be changed from just guaranteeing "Car A must follow car B" into "Assuming car C is not between cars A and B, then car A must follow car B", by adding the assumption "Car C is not between cars A and B".

Example 1.2 ([NMM+19])**.** *The landing gear system (LGS) consists of a gear box that stores the landing gear beneath a plane and a handle in the cockpit controlling the gear's position (extended or retracted). For illustrative purposes, we introduce the following model of this system, consisting of two input variables* $\mathcal{X} = \{\mathsf{handle_up}, \mathsf{handle_down}\}$ *and one output variable* $\mathcal{Y} = \{\mathsf{gear_extended}\}$*. The input variables respectively indicate whether the handle is up or down, and the output variable denotes the position of the gear (*$\mathsf{gear_extended} = \mathsf{true}$ *if the gear*

is in the extended position, false *if it is in the retracted position).*

The specification consists of two guarantees requiring that if the handle is pulled down and remains down for two consecutive time steps, then the gear must be extended; if instead the handle is pulled up for two consecutive states, the gear must be retracted:

$$\rho_1^{\mathcal{S}} := \mathbf{G}(\mathsf{handle_down} \rightarrow \mathbf{X}(\neg\mathsf{handle_down} \vee \mathsf{gear_extended}))$$

$$\rho_2^{\mathcal{S}} := \mathbf{G}(\mathsf{handle_up} \rightarrow \mathbf{X}(\neg\mathsf{handle_up} \vee \neg\mathsf{gear_extended}))\ .$$

The meaning of **G** *and* **X** *will be formalized in Section 2.1. For now, we just say that the operator* **G** *is read "always", and the Boolean expression in the operand is required to hold in every execution state of the system;* **X** *is read "next" and expresses the requirement that the Boolean expression in the operand hold in the execution state that immediately follows the current one.*

This specification is unrealizable: indeed, there is no explicit assumption that prevents handle_down *and* handle_up *from being both* true *at the same time, in which case $\rho_1^{\mathcal{S}}$ and $\rho_2^{\mathcal{S}}$ cannot be satisfied at the same time by any controller.*

The problem lies in the absence of an assumption of the kind $\mathbf{G}(\mathsf{handle_down} \leftrightarrow \neg\mathsf{handle_up})$. *This assumption may seem implicit to the person who provides the specification; however, it needs to be elicited in order for the specification to be realizable. Although the solution seems obvious for this very simple problem, assumptions like this can be easily missed in more complex specifications with of variables and hundreds of assumptions/guarantees.*

These two examples highlight different causes of missing assumptions. In the first example, there is an unforeseen event that can make the specification unrealizable, and an additional assumption is needed to exclude that event from occurring. The root cause can be an incompleteness in the natural language specification of the environment, a common problem to software engineering contexts [ASB19]. In the second example, the missing assumption comes from a mistake in modelling the environment: using two Boolean variables for the position of

the handle allows for more behaviours than the ones that could really occur. The additional assumption here is needed to reflect the actual behaviour of the system.

Assumptions refinement is the problem of discovering (automatically) the assumptions needed for achieving realizability [LDS11, AMT13, CA17, MRS19]. It poses the following question: *Given a set of guarantees a controller needs to satisfy, what are the environments in which some controller would be able to satisfy them?* There could be more than one solution. The goal of assumptions refinement is to perform a search in the space of assumptions in order to identify the right ones. Typically the aim is identifying the *weakest* possible refinements [Ses15], as they encompass the widest possible variety of environments that make the specification realizable. [1]

1.1.2 Counterstrategy-Guided Assumptions Refinement

The search space for GR(1) assumptions refinement is intractably large (doubly exponential in the number of Boolean variables, see also our discussion in Section 3.3, and [LDS11, CA17]) and finding all solutions is practically impossible. Heuristics have therefore been proposed such that at least some solutions can be found in a reasonable amount of time. Taking inspiration from counterexample-guided inductive synthesis (CEGIS, an approach to search problems successfully applied to program synthesis [SL13] and then applied to similar problems where the solution is in the form of some logical expression [GNMR16, Ses15, GSN$^+$16, AKRU13, JS14]), the main heuristics are based on the use of *counterstrategies* to generate assumptions. Counterstrategies[2] are models of environment behaviours that force the violation of some guarantee. The term derives from modelling the interaction between the environment and the controller as a game where the goal of the environment is to force the falsification of some guarantee [KHB09].

To disallow the environment behaviours described by counterstrategies, the assumptions refinement process constructs assumptions to be added to the specification that are violated by

[1]

[2]In recent work [KMR17, MRS19] counterstrategies are replaced by more abstract models called *Justice Violation Transition Systems* (*JVTSes*), that are faster to compute. In the context of assumptions refinement, they have the same role as counterstrategies. JVTSes are described in Section 8.5.

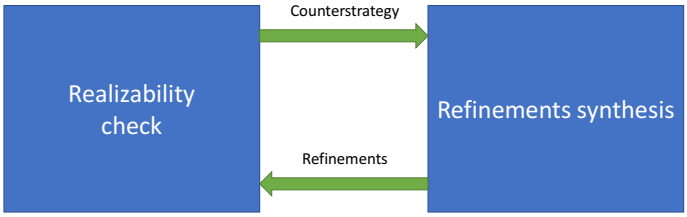


Figure 1.2: Counterstrategy-guided refinement loop. The left block checks if a refinement makes some specification realizable. If not, a counterstrategy is computed and it is used by the right block to produce a new refinement.

those behaviours. This is done in an iterative process, where for each computed counterstrategy, some assumption disallowing it is generated, until realizability is reached. The resulting loop is pictured in Figure 1.2. Recent work has defined alternative structures to counterstrategies [KMR17, MRS19] that are faster to compute than counterstrategies but represent conceptually the same thing.

To cope with the intractability of the search space, the procedure makes use of further heuristics to determine how to generate new candidate refinements from counterstrategies. These heuristics are in the form of manually tuned parameters and formulae templates the user needs to specify based on their knowledge of the model (see Section 3.4). This poses a major limitation to the approach: if the heuristics are badly tuned, the search may miss the weakest solutions or not find solutions at all.

1.2 Thesis Statement

In this thesis, we investigate the existence of fully automated, domain-independent heuristics for finding solutions to the GR(1) assumptions refinement problem. More specifically, the heuristics should be *fully automated* in the sense that their computation does not need the user to have any knowledge of the model at hand, and can be applied in a black-box manner. They should also be *domain-independent* in the sense that they do not refer to a particular domain modelled in the specification (for instance, autonomous cars), but only exploit intrinsic features of GR(1)

specifications themselves, their syntax and their semantics. Hence, the heuristics should be applicable to any instance of the GR(1) refinement procedure. The goal is to make the search effective in finding solutions given a certain budget for the search, to be able to assess the quality of the final solutions, and to drive the search towards the best solutions according to the prescribed quality metrics.

Challenges and heuristics classification. In Chapter 3, we define the assumptions refinement problem and the details of the counterstrategy-guided approach. In doing so, we identify the main challenges tackled by the approach, in the form of quality features for the search algorithm and the solutions found. The main features to assess the quality of the search are its *effectiveness* in finding solutions after as few iterations of the approach as possible, and its *full automatedness*, as discussed earlier. The quality features of the solutions are: their *weakness*, that is, the size of the set of environments fulfilling the assumptions in the solutions; their *interpretability* by a human reader; and their *usefulness*, that is, their reflecting the behaviours of the actual environment where the controller is expected to operate.

To match these quality criteria, in this thesis we propose a set of heuristics to employ as components of the counterstrategy-guided approach. In this chapter, we provide a broad classification of these heuristics into the two macro-classes of *bias heuristics* and *sorting heuristics* (whose names trace a connection with the terminology of AI and machine learning). Bias heuristics pertain how assumptions are derived from the computed counterstrategies. Sorting heuristics determine the order in which candidate assumption refinements are explored during the search.

The implementation of the heuristics presented in this thesis and the experimental data referred across the thesis is available at `https://gitlab.doc.ic.ac.uk/dgc14/ThesisRepo`.

1.3 Main Contributions

In the thesis, we provide the following contributions:

Interpolation bias. In Chapter 4 we describe a bias heuristic based on Craig's interpolation [Cra57] to derive assumption refinements from counterstrategies. The heuristic was initially suggested by us in [Cav16].[3] In that work, we described a four-step algorithm using Craig's interpolation to generate assumption refinements. This chapter contains three extensions to that: *(i)* it frames the algorithm presented in [Cav16] in the more general framework of heuristics for GR(1) assumptions refinement; *(ii)* it elicits the required hypotheses on the structure of the input under which the heuristic works by introducing the novel concept of *abstract counterstrategies* (which induces a modification to the first two steps of the algorithm, as described in Section 4.4); and *(iii)* it presents a novel evaluation methodology for conducting experiments in assumptions refinement. This methodology is based on a modification of an existing baseline heuristic [AMT13] (we call this modified baseline *multivarbias*), to adapt it to contexts where fully-automated heuristics are required, and a novel metric to compute the effectiveness of the search, defined in Section 4.8.

We describe the notion of abstract counterstrategies, structures that abstract away from the counterstrategies the information that is not related to the guarantee violation and are used as input to the interpolation bias. Then we give the details of all the steps involved in the algorithm (as described in [Cav16], but revised in light of the definition of the abstract counterstrategies): *(i)* extracting a path from the abstract counterstrategy; *(ii)* provide a description of the path and the violated guarantee in the form of Boolean formulae; *(iii)* apply Craig's interpolation to the two Boolean formulae; *(iv)* extract GR(1) assumptions from the interpolant.

Interpolation is shown to be able to find solutions to the refinement problem in a fully-automated fashion, in a higher proportion of case studies (various versions of the AMBA-AHB specification and specifications from the SYNTECH project) than state-of-the-art bias heuristics. Moreover, it is shown experimentally to improve the effectiveness of the search compared with the state-of-the-art bias heuristics.

The content of this chapter is an extension of our publication in:

[3]The work in [Cav16] was performed in the context of a single project involving a Master of Research degree and a PhD, under the HiPEDS Centre for Doctoral Training, and has to be intended as a subset of the work we present in this thesis.

Davide G. Cavezza and Dalal Alrajeh. Interpolation-Based GR(1) Assumptions Refinement. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 281–297. Springer Berlin Heidelberg, 2017.

Weakness measure. In Chapters 5 and 6, we develop a quantitative measure of weakness that describes how permissive a set of assumptions is. The goal is to provide a sorting heuristic to drive the search towards weaker refinements. Chapter 5 provides the definitions of two quantities, the *entropy* and *Hausdorff dimension* of an ω-language, upon which the weakness measure is constructed. We extend the algorithms from [MS94, Sta98] to compute these quantities on Muller automata, in order to allow the computation also on Büchi and generalized Büchi automata. We also present a simplification of the formula to compute these two quantities, which allows to avoid dealing with long integers during the computation.

Chapter 6 introduces the weakness measure. The focus is on defining a quantitative measure $d(\phi)$, where ϕ is a conjunction of GR(1) assumptions, provably consistent with the typical ordering given by implication. This means that if ϕ_1 implies ϕ_2, then $d(\phi_1) \leq d(\phi_2)$. We prove that our weakness measure (based on a triple of entropy and Hausdorff dimension values computed over appropriate ω-languages constructed from the components of the language of ϕ) induces a lexicographic order that is consistent with implication. This consistency between the ordering induced by our weakness measure and by implication is proven for increasingly complex classes of GR(1) refinements.

The experiments show that effectively the solutions found by using this sorting heuristic are in large part weaker than the solutions found without the heuristic. Also, the comparisons between candidate refinements based on the quantitative measure are more efficient than using implication checks between every pairs of refinements explored during the search, and allows to impose a strict order also in cases where none of two formulae imply the other.

The work presented here is based on the following publications:

Davide G. Cavezza, Dalal Alrajeh, and András György. A Weakness Measure for GR(1) Formulae. In *International Symposium on Formal Methods (FM)*, pages 110–128. 2018.

Davide G. Cavezza, Dalal Alrajeh, and András György. A Weakness Measure for GR(1) Formulae. 2020. To appear in *Formal Aspects of Computing*.

Minimality of refinements. Typically, assumptions refinements are produced incrementally in the loop of Figure 1.2. Every counterstrategy induces the addition of one new assumption to a refinement, until realizability is achieved. However, in these iterations, some of the later assumptions may remove more than one counterstrategy, thus making some of the former assumptions redundant. In Chapter 7, we define the notion of minimality of a refinement with respect to a set of counterstrategies. From this definition, we provide a minimization algorithm that can be integrated to any bias heuristic to ensure that the refinements explored by the counterstrategy-guided approach are all minimal.

The experiments show that the search focuses on shorter candidate refinements, which are faster to compute and allow more refinements to be searched given a time budget. However, fewer solutions are found in the given time. This is partly because using minimization, some of the solutions returned are subsets of assumptions common to several, redundant solutions that are returned without using minimization. We define a metric of *coverage* between solutions to describe this phenomenon.

The work presented here is based on the following publication:

Davide G. Cavezza, Dalal Alrajeh, and András György. Minimal Assumptions Refinement for Realizable Specifications. In *International Conference on Formal Methods in Software Engineering (FormaliSE)*, page 66–76. ACM, 2020.

1.4 Thesis Organization

The thesis is structured in the following chapters.

- Chapter 2 reviews the basic definitions of LTL, GR(1) formulae and of their semantics. It presents the standard notions of GR(1) realizability and counterstrategies, that are

central to the rest of the thesis.

- Chapter 3 introduces the problem of GR(1) assumptions refinement and presents the counterstrategy-guided methodology for finding refinements. In this chapter we identify the challenges posed by the counterstrategy-guided methodology and how these are addressed by the heuristics employed in the methodology. We propose a classification of the heuristics into two groups, *bias* and *sorting* heuristics. We discuss this terminology in Section 3.3.

- Chapter 4 discusses the use of Craig's interpolation for generating refinements from counterstrategies.

- Chapter 5 describes two quantities, *entropy* and *Hausdorff dimension*, that are used to define a quantitative weakness measure over GR(1) formulae. In this chapter we define those quantities and propose variations of the existing algorithms for computing them.

- Chapter 6 defines a weakness measure for GR(1) formulae. This measure is then used as a sorting heuristic in the counterstrategy-guided loop of Chapter 3.

- Chapter 7 introduces a bias heuristic based on the concept of the *minimality* of refinements with respect to a set of observed counterstrategies.

- Chapter 8 surveys existing work from other authors that inspired and contextualized the work presented in this thesis.

- Chapter 9 concludes the thesis by outlining possible directions for future work.

Chapter 2

Background

The problem of synthesizing a controller from a declarative logical specification dates back to Church [Chu62]. Then, Pnueli and Rosner [PR89] showed that the problem is 2EXPTIME-complete for linear temporal logic. More recently, Piterman et al. [PPS05, BJP+12] described an efficient algorithm for the synthesis problem from generalized reactivity (GR(1)) formulae, a subset of LTL for which synthesis can be solved in time polynomial with the size of the formula. The algorithm reduces the synthesis problem to solving a game between the environment and a controller. The synthesized controller is modeled as its winning strategy in the game.

This chapter provides the basic notation about temporal logic and GR(1) used throughout the thesis, and formalizes the notions of winning strategies and counterstrategies that will be used extensively in later chapters. In Section 2.1, we define the syntax and semantics of LTL. We present the concepts of ω-language and ω-automaton. In Section 2.2, we focus on GR(1) and introduce the reactive synthesis problem. This provides the context where the problem of GR(1) assumptions refinement stems from.

2.1 Linear-Time Temporal Logic

Temporal logic is an extension of classical Boolean logic to reason about system behaviour over time. Formalized by Prior under the name of Tense logic [Pri62], this language has become

today one of the de facto standards for formal specifications of software systems. For instance, it is used to specify requirements in frameworks like KAOS [LKMU08, VL00, LV02], and to describe properties in model checking, the formal verification technique by Clarke [CGP99]. It also forms the basis of popular specification languages such as TLA+ by Lamport [Lam02] and PROMELA, the language of the SPIN model checker [FMH16].

There are several variants of temporal logic [Kon10]. We will only focus on *linear-time temporal logic* (*LTL*) [Pnu77, MP92], a language used to reason about sequences of time steps. Other logics like branching-time temporal logics or bounded-time temporal logics are outside the scope of this thesis. We will first introduce the syntax of LTL (Section 2.1.1), and then define its semantics: ω-words (Section 2.1.2), which model infinite sequences of system states, and ω-automata (Section 2.1.3), which capture infinite sets of ω-words satisfying a given LTL formula.

2.1.1 Syntax

Let $\mathcal{T}$ be a finite set of *elementary expressions*, and let $v \in \mathcal{T}$. A *Boolean formula* (or *Boolean expression*) over $\mathcal{T}$ is defined by the following BNF expression:

$$\beta \ ::= \mathsf{true} \mid \mathsf{false} \mid v \mid \neg\beta \mid \beta \wedge \beta$$

The terminal symbols $\neg$ and $\wedge$ are called *Boolean operators*. The operators $\neg$ and $\wedge$ have the usual meaning as in propositional logic. We will also make use of the (redundant) operators $\vee, \rightarrow$, and $\leftrightarrow$ as in propositional logic (and we call those Boolean operators as well). In some cases (especially in Chapter 4), the set of elementary expressions needs to be frequently referred. So, we use the notation $\beta(\mathcal{T})$ to denote that the Boolean expression β uses the non-logical symbols in $\mathcal{T}$. This is a widely accepted convention, used for instance in [BJP+12, PPS05, MRS19].

In the following, we use the Greek letters β and θ to denote Boolean formulae throughout the thesis.

An *LTL formula* (or *LTL expression*) is either a Boolean formula or one of the following

expressions in BNF:

$$\phi \ ::= \ \beta \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{X}\phi \mid \phi\mathcal{U}\phi \mid \mathbf{G}\phi \mid \mathbf{F}\phi \mid \mathbf{H}\phi \ .$$

The $\mathbf{X}$ operator is read as "next", $\mathcal{U}$ as "until", $\mathbf{G}$ as "globally" or "always", $\mathbf{F}$ as "finally" or "eventually", $\mathbf{H}$ for "always in the past up until the current state". These operators are called *temporal operators*.

In the following, the Greek letters ϕ, ψ, ρ, and the Latin uppercase letter J, denote LTL formulae.

An elementary expression containing no Boolean or temporal operators is called a *propositional* or *Boolean variable*. The set of Boolean variables in a formula is denoted by $\mathcal{V}$, and propositional variables by the Latin lowercase letters v, x and y. As we see in Section 2.2.1, the set $\mathcal{T}$ of elementary expressions can contain Boolean variables or LTL sub-expressions, as long as there is a finite number of them.

2.1.2 Semantics: ω-Words

The basic model to describe a software execution is an infinite sequence of states of execution of a system, where each state is the assignment of a Boolean value $\mathsf{true}/\mathsf{false}$ values to a set of propositional variables. This is called an ω-word, and is the building block of all LTL semantics. In this subsection, we define ω-words and the semantics of LTL formulae.

Definition 2.1 (Symbols, alphabet, ω-words)**.** *Formally, let Σ be a finite set, called* alphabet*, of elements called* symbols*. An* ω-word $w = \sigma_1\sigma_2\dots$ *over Σ is an infinite sequence of symbols $\sigma_i \in \Sigma$. It is sometimes denoted as an infinite tuple $w = (\sigma_1, \sigma_2, \dots)$.*

In the following, we use the Greek letter σ to denote symbols and w for ω-words. We denote the i-th symbol of w (starting from 1) by $w[i]$ and the *i-based suffix*, that is the ω-word $w[i]w[i+1]\dots$, by $w[i..]$. We denote the set of all ω-words that can be produced with the alphabet Σ by Σ^ω.

When using ω-words as semantics for LTL formulae, the set Σ is the set of valuations of the propositional variables in $\mathcal{V}$.

Definition 2.2 ($\mathcal{V}$-valuation)**.** *A* valuation *of the variables in $\mathcal{V}$ (or $\mathcal{V}$-valuation) is an assignment of the values* true *or* false *to each variable of $\mathcal{V}$. Equivalently, a $\mathcal{V}$-valuation is any subset σ of $\mathcal{V}$, $\sigma \in 2^{\mathcal{V}}$. A variable $v \in \mathcal{V}$ is* true *in σ if $v \in \sigma$, and* false *if $v \notin \sigma$.*

Definition 2.3 (Semantics of Boolean formulae)**.** *We say that a valuation $\sigma \in 2^{\mathcal{V}}$ satisfies a Boolean formula β ($\sigma \models \beta$) if and only if any of these conditions hold.*

$$\begin{aligned}
&\sigma \models \mathsf{true} && \textit{always}\\
&\sigma \models \mathsf{false} && \textit{never}\\
&\sigma \models v && \textit{iff } v \in \sigma\\
&\sigma \models \neg\beta && \textit{iff } \sigma \not\models \beta\\
&\sigma \models \beta_1 \wedge \beta_2 && \textit{iff } \sigma \models \beta_1 \textit{ and } \sigma \models \beta_2
\end{aligned}$$

Definition 2.4 (Semantics of LTL formulae)**.** *We say that an ω-word w* satisfies *an LTL formula ϕ* in position *i (in symbols $\langle w, i\rangle \models \phi$) if and only if one of the following conditions holds; in these conditions, $w \in \Sigma^\omega$, $v \in \mathcal{V}$, and ϕ and ψ are LTL formulae.*

$$\begin{aligned}
&\langle w, i\rangle \models \beta && \textit{where } \beta \textit{ is Boolean, if } w_i \models \beta\\
&\langle w, i\rangle \models \neg\phi && \textit{iff } \langle w, i\rangle \not\models \phi\\
&\langle w, i\rangle \models \phi \wedge \psi && \textit{iff } \langle w, i\rangle \models \phi \textit{ and } \langle w, i\rangle \models \psi\\
&\langle w, i\rangle \models \mathbf{X}\phi && \textit{iff } \langle w, i+1\rangle \models \phi\\
&\langle w, i\rangle \models \mathbf{F}\phi && \textit{iff } \exists j \geq i \textit{ s. t. } \langle w, j\rangle \models \phi\\
&\langle w, i\rangle \models \mathbf{G}\phi && \textit{iff } \forall j \geq i\ \langle w, j\rangle \models \phi\\
&\langle w, i\rangle \models \phi\mathcal{U}\psi && \textit{iff } \exists j \geq i \textit{ s. t. } \langle w, j\rangle \models \psi \textit{ and } \forall i \leq k < j\ \langle w, k\rangle \models \phi\\
&\langle w, i\rangle \models \mathbf{H}\phi && \textit{iff } \forall k, 1 \leq k \leq i\ \langle w, k\rangle \models \phi
\end{aligned}$$

We say that w satisfies ϕ ($w \models \phi$) iff $\langle w, 1\rangle \models \phi$.

Informally, LTL satisfaction is defined with respect to some position i in the ω-word. The operand of $\mathbf{X}$ operator is evaluated in the $i+1$-th position. $\mathbf{F}\phi$ is true in position i if and only if ϕ is true at position i or at some position later than i. The formula $\mathbf{G}\phi$ is true if ϕ holds in every position later than i and i itself. $\phi\mathcal{U}\psi$ is true if ϕ holds up until some position j after which ψ holds . $\mathbf{H}\phi$ is true in position i if ϕ holds at every position up to (and including) i.

The operators $\mathbf{X}, \mathbf{G}, \mathbf{F}$, and $\mathcal{U}$ are usually called *future operators*, since their satisfaction depends on the future states of execution of the system. $\mathbf{H}$ is known as *past operator* for a symmetric reason [CGP99]. . The ones we have defined are the only ones we are going to use throughout this thesis.

The notions of validity of LTL formulae and implication between LTL formulae occur frequently in our thesis. We give here the formal definitions.

Definition 2.5 (Valid LTL formula)**.** *An LTL formula ϕ is* valid *if for all $w \in \Sigma^\omega$, $w \models \phi$.*

Definition 2.6 (Implication between LTL formulae)**.** *An LTL formula ϕ_1* implies *an LTL formula ϕ_2 if $\phi_1 \rightarrow \phi_2$ is valid.*

2.1.3 Semantics: ω-Languages and ω-Automata

An LTL formula implicitly defines the set of ω-words that satisfy it.

Definition 2.7 (ω-language, language of an LTL formula)**.** *A set of ω-words is called an ω-*language*. The* language of an LTL formula *ϕ, denoted by $L(\phi)$, is the ω-language of the words that satisfy it.*

Despite being an infinite set in general, the language of an LTL formula can be finitely described (and computed) via an automaton. In fact, Vardi shows that the language of an LTL formula is *ω-regular*, that is it can be described by a finite-state ω-automaton [Var96].

An *ω-automaton*, or automaton over infinite words, is a state machine whose arcs are labeled with symbols from the alphabet Σ, such that infinite paths on the automaton correspond to

infinite words from Σ^ω. An acceptance condition determines whether or not the infinite path corresponds to an ω-word that is part of the language. The ω-language described by the automaton is the set of all words accepted by it.

Definition 2.8 (Deterministic ω-automaton, run, acceptance condition)**.** *A* deterministic ω-automaton *is a 5-tuple* $\mathcal{A} = (Q, \Sigma, q_0, \delta, A)$ *where*

- Q *is a set of* states*;*
- Σ *is the alphabet of the language;*
- $q_0 \in Q$ *is the* initial state*;*
- $\delta : Q \times \Sigma \to Q$ *is a (partial*[1]*)* transition function*, which maps a state and a symbol onto the next state;*
- $A : Q^\omega \to \{\mathsf{true}, \mathsf{false}\}$ *is an* acceptance condition*, that maps infinite sequences of states onto* true *or* false*.*

A run *of* $\mathcal{A}$ *is an infinite sequence of states* $q_0 q_1 \dots$ *such that* q_0 *is the initial state and* $\forall i \in \mathbb{N} \;.\; \exists \sigma_{i+1} \in \Sigma \;.\; q_{i+1} = \delta(q_i, \sigma_{i+1})$*. A run* $q_0 q_1 \dots$ *(as well as the corresponding* ω*-word* $w = \sigma_1 \sigma_2 \dots$*) is said to be* accepted *by* $\mathcal{A}$ *if and only if* $A(q_0 q_1 \dots) = \mathsf{true}$*.*

In this thesis, we also mention non-deterministic ω-automata, but we do not develop any theory involving them. In a non-deterministic automaton, there can be more than one initial state, and the transition function is replaced by a transition relation. Thereby, given a state and a symbol, the automaton can transit to more than one state. An ω-word is accepted if at least one of the runs induced by it is accepted.

Definition 2.9 (Language of an ω-automaton)**.** *We call the* language of $\mathcal{A}$ *the set* $L(\mathcal{A}) \subseteq \Sigma^\omega$ *of the* ω*-words accepted by* $\mathcal{A}$*.*

[1] By partial we mean that its domain might be a subset of $Q \times \Sigma$. One may require by definition a complete function δ_c that is defined on the entire $Q \times \Sigma$. To transform δ into δ_c, we add to the states Q a special sink state q_{err} such that $\delta_c(q, \sigma) = q_{err}$ for every pair state-symbol (q, σ) such that δ is not defined on it, and $\delta_c(q_{err}, \sigma) = q_{err}$ for every symbol σ.

There are several ways to describe the acceptance condition, and according to the nature of its description the ω-automaton has different names. Notice here that r, $\mathsf{Inf}(r)$ and $\mathsf{Fin}(r)$ are pieces of notation we are going to use nowhere else in the thesis, but they are necessary to define the acceptance conditions.

Definition 2.10 (Büchi acceptance condition, Büchi automaton)**.** *The* Büchi *acceptance condition [Büc66] is described by a set of* accepting states $F \subseteq Q$*: a run r is accepted by the automaton if and only if* $\mathsf{Inf}(r) \cap F \neq \varnothing$*. A* Büchi automaton $\mathcal{B}$ *has a Büchi acceptance condition and is defined by the tuple* $\mathcal{B} = (Q, \Sigma, q_0, \delta, F)$*.*

Notice that for conciseness, the set of states F replaces the acceptance condition A as the fifth element of the tuple.

Definition 2.11 (Muller acceptance condition, Muller automaton)**.** *The* Müller *acceptance condition [Mul63] is described by a* table *of subsets of states $T \subseteq 2^Q$: a run r is accepted by the automaton if and only if* $\mathsf{Inf}(r) \in T$*. An ω-automaton equipped with a Müller acceptance condition is called* Müller automaton *and is denoted by the tuple* $\mathcal{M} = (Q, \Sigma, q_0, \delta, T)$*.*

Notice the difference with the Büchi condition: the Müller condition requires that the set of states visited infinitely many times coincide exactly with one of the sets in T, while in the Büchi condition it is sufficient that this set contains any accepting states.

In this thesis, we use script uppercase Latin letters to denote ω-automata. E.g., $\mathcal{B}$ denotes a Büchi automaton and $\mathcal{M}$ a Muller automaton. Büchi and Muller automata are central in our construction of the weakness measure for GR(1) formulae, described in Chapters 5 and 6.

2.2 Generalized Reactivity of Rank 1

In this thesis, we focus on a subset of LTL formulae called *generalized reactivity of rank 1* (*GR(1)* for short). The nomenclature "generalized reactivity", to our knowledge appearing for the first time in [PPS05], derives from the class of *reactivity formulae* defined in [MP92],

as formulae of the kind $\bigwedge_i(\mathbf{GF}q_i \rightarrow \mathbf{GF}p_i)$ The term "generalized" refers to the presence of multiple $\mathbf{GF}$ conjuncts in both sides of the implication $\bigwedge_i(\bigwedge_j \mathbf{GF}q_{ij} \rightarrow \bigwedge_k \mathbf{GF}p_{ik})$, and the rank 1 refers to the presence of one implication only in the formulae, $\bigwedge_i \mathbf{GF}q_i \rightarrow \bigwedge_j \mathbf{GF}p_j$. More recent literature [LDS11, AMT13, MR15] has extended the syntax to include conditions of the kind $\mathbf{G}p_i$.

In this section, we define GR(1) formulae using the two definitions found in the literature, namely strict and non-strict (Section 2.2.1). Then we describe GR(1) games, which model the interaction between an environment and a controller, and formalize the problem of reactive synthesis as that of finding a winning strategy for the controller (Section 2.2.2). Finally, we introduce the problem of unrealizability, which motivates assumptions refinement, and define counterstrategies, which are largely used in assumptions refinement (Section 2.2.3).

2.2.1 GR(1) formulae

GR(1) formulae describe properties of systems consisting of two agents, typically called *environment* and *controller*. The environment's state is described by a set $\mathcal{X}$ of Boolean variables called *input variables*, and the controller's state by a set $\mathcal{Y}$ of *output variables*. In the following, we let $\mathcal{V} = \mathcal{X} \cup \mathcal{Y}$.

Here, we present the two definitions of GR(1) formulae, the strict and non-strict ones. The problem of reactive synthesis is formalized through the former [BJP+12], while the problem of assumptions refinement is often introduced with respect to the latter [LDS11, AMT13].

In the following, β is a Boolean expression according to the BNF defined in Section 2.1.1, with a slight variation. The set of terminal symbols in some cases does not coincide with the set $\mathcal{V}$ of propositional variables, but it may contain elements from the set $\mathbf{X}\mathcal{V} := \{\mathbf{X}v | v \in \mathcal{V}\}$, that is the set of propositional variables prepended by a $\mathbf{X}$ temporal operator[2]. We denote the set of terminal symbols appearing in β by $\mathsf{vars}(\beta)$.

[2]The elements of this set have the same meaning as *primed variables* defined in some contexts (like [CGP99, MRS19]). The choice of using the $\mathbf{X}$ operator instead appears in [BJP+12] and allows to express GR(1) specifications as a single LTL formula over variables in $\mathcal{V}$.

The definition of both strict and non-strict GR(1) formulae is based on a tuple of components called GR(1) specification [BJP+12].

Definition 2.12 (GR(1) specification)**.** *A GR(1) specification is a pair* $\langle \mathcal{E}, \mathcal{S} \rangle$*, where* $\mathcal{E} = \langle \theta^{\mathcal{E}}, \rho^{\mathcal{E}}, \{J_1^{\mathcal{E}}, \ldots, J_m^{\mathcal{E}}\} \rangle$ *is called* environment specification *and* $\mathcal{S} = \langle \theta^{\mathcal{S}}, \rho^{\mathcal{S}}, \{J_1^{\mathcal{S}}, \ldots, J_n^{\mathcal{S}}\} \rangle$ *is called* controller specification*. The elements of the environment specification are called* assumptions *and those of the controller specification* guarantees*. Assumptions and guarantees are formulae defined by the following structure and terminology:*

- $\theta^{\mathcal{E}}, \theta^{\mathcal{S}}$ *are called* initial conditions*. They are Boolean formulae such that* $\mathsf{vars}(\theta^{\mathcal{E}}) \subseteq \mathcal{X}$ *and* $\mathsf{vars}(\theta^{\mathcal{S}}) \subseteq \mathcal{Y}$*. They specify the initial valuation of a behaviour of the environment and the controller.*

- $\rho^{\mathcal{E}}, \rho^{\mathcal{S}}$ *are called* invariants*. They are defined in different ways for strict and non-strict GR(1) formulae:. In the former case, they are two Boolean formulae such that* $\mathsf{vars}(\rho^{\mathcal{E}}) \subseteq \mathcal{V} \cup \mathbf{X}\mathcal{X}$ *and* $\mathsf{vars}(\rho^{\mathcal{S}}) \subseteq \mathcal{V} \cup \mathbf{X}\mathcal{V}$*. In the latter case, they are two LTL formulae of the kind* $\rho^{\mathcal{E}} = \mathbf{G}\beta^{\mathcal{E}}$ *and* $\rho^{\mathcal{S}} = \mathbf{G}\beta^{\mathcal{S}}$*, where* $\mathsf{vars}(\beta^{\mathcal{E}}) \subseteq \mathcal{V} \cup \mathbf{X}\mathcal{X}$ *and* $\mathsf{vars}(\beta^{\mathcal{S}}) \subseteq \mathcal{V} \cup \mathbf{X}\mathcal{V}$*. They specify, for each* $\mathcal{V}$*-valuation, the next valuations the environment and the controller can have.*

- $J_i^{\mathcal{E}}, J_j^{\mathcal{S}}$ *for every* i *and* j *are called* fairness conditions*. They are defined in different ways for strict or non-strict GR(1) formulae. In the former case, they are Boolean formulae such that* $\mathsf{vars}(J_i^{\mathcal{E}}) \subseteq \mathcal{V}$ *and* $\mathsf{vars}(J_j^{\mathcal{S}}) \subseteq \mathcal{V}$*. In the latter, they are* $J_i^{\mathcal{E}} = \mathbf{GF}\beta^{\mathcal{E}}$ *and* $J_j^{\mathcal{S}} = \mathbf{GF}\beta^{\mathcal{S}}$*. They specify valuations that have to occur infinitely many times in any execution of the environment and the controller.*

Assumptions and guarantees together define a GR(1) formula.

Definition 2.13 (Strict GR(1) formula)**.** *A* strict generalized reactivity formula of rank 1 (GR(1)) *is the LTL formula*

$$\phi := (\theta^{\mathcal{E}} \rightarrow \theta^{\mathcal{S}}) \wedge (\theta^{\mathcal{E}} \rightarrow \mathbf{G}(\mathbf{H}\rho^{\mathcal{E}} \rightarrow \rho^{\mathcal{S}})) \wedge \left(\theta^{\mathcal{E}} \wedge \mathbf{G}\rho^{\mathcal{E}} \rightarrow \left(\bigwedge_{i=1..m} \mathbf{GF} J_i^{\mathcal{E}} \rightarrow \bigwedge_{j=1..n} \mathbf{GF} J_j^{\mathcal{S}} \right) \right) \quad (2.1)$$

The formula consists of three conjuncts. The first one, $\theta^{\mathcal{E}} \rightarrow \theta^{\mathcal{S}}$, expresses the requirement that if the environment satisfies its initial condition, then the controller must satisfy its initial condition too; if the environment violates it, then the controller has no constraints in setting the output variables. The expression $\theta^{\mathcal{E}} \rightarrow \mathbf{G}(\mathbf{H}\rho^{\mathcal{E}} \rightarrow \rho^{\mathcal{S}})$ states that if the environment satisfies its initial condition and its behaviour up to any point in time satisfies the invariant $\rho^{\mathcal{E}}$ (which is the meaning of $\mathbf{H}\rho^{\mathcal{E}}$), then the controller must satisfy its invariant $\rho^{\mathcal{S}}$ at that point in time. The last expression $\theta^{\mathcal{E}} \wedge \mathbf{G}\rho^{\mathcal{E}} \rightarrow \left(\bigwedge_{i=1..m} \mathbf{GF} J_i^{\mathcal{E}} \rightarrow \bigwedge_{j=1..n} \mathbf{GF} J_j^{\mathcal{S}}\right)$ states that in every ω-word, provided that the environment satisfies its initial condition and invariant, if it also satisfies all of its fairness conditions $J_i^{\mathcal{E}}$, then the controller must satisfy its fairness conditions $J_j^{\mathcal{S}}$.

Much work on GR(1) assumptions refinement (such as [AMT13, LDS11, AMT15]), and existing tools for GR(1) synthesis, like RATSY [BCG+10], use a simpler definition of GR(1) formulae where initial conditions, invariants, and fairness conditions are conjoined with each other, as formalized below.

Definition 2.14 (Non-strict GR(1) formula)**.** *A non-strict GR(1) formula is an implication of the form*

$$\phi := \left(\theta^{\mathcal{E}} \wedge \rho^{\mathcal{E}} \wedge \bigwedge_{i=1}^{m} J_i\right) \rightarrow \left(\theta^{\mathcal{S}} \wedge \rho^{\mathcal{S}} \wedge \bigwedge_{j=1}^{n} J_j^{\mathcal{S}}\right) \tag{2.2}$$

where $\theta^{\mathcal{E}}, \theta^{\mathcal{S}}, \rho^{\mathcal{E}}, \rho^{\mathcal{S}}, J_i^{\mathcal{E}}$ and $J_j^{\mathcal{S}}$ are defined in Definition 2.12.

In other words, non-strict GR(1) formulae are implications between two generalized Büchi formulae. The antecedent is a conjunction of assumptions and the consequent a conjunction of guarantees. We call the single assumptions and guarantees *GR(1) conjuncts*, and their conjunction *GR(1) conjunction.*

As pointed out in [BJP+12], this definition is not equivalent to the strict one given in Definition 2.13. In fact, there are cases where the latter cannot be implemented by any controller, while the former is realizable for a given set of initial conditions, invariants and fairness conditions. Intuitively, the strict formula requires that the controller should satisfy the invariant guarantee whenever the environment satisfies the invariant assumptions; the non-strict formula allows the controller to violate the invariant guarantees even if the environment satisfies

the invariant assumptions, in case it violates the fairness assumptions. We will show this in Section 2.3 after defining the notion of realizability.

Remark on Notation

Notice that given a GR(1) specification, a strict and non-strict GR(1) formula are uniquely determined, and vice versa. That is, given a specification $\langle \mathcal{E}, \mathcal{S} \rangle$, the expressions in formulae (2.1) and (2.2) are uniquely determined, and viceversa, given either a strict or non-strict formula, its initial conditions, invariants, and fairness conditions are uniquely identifiable. Therefore, unless a distinction needs to be made for preciseness, we will use the terms GR(1) specification and GR(1) formula interchangeably. Also, notice that the symbol $\mathcal{E}$ denotes a tuple of initial conditions, invariant and fairness conditions. In some parts of the thesis (as in Chapter 7), there is no need to distinguish between different kinds of assumptions and guarantees. In this case, $\mathcal{E}$ and $\mathcal{S}$ are taken to be just sets of assumptions and guarantees rather than tuples.

We define $\mathcal{E}$ to have only one initial condition and one invariant. In the rest of the thesis, we discuss assumptions refinement, where assumptions are progressively added to $\mathcal{E}$. This can lead to specifications with multiple initial conditions and guarantees. In both the strict and non-strict setting, the case of multiple initial conditions and/or invariants can be reduced to the one of Definitions 2.13 and 2.14. If the specification contains two initial conditions θ_1 and θ_2, it is equivalent to a specification with a single initial condition $\theta = \theta_1 \wedge \theta_2$. If the specification contains two invariants ρ_1 and ρ_2 in the strict definition, this is equivalent to a specification with a single invariant $\rho = \rho_1 \wedge \rho_2$. If in the non-strict definition $\rho_1 = \mathbf{G}\beta_1$ and $\rho_2 = \mathbf{G}\beta_2$, this is equivalent to the specification with a single invariant $\rho \equiv \rho_1 \wedge \rho_2 \equiv \mathbf{G}(\beta_1 \wedge \beta_2)$.

2.2.2 GR(1) Games

Assumptions and guarantees, as given in Definition 2.12, define a game between the environment and the controller. Reactive synthesis consists in finding a strategy for the controller to win the game; winning the game coincides with satisfying the formula ϕ in every infinite execution

of the strategy. The following definitions formalize the notion of games and strategies and are necessary for the completeness of this exposition. However, an informal comprehension of these concepts is sufficient for the understanding of the core chapters of the thesis, since there will be no more reference to games and strategies.

Definition 2.15 (GR(1) Game Structure [BJP+12]). *A* GR(1) game structure *is a tuple* $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, \theta^{\mathcal{E}}, \theta^{\mathcal{S}}, \rho^{\mathcal{E}}, \rho^{\mathcal{S}}, \mathsf{Win} \rangle$*, where*

- $\mathcal{X}, \mathcal{Y}, \theta^{\mathcal{E}}, \theta^{\mathcal{S}}, \rho^{\mathcal{E}}$ *and* $\rho^{\mathcal{S}}$ *are defined in Definition 2.12,*
- $\mathsf{Win} = \bigwedge_{i=1..m} \mathbf{GF} \mathsf{J}_i^{\mathcal{E}} \rightarrow \bigwedge_{j=1..n} \mathbf{GF} \mathsf{J}_j^{\mathcal{S}}$ *is a formula representing the* winning condition *of the game.*

Informally, the game structure models the possible interactions between two players, the environment and the controller. The environment first selects a valuation for the variables in $\mathcal{X}$ and then the controller responds with a valuation of $\mathcal{Y}$. Each $\mathcal{V}$-valuation forms a *state* of the game. The initial conditions determine which state the interactions are allowed to start from. The invariants determine which states are reachable from any current state: the assumptions constrain the environment's choices and the guarantees the controller's ones. Either the interaction terminates when some of the players have no choices to satisfy their invariants, or it plays out for infinite steps: in the latter case, the controller wins if the infinite sequence of states (ω-word) generated by this interaction satisfies the winning condition. The following definitions formalize this.

Definition 2.16 (Game state). *A* state *of the game is any valuation* σ *of* $\mathcal{V} = \mathcal{X} \cup \mathcal{Y}$*. A state* σ *is* initial *for the game if* $\sigma \models \theta^{\mathcal{E}} \wedge \theta^{\mathcal{S}}$*. A state* σ' *is a* successor *of* σ *in* $\mathcal{G}$ *iff any* ω*-word* $w = \sigma\sigma' \cdots \in \Sigma^{\omega}$ *satisfies* $\rho^{\mathcal{E}} \wedge \rho^{\mathcal{S}}$*.*

Definition 2.17 (Play [BJP+12]). *A* play *of* $\mathcal{G}$ *is a maximal* [3] *sequence of states* $w = \sigma_0\sigma_1 \ldots$ *such that* σ_0 *is an initial state and* σ_{j+1} *is a successor of* σ_j *for every index* j*.*

From this definition, it is clear that a play is a finite word or an ω-word with alphabet $2^{\mathcal{V}}$.

[3] This means that there is no subsequence of a play that is itself a play.

To distinguish between environment and controller choices, we use the notion of projections[4] of valuations.

Definition 2.18 (Projection of a valuation)**.** *Let σ be a $\mathcal{V}$-valuation and $\mathcal{V}' \subseteq \mathcal{V}$. Then $\sigma|_{\mathcal{V}'} := \sigma \cap \mathcal{V}'$ is called the* projection of σ onto $\mathcal{V}'$.

Example 2.1. *Consider $\mathcal{V} = \{a, b, c\}$, $\mathcal{V}' = \{a\}$, and $\sigma = \{a, c\}$, that is, the valuation where a and c are* true *and b is* false*. The projection of σ onto $\mathcal{V}'$ is $\sigma|_{\mathcal{V}'} = \{a\}$, that is, the valuation where a is* true*. In other words, the projection is just the restriction of the valuation onto a subset of the propositional variables.*

Reactive synthesis in the context of GR(1) games consists in finding a winning strategy for the controller. A controller strategy is any function that given the current state, the last choice of the environment, and a memory value that keeps track of the previous states traversed, decides on the next valuation of the output variable. Symmetrically for the environment: an environment strategy is a decision function that selects the next input variables given the current state and the memory of previous states. We denote by $M^{\mathcal{S}}$ and $M^{\mathcal{E}}$ the domain of values the memory can contain in controller and environment strategies respectively.

Definition 2.19 (Controller and environment strategies [BJP+12, KHB09])**.** *A* controller strategy *is a partial function $f^{\mathcal{S}} : M^{\mathcal{S}} \times 2^{\mathcal{V}} \times 2^{\mathcal{X}} \to M^{\mathcal{S}} \times 2^{\mathcal{Y}}$, .*[5]

An environment strategy *is $f^{\mathcal{E}}$ is a partial function $f^{\mathcal{E}} : M^{\mathcal{E}} \times 2^{\mathcal{V}} \to M^{\mathcal{E}} \times 2^{\mathcal{X}}$., .*[6]

In this definition, the memory value is needed as input to the function because a strategy may choose different next states from the same current states of the game depending on the history of states visited. Apart from this chapter, though, we will not use memory in the rest of the thesis.

Definition 2.20 (Compliance of plays with strategies [BJP+12])**.** *We say that a play $w = \sigma_0\sigma_1\ldots$ is* compliant with a controller strategy *$f^{\mathcal{S}}$ iff, for all $i \geq 0$, there exists a sequence*

[4]Notice that the projections defined here are different from the projections of Boolean formulae defined in Chapter 4, although similar. We use again Definition 2.18 in Chapter 5 to prove a property of Hausdorff dimension

[5]

[6]

of memory values $m_0, m_1, \ldots$ *such that* $f^{\mathcal{S}}(m_i, \sigma_i, \sigma_{i+1}|_{\mathcal{X}}) = (m_{i+1}, \sigma_{i+1}|_{\mathcal{Y}})$. *Similarly, we say that* w *is* compliant with an environment strategy $f^{\mathcal{E}}$ *iff for all* $i \geq 0$ *there exists a sequence of memory values* $m_0, m_1, \ldots$ *such that* $f^{\mathcal{E}}(m_i, \sigma_i) = (m_{i+1}, \sigma_{i+1}|_{\mathcal{X}})$.

While reactive synthesis is concerned with finding a controller strategy, the problem of assumptions refinement only uses environment strategies. Therefore, in the following we will focus on the latter only.

2.2.3 Unrealizability and Counterstrategies

In order to win the game, a controller strategy must ensure that any play compliant with it satisfies the formula (2.1). Symmetrically, an environment strategy must ensure that any play compliant with it violates the formula (2.1).

Definition 2.21 (Winning strategies)**.** *Given a strict GR(1) formula* ϕ *and a GR(1) game* $\mathcal{G}$*, a controller strategy* $f^{\mathcal{S}}$ *is* winning for the controller *if for every play* w *compliant with* $f^{\mathcal{S}}$*,* $w \models \phi$*. Equivalently,* $f^{\mathcal{S}}$ *is winning for the controller if and only if every play* w *compliant with* $f^{\mathcal{S}}$ *satisfies the following three conditions:*

1. *;*
2. *for every two consecutive symbols* σ *and* σ' *in* w*,* $\sigma\sigma' \ldots \models \rho^{\mathcal{E}} \wedge \rho^{\mathcal{S}}$*;*
3. *and either (i)* $w = \sigma_0 \ldots \sigma_n$ *is finite and there exists no state* $\sigma' \in 2^{\mathcal{V}}$ *such that* $\sigma_n\sigma' \ldots \models \rho^{\mathcal{E}}$*, or (ii)* w *is infinite and* $w \models \mathsf{Win}$.

An environment strategy $f^{\mathcal{E}}$ *is* winning for the environment *if for every play* w *compliant with* $f^{\mathcal{E}}$*,* $w \models \neg\phi$.

In reactive synthesis, successful implementations of GR(1) formulae coincide with winning strategies for the controller. The existence of such a strategy is termed the *realizability* of a GR(1) formula (or equivalently, of a GR(1) specification, or a GR(1) game).

Definition 2.22 (Realizability, unrealizability)**.** *A strict GR(1) formula (or specification, or game) is said to be* realizable *if there exists a winning strategy for the controller in the game associated with it (in the game itself). Otherwise, it is said to be* unrealizable*.*

In our context, we deal with repairing unrealizable formulae. If a formula is unrealizable, then there exists a strategy winning for the environment [KHB09]. In the context of assumptions refinement, such strategies are known as *counterstrategies* [AMT13, LDS11].

Definition 2.23 (Counterstrategy [KHB09])**.** *Given an unrealizable GR(1) formula ϕ, a* counterstrategy *c for the formula is a winning strategy for the environment.*

we use the symbol c to denote counterstrategies throughout the thesis.

Counterstrategies can be depicted as graphs in which any infinite path exhibits the violation of a guarantee while satisfying all the assumptions. More precisely:

- The initial state of the counterstrategy satisfies $\theta^{\mathcal{E}}$ in such a way that no valuation of $\mathcal{Y}$ can satisfy $\theta^{\mathcal{S}}$; or
- all pairs of consecutive states along a finite path satisfy $\rho^{\mathcal{E}}$ and the last state along the path is such that no valuation of $\mathcal{Y}$ can satisfy $\rho^{\mathcal{S}}$; or
- all infinite paths satisfy every $J_i^{\mathcal{E}}$ but violate at least one $J_j^{\mathcal{S}}$.

Informally, the nodes of a counterstrategy graph group together all the $\mathcal{V}$-valuations with the same memory value such that the choice of the next $\mathcal{X}$-valuation is the same. The edges are labelled with all the possible responses of the controller in the form of $\mathcal{Y}$-valuations[7].

Definition 2.24 (Counterstrategy graph)**.** *The* graph of a counterstrategy *c is a tuple $\mathcal{C} = (S, s_0, E)$ where*

[7]The definition of counterstrategy graph we give here is a more detailed version than the one given in [AMT13]. There, counterstrategies are defined as Moore transducers. Here, we include more detail by describing what each state of the Moore transducer represents.

- $S \subseteq 2^{2^{\mathcal{V}}} \times M \times 2^{\mathcal{X}}$ *is a set of nodes. Every element* $s = (\Sigma_s, m, \sigma'|_{\mathcal{X}}) \in S$*, called* counterstrategy state*, is a triple consisting of a set* Σ_s *of* $\mathcal{V}$*-valuations, a memory value* m*, and an* $\mathcal{X}$*-valuation* $\sigma'|_{\mathcal{X}}$*;*
- $s_0 = (\Sigma_0, m, \sigma'_{\mathcal{X}}) \in S$ *is the initial state, where for every* $\sigma_0 \in \Sigma_0$*,* $\sigma_0 \models \theta^{\mathcal{E}}$*;*
- *is a set of edges labelled with sets of* $\mathcal{Y}$*-violations.*

The graph satisfies the following conditions.

(i) For every node $s = (\Sigma_s, m, \sigma'|_{\mathcal{X}})$*, if* $\sigma_1, \sigma_2 \in \Sigma_s$*, then for some* $m' \in M$*,* $c(\sigma_1, m) = c(\sigma_2, m) = (\sigma'|_{\mathcal{X}}, m')$*.*

(ii) For every edge $e = (s_1, \Sigma_e, s_2)$ *let* $s_1 = (\Sigma_1, m_1, \sigma'_1|_{\mathcal{X}})$ *and* $s_2 = (\Sigma_2, m_2, \sigma'_2|_{\mathcal{X}})$*. Then for every* $\sigma_2 \in \Sigma_2$*, we have* $\sigma'_1|_{\mathcal{X}} = \sigma_2 \cap \mathcal{X}$*, and there exists* $\sigma'_1|_{\mathcal{Y}} \in \Sigma_e$ *such that* $\sigma'_1|_{\mathcal{Y}} = \sigma_2 \cap \mathcal{Y}$*. Moreover,* m_2 *is such that for every* $\sigma_1 \in \Sigma_1$ *and* $\sigma_2 \in \Sigma_2$*,* $c(\sigma_1, m_1) = (\sigma_2 \cap \mathcal{X}, m_2)$*.*

The two conditions above say that: *(i)* if two $\mathcal{V}$-valuations are grouped in the same node with the same memory value m, then the counterstrategy's choice for the next $\mathcal{X}$-valuations is the same in both valuations and is equal to $\sigma'|_{\mathcal{X}}$; *(ii)* if two nodes are connected by an edge, then the $\mathcal{X}$-valuation of the destination node corresponds to the input choice made by the counterstrategy in the source node, and the $\mathcal{Y}$-valuation of every state in the destination node is in the edge's label. Moreover, the memory value m_2 of the destination node is the one set by the counterstrategy in the source node.

We now provide the example of an unrealizable GR(1) formula with its counterstrategy.

Example 2.2 (Request-grant protocol [CHJ08, AMT13])**.** *Consider a system consisting of a device that can require access to some shared resource (a memory location, for instance). We want to specify a simplified request-grant protocol between the device and the resource in order to allow communication between the two.*

The protocol consists of two input variables $\mathcal{X} = \{\mathsf{req}, \mathsf{cl}\}$ *controlled by the device (environment), and two output variables* $\mathcal{Y} = \{\mathsf{gr}, \mathsf{val}\}$ *set by the resource (controller). The device can request*

access by setting req *to* true*, and the resource can grant access by setting* gr *to* true*. The device can request to clear the memory location by setting* cl *to* true*, and the resource signals whether its content is valid or cleared through the* val *bit.*

The specification has a single fairness condition for assumption: it forces the signal req *not to be* false *in infinitely many steps.*

$$J^{\mathcal{E}} = \neg \mathsf{req}$$

The guarantees express the requirement that (i) every time the environment issues the cl *signal, the data in the memory location must not be considered valid, and (ii) the environment must be granted access to valid data infinitely many times.*

$$\rho^{\mathcal{S}} = \mathsf{cl} \rightarrow \neg \mathsf{val}$$

$$J^{\mathcal{S}} = \mathsf{gr} \wedge \mathsf{val} \ .$$

This specification is unrealizable. In fact, the environment can force the violation of at least one guarantee by the input variable cl *to at every step: in this way, if the controller sets* val *to be always* false*, the invariant $\rho^{\mathcal{S}}$ is satisfied but the fairness condition $J^{\mathcal{S}}$ is violated; on the contrary, if at some point the controller sets* val *to* true*, the invariant is violated .*

A counterstrategy exploiting this behavior is shown in Figure 2.1. The path $s_0(s_1 s_2)^{\omega}$, with the loop between s_1 and s_2, compose the infinite behavior in which the controller keeps val *to* false*, thus violating $J_1^{\mathcal{S}}$. The loop between s_1, where* req *is forced to* false*, and s_2, where it is forced to* true *ensures the environment satisfies $J_1^{\mathcal{E}}$. The state s_{f1} is the destination state of the edges labeled with* val = true*; this, together with* cl = false*, forces the violation of $\rho_1^{\mathcal{S}}$. Any infinite path going through s_{f1} violates this invariant, and s_{f2} is a sink state collecting every such path.*

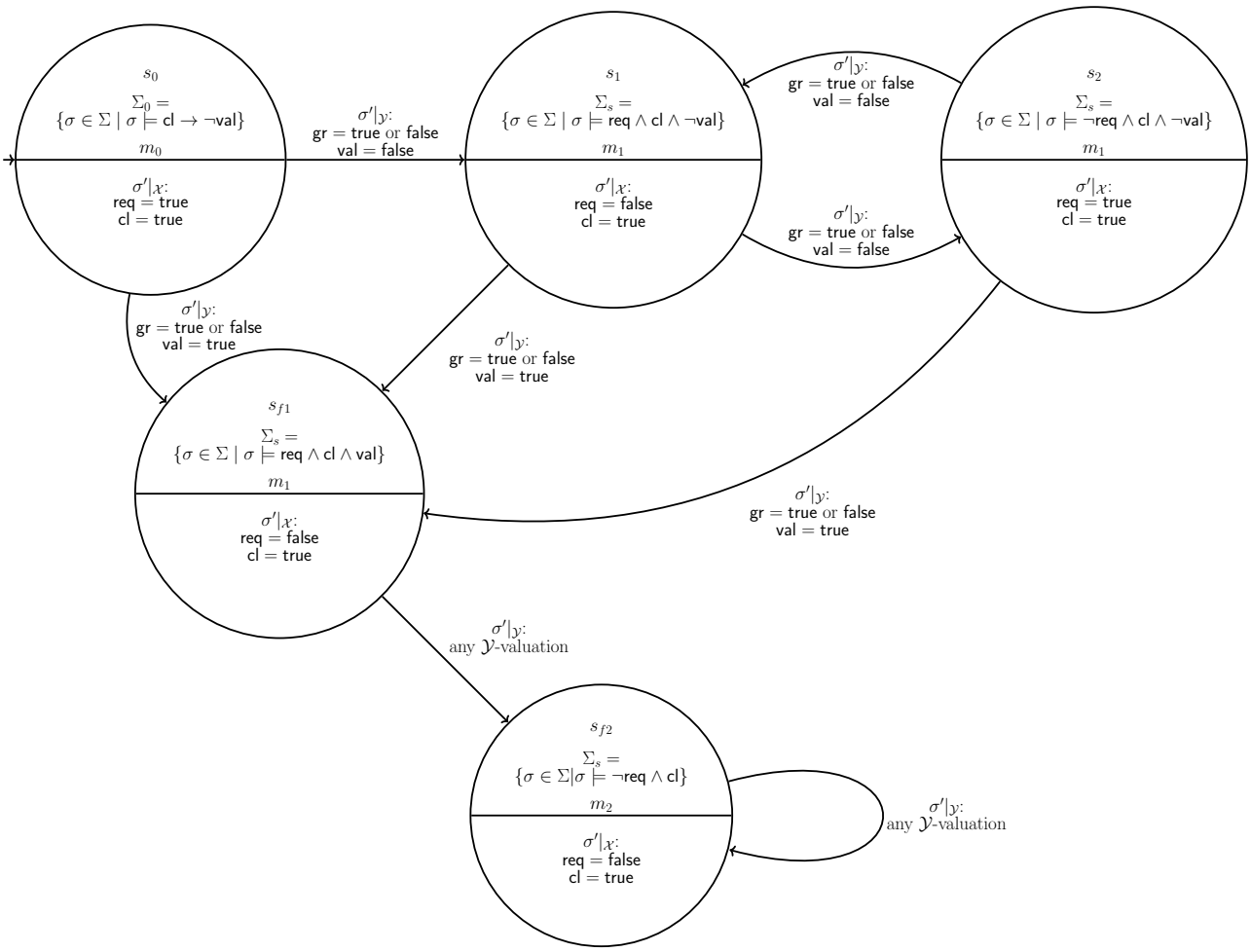


Figure 2.1: Counterstrategy for Example 2.2. In each node the upper half contains an identifier for the state, the set Σ_s, and a memory value. The lower half contains the next choice $\sigma'|_{\mathcal{X}}$ for the $\mathcal{X}$-valuation. The edges are labeled with sets of $\mathcal{Y}$-valuations $\sigma'|_{\mathcal{Y}}$, as in Definition 2.24 (which in the figure are represented as alternative $\mathcal{Y}$-valuations on the same edge). The states s_{f1} and s_{f2} are called *failing states*: any play of the counterstrategy passing through these states violates the invariant guarantee $\rho^{\mathcal{S}}$.

2.2.4 Unrealizable Cores

When a GR(1) specification is unrealizable, a common practice for finding the cause of unrealizability is to reduce the set of guarantees[8] while still retaining the unrealizability [CRST08]. The simplified specification is called an *unrealizable core*.

Definition 2.25 (Unrealizable core)**.** *Given an unrealizable GR(1) specification $\langle \mathcal{E}, \mathcal{S} \rangle$, an* unrealizable core $\mathcal{S}^{uc} \subseteq \mathcal{S}$ *of the specification is a subset of guarantees such that the specification $\langle \mathcal{E}, \mathcal{S}^{uc} \rangle$ is unrealizable, and for every $\mathcal{S}' \subseteq \mathcal{S}^{uc}$, the specification $\langle \mathcal{E}, \mathcal{S}' \rangle$ is realizable.*

[8] In the original definition, both assumptions and guarantees were eliminated [CRST08]. Later work [KHB09] points out that removing assumptions makes the analysis of unrealizability more difficult, since the environment has more freedom in undertaking behaviours that may guarantees. In this thesis we use the latter approach.

Unrealizable cores have two roles in the analysis of unrealizability: *(i)* they allow to focus the attention on a subset of guarantees among which a violation can be forced by the environment; *(ii)* it makes the computation of counterstrategies much faster [KHB09]. Indeed the main tools for the analysis of GR(1) specifications (RATSY [BCG+10] and SPECTRA [MRS19]) compute by default an unrealizable core before computing counterstrategies.

2.3 Relationship between Strict and Non-Strict GR(1) Realizability

The notion of strict GR(1) formulae was given in compliance with a discussion, carried out in [DW90, AL91], about the importance of separating out safety specifications from liveness specifications, that need different methods to be verified on systems. However, the non-strict version is the most common in GR(1) assumptions refinement works [LDS11, AMT13, LSSS14] (which are the baseline of our work and will be the focus of our thesis). We give here the definition of non-strict GR(1) realizability.

Definition 2.26 (Non-strict GR(1) realizability)**.** *Given a non-strict GR(1) formula ϕ (as defined in Definition 2.14), the formula is* realizable *(or equivalently, we say that the GR(1) specification is* realizable in the non-strict sense*) if there exists a controller strategy (as defined in Definition 2.19) such that all the plays compliant with it satisfy ϕ.*

We discuss here the impact of the difference between strict and non-strict GR(1) formulae on realizability. Consider the following example from [BJP+12].

Example 2.3. *Let $\mathcal{X} = \{x\}$ and $\mathcal{Y} = \{y\}$, the assumptions $\rho^{\mathcal{E}} = \mathbf{X}x$ ($\mathbf{G}(\mathbf{X}x)$ in the non-strict definition), $J^{\mathcal{E}} = x \leftrightarrow y$ ($\mathbf{GF}(x \leftrightarrow y)$ in the non-strict definition), $\rho^{\mathcal{S}} = \mathbf{X}x \leftrightarrow \mathbf{X}y$, $J^{\mathcal{S}} = \neg y$. A strategy satisfying the non-strict formula consists in setting the output y always to the inverse of x, so that $J^{\mathcal{E}}$ is violated despite $\rho^{\mathcal{S}}$ being violated too. The strict formula, instead, is unrealizable. Indeed, if the environment satisfies $\rho^{\mathcal{E}}$, any controller has to satisfy $\rho^{\mathcal{S}}$ and y*

must be set to the same value as x at every step but the initial; therefore, $J^{\mathcal{E}}$ is satisfied and the environment where x is always true *forces the controller to violate the fairness guarantee $J^{\mathcal{S}}$.*

As shown in this example, in non-strict formulae the controller is allowed to violate any of the guarantees whenever the environment violates any of the assumptions. On the contrary, in strict formulae initial conditions, invariants, and fairness conditions are decoupled: when the environment satisfies the invariant assumptions, the controller is still required to satisfy the invariant guarantees even in case the environment violates the fairness assumptions.

Realizability of non-strict GR(1) formulae is defined as in Definition 2.22, where the plays compliant with the winning strategy are required to satisfy the formula in Definition 2.14. The specification is realized by a controller that satisfies all the guarantees whenever the environment satisfies all the assumptions, or by a controller that forces any of the assumptions to be falsified. The work in [BJP+12] shows how to set up the GR(1) game in order to find the winning strategy with the same algorithm as in the strict case.

The nomenclature strict/non-strict is due to the following relationship between the realizability of the two formulae. In the following statement, ρ and J_i are defined as in the strict version, without the operators **G** and **GF** respectively. Hence, the formula (2.2) is modified accordingly in order to include these operators.

In the next chapter, we define the problem of assumptions refinement in both the strict and non-strict setting. The heuristics we develop in the later chapters are applicable to either setting, except for the weakness in Chapter 6, which makes explicit reference to GR(1) conjunctions and thereby to non-strict GR(1) formulae. For this reason, in the rest of the thesis we only consider non-strict GR(1) formulae.

However, we choose to include strict GR(1) formulae here in the background because the main tools developed for assumptions refinement, mainly RATSY [BCG+10] and SPECTRA [MRS19], despite presenting a front-end user interface that expresses GR(1) specifications in the non-strict sense, have back-end algorithms to check realizability and produce counterstrategies in the strict sense. The soundness of the overall results is guaranteed by the following property.

Proposition 2.1. *For a given set of initial conditions θ^α, invariants ρ^α and fairness conditions J^α, if the strict formula (2.1) is realizable, then so is the non-strict formula (2.2).*

Proof. It suffices to see that (2.1) implies (2.2). Let $f^{\mathcal{S}}$ be a winning strategy. Any play compliant with $f^{\mathcal{S}}$ satisfies (2.1) by definition, therefore it also satisfies (2.2). In conclusion, $f^{\mathcal{S}}$ realizes formula (2.2). □

2.4 Entropy and Hausdorff Dimension

In this section, we provide algebraic and topological definitions regarding ω-languages, including the central definition of *closed ω-languages*. Then we survey some properties of graphs that are involved in the definition of the algorithms we describe in Chapter 6. Finally, we give the central definitions of entropy and Hausdorff dimension, with illustrative examples to help the reader to understand what these quantities represent.

Concatenation of words and closed ω-languages. We present here some additional notation about languages and automata that completes the one introduced in Section 2.1.3 and is used throughout this chapter and Chapter 6. This includes the concatenation operator over words and the definition of some topological properties of interest over ω-languages.

Recall from Section 2.1.2, a word w is explicitly denoted as a sequence of its symbols $\sigma_1\sigma_2\ldots\sigma_n$, or with a parenthesis notation $(\sigma_1, \sigma_2, \ldots, \sigma_n)$, where the symbols are separated by commas. This notation is also used for ω-words. The notation $w[i..]$ denotes the suffix of w starting with the symbol σ_i. We write $w[i]$ to denote the symbol appearing in the ith position in w, i.e., σ_i. $|w|$ denotes the number of symbols appearing in the word w (aka the length of w).

Given two words v and w, their concatenation is denoted as $v \cdot w$ or simply vw. The same notation is used for the concatenation of a word v and an ω-word w; of course, the concatenation of an ω-word and a word, where the ω-word comes first, has no existing definition. Given a set V of finite-length words and a set W of finite-length words or ω-words over the same alphabet

Σ, the set $V \cdot W$ is the set of words obtained by concatenating a word in V with a word in W. *Kleene's star operator* yields the set V^* of finite words obtained by concatenating an arbitrary number of words in V. The *omega operator* applied to V yields the set V^ω of ω-words obtained by concatenating a (countably) infinite number of words in V. Naturally, Σ^* and Σ^ω represent the set of all finite words and all ω-words over the alphabet Σ respectively. We also borrow the convention from the literature on languages (such as in [CJS09]) of dropping brackets from singleton sets of words: hence, the set $\{w\}$ containing only the word w is written as w, whilst the set $\{w\} \cdot \Sigma^*$ of all finite words starting with a w is written as $w\Sigma^*$.

Given an ω-language $L \subseteq \Sigma^\omega$, we denote the set of all $v \in \Sigma^*$ of length n such that v is a prefix of some word in L by $\mathsf{pref}_n(L)$. We also define $\mathsf{pref}(L) = \bigcup_{n \in \mathbb{N}} \mathsf{pref}_n(L)$ to be the set of all the prefixes of ω-words in L.

Definition 2.27 (Closed ω-language)**.** *An ω-language L is* closed *if and only if, for any ω-word w such that $\mathsf{pref}(\{w\}) \subseteq \mathsf{pref}(L)$, $w \in L$.*

In other words, L is closed if whenever a word w is arbitrarily close to some word w' in L (that is, for every finite prefix of w there is a word w' in L having the same prefix), then $w \in L$.

Definition 2.28 (Closure of an ω-language)**.** *The* closure *of an ω-language L, denoted by $\mathcal{C}(L)$, is the smallest closed ω-language that contains L.*

These two concepts come from the topology over Σ^ω defined in [Sta15].

Despite being infinite sets, the ω-languages of interest to us allow for a computable finite representation through automata. Formally, we define a *regular ω-language* as an ω-language which is accepted by a deterministic Muller automaton (DMA, introduced in Definition 2.11).

While DMAs are sufficient to define regular languages, many applications use different accepting conditions, e.g., Büchi automata . A *deterministic Büchi automaton* (DBA) $\mathcal{B}$ is defined in the same way as a DMA except for the acceptance condition (see Definition 2.10). Given a DBA it is always possible to obtain an equivalent DMA by replacing the Büchi acceptance condition. However, there exist DMAs for which no equivalent deterministic Büchi automaton

exists [GTW02]. In this case, nondeterministic automata may be used to extend the expressiveness of Büchi automata. We do not consider nondeterministic automata in our work, as the computation of the quantities defined below requires deterministic automata.

We define here reachable and co-reachable states in a DBA [ABD14a]. The same definitions hold for DMAs.

Definition 2.29 (Reachable/co-reachable state)**.** *Given a Büchi automaton $\mathcal{B} = \langle Q, \Sigma, q_0, \delta, F \rangle$, a state q is* reachable *if there exists a word $v \in \Sigma^*$ such that $\delta(q_0, v) = q$. It is* co-reachable *if there exists a word $\bar{v} \in \Sigma^*$ such that $\delta(q, \bar{v}) = f$, for some accepting state $f \in F$.*

Given an ω-language L, there exists at least one Büchi automaton such that all states are reachable and co-reachable. In fact, if some state is not reachable, it is never reached by any run starting from the initial state, and can be safely removed without changing the accepted language L. Likewise, suppose a state q is not co-reachable, and suppose there exists a word $v \in \Sigma^*$ such that $\delta(q_0, v) = q$. Then, any ω-word $w \in \Sigma^\omega$ starting with v cannot be accepted by the automaton, since the run induced by it traverses q and cannot reach any accepting state from there. Hence, non-co-reachable states can be safely removed without changing the accepted language L. The same observations can be referred to DMAs. In the rest of the thesis, we suppose all DBAs and DMAs to have only reachable, co-reachable states.

Graphs and Adjacency Matrices. Muller and Büchi automata are typically represented by graphs. In particular, a Büchi automaton $\mathcal{B}$ (or a Muller automaton $\mathcal{M}$) is represented as a labelled graph $\mathcal{G} = (Q, E)$, where the set of *nodes* Q is the set of states in $\mathcal{B}$, $E \subseteq Q \times \mathbb{N} \times Q$ is a set of (multi-)*edges* such that $(q_i, n, q_j) \in E$ if and only if there exist exactly n symbols $\sigma \in \Sigma$ such that $\delta(q_i, \sigma) = q_j$. Throughout this and the next chapter, we heavily rely on the properties of the graph underlying an automaton. Specifically, we use the adjacency matrix, the notion of a subgraph induced by a subset of states, and strong connectedness.

Given a subset $Q' \subseteq Q$, the *subgraph* induced by Q' on $\mathcal{G}$ is the graph $\mathcal{G}[Q'] = (Q', E')$ such that $(q_i, n, q_j) \in E'$ if and only if $(q_i, n, q_j) \in E$ and $q_i, q_j \in Q'$.

We give here the central definition of strong connectedness, that will occur frequently throughout this and the next chapter.

Definition 2.30 (Strong connectedness)**.** *A graph is* strongly connected *if, for any two states $q_i, q_j \in Q$, there exists a path (a sequence of consecutive edges) from q_i to q_j and vice versa.*

If a graph $\mathcal{G}$ is not strongly connected, it can admit one or more *strongly connected components*.

Definition 2.31 (Strongly connected components (SCC))**.** *A* strongly connected component *of $\mathcal{G}$ is a maximal*[9] *strongly connected subgraph of $\mathcal{G}$.*

Naturally, different SCCs are disjoint.

Let A_{ij} denote an element in position (i, j) of matrix A. A graph can be represented by its *extended adjacency matrix* A, defined as the square matrix of size $|Q| \times |Q|$ such that: $A_{ij} = n$ if and only if $(q_i, n, q_j) \in E$. Since the elements of A are nonnegative, A is a *nonnegative matrix*. Given a subgraph $\mathcal{G}[Q']$ induced by $Q' \subseteq Q$, we denote its extended adjacency matrix by $A[Q']$; this $|Q'| \times |Q'|$ matrix can be obtained by removing from A the rows and columns corresponding to the states not in Q'. Any matrix obtained by removing from A a set of rows and columns with the same indices is called a *principal submatrix* of A [HJ12]. We say that an adjacency matrix is *irreducible* if it describes a strongly connected graph [HJ12].

Given an automaton's graph, we are interested in the growth rate of the number of paths of a certain length as a function of the length. For large enough lengths, this number grows approximately exponentially, and the growth rate (i.e., the base of this exponential function) is the *spectral radius* of the adjacency matrix A [MS94], which in turn is the Perron-Frobenius eigenvalue of A (a positive real eigenvalue which has the largest absolute value among all eigenvalues), and is denoted by $\nu(A)$ [Sen06] (more details are given in Section 2.4). In what follows, we use a property of the spectral radius that states a sufficient condition for a strict ordering of the spectral radii of two matrices.

Theorem 2.1 (Monotonicity of spectral radius [BP94, Chapter 2 Corollary 1.5])**.** *Let A and B be two nonnegative square matrices of the same size such that $A_{ij} \leq B_{ij}$ for all (i, j) that*

[9] A set S satisfying some property is *maximal* if there is no superset $S' \supset S$ satisfying the same property.

index positions in A and B, and $A_{ij} < B_{ij}$ for some (i, j). Suppose $A + B$ is irreducible. Then $\nu(A) < \nu(B)$.

Accepting states and accepting SCCs. Recall from Definition 2.10 that a run $q_0 q_1 \dots$ of a deterministic Büchi automaton is accepted if it traverses at least one accepting state infinitely many times. This is possible only if the accepting state is reachable from itself, that is, it belongs to some SCC. We denote by *accepting SCC* an SCC containing at least one accepting state.

Definition 2.32. *Given a DBA $\mathcal{B} = \langle Q, \Sigma, q_0, \delta, F \rangle$, an* accepting SCC *$S \subseteq Q$ is an SCC of the graph of $\mathcal{B}$ such that $S \cap F \neq \varnothing$.*

Suppose an accepting state $f \in F$ is not reachable from itself. Then, no accepting run can traverse it infinitely many times. Therefore, by removing f from F, the automaton $\mathcal{B}$ does not lose any accepted run. In other words, if f is removed from the accepting states, an equivalent DBA is obtained. When we refer to co-reachable states, we are interested in states from which an accepting SCC is reachable, rather than an accepting state. This prevents from including under this definition those states from which only such "degenerate" accepting states like f are reachable, which are traversed by no accepting run. For this reason, in the rest of the thesis we use the following alternate definition of co-reachability (slightly different from Definition 2.29).

Definition 2.33 (Co-reachable states)**.** *Given a DBA $\mathcal{B} = \langle Q, \Sigma, q_0, \delta, F \rangle$, a state $q \in Q$ is co-reachable if there exists a word $v \in \Sigma^*$ such that $q' = \delta(q, v)$ is in some accepting SCC.*

Entropy, Hausdorff dimension, and algorithms to compute them. *Hausdorff dimension* and *Hausdorff measure* are basic concepts in fractal geometry and represent a way of defining measures—that is, analogous concepts to length, area, volume from classical geometry—for fractals [Fal04]. Staiger [Sta15] pointed out a homeomorphism between fractals and regular ω-languages and proposed an analogous interpretation of the Hausdorff dimension and measure for ω-languages.

Intuitively, given an ω-language L, its Hausdorff dimension quantifies the growth rate of the number of distinct n-long prefixes of words in the language as a function of n. This makes it a good candidate for quantifying weakness: the less constrained the language is, the more prefixes of a fixed length are contained in it, implying a higher Hausdorff dimension.

The formal definition of Hausdorff dimension is tightly related to the notion of Hausdorff measure. The following definitions are taken from [Sta98].

Definition 2.34 (γ-dimensional Hausdorff outer measure)**.** *Given a regular ω-language L over an alphabet Σ with cardinality r, and a nonnegative real value γ, the* γ-dimensional Hausdorff outer measure *of L is defined as*

$$m_\gamma(L) = \lim_{n\to\infty} \inf_{V\in\mathcal{L}_n} \sum_{v\in V} r^{-\gamma|v|} \tag{2.3}$$

where $\mathcal{L}_n = \{V \subseteq \Sigma^* \mid V \cdot \Sigma^\omega \supseteq L \text{ and } |v| \geq n \text{ for all } v \in V\}$ is the collection of languages V containing finite words of length at least n such that every word in L has at least one prefix in V.

Definition 2.35 (Hausdorff dimension and measure)**.** *Given an ω-language L, its* Hausdorff dimension*, denoted by $\mathsf{dim}\,(L)$, is the (unique) value $\bar{\gamma}$ such that*

$$m_\gamma(L) = \infty \;\; \gamma < \bar{\gamma}$$

$$m_\gamma(L) = 0 \;\; \gamma > \bar{\gamma}$$

The value $m_{\mathsf{dim}(L)}(L)$ is called the Hausdorff measure *of L, and is denoted by $m(L)$.*

In other words, Hausdorff measure is the limit of the process of approximating the ω-language L by a set V of finite prefixes with length at least n, and weighting each prefix w with a quantity $r^{-\gamma|v|}$ that decreases exponentially as the prefix length increases. This limit can be finite and positive for at most one value of the γ parameter. This value is called *Hausdorff dimension*. We illustrate the notions above in the following examples.

Example 2.4. *Consider the set of variables $\mathcal{V} = \{a, b\}$, and the associated alphabet $\Sigma =$*

$\{\varnothing, \{a\}, \{b\}, \{a, b\}\}$, *with number of symbols* $r = 4$*; we denote the symbols in* Σ *as* $\sigma_0, \sigma_1, \sigma_2$*, and* σ_3 *respectively. We want to determine the* γ*-dimensional Hausdorff measure of the language* L *of the formula* $\mathbf{G}a$*, i.e., the set of all* ω*-words containing only symbols* σ_1 *and* σ_3*:*

$$L = \{\sigma_1, \sigma_3\}^\omega \ .$$

To achieve this, we determine a $V_n \in \mathcal{L}_n$ *that achieves the limit of the infimum in* (2.3)*, and in particular, for* $\gamma < \mathsf{dim}\,(L)$*, minimizes the sum* $s_V = \sum_{v \in V} 4^{-\gamma|v|}$ *over* $\mathcal{L}_n$ *(that is over the set of languages* V *containing only words of length at least* n *such that every* ω*-word in* L *has at least one prefix in* V*).*

The choice of $V_n = \{\sigma_1, \sigma_3\}^n$*, which trivially belongs to* $\mathcal{L}_n$*, is appropriate. Since* V_n *contains* 2^n *words of length* n*,*

$$s_{V_n}(n) = \sum_{v \in V_n} 4^{-\gamma n} = 2^n 4^{-\gamma n} = 2^{(1-2\gamma)n} = \begin{cases} 2^{(1-2\gamma)n} \xrightarrow{n\to\infty} 0 & \text{for } \gamma > 1/2; \\ 1 & \text{for } \gamma = 1/2; \\ 2^{(1-2\gamma)n} \xrightarrow{n\to\infty} \infty & \text{for } \gamma < 1/2. \end{cases}$$

Thus, for $\gamma > 1/2$*,*

$$0 \leq m_\gamma(L) = \lim_{n\to\infty} \inf_{V \in \mathcal{L}_n} s_V \leq \lim_{n\to\infty} s_{V_n} = 0 \ .$$

Hence $m_\gamma(L) = 0$ *for* $\gamma > 1/2$*.*

For $\gamma \leq 1/2$ *we show that* $s_{V_n} = \mathsf{inf}_{V \in \mathcal{L}_n} s_V$*, that is* V_n *achieves the infimum over* $\mathcal{L}_n$*. Intuitively, what happens is that if we take any* $V \in \mathcal{L}_n$ *and replace a word* $v \in V$ *with* $v\sigma_1$ *and* $v\sigma_3$*, then the corresponding sum* s_V *increases, since the difference is* $2 \cdot 4^{-\gamma(|v|+1)} - 4^{-\gamma|v|} = 4^{-\gamma|v|}(2 \cdot 4^{-\gamma} - 1) \geq 0$*. Reversing the argument, starting from any finite* V*, we can get to* V_n *by reducing (non-increasing)* s_V *in every step. Unfortunately this argument does not work when* V *is infinite. To address this case rigorously, we take a different route.*[10] *Consider first the case of* $\gamma = 1/2$*, and let* $V \in \mathcal{L}_1$ *(note that* $\mathcal{L}_{i+1} \subset \mathcal{L}_i$ *for any* i*, hence* $V \in \mathcal{L}_n$ *for all* n*). Introducing the*

[10]The argument presented is based on the ideas used in the proof of the extended Kraft inequality in information theory [CT06].

notation $\sigma_1 = 0$ *and* $\sigma_3 = 1$, *for any* $v \in V$ *of length* $k = |v|$, *let* r_v *denote the real number with binary expansion* $0.v_1v_2\dots v_k$. *Then the interval* $I_v = [v, v + 1/2^k]$ *is the set of real numbers whose binary expansion begins with* $0.v_1v_2\dots v_k$ *(where* $v + 1/2^k$ *is interpreted here as* $0.v_1v_2\dots v_k111\dots$*). Since* L *contains all infinite binary strings, and each of them have a prefix in* V, $\bigcup_{v\in V} I_v$ *contains the whole unit interval* $[0, 1]$. *Therefore, denoting by* $|I|$ *the length of interval* I,

$$s_V = \sum_{v\in V} 4^{-\frac{1}{2}|v|} = \sum_{v\in V} 2^{-|v|} = \sum_{v\in V} |I_v| \geq \left|\bigcup_{v\in V} I_v\right| \geq |[0, 1]| = 1,$$

which readily shows that V_n *is minimal for* $\gamma = 1/2$ *and* $m_{1/2}(L) = 1$. *Using the above inequality, for* $\gamma < 1/2$ *and* $V \in \mathcal{L}_n$,

$$s_V = \sum_{v\in V} 4^{-\gamma|v|} = \sum_{v\in V} 2^{-|v|}4^{(1/2-\gamma)|v|} \geq 4^{(1/2-\gamma)n} \sum_{v\in V} 2^{-|v|} \geq 4^{(1/2-\gamma)n}$$

showing the minimality of V_n *again. Therefore,* $m_\gamma(L) = \infty$ *for* $\gamma < 1/2$. *In summary, we have*

$$\mathsf{dim}\,(L) = 1/2 \qquad \textit{and} \qquad m(L) = 1.$$

The Hausdorff dimension properly captures the notion that when generating a word from L, *every time half of the possibilities (relative to* Σ*) are excluded.*

Example 2.5. *Consider the set of variables* $\mathcal{V} = \{a\}$ *and the associated alphabet* $\Sigma = \{\varnothing, \{a\}\}$, *with number of symbols* $r = 2$. *We denote the symbol* $\{a\}$ *by* σ *for short. Let the language* L *be the language of the formula* $\mathsf{FG}a$, *which is the set of all* ω*-words ending in an infinite sequence of* σ*s,*

$$L = \Sigma^*\sigma^\omega\ .$$

We want to determine the γ*-dimensional outer measure of* L, *and–from it–its Hausdorff dimension and measure.*

Definition 2.34 requires the identification of a set $V \in \mathcal{L}_n$, *for a fixed* n, *such that* $s(n) = \sum_{v\in V} r^{-\gamma|v|}$ *is as small as possible. This sum* s *is obtained by attaching a measure* $r^{-\gamma|v|}$ *to each word* $v \in V$. *This quantity is non-negative and decreasing in* $|v|$, *thereby the sum can be*

minimized by choosing a V containing a few short words and several long words.

Consider a set

$$V_1 = \sigma^n \cup \varnothing\sigma^n \cup \Sigma\varnothing\sigma^{n+1} \cup \Sigma^2\varnothing\sigma^{n+2} \cup \cdots = \sigma^n \cup \bigcup_{i=0}^{\infty} \Sigma^i\varnothing\sigma^{n+i} \ .$$

This is constructed such that any word having i symbols before the last occurrence of $\varnothing$ has also i σs at the end. Notice that any ω-word of L has at least one prefix in V_1, and all elements of V_1 have length at least n, hence $V_1 \in \mathcal{L}_n$. The set V_1 contains one word of length n and 2^i words of length $n + 2i + 1$; hence the sum is

$$s_1(n) = 2^{-\gamma n} + \sum_{i=0}^{\infty} 2^i 2^{-\gamma(n+2i+1)} = \begin{cases} 2^{-\gamma n}\left(1 + \frac{2^{-\gamma}}{1-2^{1-2\gamma}}\right) & \text{for } \gamma > 1/2 \\ \infty & \text{for } \gamma \leq 1/2 \end{cases} .$$

For any fixed γ, we can find another set which has a lower sum. Consider the set

$$V_2 = \sigma^n \cup \varnothing\sigma^n \cup \Sigma\varnothing\sigma^{n+2} \cup \Sigma^2\varnothing\sigma^{n+4} \cup \cdots = \sigma^n \cup \bigcup_{i=0}^{\infty} \Sigma^i\varnothing\sigma^{n+2i} \ .$$

That is, a word $v \in V_2$ ends in twice as many σs as symbols it has before its last $\varnothing$. For the same reasons as V_1, also V_2 belongs to $\mathcal{L}_n$. V_2 contains one word of length n and 2^i words of length $n + 3i + 1$. Its corresponding sum is

$$s_2(n) = 2^{-\gamma n} + \sum_{i=0}^{\infty} 2^i 2^{-\gamma(n+3i+1)} = \begin{cases} 2^{-\gamma n}\left(1 + \frac{2^{-\gamma}}{1-2^{1-3\gamma}}\right) & \text{for } \gamma > 1/3 \\ \infty & \text{for } \gamma \leq 1/3 \end{cases} .$$

A sequence of sets $V_j \in \mathcal{L}_n$ can be constructed by following the same rule:

$$V_j = \sigma^n \cup \bigcup_{i=0}^{\infty} \Sigma^i\varnothing\sigma^{n+j\cdot i}$$

with sums

$$s_j(n) = 2^{-\gamma n} + \sum_{i=0}^{\infty} 2^i 2^{-\gamma(n+(j+1)\cdot i+1)} = \begin{cases} 2^{-\gamma n}\left(1 + \frac{2^{-\gamma}}{1-2^{1-(j+1)\gamma}}\right) & \text{for } \gamma > 1/(j+1) \\ \infty & \text{for } \gamma \leq 1/(j+1) \end{cases}.$$

Of course $\mathsf{inf}_{V\in\mathcal{L}_n} s(n) \leq s_j(n)$ *for every* $j \in \mathbb{N}$. *Taking the limit over* n *on both sides, we obtain that regardless of the parameter* $\gamma > 0$

$$m_\gamma(L) = \lim_{n\to\infty} \inf_{V\in\mathcal{L}_n} s(n) \leq \lim_{n\to\infty} s_j(n) = 0 \text{ for every } j \text{ and } \gamma > 0.$$

Since the γ*-dimensional Hausdorff measure is nonnegative, equality holds in the expression above. From this we conclude the Hausdorff dimension of* L *is*

$$\mathsf{dim}\,(L) = 0$$

and its Hausdorff measure is

$$m_{\mathsf{dim}(L)}(L) = \infty\,.$$

This example shows how the Hausdorff dimension is mainly affected by the infinite suffixes allowed in L. *If all* $w \in L$ *have a restriction on the allowed suffixes, it is possible to reduce* $s(n)$ *by extending the words in* $V \in \mathcal{L}_n$ *with longer suffixes fulfilling these restrictions.*

A related concept appearing in the literature which we will make use of in our weakness measure is entropy:

Definition 2.36 (Entropy [MS94])**.** *Given an* ω*-language* $L \subseteq \Sigma^\omega$ *over an alphabet of size* r*, the* entropy *of* L *is* $H(L) = \mathsf{lim\,sup}_{n\to\infty} \frac{1}{n} \mathsf{log}_r\, |\mathsf{pref}_n(L)|$*, where* $\mathsf{pref}_n(L)$ *is the set of all prefixes of length* n *of the* ω*-words in* L.

In other words, the entropy quantifies the growth rate of the number of n-long prefixes of the ω-words in L as n grows. Since the number of prefixes grows exponentially in L (its maximum value is r^n, in case L allows any symbol at any position in the ω-word), its logarithm

is considered.

Example 2.6. *Consider the language L of the formula $\mathbf{FG}a$ over the variables $\mathcal{V} = \{a\}$ as in Example 2.5, $L = \Sigma^*\{a\}^\omega$. The set of prefixes of length n is*

$$\mathsf{pref}_n(L) = \Sigma^n \ .$$

Applying Definition 2.36, the entropy can be calculated as

$$H(L) = \limsup_{n\to\infty} \frac{1}{n} \log_r |\mathsf{pref}_n(L)| = \limsup_{n\to\infty} \frac{1}{n} \log_2 2^n = 1 \ .$$

Notice that the entropy is greater than the Hausdorff dimension, since it is affected by the number of prefixes of L rather than by its suffixes. In contrast, the entropy for Example 2.4 is $1/2$, which is equal to the Hausdorff dimension.

It has been proved [MS94] that $\mathsf{dim}\,(L) \leq H(L)$ in general, and $\mathsf{dim}\,(L) = H(L)$ if L is a closed ω-language.

The entropy of a closed language can be computed on its Büchi automaton $\mathcal{B}$ if all states are accepting [MS94]. Given a Büchi automaton with all states accepting and its interpretation as a graph $\mathcal{G}$, [MS94] presents the following algorithm to compute the entropy:

1. Compute all the strongly connected components $\{C_j\}$ and their adjacency matrices $\{M[C_j]\}$;
2. For every $M[C_j]$ compute the Perron-Frobenius eigenvalue $\nu(M[C_j])$;[11]
3. Return the maximum $\log_r \nu(M[C_j])$ (over all SCCs).

This algorithm generalizes to the case where even if the language is not closed (and therefore some SCCs may not be accepting), provided that every SCC of the DBA is both reachable and

[11]Since A_i is the adjacency matrix of a strongly connected component, it is an irreducible non-negative matrix. Therefore, it has a real eigenvalue $\nu(A_i)$ (of multiplicity 1) which is at least as large as the absolute value of any other eigenvalue of A_i. This eigenvalue is called the Perron-Frobenius eigenvalue (see, e.g., [Sen06, Theorem 1.5]).

co-reachable. Algorithm 1 shows these steps in pseudocode. The procedure StronglyConnectedComponents, which returns the strongly connected components of a graph defined by a set of states Q and a transition function δ, can be implemented by using Tarjan's algorithm [Tar72], which runs in $O(|Q|+|E|)$ time, linear with the sum of the number of nodes and edges in the graph.

Algorithm 1: EntropyDBA

Input: A DBA $\mathcal{B} = (Q, \Sigma, q_0, \delta, F_{\mathcal{B}})$, with adjacency matrix M
Output: Entropy H of $L(\mathcal{B})$
$C \leftarrow$ StronglyConnectedComponents(Q, δ);
$i \leftarrow 0$;
foreach $C_j \in C$ **do**
  $\nu_j \leftarrow$ maximum eigenvalue of $M[C_j]$;
  $j \leftarrow j + 1$;
end
$\nu \leftarrow \mathsf{max}_j\, \nu_j$;
$H \leftarrow \mathsf{log}_r\, \nu$;
return H

To give an intuition of the relationship between the entropy and the maximum eigenvalues of the A_i, we recall that the power matrix A_i^n contains, as its (k, h) element, the number of paths of length n from state k to state h. If k and h belong to the same SCC, the growth rate of $A_i^n(k, h)$ is $\nu(A_i)^n$, that is, $\mathsf{lim}_{n\to\infty} \frac{1}{n} \mathsf{log}_r\, A_i^n(k, h) = \mathsf{log}_r\, \nu(A_i)^n$, by the Perron-Frobenius theory [LM95, Section 4.2]. Therefore, for every SCC,

$$\limsup_{n\to\infty} \frac{1}{n} \log_r |\mathsf{pref}_n(L)| = \limsup_{n\to\infty} \frac{1}{n} \log_r (\nu(A_i))^n = \log_r \nu(A_i)\ . \tag{2.4}$$

The correctness of the algorithm then follows by noticing that the growth rate of the total number of prefixes over all SCCs is dominated by the largest growth rate over the components.

The following proposition (proven in [LM95]) states the correctness of Algorithm 1:

Proposition 2.2. *Let $\mathcal{B}$ be a DBA of an ω-language L over an alphabet Σ of size $|\Sigma| = r$, with adjacency matrix M and strongly connected components $C = \{C_1, \ldots, C_k\}$ that are both*

reachable and co-reachable. The entropy of L is:

$$H(L) = \max_{C_j \in C} \log_r M[C_j] \ .$$

For the case when L is not closed, a general algorithm is presented in [Sta98, Sta15], which highlights the difference between the entropy and the Hausdorff dimension. The method, shown in Algorithm 2, is based on computing a Muller automaton $\mathcal{M}_L$ accepting L whose table of accepting state sets is T. For each accepting set $S' \in T$ and for each state $s \in S'$, consider the ω-language $C_{S'}$ consisting of all the infinite paths in $\mathcal{M}_L$ starting from s and visiting no states outside S'. It can be shown that this language is closed and its entropy $H(C_{S'})$ is independent of the choice of the starting state s [Sta98]. The Hausdorff dimension of L is then

$$\dim(L) = \max_{S' \in T} H(C_{S'}), \tag{2.5}$$

where the $H(C_{S'})$ can be computed as the maximum eigenvalue of the adjacency matrix of the subgraph induced by S'.

Algorithm 2: HausdorffDMA

Input: A DMA $\mathcal{M} = \langle Q, \Sigma, q_0, \delta, T \rangle$, with adjacency matrix M
Output: Hausdorff dimension dim
foreach $S' \in T$ **do**
  $\nu_{S'} \leftarrow$ maximum eigenvalue of $M[S']$;
end
$\nu \leftarrow \max_{S'} \nu_{S'}$;
$dim \leftarrow \log_r \nu$;
return dim

As a final remark, we note that the language $C_{S'}$ plays a central role in the definition of our weakness measure (see Section 6.2).

2.5 Summary

In this chapter, we revised the main definitions used throughout the thesis. We presented the syntax and semantics of Boolean, LTL and GR(1) formulae. We distinguished between strict and non-strict GR(1) formulae, specifying that in the rest of the thesis we deal with non-strict ones. We introduced the semantics of LTL (ω-words, ω-languages and ω-automata), and in particular we introduced *deterministic Büchi automata* (DBAs) and deterministic Muller automata (DMAs).

We have given the central notions of *realizability* and *unrealizability*, related to the existence of a solution to the reactive synthesis problem. When a specification is *unrealizable*, there is no solution to synthesis, because the environment can force the violation of some guarantee of the GR(1) specification. A function modelling this environment behaviour is called *counterstrategy*, and can be represented as a graph where for each state of the system, the next response of the environment is specified, and edges connect each state to the next reachable ones. Unrealizable specifications and counterstrategies are essential to the problem of assumptions refinement, that we will introduce in the next chapter.

Chapter 3

The Assumptions Refinement Problem

As discussed in Chapter 1, our thesis focuses on the problem of *insufficient* assumptions for realizability. As demonstrated in Examples 1.1-1.2, this insufficiency may be due to unknown assumptions at the time of specification, or to assumptions the specifier considered as implicit. *Assumptions refinement* is the process of identifying a sufficient set of additional assumptions that make the formula realizable. The search for such assumptions is driven by witnesses of undesirable environments, like the counterstrategies from Definition 2.24 or equivalent notions (for instance, JVTSes [KMR17]). Given a counterstrategy, an assumption refinement can be any formula the counterstrategy violates. The space of these formulae is intractably large, and in practice, heuristics are used to drive the search towards a solution. The heuristics define criteria to select subsets of candidate refinements to be checked during the search and prioritize these as the search goes on.

We have motivated through examples the problem of GR(1) assumptions refinement in Chapter 1. This chapter formally defines this problem as a search problem and introduces the use of heuristics. Section 3.1 contains the definitions of assumptions refinement in the strict and non-strict GR(1) contexts. Sections 3.2 presents the prototypical solution to the problem, counterstrategy-guided search. In this thesis, we focus on the challenges coming from the intractability of the search space, which are outlined in Section 3.3 and motivate the need for good heuristics to perform the search. Section 3.4 introduces the two macro-classes of heuristics

(*bias* and *sorting heuristics*) that are used in counterstrategy-guided approaches, and describes the existing heuristics and their limitations.

3.1 Problem Definition

Given an unrealizable GR(1) specification $\langle \mathcal{E}, \mathcal{S} \rangle$, the goal of assumptions refinement is to find a set of additional assumptions Ψ (called *refinement*) such that the new formula $\langle \mathcal{E} \cup \Psi, \mathcal{S} \rangle$ is realizable. As noted in Chapter 2, each of the assumptions in Ψ can be of one kind among initial conditions, invariants or fairness conditions.

3.1.1 Refinement in Strict GR(1) Formulae

We start by defining assumptions refinement in strict GR(1) formulae (Definition 2.13). The definition of strict GR(1) formulae only allows for one invariant $\rho^{\mathcal{E}}$ and one initial condition $\theta^{\mathcal{E}}$. The refinement Ψ may contain more invariants $\rho'_1, \ldots, \rho'_q$ and more initial conditions $\theta'_1, \ldots, \theta'_p$. In this case the refined assumptions $\mathcal{E} \cup \Psi$ is equivalent to the set of assumptions containing only one invariant consisting of the conjunction $\rho^{\mathcal{E}} \wedge \rho'_1 \wedge \cdots \wedge \rho'_q$ and one initial condition $\theta^{\mathcal{E}} \wedge \theta'_1 \wedge \cdots \wedge \theta'_p$, as formalized in the following definition.

Definition 3.1 (Assumptions refinement problem (strict GR(1)))**.** *Let $\langle \mathcal{E}, \mathcal{S} \rangle$ be an unrealizable GR(1) specification, with $\mathcal{E} = \{\theta^{\mathcal{E}}, \rho^{\mathcal{E}}, J_i^{\mathcal{E}}\}$ and $\mathcal{S} = \{\theta^{\mathcal{S}}, \rho^{\mathcal{S}}, J_j^{\mathcal{S}}\}$. The problem of* assumptions refinement *for the associated strict GR(1) formula is finding a set of assumptions*

$$\Psi := \{\theta'_1, \ldots, \theta'_p, \rho'_1, \ldots, \rho'_q, J'_1, \ldots, J'_r\}$$

such that, given $\theta' := \theta'_1 \wedge \cdots \wedge \theta'_p$, $\rho' := \rho'_1 \wedge \cdots \wedge \rho'_q$, the formula

$$\theta^{\mathcal{E}} \wedge \theta' \wedge \mathbf{G}(\rho^{\mathcal{E}} \wedge \rho') \wedge \bigwedge_{i=1..m} \mathbf{GF} J_i^{\mathcal{E}} \wedge \bigwedge_{h=1..r} \mathbf{GF} J'_h$$

is satisfiable, and

$$\begin{aligned}\phi :=&(\theta^{\mathcal{E}} \wedge \theta' \rightarrow \theta^{\mathcal{S}}) \\ &\wedge(\theta^{\mathcal{E}} \wedge \theta' \rightarrow \mathbf{G}(\mathbf{H}(\rho^{\mathcal{E}} \wedge \rho') \rightarrow \rho^{\mathcal{S}})) \\ &\wedge \left(\theta^{\mathcal{E}} \wedge \theta' \wedge \mathbf{G}(\rho^{\mathcal{E}} \wedge \rho') \rightarrow \left(\bigwedge_{i=1..m} \mathbf{GF} J_i^{\mathcal{E}} \wedge \bigwedge_{h=1..r} \mathbf{GF} J_h' \rightarrow \bigwedge_{j=1..n} \mathbf{GF} J_j^{\mathcal{S}}\right)\right)\end{aligned} \tag{3.1}$$

is realizable.

We call the assumptions Ψ *a* solution *to the refinement problem, or equivalently a* sufficient refinement *for strict realizability.*

In the above definition, we omit the superscript $\mathcal{E}$ for the primed assumptions in order to avoid cluttering the notation. Since throughout the thesis we will always talk about *assumptions* refinement and not guarantees, it is implied that all the primed formulae refer to the environment.

3.1.2 Refinement in Non-Strict GR(1) Formulae

Throughout the thesis, we deal with non-strict GR(1) formulae. The reason is twofold: *(i)* the majority of the literature on assumptions refinement [LDS11, LSSS14, AMT13, AMT15] deals with this definition of GR(1) formulae, therefore we position ourselves in the same setting for an easier comparison; (ii) the tools that implement strict GR(1) semantics on the backend (RATSY [BCG+10] and SPECTRA [MRS19]) have a user interface presenting assumptions and guarantees like in the non-strict definition (that is, as $\mathbf{G}$ formulae and $\mathbf{GF}$ formulae rather than Boolean formulae).

Definition 3.2. *Let* $\langle \mathcal{E}, \mathcal{S} \rangle$ *be an unrealizable GR(1) specification, with* $\mathcal{E} = \{\theta^{\mathcal{E}}, \rho^{\mathcal{E}}, J_i^{\mathcal{E}}\}$ *and* $\mathcal{S} = \{\theta^{\mathcal{S}}, \rho^{\mathcal{S}}, J_j^{\mathcal{S}}\}$*. The problem of* assumptions refinement *for non-strict GR(1) formulae is finding a set of assumptions*

$$\Psi := \{\theta_1', \ldots, \theta_p', \rho_1', \ldots, \rho_q', J_1', \ldots, J_r'\}$$

such that, given $\theta' := \theta'_1 \wedge \cdots \wedge \theta'_p$, $\rho' := \rho'_1 \wedge \cdots \wedge \rho'_q$, *the formula*

$$\theta^{\mathcal{E}} \wedge \theta' \wedge \rho^{\mathcal{E}} \wedge \rho' \wedge \bigwedge_{i=1..m} J_i^{\mathcal{E}} \wedge \bigwedge_{h=1..r} J'_h$$

is satisfiable, and

$$\phi := (\theta^{\mathcal{E}} \wedge \theta') \wedge (\rho^{\mathcal{E}} \wedge \rho') \wedge \left(\bigwedge_{i=1}^{m} J_i \wedge \bigwedge_{h=1}^{r} J'_h \right) \rightarrow \theta^{\mathcal{S}} \wedge \rho^{\mathcal{S}} \wedge \bigwedge_{j=1}^{n} J_j^{\mathcal{S}} \tag{3.2}$$

is realizable.

We call the assumptions Ψ *a* solution *to the refinement problem, or equivalently a* sufficient refinement *for non-strict realizability.*

A direct consequence of Proposition 2.1, any Ψ solving the problem of assumptions refinement in the strict setting also solves it in the non-strict one. Hence, the problem in Definition 3.2 is often solved by applying realizability checks in the strict setting. We come back to this in Section 3.2.

3.1.3 Weakest Assumptions Refinement

Notice that there may be multiple solutions to the assumptions refinement problem in either sense. However, not all solutions may be equally desirable from a practical perspective. In particular, it is easy to see that if some Ψ is a solution, then any $\Psi \cup \{\lambda\}$ is also a solution, for any additional assumption λ. In fact, any strategy satisfying (3.2) also satisfies

$$\phi_\lambda := (\theta^{\mathcal{E}} \wedge \theta') \wedge (\rho^{\mathcal{E}} \wedge \rho') \wedge \left(\bigwedge_{i=1}^{m} J_i \wedge \bigwedge_{h=1}^{r} J'_h \right) \wedge \lambda \rightarrow \theta^{\mathcal{S}} \wedge \rho^{\mathcal{S}} \wedge \bigwedge_{j=1}^{n} J_j^{\mathcal{S}} \ .$$

A common requirement [Ses15, LDS11, AMT13] is for solutions to be as weak as possible. The underlying idea is that, if the assumptions are too strict, one risks generating a controller that is incapable to operate in the real world, as it is only ensured to satisfy the guarantees in an overconstrained set of environments. In the following, given a set of assumptions $\Psi =$

$\{\psi_1, \ldots, \psi_n\}$, we denote by $\bigwedge \Psi := \psi_1 \wedge \cdots \wedge \psi_n$. The relative weakness of two refinements is typically defined as follows.

Definition 3.3 (Weakness of two refinements (by implication))**.** *Let $\langle \mathcal{E}, \mathcal{S} \rangle$ be a GR(1) specification. Let Ψ' and Ψ'' be two assumptions refinements. We say that Ψ' is* weaker *than Ψ'' (*by implication*) if $\bigwedge \mathcal{E} \wedge \bigwedge \Psi'' \rightarrow \bigwedge \mathcal{E} \wedge \bigwedge \Psi'$ is a valid formula (in the sense that it holds in every ω-word).*

In other words, Ψ' is weaker than Ψ'' if, when the assumptions of Ψ'' are conjoined with the assumptions present in the original specification, the resulting formula implies the one obtained by conjoining $\mathcal{E}$ with Ψ' instead.

Example 3.1. *In the request-grant protocol of Example 2.2, the refinement $\Psi' = \{\mathbf{GF}\neg\mathsf{cl}\}$ is weaker than $\Psi'' = \{\mathbf{G}(\mathsf{cl} \rightarrow \mathbf{X}\neg\mathsf{cl})\}$ since $\mathbf{GF}\neg\mathsf{req} \wedge \mathbf{G}(\mathsf{cl} \rightarrow \mathbf{X}\neg\mathsf{cl})$ implies $\mathbf{GF}\neg\mathsf{req} \wedge \mathbf{GF}\neg\mathsf{cl}$.*

This notion induces a partial ordering on the space of refinements. Indeed, given two sets of assumptions, it is possible that neither of the two implies the other.

When taking into account this notion of weakness, the problem of assumptions refinement is reformulated as follows.

Definition 3.4 (Weakest assumptions refinement problem)**.** *The problem of* weakest assumptions refinement *consists in identifying all the sufficient refinements Ψ such that there is no sufficient refinement $\Psi' \not\equiv \Psi$ such that Ψ' is weaker than Ψ.*

Chapter 4 deals with the problem formulated in Definition 3.2. Chapters 5-7 address the problem in Definition 3.4. Specifically, in Chapter 6 the definition of weakness is reformulated, through the use of a quantitative weakness measure that allows to compare also pairs of formulae that are incomparable via implication.

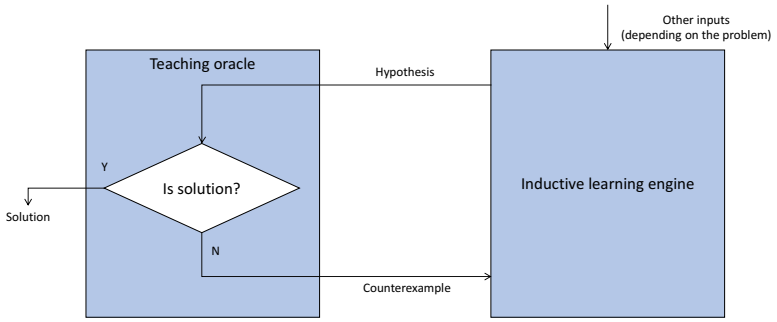


Figure 3.1: Counterexample-guided inductive synthesis

3.2 Counterstrategy-Guided Search for Assumptions Refinement

As explained in Chapter 2, a GR(1) formula is realizable if and only if there are no counterstrategies to the GR(1) game. Hence, a natural way to find a sufficient refinement to an unrealizable formula is to compute one or more counterstrategies and to produce assumptions that are violated by at least one of the counterstrategies. Using witnesses of the violation of some desired property (such as realizability) is a common practice for inductive procedures involving logic-based hypotheses. It is at the core, for example, of SMT solving [GHN+04], program synthesis [SL13, ABJ+13], abstraction refinement [Cla00], inference of LTL requirements [AKRU13].

These procedures are collectively known as *counterexample-guided inductive synthesis* (*CEGIS*). The term, introduced in [SL08] specifically for program synthesis, has been extended to the other contexts listed above [Ses15, JS17]. CEGIS has its theoretical foundations in the L^* algorithm for learning regular languages from queries and counterexamples [Ang87]. It is an iterative approach consisting of two components: *(i)* an *inductive learning engine* that produces candidate solutions to the problem, and *(ii)* a *teaching oracle* that verifies whether the candidate solutions solve the problem and, if not, produces a counterexample to help the learning engine to reach the solution. The procedure is illustrated in Figure 3.1.

In the context of GR(1) assumptions refinement, CEGIS specializes into a counterstrategy-guided search algorithm [LDS11, AMT13], depicted in Figure 3.2.

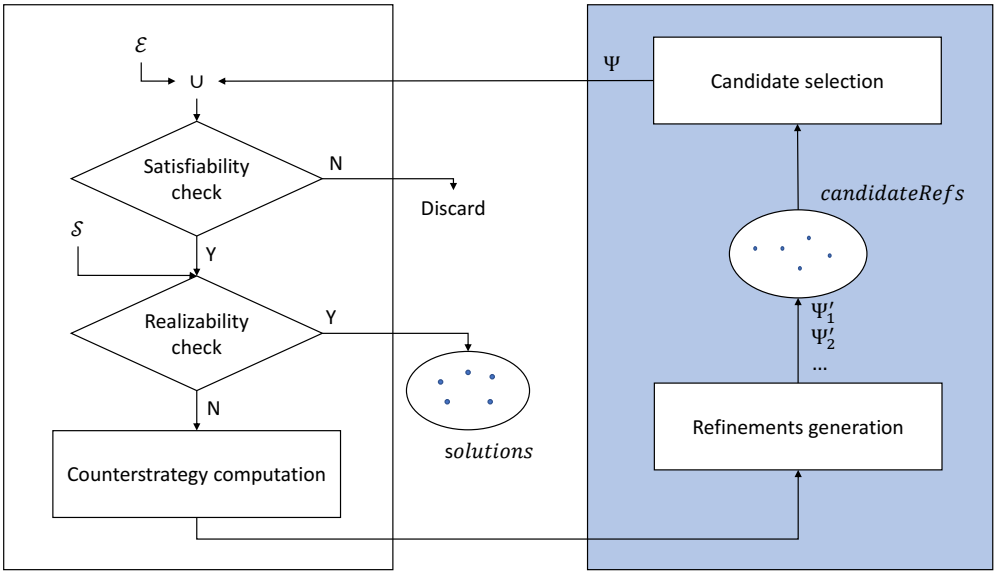


Figure 3.2: Counterstrategy-guided assumptions refinement. The components in the shaded area implement the heuristics that are the focus of this thesis.

The oracle (depicted on the left-hand side of Figure 3.2) receives a candidate refinement Ψ and returns whether it is a solution, and a counterstrategy in case it is not. Figure 3.2 shows the oracle's three phases:

1. *Satisfiability check.* The oracle checks whether the refined assumptions $\mathcal{E} \cup \Psi$ are satisfiable in conjunction, that is, whether $\bigwedge \mathcal{E} \wedge \bigwedge \Psi$ is satisfiable. If not, Ψ is discarded.

2. *Realizability check.* The refined specification $\langle \mathcal{E} \cup \Psi, \mathcal{S} \rangle$ is checked for realizability (typically in the strict sense, see Section 2.2.2). If it is realizable, the refinement Ψ is a solution to the problem in Definition 3.1 (and hence also to the problem in Definition 3.2). Therefore, it is added to the *solutions* set.

3. *Counterstrategy computation.* If the refined specification is not realizable, a counterstrategy is computed and fed to the inductive learner for inferring new candidate refinements.

The inductive learner receives counterstrategies as its input and produces a candidate refinement eliminating the counterstrategies by applying a set of *heuristics* that drive its exploration (these heuristics are the main focus of our thesis). The learner keeps a set of candidate refinements, named candidateRefs in the figure, and at every iteration, it performs two steps:

Algorithm 3: CounterstrategyGuidedRefinement procedure.

Data: $\mathcal{E}$, assumptions
Data: $\mathcal{S}$, guarantees
Result: solutions $= \{\Psi_i\}$, set of alternative refinements such that $\langle \mathcal{E} \cup \Psi_i, \mathcal{S} \rangle$ is realizable for every i

```
1  solutions ← ∅;
2  candidateRefs ← {∅};
3  repeat
4  |  Ψ ← candidateRefs.extract();
5  |  E_ref ← E ∪ Ψ;
6  |  if IsSatisfiable(E_ref) and ¬IsRealizable(⟨E_ref, S⟩) then
7  |  |  c ← ComputeCounterstrategy(E_ref, S);
8  |  |  altRefs ← GenerateRefinements(E, S, Ψ, c);
9  |  |  foreach Ψ_i ∈ altRefs do
10 |  |  |  candidateRefs.add(Ψ_i);
11 |  |  end
12 |  else if IsSatisfiable(E_ref) then
13 |  |  solutions.add(Ψ);
14 until candidateRefs = ∅;
15 return solutions;
```

1. *Refinements generation.* The learner produces one or more alternative refinements eliminating the observed counterstrategy and adds it into candidateRefs.

2. *Candidate selection.* One candidate at a time is selected from candidateRefs and sent to the oracle to be checked.

Algorithm 3 shows the same steps in pseudocode. It generalizes the algorithms used in every existing approach to find multiple solutions to the refinement problem [LDS11, AMT13, LSSS14, MRS19]. The loop in lines 3-14 is referred to as the *counterstrategy-guided loop* throughout the thesis. We call *candidate refinements* all the refinements Ψ generated throughout the iterations of the counterstrategy-guided loop (they are candidate to be solutions to the problem).

The refinement tree. In several of the existing approaches to assumptions refinement [LDS11, AMT13, CA17, MRS19], the candidate refinements generated by the counterstrategy-guided loop are organized in a tree structure.[1]

[1]The only exception is the approach we present in Chapter 7.

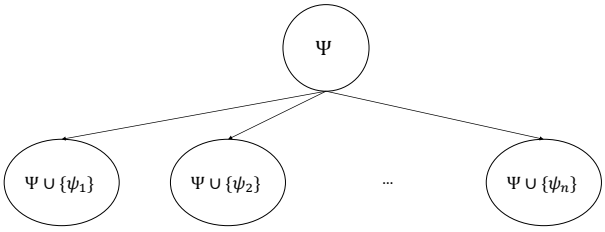


Figure 3.3: Stump containing the input refinement Ψ and the generated refinements $\Psi_i = \Psi \cup \psi_i$ in a single iteration of the counterstrategy-guided loop

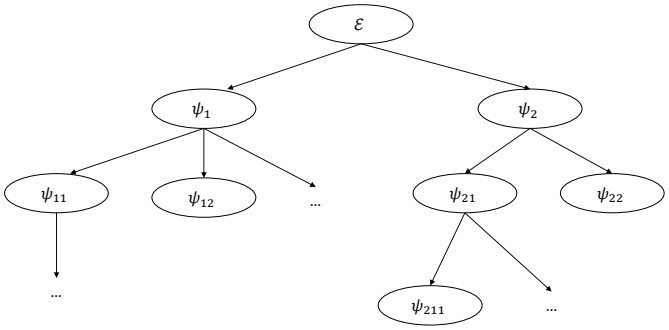


Figure 3.4: A refinement tree. Each node is the refinement tested in a single iteration of Algorithm 3. For conciseness, it is labelled with the assumption ψ_i added in the latest iteration of the loop, so that the entire refinement Ψ can be read on the path from the root to the node. The solutions are $\{\psi_1, \psi_{12}\}, \{\psi_2, \psi_{21}.\psi_{211}\}$, and $\{\psi_2, \psi_{22}\}$.

Given a candidate refinement Ψ, the function **GenerateRefinements** in Algorithm 3 line 8 produces new candidate refinements Ψ_i each one obtained by conjoining an additional assumption ψ_i to Ψ. Thus, all these refinements share the assumptions in Ψ and only differ by the latest assumption ψ_i. Therefore, they can be organized in a stump structure (that is, a one-level tree) where the parent node is labelled with the refinement Ψ and the children with $\Psi \cup \{\psi_i\}$, as shown in Figure 3.3. By using this representation for all the iterations, we obtain a tree of refinements, where the root corresponds to the original assumptions $\mathcal{E}$, the internal nodes are insufficient candidate refinements, and the leaves are the solutions found by the loop. We will refer to this structure as the *refinement tree* throughout the thesis. An example of a refinement tree is illustrated in Figure 3.4.

With this representation, Algorithm 3 implements a classical tree search algorithm [RN09]. Therefore, we often refer to the steps in each iteration with the common terminology used in AI to refer to the steps of tree search. In particular, we refer to the satisfiability and realizability

checks of line 6 as the *exploration* of node Ψ,[2] as this is the step where the algorithm checks whether Ψ is a solution; and to lines 7-11 as the *expansion* of node Ψ, as this is the step where new nodes are generated.

3.3 Assumptions Refinement Challenges

Approaching the refinement problem as a search guided by counterstrategies poses a series of challenges tackled by the heuristics we study in this thesis.

Size of the Search Space. Algorithm 3 performs a search over a combinatorially large space, where the solutions may be sparse. This space contains: *(i)* all the non-equivalent initial conditions $\theta^{\mathcal{E}}$ that can be constructed over the variables in $\mathcal{V}$; *(ii)* all the non-equivalent invariants $\rho^{\mathcal{E}}$ that can be constructed over the variables in $\mathcal{V} \cup \mathbf{X}\mathcal{X}$; *(iii)* all the non-equivalent fairness conditions $J^{\mathcal{E}}$ that can be constructed over the variables in $\mathcal{V}$; *(iv)* any combination of an initial condition, an invariant, and any number of fairness conditions.

Given a set of variables $\mathcal{V}$, there are $2^{2^{|\mathcal{V}|}}$ semantically non-equivalent Boolean formulae over these variables (where $|\mathcal{V}|$ denotes the number of elements in $\mathcal{V}$); hence, the search space comprises $2^{2^{|\mathcal{V}|}}$ initial conditions and fairness conditions, and $2^{2^{|\mathcal{V}\cup\mathcal{X}|}}$ invariants. In total, the number of combinations of such assumptions that can form a candidate refinement is therefore upper-bounded by

$$2^{2^{|\mathcal{V}|}} \cdot 2^{2^{|\mathcal{V}\cup\mathcal{X}|}} \cdot \sum_{i=0}^{2^{2^{|\mathcal{V}|}}} \binom{2^{2^{|\mathcal{V}|}}}{i}$$

To make the computations feasible, and be able to hit solutions, the function **GenerateRefinements** of Algorithm 3 needs to use a set of heuristics that select only a small subset of refinements. Among the biases proposed before our work, the *variable-choice* bias, described in Section 3.4.2, is the latest, and we choose it as a baseline for our experiments in the following chapters.

[2]This is not to be confused with the meaning of *exploration* in the expression *exploration vs. exploitation*, popular in reinforcement learning. The terminology used here refers to [RN09].

Quality of Refinements. Given the size of the search space, the search needs to focus early on the most promising refinements in order to obtain good solutions; moreover, the best solutions can easily be missed. Therefore, Algorithm 3 needs heuristics to select the best candidate refinements to explore at each iteration.

A criterion of quality that is generally acknowledged as desirable, independently of the application domains, is *weakness* [AMT13, Ses15, MRS19], as pointed out in Definition 3.4. As discussed earlier in this thesis, weaker assumptions are more desirable given that the controllers implemented by using weaker assumptions satisfy their guarantees in a wider set of environments.

A second criterion is *interpretability* [MRS19]: the solutions should be understandable by a human reader in order to be applicable in practical contexts. This property is related with the syntactical structure of the refinement: if the refinement is lengthy, or contains operators nested at arbitrary levels, this may result in uninterpretable specifications. The heuristics discussed here and in Chapter 4 keep the operator nesting under control by using fixed templates for the assumptions in the explored refinements. In Chapter 7, we propose a heuristic that keeps the number of assumptions in a refinement as small as possible, thus enhancing interpretability.

One last quality criterion mentioned in the literature is the *usefulness* of the refinement [MRS19]. A refinement is useful if its assumptions reflect the behaviour of the real environment where the controller is expected to operate. We do not focus on this criterion in this thesis. In the current state of the art, ensuring a refinement is useful is a post-processing phase of the solutions: a sample of behaviours (called *positive traces*) of the real environment is given in input and only the solutions that are satisfied by all the positive traces are returned by the search [LDS11, Li14]. The same may be applied to the approaches presented in this thesis.

Quality of the Search. An implicit requirement for the search algorithm is that it should be effective in finding solutions: ideally, it should only explore refinements that are solutions, or candidates that are likely to lead to a solution in as few steps as possible. We say that a search algorithm is *more effective* than another if it finds more solutions given the same budget

of refinements.

Moreover, to be of effective help to specification engineers, the search should be *fully automated* and should need as little prior knowledge as possible by the user. In other words, it should provide engineers with the knowledge they lack about the assumptions needed in order to reach the guarantees. This is the opposite viewpoint to the one expressed in [Li14], where instead the importance of human interaction with specification mining techniques is emphasized. In our case, we deal with the situation where the human lacks the knowledge needed to drive the search and the algorithm itself leverages the intrinsic structure of the specification in order to fill this gap of knowledge.

3.4 Heuristics Classification

The counterstrategy-guided approach illustrated in Figure 3.2 and in Algorithm 3 makes use of heuristics to drive the search towards regions of the search space containing more solutions, and to find the best refinements first.

More specifically, two components of the algorithm are affected by the choice of the heuristics:

- the *Refinement generation* process, where a subset of the possible refinements eliminating the observed counterstrategies is selected. We call these heuristics *bias heuristics*, or simply *biases*, as they play the role of the inductive bias common to machine learning techniques [Mit97]; and
- the extraction of Ψ from the set of candidate refinements in line 4. This set is sorted according to one or more heuristics, in order to prefer candidate refinements that are more likely to produce better solutions. We call these heuristics *selection* or *sorting heuristics*.

Past research [LDS11, AMT13] has focused on bias heuristics based on user-provided templates of formulae and/or parameters for the search. This has the effect of addressing the interpretability of the solutions, ; however, it neglects the other quality metrics. In this thesis

we propose novel metrics that address some of the challenges outlined above. In particular, we propose two bias heuristics: *interpolation* (extending our work in [Cav16]), which addresses the effectiveness of the search and the automation of the search (Chapter 4); and *minimality*, which deals with interpretability and (to some extent) weakness (Chapter 7). In addition, we propose a sorting heuristic based on our definition of a weakness measure (Chapters 5-6) to deal with the weakness of the solutions.

In the following subsections, we describe two bias heuristics that have been proposed in [LDS11, AMT13] respectively for comparison purposes. These two approaches are the only ones existing prior to our work that apply to the exact same setting given in Definition 3.2 and use counterstrategies as defined in Definition 2.24. The latter of the two (Section 3.4.2) is the state-of-the-art approach that we use as a baseline throughout the thesis. Other techniques that use a different formalization of specifications (like automata-based instead of GR(1) formulae), or of counterstrategies (like JVTSes), are surveyed in the related work (Chapter 8).

In Section 3.4.3, we describe the only sorting heuristic used in existing approaches, namely first-in-first-out (FIFO) [LDS11, AMT13].

3.4.1 Syntax-Based Bias Heuristic

The heuristic proposed in [LDS11] and applied in [LSSS14] is a bias heuristic. We name this heuristic *syntax-based*. It requires the user to provide one or more syntactic templates of GR(1) assumptions that will be instantiated when generating new assumptions.[3] The templates are GR(1) assumptions in which literals are replaced with placeholders '?'. For instance, a template for an invariant is $\mathbf{G}(?\wedge?)$. When instantiated, each placeholder is replaced with a single positive or negative literal of a variable of the model (where a *literal* is defined as usual for Boolean logic, that is either a Boolean variable or its negation). For instance, if the set of model variables is

[3]Incidentally, this work also requires a set of desirable environment executions to be provided. However, these traces are not used in the generation procedure. They are only used *a posteriori* to discard refinements not allowing some of them. Such a feature can be implemented seamlessly in any other approach presented in this thesis and is therefore not discussed throughout the thesis. This aspect is related to the notion of usefulness described in Section 3.3.

Algorithm 4: SyntaxBasedCounterstrategyGuidedRefinement procedure.

Data: $\mathcal{E}$, assumptions
Data: $\mathcal{S}$, guarantees
1 **Data:** T, set of assumption templates
Result: *solutions* $= \{\Psi_i\}$, set of alternative refinements such that $\langle \phi^{\mathcal{E}} \cup \Psi_i, \phi^{\mathcal{S}} \rangle$ is realizable for every i
2 candidateRefs $\leftarrow \{\varnothing\}$;
3 $I \leftarrow$ **InstantiateTemplates**(T, $\mathcal{V}$);
4 **repeat**
5 $\quad \Psi \leftarrow$ candidateRefs.extract();
6 $\quad \mathcal{E}_{ref} \leftarrow \mathcal{E} \cup \Psi$;
7 $\quad$ **if** ***IsSatisfiable***$(\mathcal{E}_{ref})$ *and* $\neg$***IsRealizable***$(\langle \mathcal{E}_{ref}, \mathcal{S} \rangle)$ **then**
8 $\quad\quad c \leftarrow$ **ComputeCounterstrategy**$(\mathcal{E}_{ref}, \mathcal{S})$;
9 $\quad\quad$ *altRefs* $\leftarrow$ **GenerateRefinements**$(\mathcal{E}, \mathcal{S}, \Psi, c, I)$;
10 $\quad\quad$ **foreach** $\Psi_i \in$ *altRefs* **do**
11 $\quad\quad\quad$ candidateRefs.add(Ψ_i);
12 $\quad\quad$ **end**
13 $\quad$ **else if** ***IsSatisfiable***$(\mathcal{E}_{ref})$ **then**
14 $\quad\quad$ *solutions*.add(Ψ);
15 **until** candidateRefs $= \varnothing$;
16 **return** *solutions*;

$\mathcal{V} = \{a, b\}$, possible instantiations of this template are $\mathbf{G}(a \wedge a)$, $\mathbf{G}(a \wedge \neg a)$, $\mathbf{G}(a \wedge b)$, $\mathbf{G}(a \wedge \neg b)$, and so on.

This approach requires a small modification to the general refinement procedure presented in Algorithm 3: the templates T are instantiated with all the possible combinations of literals constructed with the variables in $\mathcal{V}$, forming the set I; these instantiations are then passed to the function **GenerateRefinements**. This is shown in Algorithm 4.

The function **GenerateRefinements** just loops through the assumptions in I to select the ones being violated by the counterstrategy c. When this happens, the assumption is concatenated with the current refinement Ψ and the resulting refinement is added to the set altRefs of generated refinements. Algorithm 5 shows this.

The syntax-based bias has the advantage of giving the user full control of the interpretability of the solutions [Li14]: the structure and length of the assumptions are fully determined by the template provided by the user.

Algorithm 5: SyntaxBasedGenerateRefinements procedure.

Data: $\mathcal{E}$, assumptions
Data: $\mathcal{S}$, guarantees
Data: Ψ, current refinement
Data: c, counterstrategy for the formula $\langle \mathcal{E} \cup \Psi, \mathcal{S} \rangle$
Data: I, set of candidate assumptions instantiated from the templates
Result: altRefs, set of alternative refinements eliminating c

```
1 altRefs ← ∅;
2 foreach α ∈ I do
3 |   if c ⊭ α then
4 |   |   altRefs.add(Ψ ∪ {α})
5 |   end
6 end
```

The complexity of the search is also under the control of the user. At every iteration of Algorithm 5, the number of refinements generated is at most the number of instantiations in I. Since every placeholder can be replaced independently of each other with any of the $2|\mathcal{V}|$ literals of the variables in $\mathcal{V}$, this number is equal to

$$\sum_{t \in T} (2|\mathcal{V}|)^{p_t} \ ,$$

where p_t is the number of placeholders in the template t. By reducing the number of templates and placeholders per template, the user reduces drastically the complexity of the search.

The main disadvantage of the syntax-based bias heuristic is that the solution found depends on the user's prior knowledge of the problem. In the context of the weakest assumptions refinement problem (Definition 3.4), weakest solutions can be missed if none of them can be instantiated from the templates in T. Even more drastically, if none of the solutions fits the template, the approach does not return any solution and the search can only terminate with the false assumption. We show examples of these two cases.[4]

Example 3.2. *Consider the request-grant protocol of Example 2.2. A weakest solution is* $\mathsf{GF}\neg\mathsf{cl}$. *However, if the template* $\mathsf{G}?$ *is provided to the algorithm, the only solution that can be induced*

[4]These examples are clearly simplistic for illustrative purposes. They serve the goal of showing that a template may be poorly chosen for some problem. If this can happen in simple examples, there is higher chance of this happening in larger specifications with of variables (like the AMBA-AHB protocol presented in Section 3.5).

is $\mathbf{G}\neg\mathsf{cl}$*, which is stronger.*

Example 3.3. *Consider a more complex version of the request-grant protocol, with assumption*

$$J^{\mathcal{E}} = \mathbf{GF}\neg\mathsf{req}$$

and guarantees

$$\rho_1^{\mathcal{S}} = \mathbf{G}(\mathsf{req} \leftrightarrow \mathsf{gr})$$

$$\rho_2^{\mathcal{S}} = \mathbf{G}(\mathsf{cl} \rightarrow \neg\mathsf{val})$$

$$J^{\mathcal{S}} = \mathbf{GF}(\mathsf{gr} \wedge \mathsf{val}) \ .$$

The weakest solution to this problem is $\mathbf{GF}(\mathsf{req} \wedge \neg\mathsf{cl})$*. However, if the template* $\mathbf{GF}?$ *is provided to the algorithm, no solutions can be found. In fact, the only assumptions that can be induced with this template are* $\mathbf{GF}\mathsf{req}$ *and* $\mathbf{GF}\neg\mathsf{cl}$*. None of the two is sufficient to achieve realizability. Indeed, the counterstrategy that alternates the input valuations* $\mathsf{req} = \mathsf{true}, \mathsf{cl} = \mathsf{true}$ *and* $\mathsf{req} = \mathsf{false}, \mathsf{cl} = \mathsf{false}$ *times satisfies both these assumptions and prevents the guarantees from being satisfied.*

The tight dependence on the templates is the reason why we do not compare our heuristics to the syntax-based bias in the rest of the thesis. Indeed, it is possible to make this heuristic perform outstandingly well or outstandingly poorly in the comparison by selecting the templates appropriately. Moreover, unlike the bias presented in the next subsection, it is not straightforward how the choice of the template could be randomized. Indeed, the space of possible templates is infinite, and varies along several dimensions (number of operators, type of operators, nesting levels between operators).

3.4.2 Variable-Choice Bias Heuristic

To overcome the limitations imposed by user-defined templates, the work by Alur et al. [AMT13] proposes a different kind of bias. Here, the templates are fixed to formulae parameterized by

state valuations drawn from counterstrategies, allowing more flexibility on how assumptions are generated. However, since this potentially expands the search space to an intractable size, the set of variables to use when instantiating the templates is fixed by the user. We call this approach *variable-choice bias* (*varbias* for short), as the generated assumptions depend on the choice of variables.

More specifically, the templates used in this approach are three. In the following, the symbol $[\![s]\!]_c$ denotes a placeholder for a Boolean formula that is true in the state s of the counterstrategy c.

1. $\mathbf{G} \bigwedge_{s \in S'} \neg [\![s]\!]_c$, where S' is a subset of states in the counterstrategy;
2. $\mathbf{GF} \bigwedge_{s \in S^{loop}} \neg [\![s]\!]_c$, where S^{loop} is a set of states forming a loop in c;
3. $\mathbf{G} \left(\neg [\![s]\!]_c \vee \bigvee_{s' \in S_s^{next}} \neg \mathbf{X} [\![s']\!]_c \right)$, where S_s^{next} the set of all successor states of s in c.

In order for these templates to be instantiated, first the procedure **GenerateRefinements** needs to work out suitable sets S', S^{loop}, and S_s^{next} in the counterstrategy c.

In principle, any subset S' of states in c such that at least one is reachable from the initial state can be used to obtain an assumption violated by c. Indeed, if $s \in S'$ and s_0 is the initial state of c, the run $s_0 \dots s' \dots$ violates the formula $\mathbf{G} \bigwedge_{s \in S'} \neg [\![s]\!]_c$. The procedure includes a parameter b denoting the maximum size of the subsets to include in the instantiation.

In order to identify S^{loop}, one needs to find all [5] the strongly connected components of c. A *strongly connected component* (*SCC*) of a graph is a subset of states such that any state is reachable from any other state in the subset. The most common solution to finding SCCs is Tarjan's algorithm [Tar72], which runs in linear time with the number of states and edges of the graph. We will analyze SCCs in more detail in Chapter 5, when we will see them applied to the computation of a quantitative weakness measure.

Finding S_s^{next} for every state s is straightforward, since it is just the set of all successor states of s.

[5]

Algorithm 6: InstantiateVarbiasTemplate procedure.

Data: s, counterstrategy state
Data: vars, set of variables to use in the instantiation
Result: $[\![s]\!]_c$, instantiation of the template in s using the variables in vars

```
1 [[s]]_c ← true ;
2 foreach v ∈ vars do
3 |   if s ⊨ v then
4 |   |   [[s]]_c ← [[s]]_c ∧ v ;
5 |   else
6 |   |   [[s]]_c ← [[s]]_c ∧ ¬v ;
7 end
8 return [[s]]_c
```

Once all the templates are generated, the following step is instantiating all the placeholders with Boolean formulae. Consider the placeholder $[\![s]\!]_c$ for some state s of c, and a subset of variables $\mathcal{V}' \subseteq \mathcal{V}$ provided by the user. Intuitively, the way of constructing the Boolean formula for s is forming the conjunction of the positive literals of every variable of $\mathcal{V}'$ that is true in s and the negative literals of every variable of $\mathcal{V}'$ that is false in s.

The procedure is summarized in Algorithm 7. The procedure makes use of a helper function **InstantiateVarbiasTempate** (Algorithm 6). This function takes in input a state s of the counterstrategy graph and a set of variables vars, and returns a conjunctive formula containing the positive literals of the variables in vars that are true in s, and the negative literals of the variables in vars that are false in s. In other words, this function instantiates the template $[\![s]\!]_c$ on state s.

Algorithm 7 calls this function several to construct refinements. Lines 2-9 construct refinements of type 1 (invariants without the **X** operator), one for every set of states S' defined as above. Lines 10-17 produce refinements of type 2 (fairness conditions), one for every SCC S^{loop}. Finally, lines 18-26 produce refinements of type 3 (invariants with the **X** operator) in the way described above.

This approach has the advantage of exploring more refinements than the syntax-based one, but shares the disadvantage of depending on the user's prior knowledge: if the variable set $\mathcal{V}'$ is badly chosen, no solution is found or weakest solutions are missed.

Algorithm 7: VarBiasGenerateRefinements procedure.

Data: $\mathcal{E}$, assumptions
Data: $\mathcal{S}$, guarantees
Data: Ψ, current refinement
Data: $c = (S, s_0, E)$, counterstrategy for the formula $\langle \mathcal{E} \cup \Psi, \mathcal{S} \rangle$
Data: b, maximum size of subset of states in $\mathbf{G} \bigwedge_{s \in S'} \neg [\![s]\!]_c$ templates
Data: $\mathsf{vars_G}, \mathsf{vars_{GF}}, \mathsf{vars_{GX}}$, sets of variables to use in the three templates
Result: altRefs, set of alternative refinements eliminating c

```
1  altRefs ← ∅ ;
2  foreach S' ⊆ S such that |S'| ≤ b do
3  |  refinement ← true ;
4  |  foreach s ∈ S' do
5  |  |  [[s]]_c ← InstantiateVarBiasTemplate(s, vars_G) ;
6  |  |  refinement ← refinement ∧ ¬[[s]]_c ;
7  |  end
8  |  altRefs ← altRefs ∪ {refinement} ;
9  end
10 foreach S^loop ∈ GetSCCs(c) do
11 |  refinement ← true ;
12 |  foreach s ∈ S^loop do
13 |  |  [[s]]_c ← InstantiateVarBiasTemplate(s, vars_GF) ;
14 |  |  refinement ← refinement ∧ ¬[[s]]_c ;
15 |  end
16 |  altRefs ← altRefs ∪ {refinement} ;
17 end
18 foreach s ∈ S do
19 |  S_s^next ← set of successors of s in c ;
20 |  [[s]]_c ← InstantiateVarBiasTemplate(s, vars_GX) ;
21 |  refinement ← ¬[[s]]_c ;
22 |  foreach s' ∈ S_s^next do
23 |  |  [[s']]_c ← InstantiateVarBiasTemplate(s', vars_GX) ;
24 |  |  refinement ← refinement ∨ ¬X[[s']]_c ;
25 |  end
26 |  altRefs ← altRefs ∪ {refinement} ;
27 end
```

3.4.3 FIFO Selection Heuristic

In all the approaches mentioned above, no attention is given to the selection heuristic. The set candidateRefs is generally implemented as a queue, and the selection of candidate refinements occurs in a first-in-first-out (FIFO) fashion. This has the effect of selecting refinements of smaller size before the ones of greater size, thus identifying simpler solutions earlier. On the refinement tree illustrated in Figure 3.4, this translates into a *breadth-first search* (*BFS*), where

the refinements in upper levels of the tree are all explored before the ones in lower levels of the tree.

This criterion, however, does not allow to express a preference between refinements with the same size, nor does it capture entirely the notion of weakness: if a refinement of size 2 is weaker than a refinement of size 1, the latter will be explored first. This is a problem when the search has a limited budget of time and/or number of candidates to explore: in fact, a weakest refinement may be missed because pushed too late in the search. In Chapter 6, we introduce a new quantitative measure of weakness that allows to sort all the candidates consistently with their relative weakness as defined in Definition 3.3.

3.5 Case studies

In the subsequent chapters describe an extensive experimental campaign on popular benchmarks from the relevant literature. These include embedded software engineering case studies, in particular on-chip communication protocols [LDS11, AMT13, PMAB19], and robot controllers [MR15, KMR17, MRS19]. This benchmark is used in later chapters to assess all of our heuristics.

The *Advanced High-performance Bus* (*AHB*) is part of the *Advanced Microcontroller Bus Architecture* (*AMBA*) specification. It is an open-source communication protocol for on-chip devices through a shared bus. Devices are divided into *masters*, which initiate a communication, and *slaves*, which respond to requests. Multiple masters can request the bus simultaneously, but only one at a time can communicate through it. Masters and slaves constitute the environment, while the system is the bus arbiter implementing the protocol. The specification of the AHB protocol is a GR(1) description of the protocol developed by ARM and summarized in [BJP+12]. The original specification is realizable: by removing one or more assumptions one obtains an unrealizable specification [AMT13, LDS11, CRST08]. We consider specifications for two, four and eight masters (AMBA02, AMBA04, AMBA08 respectively). To the best of our knowledge, ours is the first work to deal with the specification for 8 masters (the most recent

paper on the topic [MRS19] deals with at most 3 masters).

We tested our approach on additional case studies part of the SYNTECH project, run by the Universities of Leicester and Tel Aviv with the goal of implementing tools and languages to support software engineers the specification and synthesis of reactive systems [KMR17, FMR20, MR15, MR15, MRS19]. The case studies, presented in [KMR17], include 15 variants of:

- a robot sorting Lego pieces by color (*ColorSort*);
- a mobile robot of humanoid shape (*Humanoid*);
- a robot with self-balancing capabilities (*Gyro*);
- a self-parking car (*PCar*).

The authors of the mentioned work consider 15 unrealizable versions of these three specifications, with slight changes in the initial assumptions and guarantees, and check the computation time of JVTSes on those. We refer to these case studies as the *original SYNTECH case studies.* In [MRS19] they also extend the benchmark with 136 additional variants of the same case studies. We refer to these as the *additional SYNTECH case studies.*

More details on the size of the benchmarks case studies are given in later chapters along with the presentation of the experiments executed on each of them (see Sections 4.8, 6.4, 7.5).

3.6 Summary

In this chapter, we have defined the problem of GR(1) assumptions refinement, and its specialization into the problem of weakest assumptions refinement. We have presented the typical counterstrategy-guided class of approaches that at the state of the art is the most effective way of refining GR(1) assumptions. Inspired by the popular counterexample-guided inductive synthesis approaches used to discover logic-based specifications, a counterstrategy-guided approach finds additional assumptions to a GR(1) specifications by deriving them from counterstrategies

(defined in Section 2.2.3). Counterstrategy-guided approaches alternate between counterstrategy computations and candidate refinement generations until realizability is reached, resulting in a looping search algorithm known as the *counterstrategy-guided loop*.

The main challenges relevant to the counterstrategy-guided loop lie in how to select the best candidates that eliminate all the observed counterstrategies at every step of the loop. This results in the use of *heuristics* that drive the search towards appropriate regions of the search space where to find solutions. In this chapter, we have outlined the main quality criteria for the search that depend on the choice of the heuristics, namely *effectiveness* (the probability of hitting a solution among the searched candidates), and weakness of the solutions (that is, how constraining the solutions are on the environment). We have proposed a categorization of the heuristics into two macro-classes, *bias heuristics* (the heuristic used to infer refinements from counterstrategies), and *sorting heuristics* (the heuristics used to sort the candidates explored during the search).

Finally, we have surveyed the existing bias and sorting heuristics from the literature that serve as a baseline for the heuristics we are going to propose in the remainder of this thesis (syntax-based bias heuristic, varbias, and FIFO sorting heuristic).

Chapter 4

Interpolation-Based Assumptions Generation

In Chapter 3, we underlined the importance for heuristics to be fully automated, that is, to require the minimum amount of prior knowledge from the user (Section 3.3). A major setback of most existing bias heuristics (like *varbias*, see Section 3.4.2) is that their success depends on the correct parameter choice by the user. In some cases, as we illustrate in this chapter, this knowledge almost coincides with knowing the solution beforehand, thus rendering the whole automated process of assumptions refinement pointless.

In this chapter, we revise and extend our proposal in [Cav16] of a bias heuristic aimed at automating all the reasoning steps one needs to perform when looking for a solution by observing a counterstrategy. Our heuristic is based on Craig's interpolation [McM03], an automated procedure widely used in verification to identify the causes underlying the violation of a temporal logic formula in a given model. We show that interpolation improves the efficiency of the counterstrategy-guided search (measured as number of solutions found over all the nodes explored) in contexts where no prior knowledge is available.

In Section 4.1 we define Craig's interpolation and present McMillan's algorithm to compute interpolants [McM03]. Section 4.2 motivates the use of Craig's interpolation as a bias heuristic in the counterstrategy-guided loop. Section 4.3 introduces the notion of abstract counterstrate-

gies. This is an abstract view of a counterstrategy graph in which only the information on the output variables that affect the next environment choice is retained. The interpolation bias expects in input this form of counterstrategy. Section 4.4 describes in detail the four steps of the interpolation bias, revising our algorithm in [Cav16] in light of the definition of abstract counterstrategies. Section 4.5 describes a variation of the bias, based on unrolling looping counterruns, that helps finding weaker assumptions. Section 4.6 presents an example application of the interpolation bias with unrolling on a popular proof-of-concept example of a lift controller. Section 4.7 proves that the counterstrategy-guided loop of Chapter 3, equipped with the interpolation bias, eventually terminates. Section 4.8 introduces a modified version of Alur's bias heuristic described in Section 3.4.2, which we call *multivarbias*, and describes the experimental campaign we conducted on a set of popular benchmark case studies, using multivarbias as a baseline. Section 4.9 contextualizes the use of interpolation in formal methods by listing a set of further applications that inspired its use as a bias heuristic. Finally, Section 4.10 summarizes the key concepts of this chapter.

Most of this chapter is based on our publication [CA17] and its extension [CA18].

4.1 Preliminaries: Craig's Interpolation

Craig's interpolation was originally defined for first-order logic [Cra57] and later for propositional logic [Kra97]. No interpolation theorems have been proved for the general LTL. Extensions have been proposed recently for LTL fragments [Kam15, GtC09]. However these do not include GR(1) formulae and therefore are not applicable in our case. In this chapter we will refer to Craig's interpolation for propositional logic.

Formally, given an unsatisfiable conjunction of formulae $\beta_1(\mathcal{V}_1) \wedge \beta_2(\mathcal{V}_2)$, a Craig interpolant $I(\mathcal{V}_I)$ is a formula that is implied by β_1, is unsatisfiable in conjunction with β_2, and is defined on the common alphabet $\mathcal{V}_I$ of β_1 and β_2. Recall that a *valid* Boolean formula is a formula that yields true for any assignment of its variables.

Definition 4.1 (Interpolant [Kra97])**.** *Let $\beta_1(\mathcal{V}_1)$ and $\beta_2(\mathcal{V}_2)$ be two logical formulae such that*

their conjunction $\beta_1(\mathcal{V}_1) \wedge \beta_2(\mathcal{V}_2)$ is unsatisfiable. Then there exists a third formula $I(\mathcal{V}_I)$, called interpolant *of β_1 and β_2, such that, $\beta_1 \rightarrow I$ is valid, $I \rightarrow \neg\beta_2$ is valid and $\mathcal{V}_I \subseteq \mathcal{V}_1 \cap \mathcal{V}_2$.*

As a simple example of interpolant, suppose $\beta_1(\mathcal{V}_1) = a \wedge b$ and $\beta_2(\mathcal{V}_2) = \neg b$ are two Boolean formulae using overlapping sets of variables. Then an interpolant is $I(V_I) = b$, since it only uses the common variables between β_1 and β_2, is implied by β_1, and implies $\neg\beta_2$.

An interpolant can be considered as an over-approximation of β_1 that is still unsatisfiable in conjunction with β_2. As stated in Craig's interpolation theorem, although an interpolant always exists, it is not unique. Several efficient algorithms have been proposed for interpolation in propositional logics [Kra97, Pud97, Hua95, Cra57]. The resulting interpolant depends on the internal strategies of these algorithms (e.g., SAT solvers, theorem provers). Our approach is based on McMillan's interpolation algorithm described in [McM03] and implemented in MathSAT [CGS08]. The algorithm considers a proof by resolution for the unsatisfiability of $\beta_1 \wedge \beta_2$.

An unsatisfiability proof by resolution assumes that a Boolean formula is expressed in *conjunctive normal form* (CNF).

Definition 4.2 (Conjunctive normal form)**.** *A Boolean formula β over the variables $\mathcal{V} = \{v_1, \ldots, v_k\}$ is in* conjunctive normal form *(CNF) if it is a conjunction of disjunctions. That is, it is in the form*

$$\beta = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} a_{ij}$$

where for every i, j, there exists $v \in \mathcal{V}$ such that either $a_{ij} = v$ or $a_{ij} = \neg v$. Every a_{ij} is called a literal*. Every conjunct $\bigvee_{j=1}^{m_i} a_{ij}$ is called* clause *and denoted by c_i.*[1] *We denote by C the set of clauses in β.*

It is well-known that any Boolean formula can be converted in an equivalent CNF.

Resolution is based on finding resolvents between clauses, as defined below.

[1]The definition of clause is only used in this section to introduce the interpolation algorithm, and never used again in the rest of the chapter. Therefore, we take the liberty of using the same letter c for clauses as we use for counterstrategies throughout the thesis.

Definition 4.3 (Resolvent)**.** *Given two clauses c_1 and c_2 such that, for some $v \in \mathcal{V}$, $c_1 = a_{11} \vee \cdots \vee a_{1n} \vee v$ and $c_2 = a_{21} \vee \cdots \vee a_{2m} \vee \neg v$. Their* resolvent *is the clause $c = a_{11} \vee \cdots \vee a_{1n} \vee a_{21} \vee \cdots \vee a_{2m}$. The variable v is called the* pivot variable*.*

Definition 4.4 (Unsatisfiability proof by resolution [McM03])**.** *A* proof of unsatisfiability by resolution *for a set of clauses C is a directed acyclic graph $\Pi = (V, E)$, where the vertices V is a set of clauses, such that for every vertex $c \in V$, either*

- *$c \in C$, and c is a root, or*
- *c has exactly two predecessors, c_1 and c_2, and c is their resolvent,*

and the empty clause false *is the unique leaf.*

Then the following (operational) definition describes McMillan's algorithm for computing an interpolant.

Definition 4.5 (Interpolation algorithm [McM03])**.** *Let β_1 and β_2 be two CNF Boolean formulae, and C_1 and C_2 the clause sets of β_1 and β_2. Let $g(c)$ be the disjunction of the literals in the clause c using variables common to β_1 and β_2 (it is* false *if c does not contain any such literal). Let $\Pi = (V, E)$ be a proof of unsatisfiability of $C_1 \cup C_2$. For all vertices $c \in V$, let $p(c)$ be a Boolean formula, such that*

- *if c is a root, then*
  - *if $c \in C_1$ then $p(c) = g(c)$,*
  - *otherwise $p(c)$ is the constant* true*.*
- *else, denoting the predecessors of c by c_1 and c_2, and v their pivot variable,*
  - *if v is in $\mathcal{V}_1 \setminus \mathcal{V}_2$, then $p(c) = p(c_1) \vee p(c_2)$,*
  - *otherwise $p(c) = p(c_1) \wedge p(c_2)$.*

*Then an interpolant of β_1 and β_2 is $I = p($*false*$)$.*

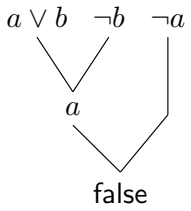


Figure 4.1: Resolution tree proof of the unsatisfiability of $\beta_1 \wedge \beta_2$

Example 4.1. *Let $\beta_1 = (a \vee b) \wedge (c \vee d)$ and $\beta_2 = \neg a \wedge \neg b$. The two Boolean formulae are unsatisfiable together. The two formulae are in CNF and their clauses are $C_1 = \{a \vee b, c \vee d\}$ and $C_2 = \{\neg a, \neg b\}$. A proof by resolution is shown in Figure 4.1. According to Definition 4.5:*

1. *$p(a \vee b) = g(a \vee b) = a \vee b$ since $a \vee b \in C_1$ and a and b appear in both β_1 and β_2.*
2. *$p(\neg b) = \mathsf{true}$ since $\neg b \in C_2$.*
3. *$p(\neg a) = \mathsf{true}$.*
4. *$p(a) = p(a \vee b) \wedge p(\neg b) = (a \vee b) \wedge \mathsf{true}$, since the pivot variable b is common to β_1 and β_2.*
5. *$p(\mathsf{false}) = I = p(a) \wedge p(\neg a) = (a \vee b) \wedge \mathsf{true}$, since the pivot variable a is common to β_1 and β_2.*

Hence, an interpolant of β_1 and β_2 is $I = a \vee b$.

4.2 Motivation

The counterstrategy-guided loop in Chapter 3 derives assumptions from counterstrategies. The intuition behind using Craig's interpolation to this goal is the following.

If one wanted to extract assumptions from counterstrategies manually, this process would involve the following steps: *(i)* checking some execution of the counterstrategy, that is one particular sequence of interactions between the environment and the controller, *(ii)* identifying the guarantee(s) that was violated in that sequence, *(iii)* , and *(iv)* producing an assumption that

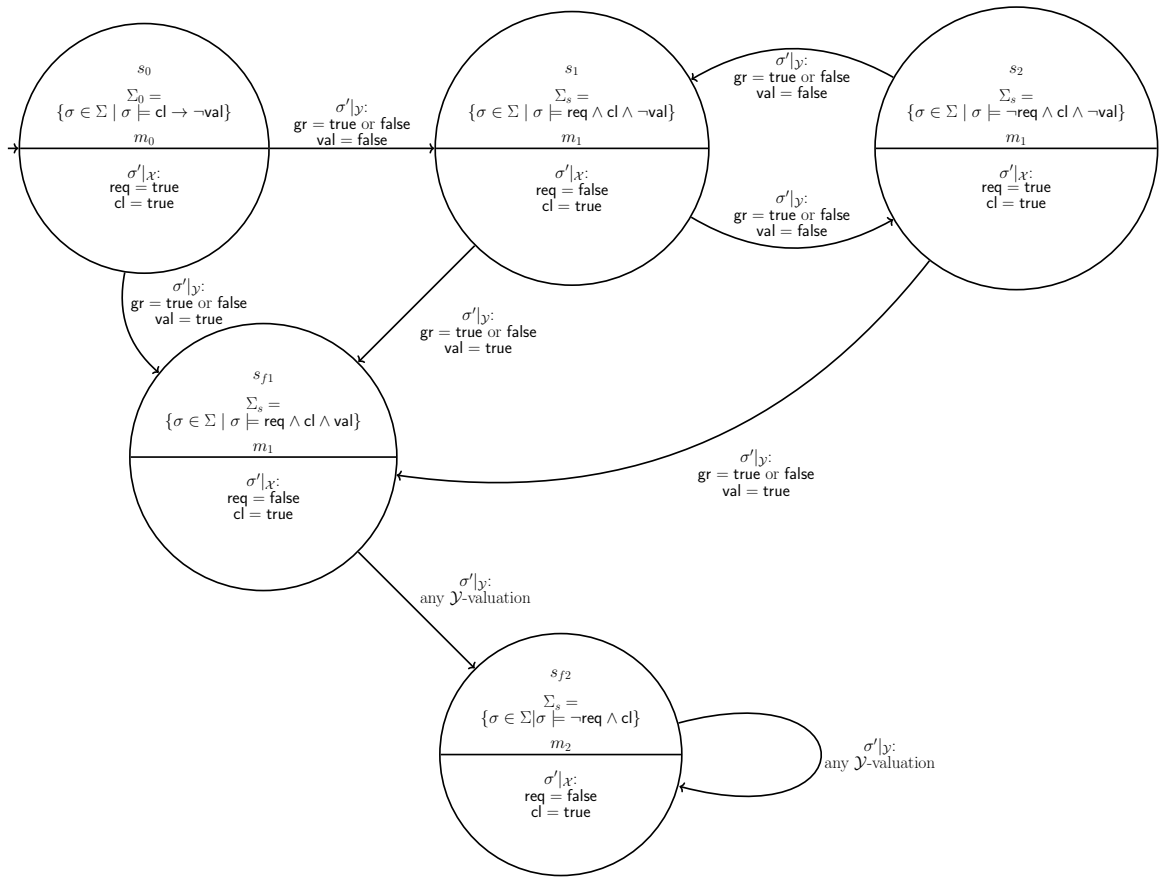


Figure 4.2: Counterstrategy for specification in Example 2.2. The full description is given in Figure 2.1.

forbids those variable assignments. As a concrete instance of this reasoning, let us consider again the request-grant specification presented in Example 2.2.

Example 4.2. *Consider the request-grant protocol in Example 2.2. A counterstrategy is shown in Figure 4.2 . It represents a set of system behaviors where the environment forces the violation of* $J^{\mathcal{S}}$ *by keeping the* cl *variable constantly* true*, while satisfying* $J^{\mathcal{E}}$ *by alternating between* req = true *and* req = false*.*

In step (i) *we extract some path (e.g. the top path* $s_0 s_1 (s_2)^{\omega}$ *from the counterstrategy. In step* (ii)*, we see that the violated guarantee is* $J^{\mathcal{S}}$*. Indeed the output variable* val *is never* true *throughout the infinite path. Step* (iii) *involves checking the specification to understand what causes* val *to never be* true *in the counterstrategy. The invariant guarantee* $\rho^{\mathcal{S}} = \mathsf{cl} \rightarrow \neg\mathsf{val}$ *expresses a relationship between the input variable* cl *and* val*, that is, whenever* cl *is* true*,* val *must be* false *by virtue of the implication. This leads to identifying the cause of the violation in the value of* cl *along the path, which is always* true*. Therefore, we identify in* cl *being always*

true *the cause of the violation of* $J^{\mathcal{S}}$*. An assumption forbidding this behaviour (step* (iv)*) is* $\mathbf{GF}\neg\mathsf{cl}$.

Some of the steps outlined above correspond to the steps performed automatically by the interpolation algorithm in Definition 4.5. That motivates our choice of Craig's interpolation as a fully-automated bias heuristic. In particular, step *(i)* of the procedure outlined above consists in extracting a path on the counterstrategy and using its description as a Boolean formula as a first term of interpolation. Step *(ii)*, that is identifying a violated guarantee, corresponds to identifying the second term of interpolation, which is violated in the extracted path. Step *(iii)*, which in the example above involves following implication steps, is equivalent to performing resolution steps as the ones described in Definition 4.5. This is where Craig's interpolation fits in the bias heuristic. The resulting interpolant, then, represents the cause of unrealizability. Step *(iv)*, finally, requires to produce a GR(1) assumption that prevents this cause from occurring in the environment.

This introductory discussion also points out that the procedure requires some form of pre-processing of the input and post-processing of the output of Craig's interpolation. We elaborate more on this when we describe our full bias heuristic in Section 4.4.

4.3 Abstract Counterstrategies

Like any other bias described in Chapter 3, the interpolation bias we propose here receives in input a counterstrategy. However, it relies on an abstraction of the representation presented in Definition 2.24 that removes all unnecessary information from the graph. The first simplification consists in removing all the controller choices that lead to finite-time violations of the guarantees (that is, violations occurring after a finite number of steps in the game). We can remove them under the premise that a controller trying to win the GR(1) game will never choose those valuations. The example below shows how keeping them hinders assumptions refinement.

Example 4.3. *Consider the counterstrategy of the request-grant example shown in Figure 4.2. Suppose the path* $s_0 s_{f1}(s_{f2})^{\omega}$ *is selected as input to interpolation. It involves a controller choice*

from s_0 to s_1 causing the violation of $\rho^{\mathcal{S}}$. Indeed, the controller sets val *to* true *even if the environment has set* cl *to* true*. The violation is not due to an action of the environment, but to an action of the controller that would not be allowed in the GR(1) game (see Section 2.2.2).*

From this path, the bias can infer the invariant assumption $\mathbf{G}\neg$cl*, that prevents the system from reaching the state s_{f1}.*[2] *However, this is much more constraining than the assumption* $\mathbf{GF}\neg$cl*, inferred in Example 4.2. In the context of weakest assumptions refinement (see Definition 3.4), it is preferable to avoid producing invariants when a weaker fairness assumption can be inferred.*

The second consists in removing from the transition labellings all the output variables whose assignments do not affect the choice of the next state (such as gr in Example 2.2). This leaves on all the transition only the so-called *influential output variables*, that affect the future behaviour of the environment. The example shows why it is not appropriate to keep the non-influential output variables in the description of a counterstrategy path.

Example 4.4. *Consider again the counterstrategy of Figure 4.2, and in particular its path . When producing a Boolean description for it, we may not need to include the variable* gr*. Indeed, its value is unspecified in s_1, as it can be either* true *or* false*. We say that* gr*, in the predecessor state s_0 is not an influential output variable, since the future choices of the counterstrategy do not depend on the value of* gr *from s_0. Likewise, it is not influential in any state forever.*

Considering non-influential output variables in the description of the path may be misleading in the identification of the cause of unrealizability. Since gr *is unspecified in the path, one may set it to be always* false*. Then, the violation of the fairness condition* $\mathbf{GF}(\mathsf{gr}\wedge\mathsf{val})$ *may be mistakenly be attributed to* gr *being always* false *rather than* val *being always* false*. In fact, while* val *is forced to be* false *due to the behaviour of the environment (see Example 4.2),* gr *is always* false *due to an arbitrary assignment by the controller. There is no environment behaviour that could be removed through an assumption in order to prevent* gr *from being always* false*. Therefore,* gr *should not be considered when looking for an explanation of the guarantee violation.*

In the following we formalize the notion of *abstract counterstrategies*, that modify Definition 2.24

[2]

to apply the two simplifications above.[3]

Failing and admissible output valuations. In the following, consider a counterstrategy graph $c = (S, s_0, E)$. Let $\rho^{\mathcal{S}}$ be the invariant guarantee in an unrealizable GR(1) formula ϕ. We are going to characterize, for a state $s \in S$ of the counterstrategy, the set of *failing output valuations*. Let $s = (\Sigma_s, m, \sigma'|_{\mathcal{X}})$ be the counterstrategy's state and $\sigma \in \Sigma_s$ be any $\mathcal{V}$-valuation of the state (remember that a state in a counterstrategy graph contains one or more valuations such that the future behavior of the environment from each of those valuations is the same, see Definition 2.24). We define the set of *failing output valuations from game state* σ denoted with $2^{\mathcal{Y}}_{fail}(\sigma)$ as the set containing the $\mathcal{Y}$-valuations $\sigma'|_{\mathcal{Y}}$ such that any play transitioning from σ into $\sigma' = \sigma'|_{\mathcal{X}} \cup \sigma'|_{\mathcal{Y}}$ violates the invariant $\rho^{\mathcal{S}}$:

Definition 4.6. *Given a counterstrategy state $s = (\Sigma_s, m, \sigma'|_{\mathcal{X}})$ and a $\mathcal{V}$-valuation $\sigma \in \Sigma_s$, the* failing output valuations *from the valuation σ are the valuations in the set*

$$2^{\mathcal{Y}}_{fail}(\sigma) = \left\{ \sigma'|_{\mathcal{Y}} \in 2^{\mathcal{Y}} \middle| , w = (\sigma, \sigma'|_{\mathcal{X}} \cup \sigma'|_{\mathcal{Y}}, w[3], w[4], \dots) \not\models \rho^{\mathcal{S}} \right\} .$$

If an output valuation is failing from every game state $\sigma \in \Sigma_s$, we say it is failing from the counterstrategy state s.

Definition 4.7. *Given a counterstrategy state $s = (\Sigma_s, m, \sigma'|_{\mathcal{X}})$, an output valuation is* failing from the state s *if it is failing from every valuation $\sigma \in \Sigma_s$. We denote the set of failing output valuations from s by $2^{\mathcal{Y}}_{fail}(s)$.*

The set of *admissible output valuations from s* is just the complement of the failing valuations.

Definition 4.8. *The set of the* admissible output valuations from s *(or* admissible $\mathcal{Y}$-valuations from s*) is*

$$2^{\mathcal{Y}}_{adm}(s) = 2^{\mathcal{Y}} \setminus 2^{\mathcal{Y}}_{fail}(s) .$$

[3] Abstract counterstrategies formalize the actual counterstrategy output by the RATSY tool for realizability checks. Presumably, this is the kind of structure actually used in our baseline works [LDS11, AMT13], although the involved abstractions are not formalized there. We are the first to elicit this formalization.

Then we can give the following definition of abstract counterstrategy.

Definition 4.9. *An* abstract counterstrategy *is a 4-tuple* $c = (S, s_0, E, T)$ *such that:*

- $S \subseteq 2^{2^{\mathcal{V}}} \times M \times 2^{\mathcal{X}}$ *is a set of counterstrategy states (as in Definition 2.24);*
- $s_0 = (\Sigma_0, m_0, \sigma_1|_{\mathcal{X}})$ *is the initial state;*
- $\mathcal{T} = \{T_s\}_{s \in S}$ *is a collection of transition functions, indexed by states in* S*; for every* $s \in S$, $T_s : 2^{\mathcal{Y}}_{adm}(s) \to S$ *is a function that, given an admissible output valuation for state* s*, returns the next state.*

The collection of transition functions T in the above definition operates the first reduction of counterstrategy graphs discussed at the beginning of this section, that is it excludes transitions labelled with failing output valuations.

Example 4.5. *Let us consider again the counterstrategy of Example 2.2 (Figure 4.2). One thing to note is that, since the violation regards a fairness condition, the path drawn in the upper part of the figure ends in a strongly connected subset of nodes where* $\mathsf{gr} \wedge \mathsf{val}$ *is* false.

Notice also that in each state, the choice of the next input does not depend on gr*, since the state reached in any transition is the same regardless of the value of* gr*. Moreover, if at any step the controller chooses to set* val *to* true*, the invariant* $\rho^{\mathcal{S}}$ *is violated. Therefore the counterstrategy enters the state labelled* s_{f1} *in the figure and from that point any subsequent infinite play is violating.*

Formally,

$$2^{\mathcal{Y}}_{fail}(s_0) = 2^{\mathcal{Y}}_{fail}(s_1) = 2^{\mathcal{Y}}_{fail}(s_2) = \{\sigma|_{\mathcal{Y}} \in 2^{\mathcal{Y}} \mid \mathsf{val} \in \sigma|_{\mathcal{Y}}\}$$

and

$$2^{\mathcal{Y}}_{adm}(s_0) = 2^{\mathcal{Y}}_{adm}(s_1) = 2^{\mathcal{Y}}_{adm}(s_2) = \{\sigma|_{\mathcal{Y}} \in 2^{\mathcal{Y}} \mid \mathsf{val} \notin \sigma|_{\mathcal{Y}}\}$$

The abstract counterstrategy obtained by eliminating the failing output valuations is shown in Figure 4.3. □

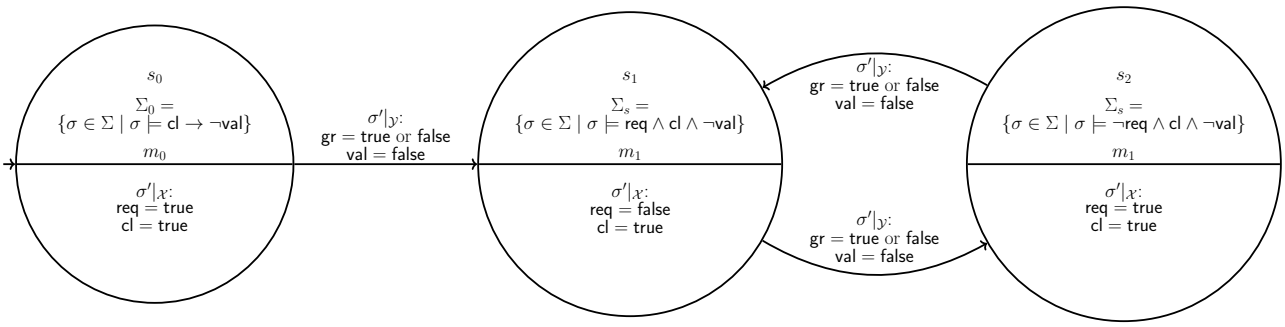


Figure 4.3: Counterstrategy of the specification in Example 2.2 after removing the failing output valuations

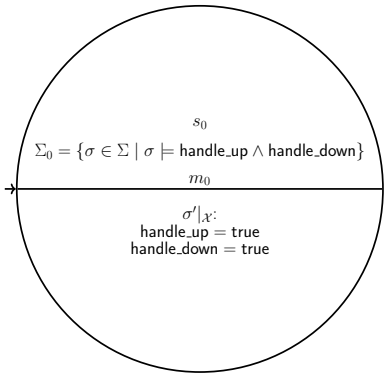


Figure 4.4: Abstract counterstrategy of the LGS example. It consists of an initial state only. When the environment sets both input variables to true in two consecutive steps, the two guarantees cannot be satisfied simultaneously.

It is possible that the set of admissible output valuations for some state s is empty, $2^{\mathcal{Y}}_{adm}(s) = \varnothing$. This happens if the environment's next input choice $\sigma'|_{\mathcal{X}}$ forces the violation of some invariant guarantee. In this case, according to Definition 4.9, s has no successors.

Example 4.6. *Consider the LGS specification presented in Example 1.2. The specification has a one-state abstract counterstrategy, as pictured in Figure 4.4. The initial state s_0 has no successor. In fact, in s_0 both* handle_down *and* handle_up *are* true*, and the counterstrategy sets* handle_down *and* handle_up *again to true in the following state. This prevents the guarantees $\rho_1^{\mathcal{S}}$ and $\rho_2^{\mathcal{S}}$ of Example 1.2 from being satisfiable at the same time, regardless of the output valuation.*

Influential output variables. We finally define the notion of influential output variable for a state.

Definition 4.10. *We define the set of* influential output variables *of a counterstrategy state*

$s = (\sigma, m, q'_{\mathcal{X}})$ *as*

$$Y(s) = \{p \in \mathcal{Y} \mid \exists \sigma_1|_{\mathcal{Y}}, \sigma_2|_{\mathcal{Y}} \in 2^{\mathcal{Y}}_{adm}(s), p \in \sigma_1|_{\mathcal{Y}} \wedge p \notin \sigma_2|_{\mathcal{Y}} \wedge T_s(\sigma_1|_{\mathcal{Y}}) \neq T_s(\sigma_2|_{\mathcal{Y}})\} \ .$$

The function $Y : S \to 2^{\mathcal{Y}}$ *maps every state to the set of its influential output variables.*

In other words, an influential output variable is a variable whose assignment the next state in the counterstrategy. If one assigns true or false to an influential variable, the counterstrategy transitions into different states. We can provide a more concise description of a counterstrategy by using only influential variables to label its transitions.

Example 4.7. *Consider the graph in Figure 4.3, and the state* s_0*. We want to identify the set of influential output variables* $Y(s_0)$*.*

First, consider the output variable val*. As shown in the previous example,* $y \in 2^{\mathcal{Y}}_{adm}(s_0)$ *if and only if* $\mathsf{val} \notin y$*. Therefore,* val *does not affect the state into which the counterstrategy transitions after visiting* s_0*.*

Then, consider the variable gr*. Both* true *and* false *appear in some admissible valuations of* s_0*, but the next state does not depend on the value assigned to* gr*. In symbols:*

$$\exists y_1, y_2 \in 2^{\mathcal{Y}}_{adm}(s_0) \textit{ such that } p \in y_1 \wedge p \notin y_2 \wedge T_{s_0}(y_1) = T_{s_0}(y_2) \ .$$

Therefore, $\mathsf{gr} \notin \mathsf{Y}(\mathsf{s_0})$*. We can then conclude that there are no influential output variables in* s_0*, and* $Y(s_0) = \varnothing$*.*

Since the same reasoning can be performed on all other states, the abstract counterstrategy can be reduced to the simple, one-path graph with unlabeled transitions in Figure 4.5.

Abstract counterruns and counterplays. Any path in an abstract counterstrategy starting from the initial state is called *abstract counterrun*. It induces a sequence of partial valuations where the subset of variables valuated is different at each step. Given an infinite counterrun

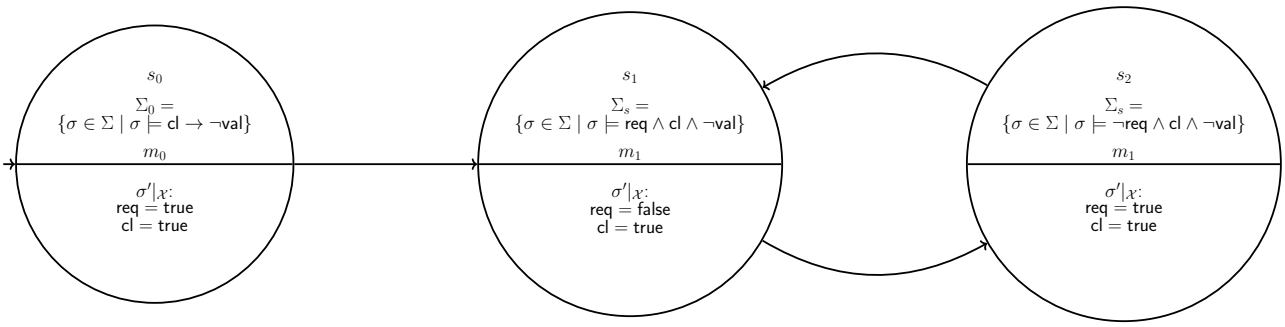


Figure 4.5: Abstract counterstrategy for the specification in Example 2.2. The non-influential output variables are removed from the edges.

$r = s_0 s_1 s_2 \ldots$, where s_0 is the initial state, we define a *counterplay* $w = \sigma_0 \sigma_1 \ldots$ *induced by* r as any sequence of $\mathcal{V}$-valuations such that

- for all $i \geq 0$, given $s_i = (\Sigma_{s,i}, m_i, \sigma'_i|_{\mathcal{X}})$, $\sigma_i \in \Sigma_{s,i}$.

Likewise, given a finite counterrun $r = s_0 s_1 \ldots s_n$, a *counterplay* $w = \sigma_0 \sigma_1 \ldots$ *induced by* r is defined as an infinite sequence of $\mathcal{V}$-valuations w such that

- for all $0 \leq i \leq n$, given $s_i = (\Sigma_{s,i}, m_i, \sigma'_i|_{\mathcal{X}})$, $\sigma_i \in \Sigma_{s,i}$;
- $\sigma_{n+1}|_{\mathcal{X}} = \sigma'_n|_{\mathcal{X}}$;
- for all $i > n+1$, σ_i is arbitrary.

We say that an abstract counterrun $r = s_0 s_1 \ldots$ (finite or infinite) *satisfies* an LTL formula ϕ in state s_i, denoted by $\langle r, s_i \rangle$, if every counterplay w induced by r satisfies ϕ in position $i+1$, $\langle w, i+1 \rangle \models \phi$ in the sense of Definition 2.4.

In this chapter, whenever we talk about counterruns and counterplays we refer to the abstract definitions just given. Counterruns are abstract descriptions of sequences of system states leading to a specific guarantee violation in the counterstrategy. Our approach makes use of counterruns to generate assumptions refinements (see Section 4.4.1).

Remark 4.1. *The abstract counterstrategy, as defined in this section, is the structure used in state-of-the-art approaches discussed in Chapter 3 [LDS11, AMT13], and is the structure*

returned by the RATSY tool for GR(1) synthesis [BCG+10] when the input is an unrealizable formula. In this work a counterstrategy is defined as a Moore transducer, where the input alphabet is the set of output valuations and the output alphabet is the set of input valuations (recall that in a Moore transducer the output alphabet is the set of state labels, while the input alphabet is the set of transition labels). This formalism is not sufficiently fine-grained in our case as it does not allow one to have different input alphabets per state. We introduced the notion of influential output variables to allow different input alphabets per state, so that the outgoing transitions of some state can have a different set of labels from the outgoing transitions of another state.

We need to define the influential output variables since those are the only ones included in the translation of the counterplay into a Boolean formula, as we shall see in Section 4.4.2.

4.4 The Interpolation Bias

The contents of this section describe the same steps as presented in [Cav16] with two differences. *(i)* The input to the procedure is not a Moore transducer but an abstract counterstrategy. This affects the first two steps of the procedure, illustrated in Sections 4.4.1 and 4.4.2. *(ii)* Counterrun unrolling, which was an integral part of the procedure in [Cav16], here becomes an optional improvement. The reason is that we only observed the prospected improvement given by unrolling in the lift example (described in Section 4.6), while it made no difference in the AMBA-AHB case studies (described in Section 3.5). Moreover, new examples, not present in [Cav16], have been added throughout the presentation.

The steps of our bias based on interpolation are illustrated in Figure 4.6. It requires in input an abstract counterstrategy c, as described in Section 4.3. Unlike the bias heuristics described in Section 3.4, which generate assumptions violated by collections of counterruns, interpolation is applied to single counterruns. We discuss the reasons of this choice in Section 4.4.1. The first step consists in extracting a counterrun r from c.

The second step consists in filling the gap between the language of GR(1) (involving temporal

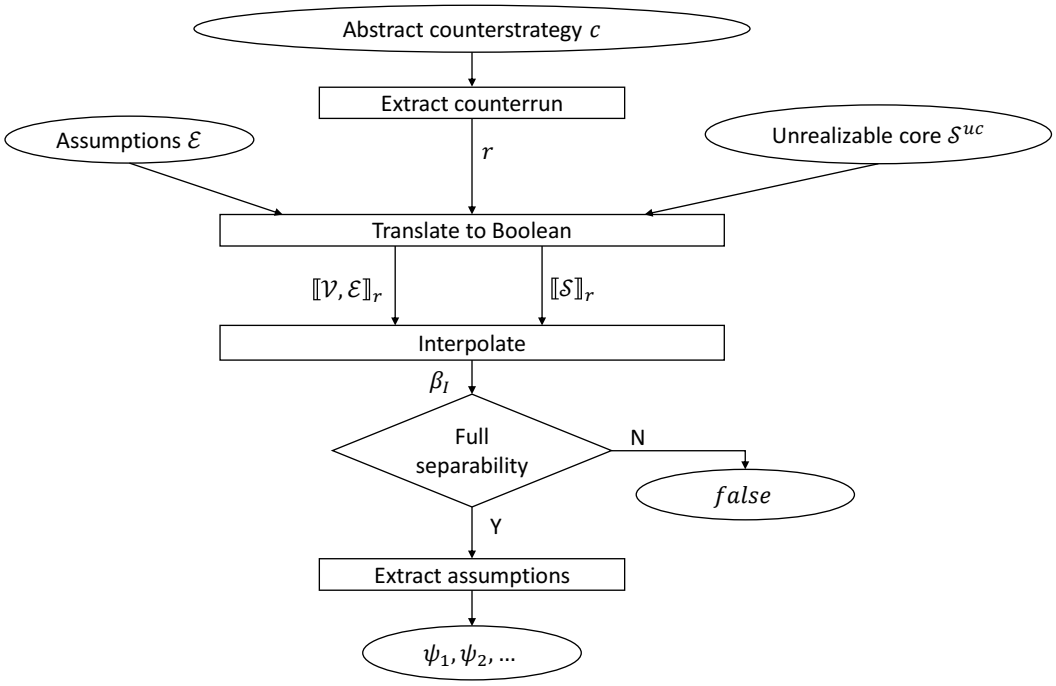


Figure 4.6: Main steps of interpolation bias

logic formulae and sequences of valuations), and that of interpolation (propositional logic). The block *Translate to Boolean* takes in input a counterrun, the set of assumptions $\mathcal{E}$, and an unrealizable core $\mathcal{S}^{uc}$, and produces two Boolean formulae, $[\![\mathcal{V}, \mathcal{E}]\!]_r$ and $[\![\mathcal{S}^{uc}]\!]_r$. The former is a formula satisfied in the counterrun, while the latter contains the guarantees of the unrealizable core, one of which is violated in the counterrun. The translation is discussed in Section 4.4.2[4].

The third step is Craig's interpolation, where $[\![\mathcal{V}, \mathcal{E}]\!]_r$ and $[\![\mathcal{S}^{uc}]\!]_r$ are interpolated with each other and an interpolant I is produced. We define a sufficient condition over the interpolant in order to extract GR(1) conjuncts from it. We call this condition *full separability* and define it in Section 4.4.3.

If I is fully separable, it undergoes the fourth step where a set of alternative assumptions $\psi_1, \ldots, \psi_k$ is extracted from it. This step involves translating the interpolant back into an

[4]Kamp's theorem [Kam68] states an equivalence between LTL and the first-order theory of linear order. Hence, for our bias we could use interpolation in first-order logic. However, we design our bias around interpolation in Boolean logic due to the wide availability of tools that support this kind of interpolation (included in popular SMT-solvers like MathSAT [BCF+08] and Z3 [DMB08]).

One can argue that our translation, involving replicates of the propositional variables over a linear sequence of states in a counterstrategy (see Section 4.4.2) is actually a translation to first-order logic. Indeed, the replicates $v(s)$ of the Boolean variables in $\mathcal{V}$ can be interpreted as predicates over constants that range on a linear order. A formalization of the procedure in this direction is matter for future work.

Algorithm 8: InterpolationBasedGenerateRefinements($\phi^{\mathcal{E}}$, $\mathcal{S}^{uc}$, c).

Data: $\mathcal{E}$, environment assumptions
Data: $\mathcal{S}^{uc}$, controller guarantees (in an unrealizable core)
Data: c, abstract counterstrategy
Result: altRefs, alternative assumptions eliminating the counterstrategy

1 r := **ExtractCounterrun**(c);
2 $[\![\mathcal{V}, \mathcal{E}]\!]_r$:= **TranslateCounterrunAssumptions**(r, $\mathcal{E}$);
3 $[\![\mathcal{S}^{uc}]\!]_r$:= **TranslateGuarantees**(r,$\mathcal{S}$);
4 I := **Interpolate**($[\![\mathcal{V}, \mathcal{E}]\!]_r$, $[\![\mathcal{S}^{uc}]\!]_r$);
5 **if** *I is not fully-separable* **then**
6 | altRefs := {false};
7 **else**
8 | $\mathcal{T}(I)$:= **TranslateInterpolant**(r, I);
9 | altRefs := **ExtractDisjuncts**($\neg\mathcal{T}(I)$);
10 **end**
11 **return** altRefs;

LTL formula satisfied by the counterrun r, whose negation is violated by the counterstrategy c. In Section 4.4.4 we discuss this last step and prove its soundness. If I is not fully separable, the interpolation bias produces no useful assumptions (a condition denoted as returning the constant false).

The entire approach is summarized by the function **InterpolationBasedGenerateRefinements** described in Algorithm 8. The function **ExtractCounterrun** (line 1) performs the first step by selecting a counterrun r at random.[5] The functions **TranslateCounterrunAssumptions** and **TranslateGuarantees** (line 2-3) take the GR(1) specification and the counterrun r and translate them into two Boolean formulae (second step). These two formulae are then interpolated by **Interpolate** (line 4), and the interpolant is checked for full separability (line 5); this is the third step. Finally, if the interpolant is fully separable, it is translated into an LTL formula $\mathcal{T}(I)$ and the disjuncts of $\mathcal{T}(I)$ (line 9) are the assumptions returned by the procedure (fourth step).

We now illustrate in more detail the steps of **InterpolationBasedGenerateRefinements** listed above, and how they are carried out on the two running Examples 2.2 and 1.2.

[5]

4.4.1 Extracting Counterruns

Unlike existing biases searching for assumptions that are violated by *every* path in the counterstrategy (see Section 3.4), the interpolation heuristic focuses on a *single* counterrun extracted from it. In fact, an LTL formula is satisfied by a counterstrategy if and only if every path in it satisfies it (see the semantics defined in Definition 2.21). Thereby, if a single counterrun violates an assumption, the entire counterstrategy violates it.

A bias focusing on single counterruns has the advantage of finding assumptions that cannot be stronger than assumptions found by focusing on entire counterstrategies. We prove this in the following proposition.

Proposition 4.1. *Let c be a counterstrategy. Consider an assumption ψ such that for every counterrun r of c, $r \not\models c$, and an assumption ψ' such that for some counterrun r' of c, $r' \not\models \psi'$, while for some different counterrun r'', $r'' \models \psi'$. Then it cannot be $\psi' \rightarrow \psi$.*

Proof. Suppose by contradiction that $\psi' \rightarrow \psi$. Then equivalently $\neg\psi \rightarrow \neg\psi'$. Therefore since $r'' \not\models \psi$ it must be $r'' \not\models \psi'$, against the hypothesis. □

For the purpose of this chapter, we assume that the procedure **ExtractCounterrun** (line 1) extracts a counterrun r at random.

A counterrun leading to the violation of an initial condition or an invariant guarantee is finite, while that of a fairness guarantee violation ends in a loop [Li14]. We call the latter a *looping counterrun*, and the loop an *ending loop*.[6]

A counterrun consists of four sets of states: (a) the *initial state* as the element of the singleton $S^{init} = \{s^{init}\}$; (b) the *failing state* in a finite counterrun as the element of $S^{fail} = \{s^{fail}\}$ (c) *looping states* that include the states in ending loop, $S^{loop} = \{s_1^{loop}, \ldots, s_h^{loop}\}$, (d) *transient states* including all states between the initial state and the first failing state

[6] While in [Cav16] a counterrun could be any path on a counterstrategy, here the counterrun is any path on an abstract counterstrategy.

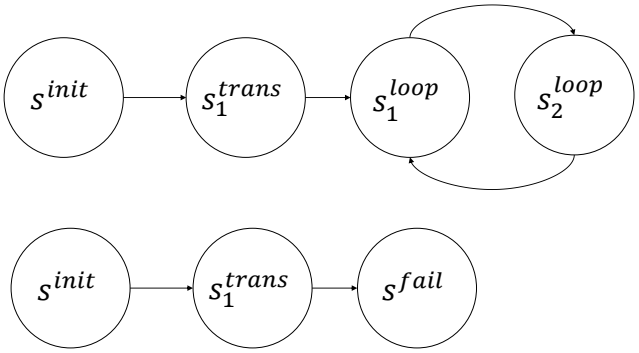


Figure 4.7: Looping counterrun (top) and finite counterrun (bottom)

or loop state (exclusive) $S^{trans} = \{s_1^{trans}, \ldots, s_k^{trans}\}$. With this classification, a finite counterrun has the form $r = s^{init} s_1^{trans} \ldots s_k^{trans} s^{fail}$; whilst a looping counterrun has the form $r = s^{init} s_1^{trans} \ldots s_k^{trans} (s_1^{loop} \ldots s_h^{loop})^{\omega}$. We call $S_r \subseteq S$ the union of all the states appearing in r. Figure 4.7 shows the two types of counterrun and their states.

Example 4.8. *Let us consider the specification in Example 2.2 and the counterstrategy in Figure 4.5. The counterstrategy has only one looping run $r = s_0(s_1 s_2)^{\omega}$, which is to be used at next stages. This is an example of a looping counterrun with $S^{init} = \{s_0\}, S^{trans} = \varnothing, S^{loop} = \{s_1, s_2\}$, and $S_r = S = \{s_0, s_1, s_2\}$.* □

Example 4.9. *In the LGS case study (see the counterstrategy in Figure 4.4) there is a single finite counterrun r consisting of the initial state s_0 and a failing state s^{fail}. So $r := s_0 s^{fail}$.*

4.4.2 Boolean Encoding of Counterplays and Unrealizable Cores

The second step of **InterpolationBasedGenerateRefinements** is implemented by the functions **TranslateCounterrunAssumptions** and **TranslateGuarantees** (lines 2-3). The procedure employs a similar translation scheme as defined in [BCC+03] for bounded model checking, which ensures that the obtained Boolean formula is satisfiable if and only if the play taken into account satisfies the LTL formula.

The translation of a GR(1) formula into the Boolean domain is a Boolean formula over the

domain $\mathcal{V}(S_r)$ obtained by replicating every variable $v \in \mathcal{V}$ for every state $s \in S_r$; we denote by $v(s)$ the replica of v referring to state s, and by $\mathcal{V}(s)$ the subset of $\mathcal{V}(S_r)$ containing all the variables referring to state s. Formally:

$$\mathcal{V}(S_r) := \{v(s) | v \in \mathcal{V},\ s \in S_r\}$$

$$\forall s \in S_r,\ \mathcal{V}(s) := \{v(s) | v \in \mathcal{V}\}$$

For instance, suppose the run r consists of three states, $r = s_0 s_1 s_2$, and the system has two variables $\mathcal{V} = \{a, b\}$. Then $S_r = \{s_0, s_1, s_2\}$, $\mathcal{V}(S_r) = \{a(s_0), a(s_1), a(s_2), b(s_0), b(s_1), b(s_2)\}$, and $\mathcal{V}(s_1) = \{a(s_1), b(s_1)\}$.

The translation operation is designed so as to be executed in linear time with the length of r, and works on GR(1) conjuncts as follows:

- an initial condition $\theta = \beta(\mathcal{V})$ is translated to $[\![\theta]\!]_r := \beta(\mathcal{V}(s^{init}))$ by replacing each occurrence of $v \in \mathcal{V}$ with $v(s^{init}) \in \mathcal{V}(s^{init})$;
- an invariant $\rho = \mathbf{G}\beta(\mathcal{V} \cup \mathbf{X}\mathcal{V})$ is translated as the conjunction over all states in S_r $[\![\rho]\!]_r := \bigwedge_{s \in S_r} \beta(\mathcal{V}(s) \cup \mathcal{V}(\mathsf{succ}\,(s)))$, where $\mathsf{succ}(s)$ is the successor state of s in r;
- a fairness condition $J = \mathbf{GF}\beta(\mathcal{V})$ is translated into a disjunction over the looping states as $[\![J]\!]_r := \bigvee_{s \in S^{loop}} \beta(\mathcal{V}(s))$, or skipped if r is not looping.

The translation produces two Boolean formulae: a formula $[\![\mathcal{V}, \mathcal{E}]\!]_r := [\![\mathcal{E}]\!]_r \wedge [\![\mathcal{V}]\!]_r$ that describes r and a formula $[\![\mathcal{S}^{uc}]\!]_r$ that describes the unrealizable core. The former consists of two conjuncts:

- a Boolean expression $[\![\mathcal{E}]\!]_r$ that translates the assumptions by the rules described above;
- a Boolean expression $[\![\mathcal{V}]\!]_r$ describing the input and output valuations last seen at every state s in r, restraining the output valuation to the influential output variables for the predecessor of s.

The expression $[\![\mathcal{V}]\!]_r = \wedge_{s \in S_r \setminus S^{init}} \beta(\mathcal{V}(s))$ is a conjunction of conjunctive formulae $\beta(\mathcal{V}(s))$ whose variables refer to a single state s. Each $\beta(\mathcal{V}(s))$ is a conjunction of literals $v(s)$ or $\neg v(s)$ for every $v \in \mathcal{X} \cup Y(\mathsf{pred}(s))$, where $\mathsf{pred}(s)$ is the predecessor of s in r, and $Y(\mathsf{pred}(s))$ is the set of influential output variables in $\mathsf{pred}(s)$.[7] The conjunction contains the positive literal $v(s)$ if and only if $v \in \sigma_s$ for every $\sigma_s \in \Sigma_s$ (recall that a state s in a counterstrategy graph is a triple $s = (\Sigma_s, m, \sigma'|_{\mathcal{X}})$, and Σ_s is the set of game valuations grouped in s, see Definitions 2.24-4.9). Dually, the formula contains the negative literal $\neg v(s)$ if and only if $v \notin \sigma_s$ for every $\sigma_s \in \Sigma_s$. Since by construction the counterplay satisfies the assumptions $\phi^{\mathcal{E}}$, the formula $[\![\mathcal{V}, \mathcal{E}]\!]_r$ is satisfiable.

The formula $[\![\mathcal{S}^{uc}]\!]_r$ only contains the translation of the unrealizable core obtained via the rules described above. Since by definition a counterrun r satisfies the assumptions and violates the guarantees, the formula $[\![\mathcal{V}, \mathcal{E}]\!]_r \wedge [\![\mathcal{S}^{uc}]\!]_r$ is unsatisfiable by construction. Therefore, according to Definition 4.1, there exists an interpolant for $[\![\mathcal{V}, \mathcal{E}]\!]_r$ and $[\![\mathcal{S}^{uc}]\!]_r$.

Example 4.10. *In the request-grant protocol, there is just one assumption* $J^{\mathcal{E}} = \mathbf{GF}\neg\mathsf{req}$. *Its translation yields the Boolean formula:*

$$[\![\mathcal{E}]\!]_r = [\![J^{\mathcal{E}}]\!]_r = \neg\mathsf{req}(\mathsf{s}_1) \vee \neg\mathsf{req}(\mathsf{s}_2) \ .$$

The formula $[\![\mathcal{V}]\!]_r$ *is obtained by inspecting Figure 4.5 for the valuations of input and influential output variables in each state:*

$$[\![\mathcal{V}]\!]_r = (\mathsf{req}(\mathsf{s}_1) \wedge \mathsf{cl}(\mathsf{s}_1)) \wedge (\neg\mathsf{req}(\mathsf{s}_2) \wedge \mathsf{cl}(\mathsf{s}_2)) \ .$$

Since there are no influential output variables, $[\![\mathcal{V}]\!]_r$ *only contains input variables. The Boolean description of the counterplay is just* $[\![\mathcal{V}, \mathcal{E}]\!]_r = [\![\mathcal{E}]\!]_r \wedge [\![\mathcal{V}]\!]_r$.

The guarantees of the unrealizable core comprise the GR(1) invariant $\rho^{\mathcal{S}}$ *and the fairness condition* $J^{\mathcal{S}}$ *from Example 2.2. The translation of the invariant guarantee* $\rho^{\mathcal{S}} = \mathbf{G}(\mathsf{cl} \rightarrow \neg\mathsf{val})$

[7] In [Cav16] influential output variables were not defined, and the translation procedure inserted a literal for every variable in $\mathcal{V} = \mathcal{X} \cup \mathcal{Y}$.

is:

$$[\![\rho^{\mathcal{S}}]\!]_r = (\mathsf{cl}(\mathsf{s}_0) \rightarrow \neg\mathsf{val}(\mathsf{s}_0)) \wedge (\mathsf{cl}(\mathsf{s}_1) \rightarrow \neg\mathsf{val}(\mathsf{s}_1)) \wedge (\mathsf{cl}(\mathsf{s}_2) \rightarrow \neg\mathsf{val}(\mathsf{s}_2)) \ .$$

That of the fairness guarantee $J^{\mathcal{S}} = \mathbf{GF}(\mathsf{gr} \wedge \mathsf{val})$ *is:*

$$[\![J^{\mathcal{S}}]\!]_r = (\mathsf{gr}(\mathsf{s}_1) \wedge \mathsf{val}(\mathsf{s}_1)) \vee (\mathsf{gr}(\mathsf{s}_2) \wedge \mathsf{val}(\mathsf{s}_2)) \ .$$

Hence, the translation of the full set of guarantees is

$$[\![\mathcal{S}^{uc}]\!]_r = [\![\rho^{\mathcal{S}}]\!]_r \wedge [\![J^{\mathcal{S}}]\!]_r$$

□

Example 4.11. *The landing gear's counterrun* r *only consists of two states* s_0 *and* s^{fail}*. There are no assumptions in the specification, hence there is no* $[\![\mathcal{E}]\!]_r$ *component. The valuations of* r*'s states are translated into*

$$[\![\mathcal{V}]\!]_r := \mathsf{handle_up}(\mathsf{s}_0) \wedge \mathsf{handle_down}(\mathsf{s}_0) \wedge \mathsf{handle_up}(\mathsf{s}^{\mathsf{fail}}) \wedge \mathsf{handle_down}(\mathsf{s}^{\mathsf{fail}})$$

The guarantees are translated into

$$\begin{aligned}[\![\mathcal{S}^{uc}]\!]_r := {} & (\mathsf{handle_down}(\mathsf{s}_0) \rightarrow (\neg\mathsf{handle_down}(\mathsf{s}^{\mathsf{fail}}) \vee \mathsf{handle_down}(\mathsf{s}^{\mathsf{fail}}) \wedge \mathsf{gear_extended}(\mathsf{s}^{\mathsf{fail}}))) \\ & \wedge (\mathsf{handle_up}(\mathsf{s}_0) \rightarrow (\neg\mathsf{handle_up}(\mathsf{s}^{\mathsf{fail}}) \vee \mathsf{handle_up}(\mathsf{s}^{\mathsf{fail}}) \wedge \neg\mathsf{gear_extended}(\mathsf{s}^{\mathsf{fail}})))\end{aligned}$$

□

4.4.3 Interpolation and Full Separability

Once the specification is translated into two Boolean formulae, we apply the function **Interpolate** to these formulae (line 4). The returned interpolant I is an over-approximation of $[\![\mathcal{V}, \mathcal{E}]\!]_r$ which by definition implies the negation of $[\![\mathcal{S}^{uc}]\!]_r$. It can be interpreted as a cause of the

guarantees not being satisfied by the counterplay, and as such a characterization of a set of counterplays not satisfying the guarantees.

From such interpolant the procedure aims at extracting a set of refinements that fit the GR(1) format. In order to do this, the Boolean to temporal translation requires the interpolant to adhere a specific structure. This is embodied in the notion of *full-separability*. To formally define full-separability, we need first to define state-separability and I/O-separability.

Definition 4.11 (State-separable interpolant)**.** *An interpolant I is said to be* state-separable *iff it is in the form*

$$\bigwedge_{s \in S_r} \beta_s(\mathcal{V}(s)) \tag{4.1}$$

where $\beta_s(\mathcal{V}(s))$ is a Boolean formula either equal to true *or expressed over variables in $\mathcal{V}(s)$ only.*

We will refer to each $\beta_s(\mathcal{V}(s))$ as a *state component* of the interpolant. In particular, a state component is equal to true if I does not use any variables from s. State-separability intuitively means that the subformulae of the interpolant involving every single state in the counterplay are linked by conjunctions. This means that in any model of the interpolant each state component must be itself true.

Definition 4.12 (I/O-separable Boolean expression)**.** *A Boolean expression $\beta_s(\mathcal{V}(s))$ is said to be* I/O-separable *if it can be written as a conjunction of two subformulae containing only input and output variables respectively:*

$$\beta_s(\mathcal{V}(s)) = \beta_s|_{\mathcal{X}}(\mathcal{X}(s)) \wedge \beta_s|_{\mathcal{Y}}(\mathcal{Y}(s)) \tag{4.2}$$

We call $\beta_s|_{\mathcal{X}}(\mathcal{X}(s))$ and $\beta_s|_{\mathcal{Y}}(\mathcal{Y}(s))$ the projections[8] *of $\beta_s(\mathcal{V}(s))$ onto $\mathcal{X}$ and $\mathcal{Y}$ respectively.*

Any model of an I/O-separable Boolean expression satisfies the projections separately. We can now define full-separability of an interpolant.

[8]Note that this definition of projection is different from the one given in Definition 2.18. There we define projections of valuations, while here we define projections of Boolean expressions of some specific form.

Definition 4.13 (Fully-separable interpolant)**.** *An interpolant is called* fully-separable *if it is state-separable and each of its state components is I/O-separable.*

An example of a fully-separable interpolant over $\mathcal{X} = \{a, b\}, \mathcal{Y} = \{c, d\}$ and states $S = \{s_0, s_1\}$ is $(a(s_0) \vee b(s_0)) \wedge c(s_0) \wedge \neg b(s_1)$; a non-fully-separable interpolant, instead, is $a(s_0) \vee a(s_1)$, since literals referring to different states are linked via a disjunction.

Remark 4.2. *A particular class of fully-separable interpolants is that of fully conjunctive interpolants, where no disjunctions appear. Whether or not the resulting interpolant is conjunctive depends on the order in which the interpolation algorithm [McM03] chooses the root clauses for building the unsatisfiability proof. A sufficient condition for obtaining a fully-conjunctive interpolant is that such root clauses be single literals from $[\![\mathcal{V}, \mathcal{E}]\!]_r$, and that the pivot variable in each resolution step belong to the shared alphabet of $[\![\mathcal{V}, \mathcal{E}]\!]_r$ and $[\![\mathcal{S}^{uc}]\!]_r$ (see Section 4.1).*

Example 4.12. *The interpolant of $[\![\mathcal{V}, \mathcal{E}]\!]_r$ and $[\![\mathcal{S}^{uc}]\!]_r$ from Example 4.10 is $I = \mathsf{cl}(\mathsf{s_1}) \wedge \mathsf{cl}(\mathsf{s_2})$, which is fully separable. Notice that I captures the environment's choices that cause the violation:* cl *being always* true *in the looping states s_1 and s_2.* □

Example 4.13. *In the LGS case study, the interpolant obtained from the expressions in Example 4.11 is $I = \mathsf{handle_up}(\mathsf{s_0}) \wedge \mathsf{handle_down}(\mathsf{s_0}) \wedge \mathsf{handle_up}(\mathsf{s^{fail}}) \wedge \mathsf{handle_down}(\mathsf{s^{fail}})$. Notice that it is equal to $[\![\mathcal{V}]\!]_r$, and being fully conjunctive it is fully separable.* □

4.4.4 Interpolant Translation to GR(1) Assumptions

The final step consists in extracting GR(1) assumptions from the interpolant I. This involves a translation from Boolean to LTL, performed by the function **TranslateInterpolant** (line 8). We devise a translation that can be performed in linear time with the size of the interpolant, and produces an LTL formula that is satisfied by the counterrun r. To this aim, the function converts a fully-separable interpolant $I = \bigwedge_{s \in S_r} \beta_s(\mathcal{V}(s))$ into the LTL formula

$$\mathcal{T}(I) := \beta^{init}|_{\mathcal{X}}(\mathcal{X}) \wedge \bigwedge_{s \in S_r} \mathbf{F}\left(\beta_s(\mathcal{V}) \wedge \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right) \wedge \mathbf{FG} \bigvee_{j=1}^{|S^{loop}|} \beta_j^{loop}(\mathcal{V}) \tag{4.3}$$

where the expression $\beta_s(\mathcal{V})$ is obtained by replacing every indexed variable $v(s) \in \mathcal{V}(S)$ in $\beta_s(\mathcal{V}(s))$ with the non-index variable $v \in \mathcal{V}$, the expression $\beta^{init}|_{\mathcal{X}}(\mathcal{X})$ is a shorthand for $\beta_{s^{init}}|_{\mathcal{X}}(\mathcal{X})$ and $\beta_j^{loop}(\mathcal{V})$ for $\beta_{s_j^{loop}}(\mathcal{V})$. Notice that formula (4.3) is computed by scanning the single state components of I, by replacing the variables in $\mathcal{V}(s)$ with the variables in $\mathcal{V}$ by dropping the state index, and by projecting the components onto the input variables where required by the GR(1) template.

The translation consists of three units: a subformula describing the initial state, a conjunction of $\mathbf{F}$ formulae each containing two consecutive state components, and an $\mathbf{FG}$ formula derived from the looping state components of I. If the counterrun is finite, the failing state s^{fail} does not have a successor: in that case we assume $\beta_{\mathsf{succ}(s^{fail})}|_{\mathcal{X}}(\mathbf{X}\mathcal{X}) = \mathsf{true}$ and $\mathbf{F}\left(\beta_{s^{fail}}(\mathcal{V}) \wedge \beta_{\mathsf{succ}\,(s^{fail})}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right)$ reduces to $\mathbf{F}\left(\beta_{s^{fail}}(\mathcal{V})\right)$.

Example 4.14. *The interpolant obtained in Example 4.12 is fully separable and the two state components refer to looping states:*

$$\beta_{s_1}(\mathcal{V}(s_1)) = \mathsf{cl}(\mathsf{s_1}) \quad \beta_{\mathsf{s_2}}(\mathcal{V}(\mathsf{s_2})) = \mathsf{cl}(\mathsf{s_2})\ .$$

The translation in formula (4.3) says that for every pair of consecutive state components an $\mathbf{F}$ *formula is generated; there are two pairs of consecutive states, namely* s_1 *followed by* s_2 *and vice versa, which give rise to two* $\mathbf{F}$ *formulae:*

$$\mathbf{F}\left(\beta_{s_1}(\mathcal{V}) \wedge \beta_{s_2}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right) = \mathbf{F}\left(\mathsf{cl} \wedge \mathbf{X}\mathsf{cl}\right) \quad \mathbf{F}\left(\beta_{s_2}(\mathcal{V}) \wedge \beta_{s_1}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right) = \mathbf{F}\left(\mathsf{cl} \wedge \mathbf{X}\mathsf{cl}\right)\ .$$

The two states also belong to a loop, and therefore the translation produces an $\mathbf{FG}$ *formula:*

$$\mathbf{FG}\left(\beta_{s_1}(\mathcal{V}) \vee \beta_{s_2}(\mathcal{V})\right) = \mathbf{FG}\left(\mathsf{cl} \vee \mathsf{cl}\right) = \mathbf{FG}\mathsf{cl}$$

Notice that the last simplification from $\mathsf{cl} \vee \mathsf{cl}$ *to* cl *is not part of our translation, but a post-processing step.*

Example 4.15. *The LGS counterrun is a finite one,* $r = s_0 s^{fail}$*. This translates into the two*

F *formulae:*

$$\mathbf{F}\left(\beta_{s_0}(\mathcal{V}) \wedge \beta_{sfail}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right) = \mathbf{F}\left(\mathsf{handle_up} \wedge \mathsf{handle_down} \wedge \mathbf{X}\mathsf{handle_up} \wedge \mathbf{X}\mathsf{handle_down}\right)$$

$$\mathbf{F}\left(\beta_{sfail}(\mathcal{V})\right) = \mathbf{F}\left(\mathsf{handle_up} \wedge \mathsf{handle_down}\right)$$

and since the interpolant contains a state component for the initial state, it gives rise to a pure Boolean formula

$$\beta^{init}|_{\mathcal{X}}(\mathcal{X}) = \mathsf{handle_up} \wedge \mathsf{handle_down}\ .$$

Formula (4.3) is guaranteed to hold in the counterplay induced by the counterrun r. Intuitively, since I is fully-separable by construction, the Boolean formula $[\![\mathcal{V}, \mathcal{E}]\!]_r$ implies each state component and its projections onto $\mathcal{X}$ and $\mathcal{Y}$ (we recall that state components and their projections are also Boolean formulae, and implication between logical formulae has been defined in Definition 2.6). A state component $\beta_s(\mathcal{V}(s))$ corresponds to a formula $\beta_s(\mathcal{V})$ satisfied by state s of the counterplay. Therefore, since the initial state satisfies $\beta^{init}(\mathcal{V})$, r satisfies $\beta^{init}|_{\mathcal{X}}(\mathcal{X})$; since there are two consecutive states s and $\mathsf{succ}\,(s)$ that satisfy $\beta_s(\mathcal{V}(s))$ and $\beta_s(\mathcal{V}(\mathsf{succ}\,(s)))$ respectively, r satisfies $\mathbf{F}\left(\beta_s(\mathcal{V}) \wedge \beta_{\mathsf{succ}\,(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right)$. Finally, for the $\mathbf{FG}$ subformula, it is sufficient to observe that the looping state j satisfies the formula $\beta_j^{loop}(\mathcal{V})$: since the counterplay remains indefinitely in each of the looping states, there is a suffix of it where such formula is true for at least one j. Based on these considerations, we prove the following soundness property.

Theorem 4.1. *Let r be a counterplay and $\phi^{\mathcal{E}}$ a set of assumptions satisfied in r, such that their Boolean translation $[\![\mathcal{V}, \mathcal{E}]\!]_r$ implies I, and let I be a fully-separable interpolant. Then $r \models \mathcal{T}(I)$.*

Proof. Since we are assuming that I is state-separable, and since by definition $[\![\mathcal{V}, \mathcal{E}]\!]_r$ implies I, each state component (which is a conjunct in I) is implied by $[\![\mathcal{V}, \mathcal{E}]\!]_r$:

$$[\![\mathcal{V}, \mathcal{E}]\!]_r \rightarrow I \rightarrow \beta_s(\mathcal{V}(s)) \tag{4.4}$$

for every s. By construction, a state component holds true iff it is satisfied by the corresponding

state in the counterrun:

$$[\![\mathcal{V},\mathcal{E}]\!]_r \rightarrow \beta_s(\mathcal{V}(s)) \Leftrightarrow \langle r,s\rangle \models \beta_s(\mathcal{V}) \tag{4.5}$$

Now let us consider s^{init}. By (4.5) and I/O-separability:

$$\begin{aligned}[\![\mathcal{V},\mathcal{E}]\!]_r \rightarrow \beta^{init}(V(s^{init})) \Rightarrow &\langle r,s^{init}\rangle \models \beta^{init}(V)\\ \Rightarrow &\langle r,s^{init}\rangle \models \beta^{init}_{\mathcal{X}}(\mathcal{X}) \wedge \beta^{init}_{\mathcal{Y}}(\mathcal{Y})\\ \Rightarrow &r \models \beta^{init}_{\mathcal{X}}(\mathcal{X})\end{aligned} \tag{4.6}$$

So, r satisfies the part of the translation that refers to the initial state.

The next step is to consider pairs of consecutive states s and $\mathsf{succ}\,(s)$. Since I is I/O-separable, we have

$$\beta_{\mathsf{succ}(s)}(\mathcal{V}(\mathsf{succ}(s))) \rightarrow \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathcal{X}(\mathsf{succ}(s)))$$

By (4.5) and LTL satisfaction definition for the temporal operator $\mathbf{X}$:

$$\begin{aligned}[\![\mathcal{V},\mathcal{E}]\!]_r \rightarrow \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathcal{X}(\mathsf{succ}(s))) \Rightarrow &\langle r,\mathsf{succ}(s)\rangle \models \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathcal{X})\\ \Rightarrow &\langle r,s\rangle \models \mathbf{X}\beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathcal{X})\\ \Rightarrow &\langle r,s\rangle \models \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\end{aligned} \tag{4.7}$$

From the conjunction of (4.5) and (4.7), and from the LTL interpretation of the operator $\mathbf{F}$ we finally get

$$\begin{aligned}[\![\mathcal{V},\mathcal{E}]\!]_r \rightarrow &\beta_s(\mathcal{V}(s)) \wedge \beta_{\mathsf{succ}(s)}(\mathcal{V}(\mathsf{succ}(s)))\\ \Rightarrow &\langle r,s\rangle \models B_s(\mathcal{V}) \wedge \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\\ \Rightarrow &r \models \mathbf{F}\left(\beta_s(\mathcal{V}) \wedge \beta_{\mathsf{succ}\,(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right)\end{aligned} \tag{4.8}$$

Therefore r satisfies the "eventually" subformulae in the translation.

Finally, let us consider the looping and unrolled states. In general, a path ending with a loop

among states $s_1, \ldots, s_{|S^{loop}|}$ satisfies the formula

$$\mathbf{FG} \bigvee_{j=1}^{|S^{loop}|} \beta_j(\mathcal{V}) \tag{4.9}$$

where β_j is any Boolean expression that holds in s_j. The reason is that there is a suffix of the path that contains only states from S^{loop}; therefore, this suffix always satisfies any of the looping states' valuations.

Hence, given

$$\langle r, s_j^{loop} \rangle \models \beta_j^{loop}(\mathcal{V}) \tag{4.10}$$

the expression $\beta_j^{loop}(\mathcal{V})$ can be used in formula (4.9) instead of $B_j(\mathcal{V})$ and we obtain:

$$r \models \mathbf{FG} \bigvee_{j=1}^{|S^{loop}|} \beta_j^{loop}(\mathcal{V}) \tag{4.11}$$

Since r satisfies each of the conjuncts in (4.3), then $r \models \mathcal{T}(I)$. □

In case a fully-separable interpolant is not generated and $\mathcal{T}(I)$ cannot be constructed, the algorithm returns false as its candidate assumption. Otherwise, the approach produces the candidate refinements by negating (4.3) and extracting the disjuncts (function **ExtractDisjuncts**, line 9) in the resulting formula:

$$\neg\beta^{init}|_{\mathcal{X}}(\mathcal{X}) \vee \bigvee_{s \in S_r} \mathbf{G}\neg\left(\beta_s(\mathcal{V}) \wedge \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right) \vee \mathbf{GF} \bigwedge_{j=1}^{|S^{loop}|} \neg\beta_j^{loop}(\mathcal{V}) \tag{4.12}$$

Each disjunct above is a GR(1) candidate assumption which, by Theorem 4.1, ensures the exclusion of the counterplay r from the models of the assumptions.

Example 4.16. *By negating the expression obtained in Example 4.14, one obtains the following assumptions for the request-grant example:*

$$\psi_1 = \mathbf{G}(\mathsf{cl} \rightarrow \mathbf{X}\neg\mathsf{cl}) \quad \psi_2 = \mathbf{GF}(\neg\mathsf{cl}) \ .$$

Notice that none of the two assumptions is sufficient to achieve realizability in the request-grant case study. Therefore, the search algorithm described in Algorithm 3 (see Chapter 3) will search for additional assumptions by calling ***InterpolationBasedGenerateRefinements*** *multiple times.*

Example 4.17. *From the expressions in Example 4.15 the generated assumptions are:*

$$\psi_1 = \mathbf{G}(\mathsf{handle_up} \wedge \mathsf{handle_down}) \rightarrow \mathbf{X}(\neg\mathsf{handle_up} \wedge \mathsf{handle_down})$$

$$\psi_2 = \mathbf{G}(\neg(\mathsf{handle_up} \wedge \mathsf{handle_down}))$$

$$\psi_3 = \neg(\mathsf{handle_up} \wedge \mathsf{handle_down})$$

Notice that $\psi_2 \equiv \mathbf{G}(\mathsf{handle_up} \rightarrow \neg\mathsf{handle_down})$ *alone is sufficient to ensure realizability. It is a weaker version of the assumption we targeted in Example 1.2 (*$\mathbf{G}(\mathsf{handle_up} \leftrightarrow \mathsf{handle_down})$*), but still it is a solution to the problem. This shows how interpolation (and in general automated assumptions refinement methods) allow to discover non-trivial solutions to realizability.*

4.5 Unrolling Looping Counterruns

A common problem to assumptions refinement approaches is that of the *vacuity* of the refined assumptions [BBDER01]. A GR(1) specification is said to be *vacuously realizable* if its assumptions are unsatisfiable. In this case any controller satisfies the specification, since the assumptions evaluate to false. In some cases, assumptions refinement approaches trivially eliminate counterstrategies with a refinement Ψ that is inconsistent with previous assumptions, making the specification vacuously realizable.

Interpolation-based synthesis produces a number of assumptions that grows proportionally with the number of state components in the interpolant, and thereby with the number of states in the counterrun. If the counterrun is small, few refinements are produced and all of them may be inconsistent with the original assumptions. In this case, counterrun unrolling can help producing additional assumptions.

Algorithm 9: InterpolationBasedGenerateRefinements($\varphi^{\mathcal{E}}$, $\varphi^{\mathcal{S}}$, C) with unrolling.

Data: $\phi^{\mathcal{E}}$, environment assumptions
Data: $\mathcal{S}^{uc}$, controller guarantees (in an unrealizable core)
Data: c, counterstrategy
Result: altRefs, alternative assumptions eliminating the counterstrategy

```
1  r := ExtractCounterrun(c);
2  u := 0;
3  r_{c,u} := r;
4  altRefs_old := ∅;
5  repeat
6  |  [[V, E]]_r := TranslateCounterrunAssumptions(r_{c,u}, φ^E);
7  |  [[S^uc]]_r := TranslateGuarantees(r_{c,u}, S^uc);
8  |  I_u := Interpolate([[V, E]]_r, [[S^uc]]_r);
9  |  if I_u = false or I_u is not fully-separable then
10 |  |  altRefs := {false};
11 |  |  stopping_condition := CheckStoppingCondition(altRefs, altRefs_old, u);
12 |  else
13 |  |  T(I_u) := TranslateUnrolledInterpolant(r_{c,u}, I_u);
14 |  |  altRefs := ExtractDisjuncts(¬T(I_u));
15 |  |  if r_{c,u} is looping then
16 |  |  |  altRefs_old := altRefs;
17 |  |  |  u := u + 1;
18 |  |  |  r_{c,u} := UnrollCounterrun(r,u);
19 |  |  |  stopping_condition := CheckStoppingCondition(altRefs, altRefs_old, u);
20 |  |  else
21 |  |  |  stopping_condition := true;
22 |  |  end
23 |  end
24 until stopping_condition;
25 return altRefs;
```

Counterrun unrolling consists in making the first traversals of looping states explicit. It is achieved by augmenting a counterrun with replicates of the looping states. The number of unrollings is referred to as the *unrolling degree* u. Each unrolling yields a new set of states $S^{unr} = \{s^{unr}_{1,1}, \ldots, s^{unr}_{h,1}, \ldots, s^{unr}_{1,u}, \ldots, s^{unr}_{h,u}\}$ called *unrolled states*. An unrolled looping counterrun has the form $s^{init} s^{trans}_1 \ldots s^{trans}_k \; s^{unr}_{1,1} \ldots s^{unr}_{h,1} \ldots s^{unr}_{1,u} \ldots s^{unr}_{h,u} (s^{loop}_1 \ldots s^{loop}_h)^{\omega}$. In our notation, $s^{unr}_{i,j}$, the index i denotes that the state is the replicate of the looping state s^{loop}_i, while the index j is used to enumerate the different replicates of the same state. Figure 4.8 illustrates the effect of unrolling on a counterrun.

Unrolling has two possible effects on the computed interpolant: on one hand, it can introduce

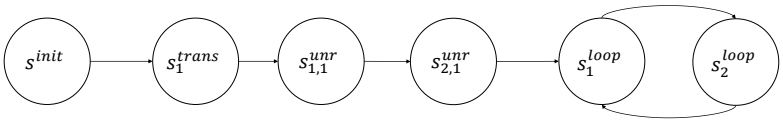


Figure 4.8: Unrolling of the looping counterrun in Figure 4.7. The unrolling degree is 1.

new state components in the interpolant, which yield new invariant refinements according to (4.12); on the other hand, the interpolant can express a more specific characterization of looping states, which corresponds to a weaker fairness refinement in (4.12). These effects are both observed in our evaluation (see Section 4.6).

Algorithm 9 shows how unrolling is integrated in the interpolation-based synthesis algorithm. It is the same as Algorithm 8 with the addition of unrolling. The translation from the temporal to the Boolean domain (lines 6-7) works as in Algorithm 8. When an interpolant is computed, provided that it is fully separable, it can contain components referring to the unrolled states. The state component referring to the r-th replica of the j-th looping state is denoted by $\beta_{j,r}^{unr}(\mathcal{V}(s_{j,r}^{unr}))$. The function **TranslateUnrolledInterpolant** (line 13) produces the formula

$$\begin{aligned}\mathcal{T}(I_u) :=& \beta^{init}|_{\mathcal{X}}(\mathcal{X}) \wedge \bigwedge_{s\in S_r} \mathbf{F}\left(\beta_s(\mathcal{V}) \wedge \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right) \wedge \\ & \mathbf{FG} \bigvee_{j=1}^{|S^{loop}|} \left(\beta_j^{loop}(\mathcal{V}) \wedge \bigwedge_{r=1}^{u} \beta_{j,r}^{unr}(\mathcal{V})\right)\end{aligned} \tag{4.13}$$

This formula is the same as (4.3) apart from the **FG** conjunct. This is replaced with a stronger disjunction, where each disjunct groups the state components referring to all the replicates of the same looping state. By negating (4.13), the algorithm produces

$$\begin{aligned}\neg\beta^{init}|_{\mathcal{X}}(\mathcal{X}) \vee & \bigvee_{s\in S_u} \mathbf{G}\neg\left(\beta_s(\mathcal{V}) \wedge \beta_{\mathsf{succ}(s)}|_{\mathcal{X}}(\mathbf{X}\mathcal{X})\right) \vee \\ & \mathbf{GF} \bigwedge_{j=1}^{|S^{loop}|} \neg\left(\beta_j^{loop}(\mathcal{V}) \wedge \bigwedge_{r=1}^{u} \beta_{j,r}^{unr}(\mathcal{V})\right)\end{aligned} \tag{4.14}$$

where each disjunct is a candidate refinement, and the fairness condition has a weaker form

than the one in (4.12).

Unrolling is stopped when some user-defined stopping condition is reached. This is checked by the function **CheckStoppingCondition** in line 19. Possible stopping conditions are:

- the interpolant has yielded the same refinements as the previous step;
- no new refinements have been produced in the last k unrolling step;
- a maximum unrolling degree has been specified.

4.6 Detailed Example: The Lift Controller

Due to its small size that allows a thorough description, we use the lift controller case study from [AMT13, BJP+12] as a complete example of the application of our approach, including unrolling. This description extends the one presented in [Cav16]. Variations on this example are also present in other software engineering applications like LTL specification mining from operational scenarios [VW98].

The case study involves the specification of a system comprising a lift controller. The lift moves between three floors. The environment consists of three buttons, whose states can be *pressed* or *unpressed*. The corresponding state is represented by three binary input variables $\{b_1, b_2, b_3\}$. The controller's state consists of three output variables $\{f_1, f_2, f_3\}$ that indicate at which floor the lift is. The assumptions are:

1. $\theta^{\mathcal{E}} = \neg b_1 \wedge \neg b_2 \wedge \neg b_3$
2. $\rho^{\mathcal{E}}_{1,i} = \mathbf{G}(b_i \wedge f_i \rightarrow \mathbf{X}\neg b_i)$
3. $\rho^{\mathcal{E}}_{2,i} = \mathbf{G}(b_i \wedge \neg f_i \rightarrow \mathbf{X} b_i)$

for $i \in \{1, 2, 3\}$. They state that the buttons are not pressed in the initial state (1); a pressed button transits to a non-pressed state when the lift arrives at the corresponding floor (2); and

the button remains in the pressed state until the lift arrives at that floor (3). The guarantees are:

1. $\theta^{\mathcal{S}} = f_1 \wedge \neg f_2 \wedge \neg f_3$
2. $\rho_1^{\mathcal{S}} = \mathbf{G}(\neg(f_1 \wedge f_2) \wedge \neg(f_2 \wedge f_3) \wedge \neg(f_1 \wedge f_3))$
3. $\rho_{2,1}^{\mathcal{S}} = \mathbf{G}(f_1 \rightarrow (\mathbf{X}f_1 \vee \mathbf{X}f_2))$
4. $\rho_{2,2}^{\mathcal{S}} = \mathbf{G}(f_2 \rightarrow (\mathbf{X}f_1 \vee \mathbf{X}f_2 \vee \mathbf{X}f_3))$
5. $\rho_{2,3}^{\mathcal{S}} = \mathbf{G}(f_3 \rightarrow (\mathbf{X}f_2 \vee \mathbf{X}f_3))$
6. $\rho_3^{\mathcal{S}} = \mathbf{G}(((f_1 \wedge \mathbf{X}f_2) \vee (f_2 \wedge \mathbf{X}f_3) \vee (f_2 \wedge \mathbf{X}f_1) \vee (f_3 \wedge \mathbf{X}f_2)) \rightarrow (b_1 \vee b_2 \vee b_3))$
7. $\rho_{4,i}^{\mathcal{S}} = \mathbf{GF}(b_i \rightarrow f_i)$
8. $\rho_{5,i}^{\mathcal{S}} = \mathbf{GF}f_i$

for $i \in \{1, 2, 3\}$. They state that the lift starts from floor 1 (1); it can never be in two floors at the same time (2); it can move only between consecutive states (3-5), and moves only when at least a button is pressed (6); plays in which the environment keeps a button b_i pressed infinitely and the lift never reaches the corresponding f_i are forbidden (7); and that the lift is required to visit all the floors infinitely often (8). Given this specification, the fairness guarantee can be satisfied if the environment sets each one of its b_i to 1 .

The specification is unrealizable, since when the buttons (environment) stay indefinitely unpressed, the lift (controller) cannot move and therefore $\rho_{5,2}^{\mathcal{S}}$ and $\rho_{5,3}^{\mathcal{S}}$ are violated. The unrealizable core consists of the whole set of assumptions and the guarantees $\theta^{\mathcal{S}}$, $\rho_{2,1}^{\mathcal{S}}$, $\rho_3^{\mathcal{S}}$ and $\rho_{5,2}^{\mathcal{S}}$. From this core, the counterstrategy c in Fig. 4.9 is computed, consisting of a unique run r. After translating the unrealizable core over the counterplay, the interpolant is $I_0 = \neg b_1(s_0) \wedge \neg b_2(s_0) \wedge \neg b_3(s_0)$. This translates back into the temporal domain as $\mathcal{T}(I_0) = \neg b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \mathbf{F}(\neg b_1 \wedge \neg b_2 \wedge \neg b_3)$. By negating this formula, we obtain the GR(1) assumptions $\psi_1 = b_1 \vee b_2 \vee b_3$ and $\psi_2 = \mathbf{G}(b_1 \vee b_2 \vee b_3)$. However, these assumption are both inconsistent with $\theta^{\mathcal{E}}$, and therefore make the specification vacuously realizable.

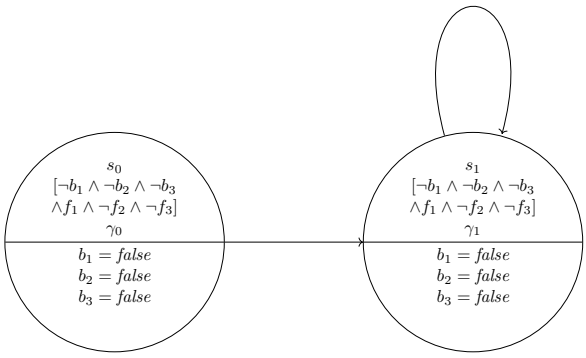


Figure 4.9: Lift counterstrategy

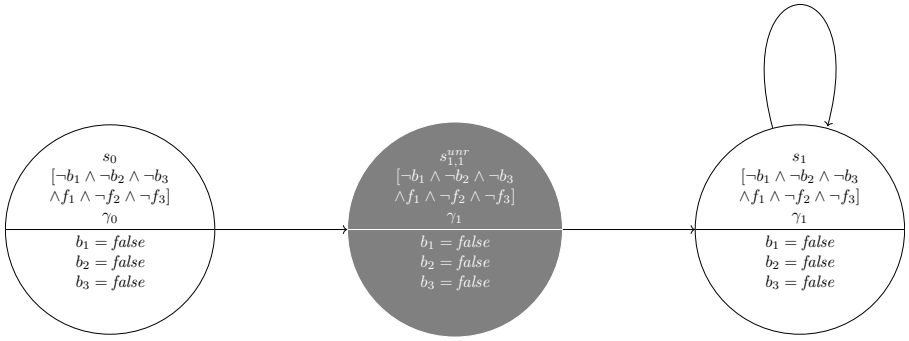


Figure 4.10: Counterrun extracted from the counterstrategy in Fig. 4.9 with one unrolling. The new state generated by the unrolling is marked in gray.

We resort to unrolling for obtaining more alternative refinements. The resulting counterrun $s_0 s_{1,1}^{unr} s_1$ is shown in Fig. 4.10. After unrolling, the procedure yields the interpolant $I_1 = \neg b_1(s_0) \wedge \neg b_2(s_0) \wedge \neg b_3(s_0) \wedge \neg b_1(s_{1,1}^{unr}) \wedge \neg b_2(s_{1,1}^{unr}) \wedge \neg b_3(s_{1,1}^{unr})$. By translating and negating this interpolant, we obtain the refinements

1. $\psi_1 = b_1 \vee b_2 \vee b_3$
2. $\psi_2 = \mathbf{G}(\neg b_1 \wedge \neg b_2 \wedge \neg b_3 \rightarrow \mathbf{X}(b_1 \vee b_2 \vee b_3))$
3. $\psi_3 = \mathbf{GF}(b_1 \vee b_2 \vee b_3)$

Notice that unrolling results in an interpolant containing an additional state component, thus allowing for more alternative assumptions (see Section 4.5). Moreover, the new state component refers to an unrolled state, from which a new fairness assumption, not inferable from I_0, is generated.

Every candidate assumption computed by our approach eliminates an unrealizable core. More-

over, each one solves the unrealizability problem for the original specification. The assumption ψ_1 does this in a trivial way, since it contradicts the initial assumption contained in the specification. Notice that all the computed refinements force at least one of the buttons to be pressed at in any play of the environment. This corresponds to the refinement output by the approach in [AMT13].

4.7 Convergence

The counterstrategy-guided loop of Algorithm 3 is guaranteed to terminate in finite time provided that

- the bias heuristic used in line 8 produces a finite number of assumptions at any call;
- the loop at lines 3-14 executes a finite number of iterations before hitting the termination condition $\mathsf{candidateRefs} = \varnothing$.

Both conditions hold in the interpolation bias. The first one derives from the finiteness of formula (4.12); the second can be shown by finding an upper bound to the number of nodes in the refinement tree, since it corresponds to the number of iterations of the loop (see Section 4.5). The argument we propose here to find this upper bound is an instantiation of the one used in [LMN16] to show convergence in the more general framework of Abstract Learning.

Theorem 4.2. *Given a satisfiable but unrealizable specification $\langle \mathcal{E}, \mathcal{S} \rangle$ Algorithm 3 equipped with the interpolation bias of Algorithm 8 in line 7 eventually terminates with a set of n_{sol} realizable specifications $\langle \mathcal{E} \cup \Psi_i, \mathcal{S} \rangle$, for every $i \in \{1, \ldots, n_{sol}\}$.*

In this formulation, notice that some of the solutions may be equivalent to the constant false; this happens when some of the interpolants are not fully-separable, or the generated refinements are inconsistent with the original specification (see Section 4.6).

Proof. In the following arguments, we refer to the refinement tree of Algorithm 3. Each node is associated with the candidate assumption tested in one specific iteration of the counterstrategy-guided loop. The root corresponds to the initial assumptions; every internal node symbolizes an insufficient refinement; the children of an internal node correspond to the alternative refinements that rule out the relevant counterstrategy. The leaves represent alternative sufficient refinements returned by Algorithm 3. We will show that this tree has finite depth and breadth.

As regards breadth, given an internal node, consider the number of its children n_c. It corresponds to the number of refinements that are generated from a single counterstrategy by applying the interpolation bias. Each of these is a disjunct in Formula (4.12); the total number of disjuncts comprises one initial condition, one fairness condition and $|S_r|$ invariants; therefore:

$$n_c = |S_r| + 2 \; .$$

Therefore, every time a node is expanded, a finite number of children is generated, proportional to the number of states in the counterrun.

We now consider the depth. The algorithm keeps refining a computed assumption until the property becomes realizable (in case the returned refinement is *false*, then the property is vacuously realizable). Consider an internal node Ψ; this gets expanded only in case it is satisfiable and not realizable (Algoritm 3, line 5), and a counterstrategy c and a set of children ψ_i are generated. Given the soundness property (Theorem 4.1), every ψ_i excludes the latest computed counterstrategy, so $c \not\models \psi_i$; at the same time by construction $c \models \psi$ for every $\psi \in \Psi$. Therefore, no node is equivalent to any of its direct children. By a simple induction step, no node is equivalent to any of its descendants.

From this descends that, given two nodes Ψ and Ψ' along any same path from the root to some leaf of the tree, they are not equivalent.

For the above reason, the depth d of the refinement tree, which is the maximum number of nodes along a single path, is bounded by the maximum number of non-equivalent GR(1) formulae.

The maximum number of initial conditions is

$$d_{init,MAX} = 2^{2^{|\mathcal{X}|}} ,$$

that is the number of all distinct (that is, semantically non-equivalent) Boolean expressions over the input variables. The maximum number of invariants is

$$d_{inv,MAX} = 2^{2^{|\mathcal{V}|}+2^{|\mathcal{X}|}} ;$$

this corresponds to the maximum number of distinct β_s that can be present in the expression (4.12) times the number of distinct $\beta_{\mathsf{succ}(s)}|_{\mathcal{X}}$. Finally, the maximum number of distinct fairness assumptions is

$$d_{fair,MAX} = 2^{2^{|\mathcal{V}|}} .$$

Therefore, the total depth d is bounded by the sum of these three quantities:

$$d \leq d_{MAX} = d_{init,MAX} + d_{inv,MAX} + d_{fair,MAX} .$$

Since the tree is finite in depth and breadth, it is finite. □

Given the above, we conclude that the refinement tree is finite. This gives us a worst-case upper bound on the depth d of the tree, which has a doubly exponential growth over $|\mathcal{V}|$. The same termination guarantee also holds for the approaches in [AMT13, LDS11]: the key point is that the refinements generated at each node expansion eliminate at least some behaviours from the language of the assumptions; using the terminology of [LMN16] and calling *learner* the function **CounterstrategyGuidedRefinement**, we say that the learner makes *progress*.

The bounds on both the breadth and the depth are very large and there is usually no hope of generating the entire refinement tree. However, in several case studies some solutions can already be found at much shallower nodes than the maximum depth. As we discussed in Section 3.4, in existing heuristics like varbias an upper bound is imposed as a user-defined

parameter to both breadth and depth [AMT13]. With our interpolation bias, we do not need to impose an upper bound to breadth: the maximum upper bound depends on the number of states in the selected counterrun, which is usually much smaller than the doubly exponential upper bound.

4.8 Evaluation

In this Section, we report on applications of our proposed approach to a number of case studies. We consider the popular benchmark presented in [AMT13, BJP+12, KHB09], the ARM's AMBA-AHB protocol, and the variants of four case studies provided by [KMR17].

The study attempts to answer the following research questions:

RQ1. Can our approach find at least one solution to some realizability problem fully automatically?

RQ2. How effective is interpolation in finding solutions compared to the state of the art?

RQ3. What impact does the full separability requirement have on interpolants on finding solutions?

RQ4. How does the size of the specification affect the effectiveness?

We compare our approach to Alur's one described in [AMT13], which at the time of performing this work (2017) was the state-of-the-art approach using counterstrategies to find refinements. Both approaches are iterative, based on the counterstrategy-guided loop of Algorithm 3, but they differ in the bias heuristic. Alur's *varbias*, described in detail in Section 3.4.2, constructs a set of formulae templates based on the counterstrategy graph, and fills it in with a set of variables provided in input by the user. Since in our setting we assume the user has no knowledge of the variables to use in order to reach a solution, we modify Alur's approach such that the choice is performed at random: specifically, at every call to the routine of assumptions generation, 5 different random variable choices are used to generate assumptions. We label this approach

multivarbias, to remark the fact that this search is biased towards the assumptions containing the variables randomly selected, and that multiple choices are made for these variables at each step.

The requirements analysis tool RATSY [BCG+10] is used to check unrealizability and compute counterstrategies. The SAT solver MathSAT [CGS08, BCF+08] is used to compute interpolants. The refinement synthesis procedure is implemented in Python 2.7. The experiment is executed on a dedicated Ubuntu machine with an Intel Core i7 CPU and 16 GiB of memory.

4.8.1 Experiment 1

Experimental setup. The first experiment is aimed at answering the RQ1 and RQ2. We apply the interpolation-based approach and multivarbias to the AMBA and the (original and additional) SYNTECH case studies in a large portion of the variants from [MRS19]. Each test consists of two phases:

- In the first phase, Algorithm 3 is run as described with a timeout of 10 minutes. This is the same duration as imposed in the experimental section of [MRS19];
- After the timeout, any refinement still remaining in candidateRefQueue is checked for realizability, but no new refinements are generated nor are new counterstrategies computed; this additional phase has also a timeout of 10 minutes. Any refinement not checked for realizability after this timeout is discarded and not considered in our results.

The metrics of comparison are:

- how many case studies can be solved by each of the two approaches, the same metric as in the first experiment of [MRS19];
- a novel metric of *effectiveness* we propose, defined as

$$effectiveness = \frac{\#solutions}{\#refinements\ explored}.$$

This definition estimates the probability of hitting some solution in any explored node, and formalizes the quality criterion for the search algorithm defined in Section 3.3.

Regarding the AMBA, we test the two approaches on several variants of the unrealizable case studies from [CRST08]:

- the case obtained by removing the fairness condition **GF**hready from the realizable specification;
- the cases obtained by removing one of the invariant assumptions;
- the cases obtained by removing all of the invariant assumptions;
- the case obtained by removing all invariants and the fairness condition;
- the case in which all assumptions (including initial conditions) are removed.

By performing in turn each of these removals on the three AMBA specifications, we obtain a total of case studies on which we test. Details on these case studies are given in Table 4.1.

Regarding the SYNTECH case studies, their specifications are available in the syntax of the synthesis tool Spectra [MRS19]. In order to make them suitable for our toolchain, we first transformed Spectra files into RAT files by using an automated translator implemented by us. Spectra allows for syntactic sugar to be used in specifications, which is not supported by RATSY. In particular:

- Spectra supports enumerative variables. We translate these into arrays of Boolean variables;
- Spectra supports macros for expressing Dwyer's specification patterns [DAC99] without explicitly referring to auxiliary variables. Our translation expands these macros;
- Spectra's syntax supports operations over integers like addition and subtraction, which are internally converted into operations over Booleans. Our automated translator does not

support those operators. Therefore we exclude from our experiment those specifications that make use of such operators.

We perform the experiment on 14 out of the 15 original SYNTECH case studies from [KMR17], and 78 of the 136 additional ones from [MRS19]. The information on the size of these case studies, grouped by the systems the specifications refer to, are shown in Tables 4.1-4.3.

Table 4.1: AMBA case studies for experiment 1.

	#In	#Out	#A	#G
amba02_nofairness	7	16	10	66
amba02_nosafety0	7	16	10	66
amba02_nosafety1	7	16	10	66
amba02_nosafety	7	16	9	66
amba02_nosafety_nofairness	7	16	8	66
amba02_noassumptions	7	16	1	66
amba04_nofairness	11	23	16	97
amba04_nosafety0	11	23	16	97
amba04_nosafety	11	23	13	97
amba04_nosafety_nofairness	11	23	12	97
amba04_noassumptions	11	23	1	97
amba08_nofairness	19	36	28	158
amba08_nosafety0	19	36	28	158
amba08_nosafety	19	36	21	158
amba08_nosafety_nofairness	19	36	20	158
amba08_noassumptions	19	36	1	158

Table 4.2: Original JVTS case studies for experiment 1

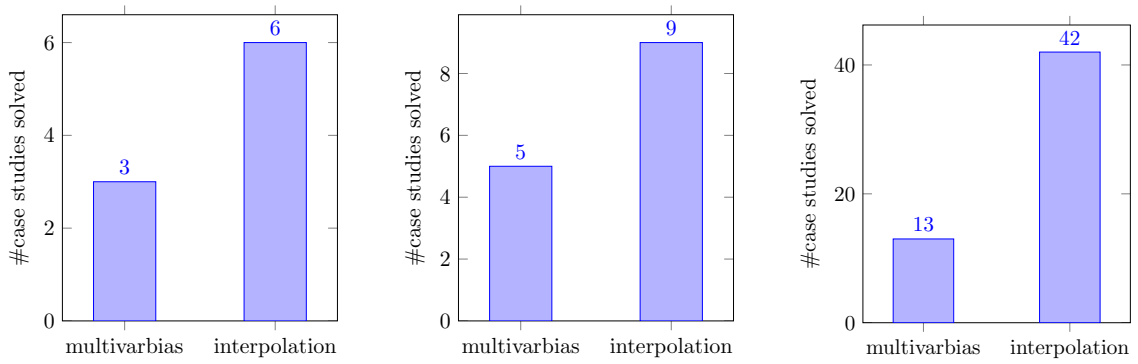
	#Variants	Min-Avg-Max #In	Min-Avg-Max #Out	Min-Avg-Max #A	Min-Avg-Max #G
ColorSort	3	17-18-20	15-21-30	5-6-9	36-45-55
Gyro	2	5-6-7	3-4-5	6-7-8	7-9-11
Humanoid	8	1-4-9	10-22-36	1-2-5	11-30-46
PCar	1	3-3-3	11-11-11	4-4-4	24-24-24

Table 4.3: Additional JVTS case studies for experiment 1

	#Variants	Min-Avg-Max #In	Min-Avg-Max #Out	Min-Avg-Max #A	Min-Avg-Max #G
ColorSort	39	14-17-22	13-23-29	2-4-10	26-44-53
Gyro	22	2-6-9	3-3-4	2-3-6	6-8-14
Humanoid	12	1-4-7	10-22-31	1-2-5	11-33-43
PCar	5	3-4-5	4-6-9	1-1-2	3-9-18

Results and discussion. Figure 4.11 shows the result of the comparison. The bars represent how many case studies were solved in each group by the two approaches, where we say that a case study is solved by an approach if it finds at least one solution. In all the three groups of specifications, interpolation solves significantly more case studies than multivarbias. This is because interpolation targets the cause of unrealizability without need of prior knowledge, while multivarbias generates several refinements guided by a random choice of variables irrespective

of the cause of unrealizability. Therefore, this answers RQ1 positively for a large portion of case studies.



(a) AMBA (b) Original SYNTECH (c) Additional SYNTECH

Figure 4.11: Case studies solved by each of the two approaches

Figure 4.12 shows for each group of case studies (AMBAs, original SYNTECH and additional SYNTECH) (1) the total number of solutions found, (2) the number of non-redundant solutions (where as *redundant solution* we mean a solution stronger than another one, or equivalent to another one), and (3) the effectiveness as defined above per case study and per approach.

The plots show that, except in one case study of the additional SYNTECH, the effectiveness of interpolation is consistently higher than multivarbias. This can be interpreted in the following way: whenever a refinement is generated via interpolation, the probability of it being a solution is higher than that of multivarbias. The plots on the lefthand side also show that in most case studies interpolation finds more solutions.

Multivarbias only finds more solutions than interpolation in the amba02_nofairness case study, one variant of Humanoid in the original SYNTECH case studies, several variants of ColorSort and one of Humanoid in the additional case studies. However, after removing redundant solutions in both approaches, in several of those case studies the number of remaining solutions is the same for both (see the middle plots). This remarks that in most cases not only does interpolation find at least as many solutions as randomly exploring the search space relative to the number of explored nodes, but also in absolute value.

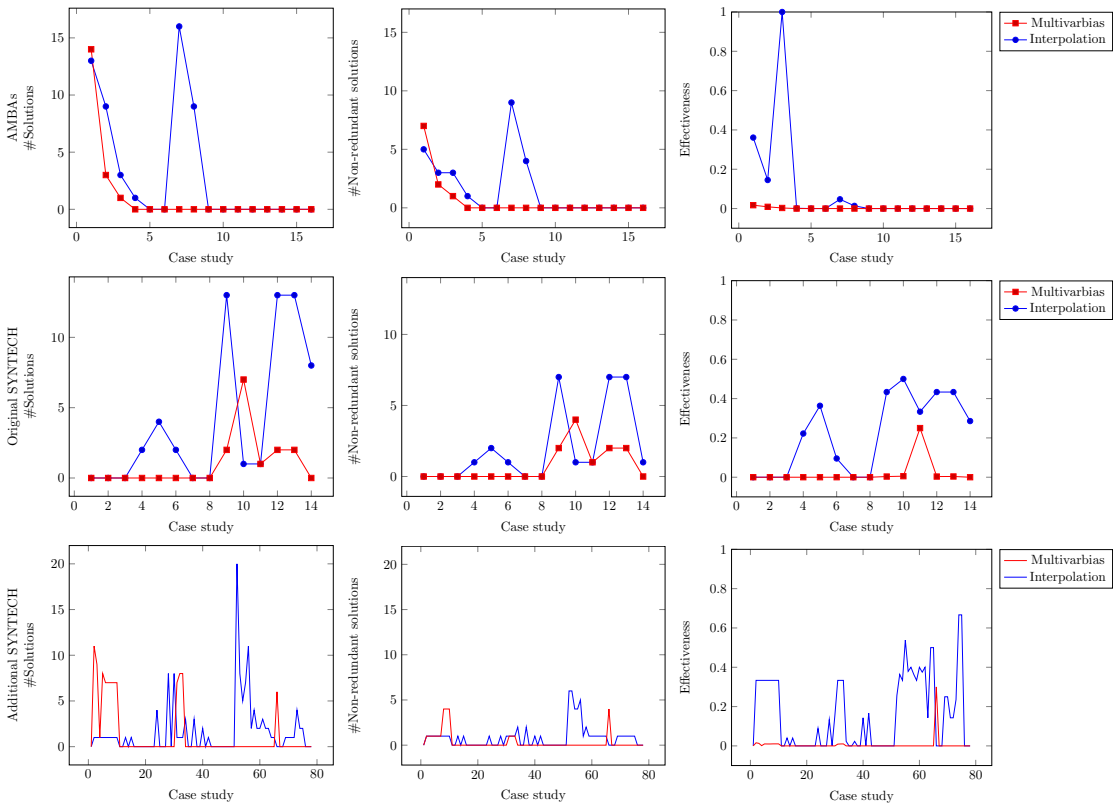


Figure 4.12: Number of solutions, of non-redundant solutions, and effectiveness of interpolation and multivarbias in each group of case studies

The only case study where interpolation is less effective than multivarbias is a variant of Humanoid. In this one, 3 insufficient refinements are generated in the first iteration of Algorithm 3. However, the interpolants generated from each of these refinements are not fully separable, hence the search cannot continue and no solutions are found. Multivarbias, instead, generates 6 solutions, out of which 4 are non-redundant.

To answer RQ3, we count how many times interpolation produced non-fully-separable interpolants across the three groups of case studies. Table 4.4 summarizes the results. In all the case studies, a very small fraction of interpolants are not state separable, both considering the total values across the case studies, and per single case study; such interpolants prevent the search to progress, since our translation from Boolean to temporal logic requires the state separability of the interpolant (see Section 4.1). However, in some cases a non-state-separable interpolant is generated in the first iteration of the search: this causes its termination without generating any candidate refinement.

Table 4.4: Statistics on full separability of the interpolants per group of case studies. *#Interpolants* shows the total number of interpolants computed in all the case studies of the group; *#NonStateSeparable* shows how many interpolants were not state separable; *Avg #NonStateSeparable/#Interpolants* shows the average fraction of non-state-separable interpolants across the case studies in a group.

	AMBAs	Original SYNTECH	Additional SYNTECH
#Interpolants	114	91	974
#NonStateSeparable	6	8	7
Avg #NonStateSeparable/#Interpolants	0.158482	0.035714	0.01038

In summary, in this experiment we demonstrate that interpolation has a high probability of hitting at least one solution in a case study when no prior information on the variables to use in the refinements is available. The higher effectiveness than multivarbias indicates that at each execution of Algorithm 8 the probability of generating at least one solution is higher too. The full-separability requirement rarely hinders the search.

Limitations. Despite the fact that the results in Figure 4.11 use the same metric as [MRS19], we choose not to compare our results directly with those in the cited work. The reason is that the translation from the Spectra syntax to the RATSY one may introduce semantic differences between the original specification and our translation. Our only criterion to accept the translated ones is that they are correctly parsed by RATSY into unrealizable specifications. Therefore, the difference in the numbers of Figure 4.11 and [MRS19] may be due to differences in performance between interpolation and their approach, but may also be due to this nuisance factor.

A threat to the validity of this evaluation, also common to the original works [CRST08, KMR17, MRS19], derives from using different variations of a specification as case studies. This hinders the generalization power of the evaluation. Indeed, common causes of success/failure of some approach are repeated across different versions of the same specification. This can make an approach seem better or worse if the specifications where the approach succeeds has more or less versions respectively. However, the choice of generating the case studies in this way is justified by the lack of GR(1) specifications in the literature. Educating engineers to write specifications of systems in GR(1) is part of the goals of the SYNTECH project [MR18] and could solve this problem in the future.

Table 4.5: Case studies for experiment 2

Case study	#Input	#Output	#A	#G
AMBA02_no_fairness	7	16	10	66
AMBA04_no_fairness	11	23	16	97
AMBA08_no_fairness	19	36	28	158
ColorSort	18	12	10	21
Humanoid	6	16	0	23
Gyro	6	4	10	8

4.8.2 Experiment 2

Experiment setup. Experiment 2 replicates the same settings as experiment 1, with a time-out of 24 hours for the first phase and 2 hours for the second phase.

This experiment has the following goals:

- To check how the effectiveness of the interpolation approach compares with the baseline over a large number of nodes explored (hence providing further evidence to answer RQ2). Indeed, if interpreted as an estimator of the probability of hitting a solution, by the law of large numbers it will converge to the estimated probability;
- To analyze the scalability of the approach (RQ4); in experiment 1 we observed that neither interpolation nor multivarbias could find a solution in the AMBA08_nofairness case study, that is considerably larger than the others in terms of number of variables and assumptions and guarantees. Our goal is to check how many solutions can be found by the two approaches when allowing more time.

During the search we record the time at which every refinement node is generated, whether or not the refinement makes the specification realizable, whether it eliminates the parent's unrealizable core, and whether the refinement is satisfiable or is equivalent to the constant false (see Section 4.5 about vacuously realizable specifications).

We perform a selection of 6 case studies from the previous sets, one per system. The selected case studies are summarized in Table 4.5. Notice we exclude PCar from this experiment since neither interpolation nor multivarbias found solutions for this system.

Results. Table 4.6 summarizes the key figures of the refinement trees generated by both approaches. In all cases, multivarbias manages to produce more nodes than interpolation in the allotted time. This is due to three causes mainly: (*i*) interpolation yields fewer nodes from every single counterstrategy than random template instantiation; this is witnessed by the value of AvgChld, which is significantly higher for multivarbias than interpolation; (*ii*) more counterstrategies need to be computed in interpolation rather than multivarbias; the number of counterstrategies corresponds to NumInternalNodes in Table 4.6, higher for interpolation than multivarbias in four case studies out of six; and (*iii*) computing interpolants requires more time than performing five random variable choices. However, in four out of six case studies, interpolation finds more solutions than multivarbias. Moreover, in AMBA08 and ColorSort only interpolation could find some solutions. For ColorSort, at the time of this experiment (2018) our approach was the first that found a realizable refinement (now, as said earlier, solutions can be found also by applying [MRS19]).

In GyroAspect and Humanoid, multivarbias finds 905 and 162 solutions respectively, compared with the 2 and 3 solutions of interpolation. However, when relating these numbers to the total number of nodes generated by the two approaches, it is worth noting that multivarbias appears less efficient than interpolation. Indeed, almost every node generated via interpolation is a solution node, while multivarbias explores around 2000 nodes to generate less than 1000 solutions. The reason why interpolation generates few nodes in the two case studies is that in a few seconds it reaches all realizable leaves of its refinement tree, and therefore no more expansions are needed. Multivarbias, instead, proceeds until the 24-hour timeout. It is also worth noting that in each case study the method that finds more solutions also hits more often an unsatisfiable refinement (row NumFalse in Table 4.6).

The depth and number of operators in a solution are indicators of the length of the formula syntactically, which is related to its understandability by a human operator. In the AMBA case studies, there is no significant difference on the average number of operators between the two approaches; however, interpolation finds some solutions that are significantly lengthier than the ones of multivarbias; in particular, in AMBA02 some interpolation solutions reach 28 Boolean operators, while multivarbias has at most 15 of them.

The number of distinct variables in the refinements provide a measurement of the portion of the model constrained by the refinement, if one is to measure such portion as the number of variables appearing in the assumptions divided by the total number of variables. There are no significant differences between the two case studies on average, although MaxVars has greater values in multivarbias in 3 out of 4 case studies where it found a solution; this means that those solutions constrain a larger portion of the model than their interpolation counterparts.

Figures 4.13-4.15 show the behavior of the search over time. Figure 4.13 shows the total number of refinement tree nodes searched by both approaches. Notice that in all case studies a larger number of nodes is explored by multivarbias than with interpolation. As already pointed out, on the GyroAspect and Humanoid case studies the interpolation-based approach terminates after less than 10 seconds, while multivarbias continues for the whole time frame of the experiment.

Figure 4.14 shows the number of realizable refinements discovered over time. In all case studies, the curve of interpolation remains above the one of multivarbias for the entire time frame where both approaches are executing. Therefore, it appears evident that interpolation is more focused on realizable nodes than multivarbias. This observation is corroborated by Figure 4.15, which shows the effectiveness over time. The interpolation approach settles around a value bigger than 15% in the first four case studies, while multivarbias remains close to 0%. In addition, the latter does not find any realizable refinement at all in the AMBA08 and ColorSort case studies, despite using more than one variable choice at every synthesis step. To our knowledge, our interpolation-based approach is the first one to find a solution for these case studies.

Discussion. A reason for the efficiency of interpolation can be sought in its addressing unrealizable cores directly. Given a GR(1) specification $\langle \phi^{\mathcal{E}}, \phi^{\mathcal{S}} \rangle$ with unrealizable core $\mathcal{S}^{uc}$, and an assumption ψ, we say that ψ *targets* $\mathcal{S}^{uc}$ if and only if the specification $\langle \phi^{\mathcal{E}} \cup \{\psi\}, \mathcal{S}^{uc} \rangle$ is realizable. Intuitively, any unrealizable core must be targeted by a refinement that solves the unrealizability problem. Therefore, when building the refinement tree it is desirable that at least an unrealizable core be targeted by a partial refinement.

Table 4.7 shows the percentage of nodes targeting their respective parent's unrealizable core

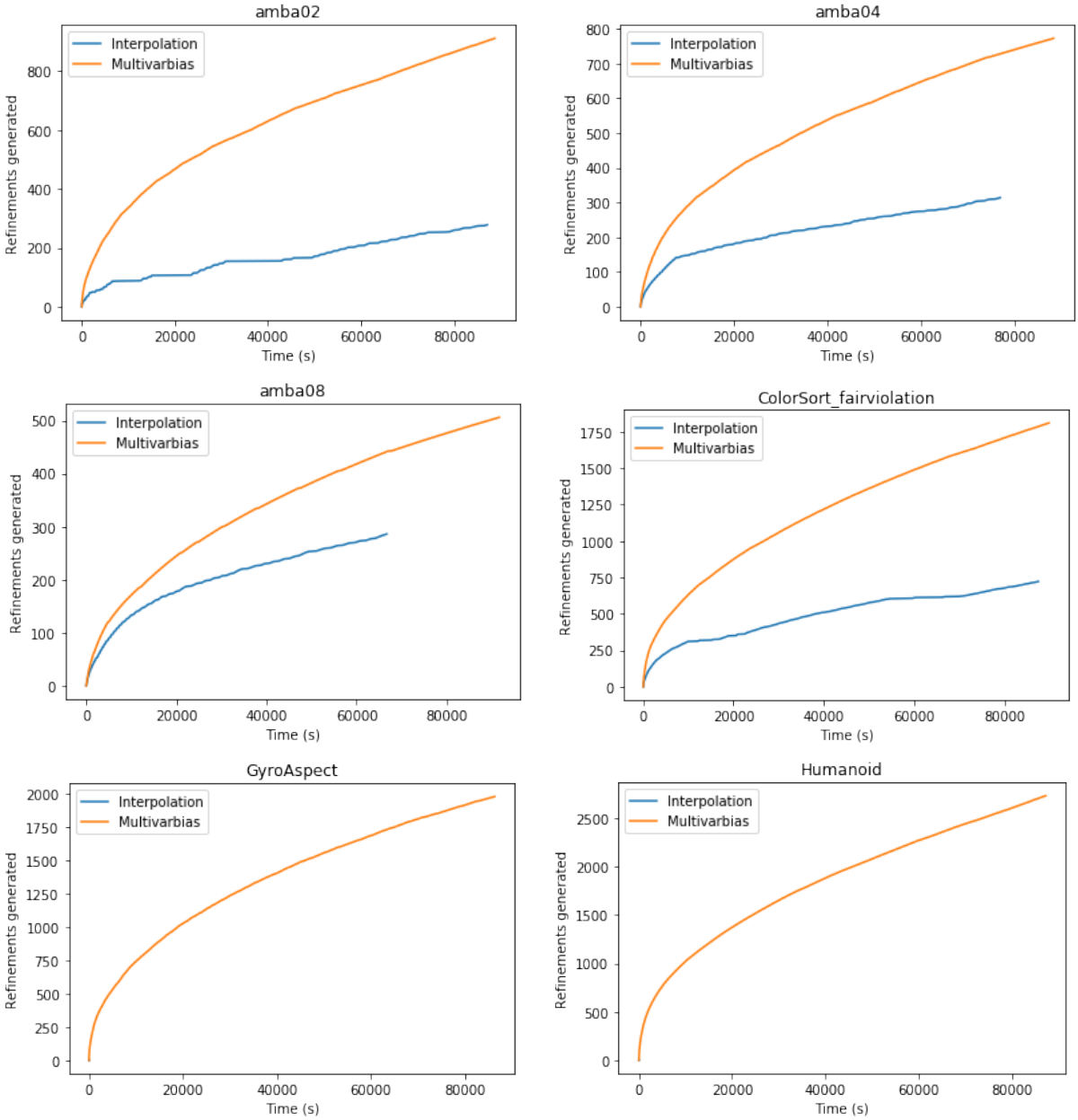


Figure 4.13: Total number of refinement nodes explored over time

violated by the produced counterstrategy (see Section 2.2.4 for the relationship between counterstrategies and unrealizable cores). By comparing this with Fig. 4.15, notice that for each approach and each case study the procedure is most efficient in identifying realizable refinements if the proportion of nodes targeting their parent's unrealizable core is higher on that case study. Therefore, producing refinements from interpolants (which in turn are computed from the translation of an unrealizable core), amounts to performing an educated choice of the variables to use in the refinements to be generated.

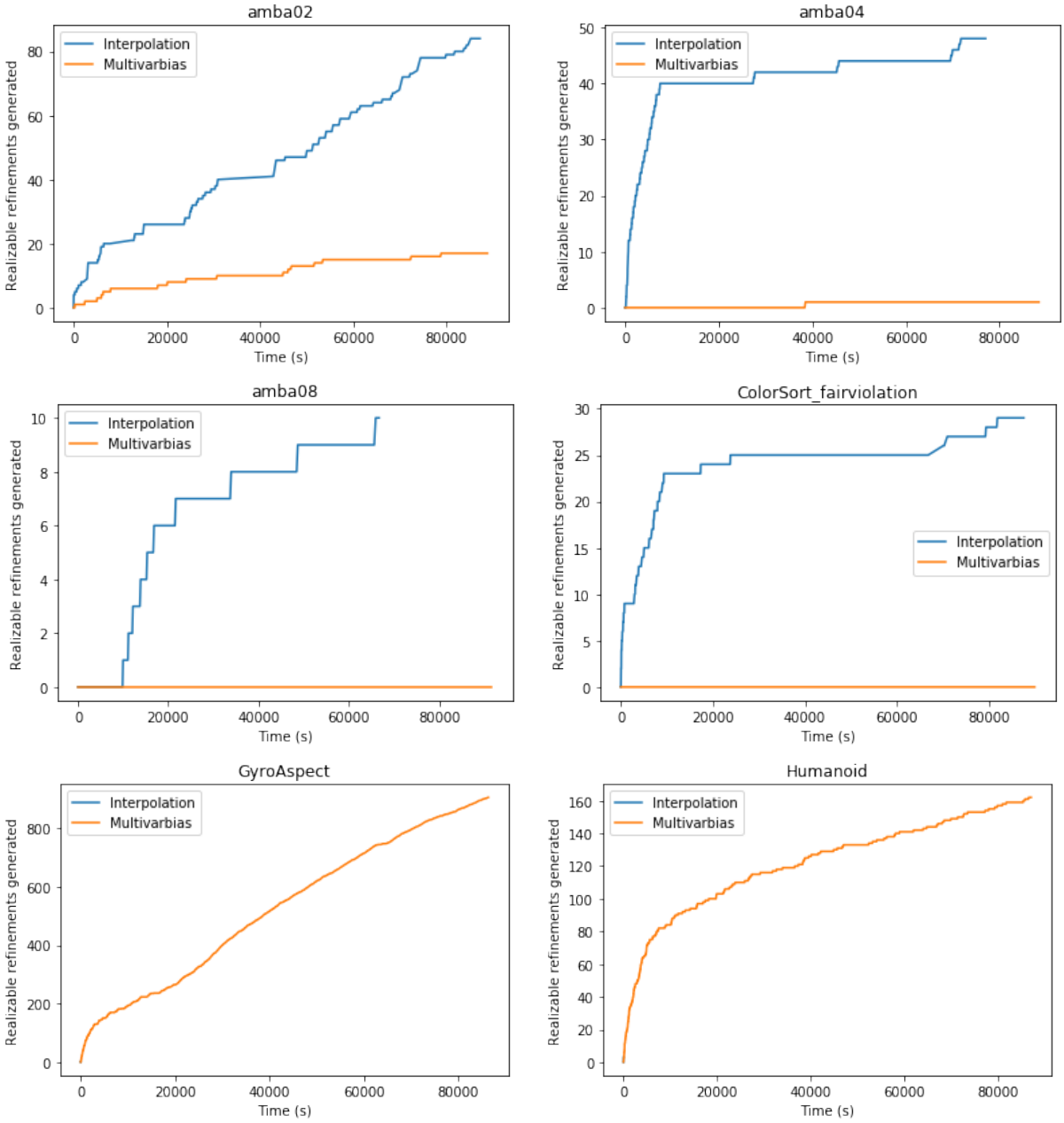


Figure 4.14: Number of realizable refinement nodes discovered over time (excluding vacuously realizable ones)

The larger number of solutions found by multivarbias in GyroAspect and Humanoid may be related to the size of the initial specification: the last two case studies are the smallest in terms of number of variables; therefore the space of possible choices for variables is small, and the likelihood of choosing a set of variables leading to a solution is higher. However, the number of variables alone may not be sufficient to characterize a class of problems where random variable selection works better than interpolation; in fact, AMBA02 has the same size in terms of number of variables, but in that case interpolation shows better performance than multivarbias.

Table 4.6: Statistics of the refinement trees. In order: *TotalNodes* is the total number of nodes generated; *NumSol* is the number of solutions at the end of the experiment; *MinSolDepth* and *MaxSolDepth* are respectively the minimum and the maximum depth at which a solution node has been found by the approach; *MinVars*, *MaxVars*, *AvgVars* and *StdDevVars* are respectively the minimum, maximum, average and sample standard deviation of the number of distinct variables included in any sufficient refinement found; for the average and the standard deviation, the denominator is the total number of solutions (*NumSol*); *MinOps*, *MaxOps*, *AvgOps* and *StdDevOps* are respectively the minimum, maximum, average and sample standard deviation of the number of Boolean operators appearing in any solution found; again, the denominator for the average and the standard deviation is *NumSol*; *NumInternalNodes* is the number of internal nodes in the tree; it coincides with the total number of counterstrategies generated; *AvgChld* and *StdDevChld* are the average and the sample standard deviation of the number of children of internal nodes, computed over the internal nodes; they indicate on average how many assumptions have been generated from a single counterstrategy; *NumFalse* is the number of times an unsatisfiable leaf is reached.

Method	Statistics	AMBA02	AMBA04	AMBA08	ColorSort	GyroAspect	Humanoid
Interpolation	TotalNodes	279	315	287	723	5	4
	NumSol	**84**	**48**	**10**	**29**	2	3
	MinSolDepth	**1**	3	**4**	**3**	2	1
	MaxSolDepth	5	5	**4**	**6**	**2**	**1**
	MinVars	**1**	**5**	**6**	**4**	4	**1**
	MaxVars	9	**6**	**7**	**10**	**5**	**3**
	AvgVars	**6**	**6**	**6**	**5**	**5**	**2**
	StdDevVars	2.04	0.5	0.42	1.4	0.7	1.15
	MinOps	**1**	**7**	**11**	**7**	4	**0**
	MaxOps	28	**15**	**18**	**18**	**5**	**3**
	AvgOps	13	**11**	**12**	**9**	**5**	**1**
	StdDevOps	5.67	2.13	2.21	2.8	0.7	1.53
	NumInternalNodes	45	85	49	154	3	1
	AvgChld	6	4	6	5	1	3
	StdDevChld	2.18	2.14	1.37	1.95	0.58	0
	NumFalse	39	42	10	**0**	**0**	**0**
Multivarbias	TotalNodes	**912**	**774**	**507**	**1812**	**1980**	**2734**
	NumSol	17	1	0	0	**905**	**162**
	MinSolDepth	2	**2**	N/A	N/A	2	1
	MaxSolDepth	**3**	**2**	N/A	N/A	3	3
	MinVars	2	10	N/A	N/A	**3**	**1**
	MaxVars	**8**	10	N/A	N/A	9	7
	AvgVars	**6**	10	N/A	N/A	7	5
	StdDevVars	1.73	0	N/A	N/A	1.34	1.15
	MinOps	2	15	N/A	N/A	**2**	**0**
	MaxOps	**15**	**15**	N/A	N/A	22	18
	AvgOps	**9**	15	N/A	N/A	12	7
	StdDevOps	3.5	0	N/A	N/A	3.65	3.28
	NumInternalNodes	39	25	12	41	71	102
	AvgChld	23	31	42	44	28	27
	StdDevChld	6.5	8.07	16.17	4.95	9.2	7.15
	NumFalse	**21**	**2**	**1**	4	401	19

Table 4.7: Percentage of nodes targeting parent's unrealizable cores

Benchmark	Interpolation	Multivarbias
AMBA02	88.17%	5.48%
AMBA04	100%	1.03%
AMBA08	100%	0.39%
ColorSort	45.09%	1.6%
GyroAspect	100%	80.5%
Humanoid	100%	100%

The other difference to be considered lies in the number of initial assumptions and guarantees, which is higher in AMBA02 than in the other two case studies. This leads to longer translations from LTL to Boolean, which in turn potentially lead to longer interpolants and therefore more nodes in interpolation's refinement tree. Therefore, in characterizing such class of problems the number of initial assumptions and guarantees plays a central role as well. Further studies are to be conducted in this direction.

4.9 Related Work

Craig's interpolants have been employed in the context of counterexample-guided abstraction refinement (CEGAR) for program verification, such as in [EKS06, DKW08, TDDO17]. These works use predicate abstraction to construct a simplified model of a program, such that model checking it against a set of given assertions is feasible; if a counterexample is generated, and it is proven to be a spurious counterexample due to an overgeneralization of the model, additional assumptions for the model are built by using interpolants. In the opposite fashion, the work in [McM03] applies Craig's interpolants to the problem of model checking: interpolants are used to construct the invariants of a model that allow to prove the satisfaction or violation of an LTL formula to check; these invariants are generalizations of the model's states that entail the violation of the formula.

The differences with our work are in specification language and overall objective. The mentioned works fit into the verification stage of software engineering, while the context of our work is the specification phase. As a consequence, the models and languages used to express properties are different too: while we deal with GR(1) formulae and games, which are used to formalize the reactive synthesis problem as described in Chapter 2, the cited works operate on models constructed from a program, which are typically more complex and fine-grained. For instance, the Symbolic Pushdown Systems defined in [EKS06] make an explicit distinction between variables representing data in a program and variables representing the program's control flow.

Despite these differences, our work takes inspiration from their use of Craig's interpolants to extract explanations from counterexamples. In particular, the notion of using replicates of the model's variables for each counterexample state in the Boolean formulae to interpolate is common to these approaches; we use it when we translate a GR(1) formula into Boolean to allow interpolation (Section 4.4.2).

A further application of interpolants is performing goal operationalization in the KAOS framework for requirements engineering [DAAU14]. Goal operationalization consists in refining a system's goal, expressed in some formal language like LTL, into constraints on the operations performed by the single agents of the system. Interpolation helps identifying constraints that are insufficient to meet the goal, and suggesting repairs to such constraints. Despite the context of KAOS is different from that of GR(1) assumptions refinement, the way we use interpolation in this chapter is similar to how it is used in [DAAU14]: indeed, they also use interpolants to identify the cause of violation of a goal, based on the same translation procedure from temporal logic to Boolean logic as we present in Section 4.4.2.

4.10 Summary

In this chapter, we have described a bias heuristic for counterstrategy-guided assumptions refinement based on Craig's interpolation. It is a fully automated procedure that addresses automatically the cause of unrealizability when constructing assumptions that eliminate counterstrategy. Being automated, it bypasses the need for manual input from the user (see Chapter 3).

We have described the requirements of this bias heuristic on the structure of the input counterstrategy, defining abstract counterstrategies. This representation of a counterstrategy eliminates from the counterstrategy graph the output valuations that the controller would never choose, since they would violate some invariant guarantee. This abstraction allows to identify a set of *influential output variables*, that are the variables on whose valuations an abstract counterstrategy branches out. In other words, these are the variables that affect the future

environment choices from some state.

We have described the steps involved in the interpolation bias: *(i)* translation from temporal to Boolean domain, *(ii)* interpolation, and *(iii)* translation from Boolean to temporal. We have proven the soundness of the approach, in the sense that the assumptions induced by this bias effectively eliminate the input counterstrategy. We have also identified a requirement on the interpolant for the procedure to be sound, namely the *state* and *I/O-separability* of the interpolant.

The experiments, performed on benchmark case studies of assumptions refinement, show that interpolation increases the effectiveness of the search as defined in Section 3.3, that is, the probability of hitting a solution among the candidate refinements produced in a single execution.

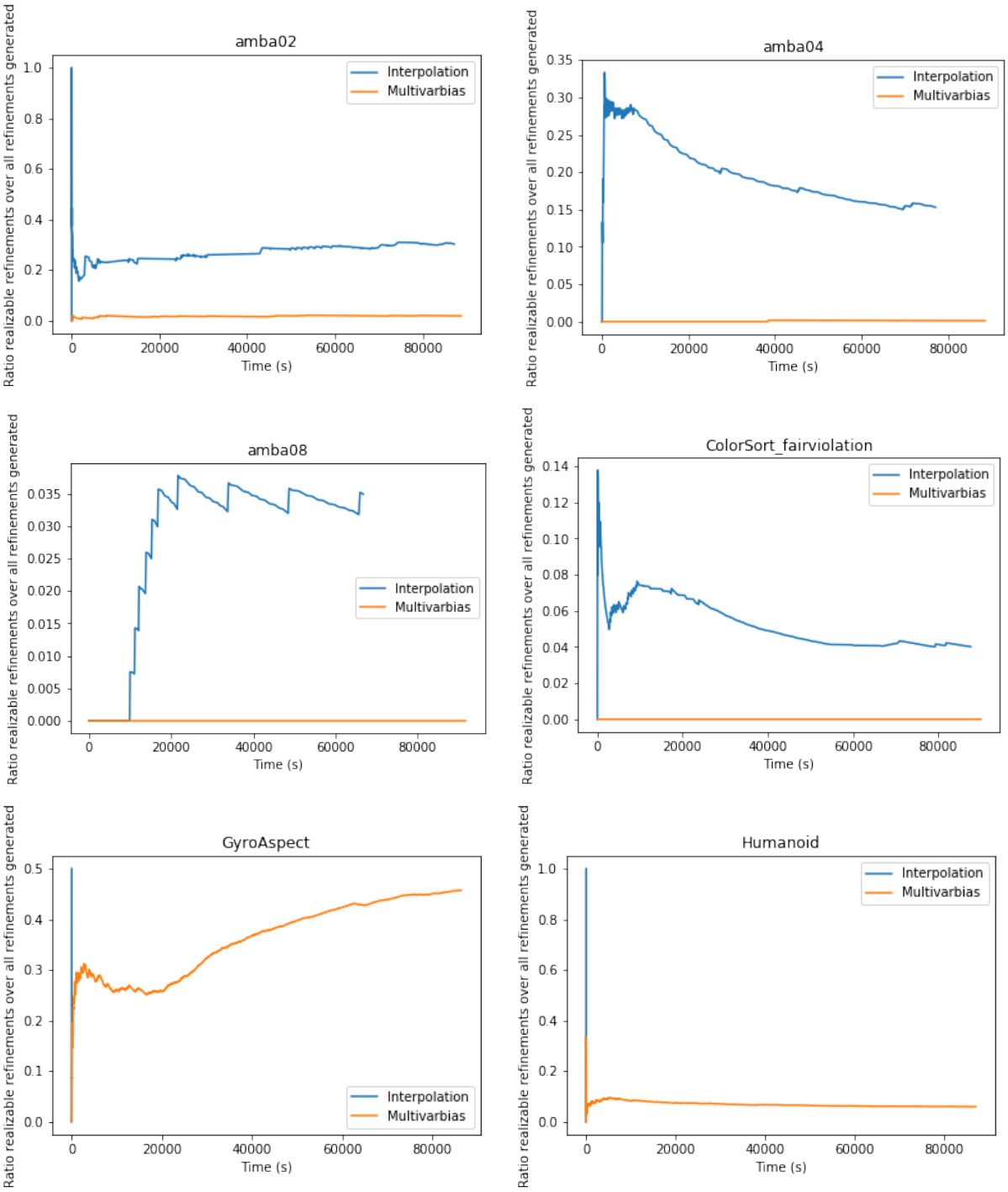


Figure 4.15: Effectiveness over time

Chapter 5

Entropy and Hausdorff Dimension of ω-languages

The ability to measure the weakness of assumption refinements is useful in two ways: *(i)* evaluating and comparing the output of different search algorithms, and *(ii)* as a novel selection heuristic that replaces the classical FIFO heuristic (Section 3.4.3) and drives the search towards weaker solutions.

This chapter Section 5.1 presents our algorithm to compute the DBA of the closure of an ω-language L from a DBA of L. In Section 5.2, we extend the algorithm from [Sta98] to other kinds of automata than Muller, namely Büchi and generalized Büchi automata, more commonly used in formal methods contexts [DLvL06, EKS16, VW94, GM03, LZP$^+$13]. Finally, in Section 5.3, we describe two simplifications that reduce the time to compute the two quantities and address a problem of overflow that can occur when dealing with formulae with a large number of variables (discussed in Section 5.3).

This chapter is largely (but not entirely) based on our work in [CAG18] and [CAG20a].

5.1 DBA of the closure of an ω-language.

Here we prove a property that is central for the proofs in this and the next chapter. Given a DBA of the ω-language L, a DBA of the closure of L can be immediately obtained by making accepting all the reachable, co-reachable states in the DBA.

Proposition 5.1 (DBA of the closure of L)**.** *Let $\mathcal{B} = \langle Q, \Sigma, q_0, \delta, F\rangle$ be a DBA of the language L, and let $S \subseteq Q$ the set of all the states being both reachable and co-reachable in the graph of $\mathcal{B}$. Then the automaton $\mathcal{B}_{\mathcal{C}} = \langle Q, \Sigma, q_0, \delta, S\rangle$ accepts the closure of L, $\mathcal{C}(L)$.*

Proof. We prove that the language $\mathcal{C}(L)$ coincides with the set of ω-words such that the runs induced by them on $\mathcal{B}$ only contain reachable, co-reachable states. Then, it is obvious that $\mathcal{B}_{\mathcal{C}}$ is the automaton accepting this language. Formally, we prove that for every $w \in \Sigma^\omega$, $\mathsf{pref}(w) \subseteq \mathsf{pref}(L)$ if and only if the states of the induced run $\mathcal{B}(w) = q_0 q_1 \dots$ are all reachable and co-reachable.

Suppose, for a given $w \in \Sigma^\omega$, that every prefix of w, $w[1..k]$ is also a prefix in an ω-word of L. Then for every k, $q_k = \delta(q_0, w[1..k])$ is reachable from q_0. Moreover, since there exists an ω-word w_k accepted by $\mathcal{B}$ that starts with the prefix $w[1..k]$, there exists a run of $\mathcal{B}$ that traverses q_k and then passes infinitely many times through an accepting state. Therefore, q_k is co-reachable.

Vice versa, suppose that, for a given $w \in \Sigma^\omega$, the induced run $\mathcal{B}(w) = q_0 q_1 \dots$ only traverses states that are both reachable and co-reachable. Then, from any q_k, there exists a word v_k such that $\delta(q_k, v_k) = f \in F$ for some accepting state F in an accepting SCC. Therefore, let $v_f \in \Sigma^*$ be a word such that $\delta(f, v_f) = f$. The ω-word $w_k = w[1..k]v_k(v_f)^\omega$ visits f infinitely many times and therefore it is a word of L sharing a prefix with w. This construction can be done for any prefix $w[1..k]$ of w. Therefore, for every prefix of w, there exists a word in L starting with the same prefix. In other words, $\mathsf{pref}(\{w\}) \subseteq \mathsf{pref}(L)$. □

As said earlier in this section, without loss of generality we assume all states in a DBA to be both reachable and co-reachable. Then, the procedure to obtain the DBA of the closure of L

reduces to making all states accepting in a DBA of L.

5.2 Hausdorff Dimension of Büchi and Generalized Büchi Automata

Algorithm HausdorffDMA requires the Muller automaton of the ω-language. However, in applications to formal methods, other types of automata are usually employed, typically Büchi automata or generalized Büchi automata [BKeS12, GL02, KGFP09]. In this section, we extend the original algorithm to these ω-automata.

The algorithm is based on finding the DMA equivalent to a given DBA and on algebraic results on nonnegative matrices. First we assume the DBA is strongly connected; later we show how the algorithm changes when this assumption is relaxed.

Strongly connected Büchi automata. Let $\mathcal{B} = (Q, \Sigma, q_0, \delta, F_{\mathcal{B}})$ be a strongly connected DBA and $\mathcal{M} = (Q, \Sigma, q_0, \delta, F_{\mathcal{M}})$ be a DMA where $F_{\mathcal{M}} = \{S'_i \in 2^Q | S'_i \cap F_{\mathcal{B}} \neq \varnothing\}$. That is, $\mathcal{M}$ has the same states and transition function as $\mathcal{B}$, and the collection of accepting states $F_{\mathcal{M}}$ consists of all the subsets of Q that have nonempty intersection with the accepting set of $F_{\mathcal{B}}$. The next proposition shows that the two automata are equivalent:

Proposition 5.2. *The DBA $\mathcal{B}$ and the DMA $\mathcal{M}$ defined above accept the same ω-language.*

Proof. Let w be a word accepted by $\mathcal{B}$. Then $\mathsf{Inf}_{\mathcal{B}}(w) \cap F_{\mathcal{B}} \neq \varnothing$, and therefore $\mathsf{Inf}_{\mathcal{B}}(w) \in F_{\mathcal{M}}$. Moreover, since the two automata have the same set of states and transition function, the sequences of states generated by the two automata are the same, that is, $\mathcal{B}(w) = \mathcal{M}(w)$; in other words, w induces the same computation on the two automata. Hence $\mathsf{Inf}_{\mathcal{M}}(w) = \mathsf{Inf}_{\mathcal{B}}(w) \in F_{\mathcal{M}}$, and by definition of Muller acceptance also $\mathcal{M}$ accepts w. Showing that $\mathcal{B}$ accepts every word accepted by $\mathcal{M}$ can be done similarly. □

Algorithm HausdorffDMA computes the Hausdorff dimension, operating directly on $\mathcal{M}$.

However, the number of accepting sets in $F_{\mathcal{M}}$ grows exponentially with the size of Q, which makes the execution of the loop in Algorithm 2 intractable. The computational complexity of the algorithm can be reduced by excluding the S'_i that do not affect the overall dimension. Those can be identified via the analysis of algebraic properties of nonnegative matrices.

Next we show that the accepting sets S'_i that affect the Hausdorff dimension are the maximal accepting sets, defined as follows:

Definition 5.1 (Maximal accepting sets)**.** *We call a set $S_j \in F_{\mathcal{M}}$ a* maximal accepting set *if there is no $S_i \in F_{\mathcal{M}}$ such that $S_j \subset S_i$.*

Then the following proposition holds:

Proposition 5.3. *Consider a language L represented by a deterministic Muller automaton with adjacency matrix M, and let T_m denote the table of maximal accepting state sets. Then $\mathsf{dim}\,(L) = \mathsf{max}_{S \in T_m} \mathsf{log}_r\, \nu(M[S])$, where $\nu(A)$ denotes the spectral radius of a matrix A. Accordingly, in Algorithm 2, if the loop in lines 1-3 is modified to iterate over maximal accepting sets only, the output is still the Hausdorff dimension of $L(\mathcal{M})$.*

Proof. Given two sets of states S'_i and S'_j such that $S'_i \subseteq S'_j$, the matrices $M[S'_i]$ and $M[S'_j]$ are principal submatrices of M by definition. More specifically, $M[S'_i]$ is a principal submatrix of $M[S'_j]$. By Corollary 8.1.20 in [HJ12], we know that $\nu(M[S'_i]) \leq \nu(M[S'_j])$. This, combined with (2.5), implies that $\mathsf{dim}\,(L) = \mathsf{max}_{S \in T_m} \mathsf{log}_r\, \nu(M[S])$. The statement on Algorithm 2 then follows trivially. □

By construction, Q is an accepting set of $\mathcal{M}$, and since it contains all states, it is the unique maximal accepting set (note that the constuction is possible because we assumed that the Büchi automatoton we start with is strongly connected). Therefore, the Hausdorff dimension of $L(\mathcal{M})$ is just the spectral radius of the adjacency matrix of $\mathcal{M}$. This in turn is the same adjacency matrix as that of the original Büchi automaton $\mathcal{B}$, resulting in the following theorem:

Theorem 5.1 (Hausdorff dimension of strongly connected Büchi automata)**.** *Given a strongly connected Büchi automaton $\mathcal{B}$ with adjacency matrix M, its Hausdorff dimension is*

$$\mathsf{dim}\,(L(\mathcal{B})) = \mathsf{log}_r\,\nu(M)\ .$$

General Büchi automata. Extending the above simplification in computing the Hausdorff dimension to non-strongly connected Büchi automata is straightforward. Consider a Büchi automaton $\mathcal{B} = (Q, \Sigma, q_0, \delta, F_{\mathcal{B}})$ such that its strongly connected components are $C_1, \ldots, C_k \subseteq Q$. Let $\mathcal{M} = (Q, \Sigma, q_0, \delta, F_{\mathcal{M}})$, where $F_{\mathcal{M}} = \{S_i \in Q \mid S_i \subseteq C_j \text{ for some } j \ \wedge\ S_i \cap F_{\mathcal{B}} \neq \varnothing\}$. It is easy to verify that $\mathcal{B}$ and $\mathcal{M}$ accept the same language.

In this case, the maximal accepting subsets in $F_{\mathcal{M}}$ are all the SCCs C_j having nonempty intersection with $F_{\mathcal{B}}$; we call these *accepting SCCs*. Therefore, similarly to the previous section, the algorithm only needs to compute the spectral radii of $M[C_j]$ for any such C_j; the Hausdorff dimension of $L(\mathcal{B})$ is the maximum of these spectral radii. This proves the following theorem:

Theorem 5.2. *Consider a Büchi automaton $\mathcal{B}$ with accepting SCCs $C_1, \ldots, C_k$ and their adjacency matrices $M[C_1], \ldots, M[C_2]$ respectively. Then* $\mathsf{dim}\,(L(\mathcal{B})) = \mathsf{max}_{i\in\{1,\ldots,k\}}\,\mathsf{log}_r\,\nu(M[C_i])$.

The formula for computing the Hausdorff dimension of an ω-language given its Büchi automaton in the above theorem is implemented in Algorithm 10.

Algorithm 10: HausdorffDBA

Input: A DBA $\mathcal{B} = (Q, \Sigma, q_0, \delta, F_{\mathcal{B}})$, with adjacency matrix M
Output: Hausdorff dimension *dim* of $L(\mathcal{B})$
$C \leftarrow$ StronglyConnectedComponents(Q, δ);
$i \leftarrow 0$;
foreach $C_j \in C$ **do**
  if $C_j \cap F_{\mathcal{B}} \neq \varnothing$ **then**
    $\lambda_i \leftarrow$ maximum eigenvalue of $M[C_j]$;
    $i \leftarrow i + 1$;
  end
end
$\lambda \leftarrow \mathsf{max}_i\,\lambda_i$;
$dim \leftarrow \mathsf{log}_r\,\lambda$;
return *dim*

Generalized Büchi automata. *Generalized Büchi automata* allow for more than one accepting set of states. It is defined as a tuple $\mathcal{G} = \{Q, \Sigma, q_0, \delta, F_\mathcal{G}\}$, where $F_\mathcal{G} = \{F_{\mathcal{G},1} \dots F_{\mathcal{G},k}\}$. An ω-word w is accepted by $\mathcal{G}$ if $\mathsf{Inf}(w) \cap F_{\mathcal{G},i} \neq \varnothing$ for every $i \in [k]$. The corresponding Muller automaton has the same set of states and transition function, and the accepting condition is defined in terms of $F_\mathcal{M} = \{Q_j \in Q \mid \forall i \in [k],\ Q_j \cap F_i \neq \varnothing\}$. The maximal accepting subsets of this Muller automaton correspond to the strongly connected component that have nonempty intersection with each F_i. The algorithm to compute Hausdorff dimension is the same as Algorithm 10 where the **if** condition is replaced with $C_j \cap F_{\mathcal{G},1} \neq \varnothing \wedge \dots \wedge C_j \cap F_{\mathcal{G},k} \neq \varnothing$.

5.3 Simplification of Entropy and Hausdorff Dimension

In this section we discuss two simplifications that can be applied to the computation of entropy and Hausdorff dimension when dealing with languages of GR(1) conjunctions.[1] These simplifications allow in some cases to reduce the size of the automata involved in the computation by decomposing the conjunction into its conjuncts and computing the two quantities on the conjuncts individually.

5.3.1 Conjunctions of Independent Formulae

Computing the of an LTL formula requires a doubly exponential time in the number of operators in the formula [KR10]. For conjunctive formulae like GR(1) conjunctions . We prove that under some restrictive assumptions it is possible to compute the entropy and Hausdorff dimension of the whole conjunction by combining the entropies and Hausdorff dimensions of the individual conjuncts. In this way the translation from LTL to DBAs involves shorter formulae (the conjuncts instead of the whole conjunction) and the time requirements can be significantly reduced.

The following theorem states this property.

[1]

Theorem 5.3. *Let $\phi = \phi_1 \wedge \phi_2$ be the conjunction of two LTL formulae such that $\mathsf{vars}(\phi_1) \cap \mathsf{vars}(\phi_2) = \varnothing$. Let $\mathcal{V}_1 = \mathsf{vars}(\phi_1)$, $\mathcal{V}_2 = \mathsf{vars}(\phi_2)$, $\mathcal{V} = \mathsf{vars}(\phi) = \mathcal{V}_1 \cup \mathcal{V}_2$, $2^{|\mathcal{V}_1|} = r_1$, $2^{|\mathcal{V}_2|} = r_2$, and $2^{|\mathcal{V}|} = r = r_1 r_2$. Let $L(\phi_1)$ and $L(\phi_2)$ be the languages of the two formulae, with entropy H_1 and H_2 respectively, and Hausdorff dimension d_1 and d_2 respectively. Then the entropy of $L(\phi)$ is*

$$H(L(\phi)) = H_1 \log_r r_1 + H_2 \log_r r_2$$

Proof. Given an ω-word $w \in (2^{\mathcal{V}})^\omega$, we denote as $w|_{\mathcal{V}_i}$, for $\mathcal{V}_i \subseteq \mathcal{V}$ the *projection of w onto $\mathcal{V}_i$*, that is, the ω-word $\bar{w}$ such that for every j $\bar{w}[j] = w[j] \cap \mathcal{V}_i$. Since ϕ_1 and ϕ_2 have no variables in common, $w \models \phi$ if and only if $w|_{\mathcal{V}_1} \models \phi_1$ and $w|_{\mathcal{V}_2} \models \phi_2$. Hence, for every prefix of length n $w[1..n] \in \mathsf{pref}_n(L(\phi))$ there exists a unique pair of prefixes $w|_{\mathcal{V}_1}[1..n] \in \mathsf{pref}_n(L(\phi_1))$, $w|_{\mathcal{V}_2}[1..n] \in \mathsf{pref}_n(L(\phi_2))$ and vice versa. Therefore, the number of prefixes of length n $|\mathsf{pref}_n(L(\phi))|$ is the product of the numbers of prefixes $|\mathsf{pref}_n(L(\phi_1))|$ and $|\mathsf{pref}_n(L(\phi_2))|$ of the same length. We can conclude that

$$\begin{aligned}
H(L(\phi)) &= \limsup_{n\to\infty} \frac{1}{n} \log_r |\mathsf{pref}_n(L(\phi))| \\
&= \limsup_{n\to\infty} \frac{1}{n} \log_r \left(|\mathsf{pref}_n(L(\phi_1))| * |\mathsf{pref}_n(L(\phi_2))|\right) \\
&= \limsup_{n\to\infty} \frac{1}{n} \log_r |\mathsf{pref}_n(L(\phi_1))| + \limsup_{n\to\infty} \frac{1}{n} \log_r |\mathsf{pref}_n(L(\phi_2))| \\
&= \frac{1}{\log_{r_1} r} \limsup_{n\to\infty} \frac{1}{n} \log_{r_1} |\mathsf{pref}_n(L(\phi_1))| + \frac{1}{\log_{r_2} r} \limsup_{n\to\infty} \frac{1}{n} \log_{r_2} |\mathsf{pref}_n(L(\phi_2))| \\
&= H(L(\phi_1)) \log_r r_1 + H(L(\phi_2)) \log_r r_2 \,.
\end{aligned}$$

□

The main constraint under which entropy can be decomposed over conjuncts is that the conjuncts use non-overlapping sets of variables. Unfortunately, this condition is not fulfilled by the assumptions in the benchmark case studies for our experiments (see Section 6.4). However, the optimization presented in the next subsection is a straightforward derivation of Theorem 5.3 and finds practical application in those case studies.

Remark 5.1. *The simplification of Theorem 5.3 applies to Hausdorff dimension too when all*

the languages under analysis are closed, since in this case entropy and Hausdorff dimension are equal. We are convinced that it holds in general for Hausdorff dimension, but were not able to prove it.

5.3.2 Reduced Adjacency Matrix for Large Alphabets

The alphabet $2^{\mathcal{V}}$ of the language of a GR(1) conjunction ϕ over the variables $\mathcal{V}$ contains all possible $\mathcal{V}$-valuations. Hence it grows exponentially with the size of $\mathcal{V}$ ($r = \left|2^{\mathcal{V}}\right| = 2^{|\mathcal{V}|}$). If the number of variables exceeds the number of bits used by the compiler to represent long integers (typically 32 or 64 in C/C++), the number of symbols connecting two states in the adjacency matrix M may not be stored correctly. Of course floating point numbers with high precision may be used instead, but this would add an approximation error in the eigenvalue computation, and in several cases it is not needed. We introduce here the *reduced adjacency matrix*, that allows to recover the logarithm of the spectral radius of the original adjacency matrix but is defined over a reduced set of symbols.

We leverage the fact that not all the variables in $\mathcal{V}$ typically appear in a GR(1) conjunction. Indeed, when the conjunction under analysis is the conjunction of assumptions (as in the application to assumptions refinement discussed in Chapter 6), only the input variables in $\mathcal{X}$ and a subset of the output variables in $\mathcal{Y}$ appear in it.

The following terminology will be used in this section.

Definition 5.2 (Constrained/non-constrained variables)**.** *Given an LTL formula ϕ over the variables in $\mathcal{V}$, the* constrained variables *are the variables appearing in ϕ, and we denote their set as $\mathcal{V}_c = \mathsf{vars}(\phi)$. The* unconstrained variables *(or* non-constrained variables*) are the variables of $\mathcal{V}$ not appearing in ϕ, and their set is denoted as $\mathcal{V}_{nc} = \mathcal{V} \setminus \mathcal{V}_c$.*

The alphabet Σ of the language of ϕ is the set of $\mathcal{V}$-valuations. However, the same formula can also be interpreted as a formula over $\mathcal{V}_c$, and the alphabet in this case is the set of $\mathcal{V}_c$-valuations. We call an automaton of $L(\phi)$ using the alphabet $\Sigma = 2^{\mathcal{V}_c}$ a *reduced automaton* of $L(\phi)$, and its adjacency matrix a *reduced adjacency matrix*.

The following definition leverages the notion of projection of a valuation, defined in Definition 2.18.

Definition 5.3 (Projected ω-word and ω-language)**.** *Let $\mathcal{V}$ be a set of propositional variables, $w \in (2^{\mathcal{V}})^\omega$ an ω-word over the alphabet $2^{\mathcal{V}}$, $L \subseteq (2^{\mathcal{V}})^\omega$ an ω-language over the alphabet $2^{\mathcal{V}}$, and $\mathcal{V}_c \subseteq \mathcal{V}$ a subset of propositional variables. The* projection of w onto $\mathcal{V}_c$ *is the ω-word $w_r \in (2^{\mathcal{V}_c})^\omega$ such that $\forall i \in \mathbb{N}, w_r[i] = w[i]|_{\mathcal{V}_c}$. The* projection of L onto $\mathcal{V}_c$ *is the set $L_{red} \subseteq (2^{\mathcal{V}_c})^\omega)$ such that $L_{red} = \{w|_{\mathcal{V}_c} \mid w \in L\}$.*

Definition 5.4 (Reduced language, reduced Muller automaton and reduced adjacency matrix)**.** *Let ϕ be an LTL formula such that $\mathsf{vars}(\phi) = \mathcal{V}_c \subseteq \mathcal{V}$. The* reduced language of ϕ, *$L_{red}(\phi)$ is the projection of $L(\phi)$ onto $\mathcal{V}_c$. A* reduced Muller automaton *of $L(\phi)$ is a Muller automaton $\mathcal{M}_{red} = \langle Q, 2^{\mathcal{V}_c}, q_0, \delta, T\rangle$ of the reduced language $L_{red}(\phi)$. Its adjacency matrix is called a* reduced adjacency matrix *of $L(\phi)$.*

Example 5.1. *Let the set of variables $\mathcal{V} = \{a, b, c\}$ and a formula $\phi = \mathbf{G}(a \rightarrow \mathbf{X}b)$. The set of constrained variables is $\mathcal{V}_c = \mathsf{vars}(\phi) = \{a, b\}$, and the set of non-constrained variables is $\mathcal{V}_{nc} = \{c\}$.*

The language $L(\phi)$ is the language of all ω-words such that any valuation where a is true *is followed by a valuation where b is* true*. There are no constraints on the assignment of c. The language $L(\phi)$ is a subset of $(2^{\mathcal{V}})^\omega$, therefore every symbol of its ω-words assigns a value to all the variables $a, b, c \in \mathcal{V}$.*

The reduced language $L_{red}(\phi)$, instead, is a subset of $(2^{\mathcal{V}_c})^\omega$, and therefore every symbol of its ω-words only assigns values to the constrained variables $a, b \in 2^{\mathcal{V}}_c$.

A Muller automaton and a reduced Muller automaton of $L(\phi)$ are shown in Figure 5.1. Their adjacency matrices are shown in Figure 5.2. Notice that the origina adjacency matrix M is obtained from the reduced one M_{red} by multiplying the latter by a scalar, corresponding to the number of valuations of the non-constrained variables (in this case there is one such variable only, so the scalar is 2).

If a reduced automaton is given, the following lemma shows that obtaining an automaton of

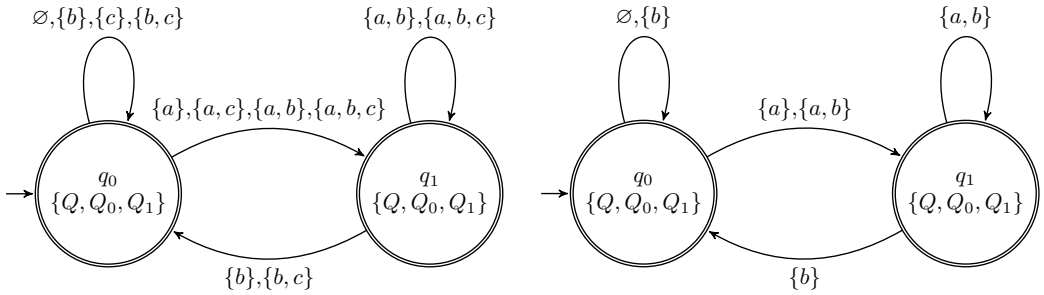


Figure 5.1: Muller (left) and reduced Muller automaton (right) of $L(\phi)$. The edges are labelled with all the symbols taking the automaton from their source to their destination state. In order to represent graphically the table of accepting sets in Muller automata (see Definition 2.11), we label each accepting state with the collection of accepting sets to which it belongs. In both automata, the table consists of the accepting sets $Q = \{q_0, q_1\}$, $Q_0 = \{q_0\}$, and $Q_1 = \{q_1\}$.

$$M = \begin{pmatrix} 4 & 4 \\ 2 & 2 \end{pmatrix} \qquad M_{red} = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$$

Figure 5.2: Adjacency matrix (left) and reduced adjacency matrix (right) of $L(\phi)$

$L(\phi)$ is immediate.

Lemma 5.1. *Given an LTL formula ϕ and its reduced automaton $\mathcal{M}_{red} = \langle Q, 2^{\mathcal{V}_c}, q_0, \delta_{red}, T\rangle$. Then a Muller automaton of $L(\phi)$ is $\mathcal{M} = \langle Q, 2^{\mathcal{V}}, q_0, \delta, T\rangle$, where*

- *Q, q_0, T coincide with those of $\mathcal{M}_{red}$;*
- *the alphabet $2^{\mathcal{V}}$ is the set of $\mathcal{V}$ valuations;*
- *the transition function δ is such that $\forall q \in Q.\, \forall \sigma \in 2^{\mathcal{V}}.\, \delta(q, \sigma) = \delta_{red}(q, \sigma|_{\mathcal{V}_c})$; that is, the label $\sigma_r \in 2^{\mathcal{V}_c}$ of every transition in M_r is replaced by a set of labels σ such that $\sigma|_{\mathcal{V}_c} = \sigma_r$.*[2]

Proof. We will prove that every ω-word $w \in L(\phi)$ is accepted by $\mathcal{M}$ and the labelling of every run accepted by $\mathcal{M}$ belongs to $L(\phi)$. Given an ω-word $w \in L(\phi)$, its projection $w_r = w|_{\mathcal{V}_c}$ belongs to $L_{red}(\phi)$ by construction. Therefore there exists an accepting run $q_0 q_1 \cdots \in Q^\omega$ on $\mathcal{M}_{red}$ labelled by w_r, such that $\mathsf{Inf}(q_0 q_1 \dots) \in T$. Since $\delta_{red}(q_i, w_r[i+1]) = q_{i+1}$, by construction $\delta(q_i, w[i+1]) = \delta_{red}(q_i, w_r[i+1]) = q_{i+1}$; hence, the ω-word w induces the same run $q_0 q_1 \dots$ on $\mathcal{M}$, which is an accepting run of $\mathcal{M}$.

[2]This set is the set of all labels $\sigma = \sigma_r \cup \sigma_{nr}$ for every $\sigma_{nr} \in 2^{\mathcal{V}_{nc}}$. In other words, M is constructed by replacing every transition label in M_r with the set of transition labels obtained by taking the unions of the original label with every valuation of the unconstrained variables.

Now, let $w \in (2^{\mathcal{V}})^\omega$ be an ω-word inducing an accepting run $q_0 q_1 \cdots \in Q^\omega$ on $\mathcal{M}$. By construction, $w_r = w|_{\mathcal{V}_r}$ induces the same run on $\mathcal{M}_{red}$, which is accepting in $\mathcal{M}_{red}$ too. Hence, $w_r \in L_{red}(\phi)$, or, in other words, $w_r \models \phi$ when w_r is a sequence of $\mathcal{V}_c$-valuations, and ϕ a formula over $\mathcal{V}_c$. The ω-word w extends the symbols of w_r with arbitrary $\mathcal{V}_{nc}$-valuations: since ϕ does not contain any constraint on $\mathcal{V}_{nc}$, and w contains the same sequence of $\mathcal{V}_c$-valuations as w_r, we have $w \models \phi$. □

This lemma leads us to prove that the entropy and Hausdorff dimension of an LTL formula can be expressed in terms of the entropy and Hausdorff dimension of its reduced language, as stated in the following theorem.

Theorem 5.4. *Let ϕ be an LTL formula over the variables $\mathsf{vars}(\phi) = \mathcal{V}_c \subseteq \mathcal{V}$. Let $L_{red}(\phi) = \{w|_{\mathcal{V}_c} | w \in L(\phi)\}$ the reduced language of ϕ and $\mathcal{V}_{nc} = \mathcal{V} \setminus \mathcal{V}_c$. Then*

$$H(L(\phi)) = \frac{|\mathcal{V}_{nc}|}{|\mathcal{V}|} + \frac{|\mathcal{V}_c|}{|\mathcal{V}|} H(L_{red}(\phi)) \ .$$

Moreover, if $0 < m(L_{red}(\phi)) < \infty$

$$\mathsf{dim}\,(L(\phi)) = \frac{|\mathcal{V}_{nc}|}{|\mathcal{V}|} + \frac{|\mathcal{V}_c|}{|\mathcal{V}|}\,\mathsf{dim}\,(L_{red}(\phi))$$

Proof. Let $\mathcal{M}_{red}$ be a DMA of $L_{red}(\phi)$ and M_{red} its reduced adjacency matrix. By Proposition 5.3 (stating the correctness of Algorithm 2), the Hausdorff dimension of $L_{red}(\phi)$ is

$$\mathsf{dim}\,(L_{red}(\phi)) = \max_{S \in T} \mathsf{log}_{r_c}\, \nu(M_{red}[S]) \ ,$$

where $r_c = \left|2^{\mathcal{V}_c}\right|$.

Let $\mathcal{M}$ be the DMA of $L(\phi)$ obtained as in Lemma 5.1. Its adjacency matrix M is

$$M = \left|2^{\mathcal{V}_{nc}}\right| M_{red}$$

where $\left|2^{\mathcal{V}_{nc}}\right|$ is the number of $\mathcal{V}_{nc}$-valuations. This is because in M every transition of M_{red}

is replaced by as many transitions as the number of arbitrary valuations of $\mathcal{V}_{nc}$. So, for every subset of states S in $\mathcal{M}$, $M[S] = \left|2^{\mathcal{V}_{nc}}\right| M_r[S]$. Hence for every accepting set S the spectral radius of $M[S]$ is:

$$\nu(M[S]) = \left|2^{\mathcal{V}_{nc}}\right| M_{red}[S] \ .$$

Hence the formula for Hausdorff dimension comes from simple algebraic steps:

$$\begin{aligned}
\mathsf{dim}\,(L) &= \max_{S \in T} \mathsf{log}_r\, \nu(M[S]) \\
&= \max_{S \in T} \mathsf{log}_r \left|2^{\mathcal{V}_{nc}}\right| M_{red}[S] \\
&= \max_{S \in T} \mathsf{log}_r \left|2^{\mathcal{V}_{nc}}\right| + \mathsf{log}_r\, M_{red}[S] \\
&= \max_{S \in T} \frac{\mathcal{V}_{nc}}{\mathcal{V}} + \frac{1}{\mathsf{log}_{r_c}\, r} \mathsf{log}_{r_c}\, M_{red}[S] \\
&= \frac{\mathcal{V}_{nc}}{\mathcal{V}} + \frac{\mathcal{V}_c}{\mathcal{V}} \mathsf{dim}\,(L_{red}(\phi)) \ .
\end{aligned}$$

The proof for entropy follows the same steps and descends from Proposition 2.2. □

This theorem tells us that all we need to compute the entropy and Hausdorff dimension of a language is the adjacency matrix of $L_{red}(\phi)$, that is the language of ϕ as if ϕ was defined only over $\mathcal{V}_c$ instead of $\mathcal{V}$. This is the reduced adjacency matrix defined above. In general the numbers in the reduced adjacency matrix are lower than the ones in the original matrix, and the chance of incurring overflow is reduced.

5.4 Summary

This chapter introduced two features of ω-languages, namely *entropy* and *Hausdorff dimension*, that represent a notion of "extension" of the ω-language. In order to clarify their meaning, we analyzed extensively two examples of ω-languages where entropy and Hausdorff dimension are computed by working out their definition. Then we presented the algorithms to compute them on deterministic Muller automata and our adaptations of these algorithms to Büchi and generalized Büchi automata. Finally, we introduced some simplifications of the algorithm that

reduce, under some constraints, their time and space requirements. In the next chapter, we show how to leverage those quantities to construct a suitable measure of weakness for GR(1) conjunctions.

Chapter 6

Measuring Weakness of GR(1) Assumption Refinements

In this chapter, we make use of entropy and Hausdorff dimension, as introduced in Chapter 5, to define a quantitative weakness measure for GR(1) conjunctions. The objective is to obtain a linear order between GR(1) conjunctions that is consistent with the usual partial order induced by logical implication, and is as discriminative as possible. These requirements are formalized into three axioms in Section 6.1.

Then we define a three-component weakness measure based on entropy and Hausdorff dimension. Each of the three components induce an order consistent with some of the axioms in different classes of GR(1) conjunctions. In Section 6.2, we introduce these classes and prove which axioms are satisfied by each of the components. Specifically, *(i)* we show that entropy satisfies all axioms in the class of strongly connected invariants; *(ii)* we introduce the notion of fairness complement and show that its Hausdorff dimension is consistent with the axioms in the class of fairness conditions; *(iii)* we generalize the notion of fairness complement to non-strongly connected GR(1) formulae, and define a three-component weakness measure consistent with implication in this general case. Section 6.3 presents an algorithm to compute the third component from the automaton of the GR(1) conjunction.

Finally, we show in Section 6.4 two applications of the weakness measure. In the first, we use it

as a sorting heuristic in the GR(1) assumptions refinement problem. In the second, as a means to the quantitative model checking problem. The latter application, in particular, is included to demonstrate that the weakness measure can be employed in a wider range of applications than the scope of this thesis.

The content of this chapter is largely based on an extension of our work published in [CAG18], accepted for publication in [CAG20a].

6.1 Weakness of LTL Formulae

In this section, we present an axiomatization of the weakness of an LTL formula. Hereafter, we denote the weakness measure of an LTL formula as a function $d : \Phi \to P$, where Φ is the set of LTL formulae defined over a set of variables $\mathcal{V}$ and P is a totally ordered set (such as the set of n-tuples $\mathbb{R}^n$ with lexicographic order). The weakness measure of a formula $\phi \in \Phi$, denoted by $d(\phi)$, is interpreted as follows: the higher this measure is, the weaker ϕ is, that is, ϕ_2 is weaker than ϕ_1 if $d(\phi_1) \leq d(\phi_2)$.

In settings such as [ADG16, AMT13, Ses12], an LTL formula ϕ_2 is *weaker* than ϕ_1 if and only if $\phi_1 \to \phi_2$ is valid (that is, it is true for any ω-word). Semantically, this translates to language inclusion: namely, ϕ_2 is weaker than ϕ_1 iff $L(\phi_1) \subseteq L(\phi_2)$. This gives us the first axiom of weakness.

Axiom 6.1 (Consistency with implication)**.** *Given two LTL formulae ϕ_1 and ϕ_2, if $\phi_1 \to \phi_2$, then $d(\phi_1) \leq d(\phi_2)$.*

In addition to this, it is desirable that a weakness measure performs at least as well as implication in discriminating pairs of formulae where one strictly implies the other.

Axiom 6.2 (Consistency with strict implication)**.** *Given two LTL formulae ϕ_1 and ϕ_2 such that $\phi_1 \to \phi_2$ is valid and $\phi_2 \to \phi_1$ is not. Then $d(\phi_1) < d(\phi_2)$.*

Notice that this criterion defines a partial ordering of specifications: if none of the two formulae

implies the other, then they are *incomparable with respect to implication* according to this criterion.

The above two axioms are for formulae that are comparable via implication. However, even for the incomparable case, it may be useful to define some ordering between formulae based on their "permissiveness".

Consider the simple case of two invariants over $\mathcal{V} = \{a, b, c\}$, $\phi_1 = \mathbf{G}(a \wedge b)$ and $\phi_2 = \mathbf{G}c$. Even if the two formulae are incomparable wrt implication, it is clear that ϕ_1 allows in some sense fewer behaviours than ϕ_2 (and hence more permissive): at each time step, the former allows for 2 distinct valuations of $\mathcal{V}$ while ϕ_2 allows 4 of them.

Furthermore, consider the formulae $\phi_3 = \mathbf{G}(a \rightarrow \mathbf{X}b)$ and $\phi_4 = \mathbf{G}((a \wedge b) \rightarrow \mathbf{X}c)$ instead. Despite neither implying the other, we note that ϕ_3 is more restrictive than ϕ_4 asymptotically: that is, for a large enough n, the number of finite prefixes of length n that satisfy ϕ_3 is less than the number of finite prefixes of length n satisfying ϕ_4 ($|L(\phi_3)| < |L(\phi_4)|$). This can be easily understood if one considers that ϕ_3 poses a restriction to the next symbol in an ω-word whenever a is true (which holds in 4 out of 8 possible valuations of $\mathcal{V}$), while ϕ_4 poses a similar restriction when $a \wedge b$ holds (in 2 out of the 8 valuations).

The number of prefixes admitted by an LTL formula up to a time step k is already used as a measure of the permissiveness of a formula (for instance in model counting [DCA+18]). This notion provides an intuitive criterion to impose a linear order when implication does not hold, captured by the following axiom:

Axiom 6.3 (Consistency with permissiveness)**.** *Given two LTL formulae ϕ_1 and ϕ_2, $d(\phi_1) \leq d(\phi_2)$ if there exists some length $\bar{n}$ such that, for every $n > \bar{n}$, the set of prefixes of length n in $L(\phi_1)$ contains fewer elements than the set of prefixes of the same length in $L(\phi_2)$; i.e., if $\exists \bar{n},\ \forall n > \bar{n},\ |\mathsf{pref}_n(L(\phi_1))| \leq |\mathsf{pref}_n(L(\phi_2))|$, then $d(\phi_1) \leq d(\phi_2)$.*

In this chapter, we focus on defining a weakness measure for GR(1) formulae satisfying Axiom 6.1. We also identify a class of formulae in which the weakness measure satisfies 6.3. We then show that, although our weakness measure is not guaranteed to satisfy Axioms 6.2 in

general, our proposed extensions to the measure aim at augmenting its resolution, increasing the set of formulae that can be strictly ordered via the measure when strict implication holds; we also identify two classes of formulae in which Axiom 6.2 is provably satisfied.

6.2 Weakness Measure of GR(1) Conjunctions

In this section, we first discuss the use of entropy and Hausdorff dimension in measuring the weakness of GR(1) conjunctions. We then demonstrate that these measures individually do not guarantee that the axioms in Section 6.1 hold for the general class of GR(1) conjunctions. To alleviate this problem, we introduce restrictions on GR(1) conjunctions for which these axioms hold. Then we gradually lift these restrictions, providing a more refined weakness measure satisfying the axioms for larger subclasses of GR(1) conjunctions.

6.2.1 Entropy and Hausdorff Dimension as Weakness Measures

Entropy and Hausdorff dimension, defined in Section 2.4, provide an ordering consistent with the weakness notion defined in Section 6.1. While entropy is, by definition, the growth rate of the number of prefixes in the language as the length of the prefixes increases (and has been used in the context of quantitative model checking, see Section 6.4.3), we can interpret Hausdorff dimension as a measure of the asymptotic degrees of freedom of an ω-language: it quantifies how many different continuations are allowed from a prefix of an ω-word given that the corresponding Muller automaton does not leave an accepting subset.

It was already demonstrated in Examples 2.5 and 2.6 that the entropy and Hausdorff dimension can be different. In what follows, we further explore the behavior of the formula $\phi_1 = \mathsf{FG}a$ from those examples.

Example 6.1. *Consider the LTL formula $\phi_1 = \mathsf{FG}a$ over the variable set $\mathcal{V} = \{a\}$ whose Muller automaton is shown in Figure 6.1.*

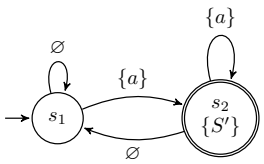


Figure 6.1: DMA of $L(\phi_1)$. In this and the following DMAs, the collection of accepting sets to which a state belongs is enclosed in curly braces inside the state.

Notice that, for any $w \in L(\phi_1)$, both valuations of $\mathcal{V}$ are allowed the accepting state, and satisfying $\mathbf{G}a$ may be delayed arbitrarily. Therefore, for any finite n, $|\mathsf{pref}_n(L)| = 2^n$, and thereby $H(L(\phi_1)) = 1 = H(L(\mathsf{true}))$ (see the definition of entropy in Section 2.4). In this simple DMA, there is only a single accepting state set $S' = \{s_2\}$, with corresponding ω-language $C_{S'} = \{\{a\}^\omega\}$. This implies $|\mathsf{pref}_n(C_{S'})| = 1$, and so the Hausdorff dimension is $\mathsf{dim}\,(L(\phi_1)) = H(C_{S'}) = 0$. This example demonstrates that the Hausdorff dimension isolates the asymptotic behaviour of $L(\phi_1)$ as it depends only on the condition $\mathbf{G}a$ that is eventually satisfied by any ω-word in the ω-language. This allows the strict ordering of the two formulae (ϕ_1 and true); one is strictly stronger than the other in the sense of implication, but has the same entropy.

The following proposition shows that both entropy and Hausdorff dimension are consistent with implication, hence satisfy Axiom 6.1.

Proposition 6.1. *Given two LTL formulae ϕ_1 and ϕ_2 such that $\phi_1 \to \phi_2$ is valid, $H(L(\phi_1)) \leq H(L(\phi_2))$ and $\mathsf{dim}\,(L(\phi_1)) \leq \mathsf{dim}\,(L(\phi_2))$.*

Proof. If $L(\phi_1) \subseteq L(\phi_2)$, for every $n \in \mathbb{N}$ $\mathsf{pref}_n(L(\phi_1)) \subseteq \mathsf{pref}_n(L(\phi_2))$. Then $|\mathsf{pref}_n(L(\phi_1))| \leq |\mathsf{pref}_n(L(\phi_2))|$ and the relation between the entropies follows immediately.

The relation between the Hausdorff dimensions follows from the language inclusion $L(\phi_1) \subseteq L(\phi_2)$ and the monotonicity of Hausdorff dimension with respect to language inclusion [MS94]. □

In addition, it follows immediately from the definition that entropy is consistent with Axiom 6.3:

Proposition 6.2. *Given two LTL formulae ϕ_1 and ϕ_2 such that there exists an $\bar{n}$ such that $|\mathsf{pref}_n(L(\phi_1))| \leq |\mathsf{pref}_n(L(\phi_2))|$ for all $n > \bar{n}$, then $H(L(\phi_1)) \leq H(L(\phi_2))$.*

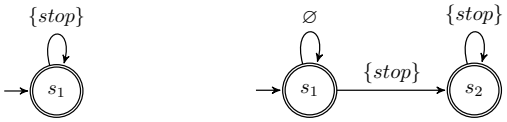


Figure 6.2: DBAs of ρ_a (left) and ρ_b (right)

Therefore, entropy is a good candidate for the weakness measure. However, it does not necessarily satisfy Axiom 6.2: there are pairs of formulae with the same entropy such that one strictly implies the other. While this was already demonstrated in Example 6.1 (for the non-GR(1) formula $\mathbf{FG}a$ and the constant *true*), next we provide two more informative examples involving GR(1) formulae only, highlighting the two reasons behind this issue: the lack of strong connectedness (Example 6.2) and the presence of fairness conditions (Example 6.3).

Example 6.2. *Consider the variable set* $\mathcal{V} = \{\mathsf{stop}\}$ *and the formulae* $\rho_a = \mathbf{G}\mathsf{stop}$ *and* $\rho_b = \mathbf{G}(\mathsf{stop} \rightarrow \mathbf{X}\mathsf{stop})$. *Their Büchi automata are shown in Figure 6.2. Clearly* $\rho_a \rightarrow \rho_b$ *in the strict sense. However the two languages have the same entropy* $H(L(\rho_a)) = H(L(\rho_b)) = 0$.

Example 6.3. *Consider the two formulae over the variables* $\mathcal{V} = \{a, b\}$*:* $\phi_a = \mathbf{G}(a \rightarrow \mathbf{X}b) \wedge \mathbf{GF}a$ *and* $\phi_b = \mathbf{G}(a \rightarrow \mathbf{X}b) \wedge \mathbf{GF}b$. *The same strongly connected invariant appears in both, and therefore the two formulae have the same entropy and Hausdorff dimension, but the fairness condition in* ϕ_b *(i.e.,* $\mathbf{GF}b$*) is always satisfied when the fairness condition of* ϕ_a *(i.e.,* $\mathbf{GF}a$*) is satisfied, by virtue of the invariant itself. However, the* ω*-word* $\{b\}^\omega$ *satisfies* ϕ_b *but not* ϕ_a. *So,* ϕ_a *implies* ϕ_b *but not vice versa.*

The language of both formulae, shown in Figure 6.7, is not closed, but is strongly connected. Therefore, like in the case of fairness conditions, the entropy and the Hausdorff dimension both coincide with the spectral radius of the only SCC of the automaton; this SCC coincides with the automaton of the common invariant alone. Therefore,

$$H(L(\phi_a)) = H(L(\phi_b)) = \mathsf{dim}\,(L(\phi_a)) = \mathsf{dim}\,(L(\phi_b)) = \mathsf{dim}\,(L(\mathbf{G}(a \rightarrow \mathbf{X}b)))\ .$$

□

In cases where the Hausdorff dimension is not equal to entropy, the Hausdorff dimension can

help to discriminate between two formulae. In other cases, the Hausdorff dimension may not be sufficient. In such cases, we propose an additional component to the weakness measure which can help distinguish between two formulae.

In the following subsections, we first identify a class of formulae where entropy alone is guaranteed to satisfy all three axioms, namely the class of *closed, strongly connected* ω-languages. For ω-languages outside of this class, we take a best-effort approach. Despite not guaranteeing Axiom 6.2, we augment the weakness measure with additional components that in some cases induce on these ω-languages a strict order consistent with strict implication.

To this end, we show how Hausdorff dimension helps in case of non-strongly connected ω-languages. Finally, for non-closed ω-languages, we identify a language decomposition based on the negation of fairness conditions such that the Hausdorff dimension of the sublanguage we define (which we call *fairness complement*, $L(J^c)$) can discriminate between formulae with the same entropy and Hausdorff dimension. We use these three components to define a weakness measure $d(\phi)$ of a formula ϕ: the first component is the entropy $H(L(\phi))$, the second is its Hausdorff dimension $\mathsf{dim}\,(L(\phi))$, and the third is the entropy of the fairness complement $\mathsf{dim}\,(L(J^c))$; d induces a lexicographic ordering of using the three components. In the following, we prove that this ordering is consistent with implication for classes of ω-languages of increasing complexity: (i) ω-languages of formulae with a fixed invariant; (ii) ω-languages of formulae with strongly connected invariants; and (iii) ω-languages of GR(1) conjunctions with the same number of accepting SCCs. Although d does not always comply with Axiom 6.2, in several cases it induces a strict order consistent with strict implication. Table 6.1 lists the classes of ω-languages along with the components of the weakness measure introduced to order them and the axioms that are guaranteed to hold.

Notice that when the entropies of two formulae are equal, the order induced by d may violate Axiom 6.3. As we will show, this is due to the fact that neither the entropy nor the Hausdorff dimension take into account the strongly connected components that have lower spectral radius than the maximum (see Section 5.2). However, in our work we choose to prioritize the consistency with implication since we are driven by the application of GR(1) assumption refinement.

Table 6.1: Summary of findings

Class of formulae	**Weakness components**	**Axioms satisfied**	**Theorem involved**
Strongly connected invariants	$H(L(\phi))$	Axiom 6.1 Axiom 6.2 Axiom 6.3	Proposition 6.1 Theorem 6.1 Proposition 6.2
Fairness conditions	$\mathsf{dim}\,(L(\neg J))$	Axiom 6.1 Axiom 6.2 Axiom 6.3	Theorem 6.3 Theorem 6.3 Trivial (fairness does not affect prefixes)
Common, strongly connected invariant	1. $H(L(\phi))$ 2. $\mathsf{dim}\,(L(\phi^c)) = \mathsf{dim}\,(L(\rho \wedge \neg J))$	Axiom 6.1	Theorem 6.5
Strongly connected invariant	1. $H(L(\phi))$ 2. $\mathsf{dim}\,(L(\phi^c)) = \mathsf{dim}\,(L(\rho \wedge \neg J))$	Axiom 6.1	Corollary 6.2, Theorem 6.5
GR(1) conjunctions	1. $H(L(\phi))$ 2. $\mathsf{dim}\,(L(\phi))$ 3. $\mathsf{dim}\,(L^c) = \mathsf{dim}\,(L(C_S \wedge \neg J))$	Axiom 6.1	Theorem 6.9 (under some conditions on the structure of the automata)

Ensuring that the order relation is consistent with permissiveness in all cases is a subject for future work.

In our analysis, we do not take into account initial conditions, which in general may or may not affect the weakness measure. This is because both the entropy and the Hausdorff dimension depend only on the reachable SCCs in the formula's automaton (see Section 5.2). The weakness measure of $\phi^{init} \wedge \rho$ is the same as that of ρ, provided that all the SCCs of an automaton of ρ are still reachable in the automaton of $\phi^{init} \wedge \phi^{inv}$.

6.2.2 Ordering Invariants and Strongly Connected Invariants

Consider the formula $\rho = \mathbf{G}\beta(\mathcal{V} \cup \mathbf{X}\mathcal{V})$. The ω-language $L(\rho)$ is closed. Hence, the entropy of $L(\rho)$ coincides with its Hausdorff dimension and is computed as the maximum eigenvalue of the adjacency matrix of its Büchi automaton (see Section 5.2). From this equivalence and Definition 2.36, it is easy to see that in this case both quantities satisfy Axiom 6.3. Therefore they can be used as a measure of weakness interchangeably.

In general, Proposition 6.1 allows one invariant to be strictly weaker than another according to implication despite both having equal entropies. Examples 6.2 and 6.3 suggest when this can happen: that is, when comparing two formulae containing fairness conditions, or when one of the formulae does not have a strongly connected automaton.

In this section, we prove the existence of a subclass of invariants for which entropy and Hausdorff dimension are strictly monotonic with respect to implication. This subclass is characterized through the concept of *strong connectedness* of an ω-language.

The DBA in Figure 6.3 accepts a strongly connected ω-language, while the automaton in Figure 6.2 (right) accepts a non-strongly connected one.

Using this notion, we provide in the next theorem a sufficient condition over invariants for Axiom 6.2 to be satisfied:

Theorem 6.1. *Let $\rho_a = \mathbf{G}B_a(\mathcal{V} \cup \mathbf{X}\mathcal{V})$ and $\rho_b = \mathbf{G}B_b(\mathcal{V} \cup \mathbf{X}\mathcal{V})$ be two non-empty invariants such that $\rho_a \to \rho_b$ is valid, $\rho_b \to \rho_a$ is not valid (that is, ρ_a implies ρ_b strictly) and ρ_b is strongly connected. Then $H(L(\rho_a)) < H(L(\rho_b))$ and $\mathsf{dim}\,(L(\rho_a)) < \mathsf{dim}\,(L(\rho_b))$.*

Proof. The proof consists of three steps.

1. First we construct, for any invariant ρ, an automaton $\mathcal{B}$ accepting $L(\rho)$ where each state keeps memory of the latest symbol read by the automaton; . This construction is similar to the one in [KE12], although, unlike them, we do not need to label states with LTL subformulae.

2. Then we show that a Büchi automaton of $L(\rho_a)$ can be constructed by removing transitions and possibly states from the automaton of $L(\rho_b)$ obtained using the construction in step 1.

3. Finally, using some algebraic properties of the adjacency matrices of the two Büchi automata, we prove that the spectral radius of the automaton of $L(\rho_a)$ is less than the spectral radius of the automaton of $L(\rho_b)$. Hence $H(L(\rho_a)) < H(L(\rho_b))$ and $\mathsf{dim}\,(L(\rho_a)) < \mathsf{dim}\,(L(\rho_b))$ (since both entropy and Hausdorff dimension are equal to the spectral radius when the corresponding automaton is strongly connected, see Eq. 2.4 and Theorem 5.1).

The construction of $\mathcal{B}$ for $L(\rho)$ uses a set of states Q which are labelled by a one-to-one function $\lambda : Q \setminus \{q_0\} \to \Sigma$. The transition function is built such that

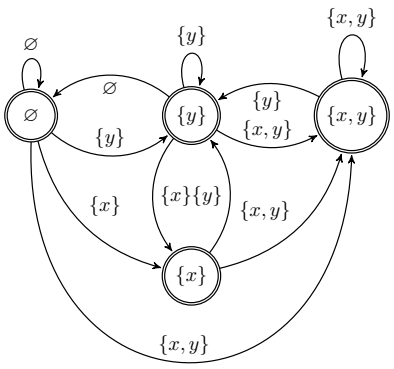


Figure 6.3: Büchi automaton of $\mathbf{G}(x \rightarrow \mathbf{X}y)$. The state labels $\lambda(q)$ are shown inside the nodes. The initial state (not shown) has transitions towards all the states in the figure, since the first symbol is unconstrained.

1. $\delta(q, \sigma)$ is defined if and only if every ω-word in $\lambda(q)\sigma \cdot \Sigma^\omega$ satisfies $\beta(\mathcal{V} \cup \mathbf{X}\mathcal{V})$;

2. in case $\delta(q, \sigma)$ is defined, $\lambda(\delta(q, \sigma)) = \sigma$.

The initial state q_0 satisfies the following properties:

1. for every $\sigma \in \Sigma$, there exists $\delta(q_0, \sigma)$ if and only if there exists $\tau \in \Sigma$ such that $\sigma\tau \cdot \Sigma^\omega \models \beta_b$;

2. for every $q \in Q \setminus \{q_0\}$, if $q = \delta(q_0, \sigma)$ then $\lambda(q) = \sigma$.

An example is depicted in Figure 6.3.

We prove that an ω-word w satisfies ρ if and only if it corresponds to an infinite path on $\mathcal{B}$. Suppose $w \models \rho$. Then we prove by induction that it induces an infinite path on $\mathcal{B}$. For every $i \in \mathbb{N}$, $w[i..] \models \beta(\mathcal{V} \cup \mathbf{X}\mathcal{V})$. Since $\beta(\mathcal{V} \cup \mathbf{X}\mathcal{V})$ constrains the first two symbols of w only, any ω-word in $w[i]w[i+1] \cdot \Sigma^\omega$ satisfies β. By construction, the automaton $\mathcal{B}$ has a transition $\delta(q_i, w[i+1]) = q_{i+1}$ such that $\lambda(q_i) = w[i]$ and $\lambda(q_{i+1}) = w[i+1]$. Therefore, if there exists a path from q_0 to q_i induced by the prefix $w[1] \dots w[i]$, there exists a path from q_0 to q_{i+1}. As a base case of the induction, consider that $w[1]w[2] \cdot \Sigma^\omega \models \beta$, and thereby there exists a path from q_0 to q_1 in $\mathcal{B}$.

Conversely, suppose that w is an infinite sequence of transition labels such that $\mathcal{B}(w) = q_0 q_1 \dots$. Then $\delta(q_i, w[i+1])$ exists for every $i \in \mathbb{N} \cup \{0\}$ and $\lambda(q_i) = w[i]$ for all $i \in \mathbb{N}$. By the construction

of $\mathcal{B}$ this means that for every $i \in \mathbb{N}$ $w[i]w[i+1] \cdot \Sigma^\omega \models \beta$, which implies $w[i..] \models \rho$. Then we can conclude $w \models \rho$. This allows us to say that $\mathcal{B}$, so constructed, is a Büchi automaton of the formula ρ.

If $L(\phi)$ is strongly connected, then $\mathcal{B}$ is also strongly connected, except for the initial state. To prove this, let $q_n = \delta(q_0, w) = \delta(\delta(\delta(\dots\delta(q_0, w[1]), \dots), w[n-1]), w[n])$ be the state reached by $\mathcal{B}$ after reading the prefix $w \in \mathsf{pref}_n(L(\phi))$, and $q_m = \delta(q_0, v)$ for $v \in \mathsf{pref}_m(L(\phi))$. By construction, $\lambda(q_n) = w[n]$ and $\lambda(q_m) = v[m]$. Since $L(\phi)$ is strongly connected, there exist $v' \in \Sigma^*$ such that $L_{vv'} = L(\phi)$. Therefore $vv'w \in \mathsf{pref}(L(\phi))$ and $\lambda(q_0, vv'w) = w[n]$, so $\delta(q_0, vv'w) = q_n$. So, there exists a path in $\mathcal{B}$ from q_m to q_n. Symmetrically, there exists a path from q_n to q_m. We have proved that for any pair of states reachable from q_0 there is a path between them in both directions, that is the graph induced by non-initial states is strongly connected.

Now consider the automata $\mathcal{B}_a$ and $\mathcal{B}_b$ of $L(\rho_a)$ and $L(\rho_b)$ respectively. Since the two formulae are pure invariants, $\rho_a \to \rho_b$ if and only if $\beta_a(\mathcal{V} \cup \mathbf{X}\mathcal{V}) \to \beta_b(\mathcal{V} \cup \mathbf{X}\mathcal{V})$. So, given the hypothesis that ρ_a is strictly stronger than ρ_b there must exist a pair of valuations σ, τ such that $\sigma\tau \cdot \Sigma^\omega \models \rho_b$ but $\sigma\tau \cdot \Sigma^\omega \not\models \rho_a$. By construction, this corresponds to at least one transition $\delta(q, \tau)$ that exists in $\mathcal{B}_b$ and does not in $\mathcal{B}_1$. Consequently, we can conclude that $\mathcal{B}_a$ is a proper subgraph of $\mathcal{B}_b$.

The next step is to construct the adjacency matrices corresponding to $\mathcal{B}_a$ and $\mathcal{B}_b$ excluding the respective initial states. Let $Q_a \setminus \{q_{0,a}\}$ and $Q_b \setminus \{q_{0,b}\}$ be the set of non-initial states of $\mathcal{B}_a$ and $\mathcal{B}_b$, respectively, and δ_a and δ_b their respective transition functions. Let M^a and M^b be the adjacency matrices of the graphs $(Q_a \setminus \{q_0\}, \delta_a)$ and $(Q_b \setminus \{q_0\}, \delta_b)$. Consider that by construction all transitions between two states are labelled by exactly one valuation: so, each element of these two matrices is either a 0 or a 1.

Since the transitions of δ_b are a proper subset of the transitions of δ_a, we have for each element (i, j) $M^a_{ij} \leq M^b_{ij}$, and $M^a_{kh} < M^b_{kh}$ for some (k, h), that is to say $M^a_{kh} = 0, M^b_{kh} = 1$. Since ρ_b is strongly connected, its adjacency matrix M^b is irreducible (for more details on the correspondence between strongly connected digraphs and irreducible matrices see Chapter 6 of [HJ86]). Moreover, $M^a + M^b$ is also irreducible, since it is the sum of two nonnegative matrices one of

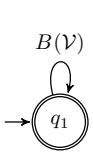


Figure 6.4: DBA of a one-state invariant.

which is irreducible. We can therefore apply the property stated in Chapter 2, Corollary 1.5 of [BP94], which guarantees that under the given conditions, $\nu(M^a) < \nu(M^b)$.

Taking the logarithm on both sides, we get $\mathsf{dim}\,(\rho_a) < \mathsf{dim}\,(\rho_b)$, finishing the proof. □

Remark 6.1. *An interesting strongly connected invariant is the* one-state invariant*, i.e., one that does not use the* $\mathbf{X}$ *operator:* $\rho_s = \mathbf{G}\beta(\mathcal{V})$ *whose DBA is shown in Figure 6.4. (For succinctness, the set of valuations that label a transition between the same states is denoted by the Boolean expression characterizing it.)*

In this case, the Hausdorff dimension has a closed form:

$$\mathsf{dim}\,(\rho_s) = \mathsf{log}_r\,|\beta(\mathcal{V})|$$

where $r = 2^{|\mathcal{V}|}$ *is the number of valuations of* $\mathcal{V}$ *and* $|\beta(\mathcal{V})|$ *is the number of valuations that satisfy* $\beta(\mathcal{V})$*. Invariants of this type are clearly strongly connected and satisfy Theorem 6.1.*

Remark 6.2. *Typical examples of GR(1) specifications are often created manually (like the ones for device communication protocols) and use of strongly connected environment assumptions. It is indeed natural to allow environments to be reset to their initial state after some steps. However, strong connectedness may not hold especially when the specifications contain Dwyer's patterns from [DAC99] (which can be converted into GR(1) by following the procedure described in [MR15]). As a simple example, consider the assumption* $a\mathcal{U}b$*: its automaton, shown in Figure 6.5, is not strongly connected. In those cases, using the Hausdorff dimension alone to compare the weakness of different formulae may lead to the problem described Example 6.2.*

An interesting property that follows directly from Theorem 6.1 is the following.

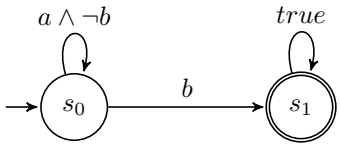


Figure 6.5: DBA of $a\mathcal{U}b$

Corollary 6.1. *Let ρ_a and ρ_b be two invariants with the same Hausdorff dimension $\mathsf{dim}\,(L(\rho_a)) = \mathsf{dim}\,(L(\rho_b))$, such that ρ_b is strongly connected and $\rho_a \to \rho_b$ is valid. Then ρ_a and ρ_b are equivalent.*

Proof. If $\rho_b \to \rho_a$ were not valid, then by Theorem 6.1 we would have $\mathsf{dim}\,(L(\rho_a)) < \mathsf{dim}\,(L(\rho_b))$ and contradict the hypothesis. Therefore $\rho_a \leftrightarrow \rho_b$ is valid, that is, the two invariants are equivalent. □

This corollary gives us a sufficient condition to determine the equivalence of two ω-languages from their entropy/Hausdorff dimension. We will use it later when defining the third component of the weakness measure.

Theorem 6.1 suggests a first version of the weakness measure for comparing strongly connected invariants. This can be either entropy or Hausdorff dimension, since both have the same value for this class of formulae. We choose entropy as it satisfies Axiom 6.3 in a wider class of formulae, as we will see.

Definition 6.1 (Weakness of strongly connected invariants)**.** *Let $\phi = \rho$ be a strongly connected invariant. Then its weakness is defined as*

$$d_1(\phi) := H(L(\rho)) \ . \tag{6.1}$$

The partial order relation of strongly connected invariants is the order induced by d_1:

$$d(\phi_a) \leq d(\phi_b) \text{ if and only if } d_1(\phi_a) \leq d_1(\phi_b)$$

Hence we have the following result:

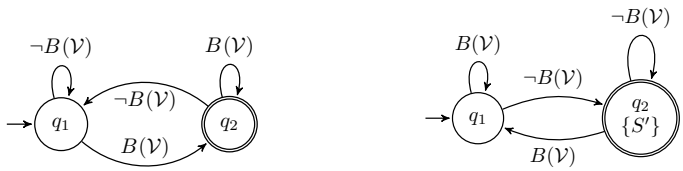


Figure 6.6: DBA of $L(J)$ (left) and DMA of $L(J^c)$ (right).

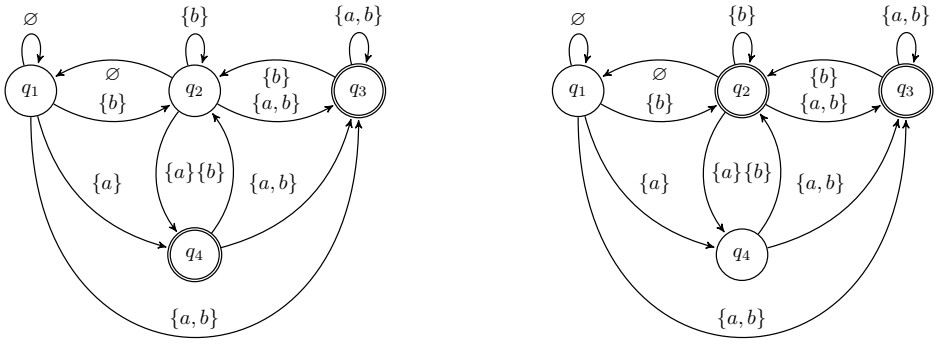


Figure 6.7: DBAs of ϕ_a (left) and ϕ_b (right) of Example 6.3.

Theorem 6.2. *In the class of strongly connected invariants the weakness measure $d_1(\phi)$ satisfies Axiom 6.1 (by Proposition 6.1), Axiom 6.2 (by Theorem 6.1), and Axiom 6.3 (by Proposition 6.2).*

6.2.3 Ordering Fairness Conditions via Fairness Complements

Consider the generic fairness condition $J = \mathbf{GF}\beta(\mathcal{V})$ whose DBA $\mathcal{B}$ is shown in Figure 6.6. This language is not closed: take a symbol $\sigma \in \Sigma$ that does not satisfy $\beta(\mathcal{V})$ and the ω-word σ^ω. It is clear that $\mathsf{pref}(\{\sigma^\omega\}) \subseteq \mathsf{pref}(L(J))$, but $\sigma^\omega \notin L(J)$.

Notice that $\mathcal{B}$ consists of only one SCC, which is an accepting one. Therefore, the entropy is the spectral radius of the adjacency matrix of $\mathcal{B}$ (see Section 2.4). Analogously, the Hausdorff dimension is computed via Algorithm 10: in this case, the only accepting SCC is also the only SCC in $\mathcal{B}$. Therefore, despite the language not being closed, the entropy and the Hausdorff dimension are the same. In particular, a simple calculation shows that $H(L(\phi)) = \mathsf{dim}\,(L(\phi)) = 1 = H(L(\mathsf{true}))$: in other words, entropy alone is insufficient to show that the constant true is weaker than any fairness condition that is not always true.

To allow for a distinction to be made, we propose to measure the weakness of the negation of this formula instead. In this section, we show that pairs of negated fairness conditions where one implies the other strictly can be discriminated by their Hausdorff dimensions.

We call an LTL formula of the form $J^c = \mathbf{FG}\neg\beta(\mathcal{V})$ a *fairness complement*[1]. It is well known that this formula cannot be represented by a DBA [GTW02]. So to compute the Hausdorff dimension we need to use Algorithm 2, which works on DMAs. The DMA of $L(J^c)$ is shown to the right of Figure 6.6. The only accepting set is $S' = \{q_2\}$. (Notice that unlike the one to the left, this automaton accepts only words that stay forever in q_2 from a certain point onwards.) The language $C_{S'}$ has an entropy of $\mathsf{log}_r\,|\neg\beta(\mathcal{V})|$. Hence

$$\mathsf{dim}\,(L(J^c)) = \mathsf{log}_r\,|\neg\beta(\mathcal{V})| = \mathsf{log}_r(r - |\beta(\mathcal{V})|)$$

where $r = 2^{|\mathcal{V}|}$, and $|\beta(\mathcal{V})|$ is the number of valuations satisfying $\beta(\mathcal{V})$.

This formula shows that the Hausdorff dimension of the fairness complement is strictly monotonic with the implication relation, but in reverse order: the weaker the fairness condition, the lower the Hausdorff dimension is. This proves the following theorem:

Theorem 6.3. *Let J_a and J_b be two fairness conditions such that $J_a \to J_b$ is valid and $J_b \to J_a$ is not valid. Then $\mathsf{dim}\,(L(\neg J_a)) > \mathsf{dim}\,(L(\neg J_b))$.*

This result suggests a version of the weakness measure that satisfies Axiom 6.2 for fairness conditions.

Definition 6.2 (Weakness of fairness conditions)**.** *Let $\phi = J = \mathbf{GF}\beta(\mathcal{V})$ a fairness condition over the variables in $\mathcal{V}$. Define $J^c := \neg J = \mathbf{FG}\neg\beta(\mathcal{V})$. The* weakness *of J, denoted by $d(J)$, is defined as*

$$d_3(\phi) := \mathsf{dim}\,(J^c) \tag{6.2}$$

The order relation on the class of fairness conditions is the following:

$$d(\phi_a) \leq d(\phi_b) \text{ if and only if } d_3(\phi_a) \geq d_3(\phi_b)$$

[1] We will refine this notion later on.

In the definition of the weakness measure for the most general class of GR(1) conjunctions (presented in Section 6.2.6), we use d_3 as the third component to be checked for measuring the weakness (this explains the use of subscript 3 in d_3).

The weakness measure d_3 satisfies all three axioms of weakness:

Theorem 6.4. *The weakness measure d_3 satisfies Axioms 6.1, 6.2 and 6.3 over fairness conditions.*

Proof. Axiom 6.3 is trivially satisfied. In fact, no fairness condition J imposes any restriction on the finite prefixes of $L(J)$, and therefore $|\mathsf{pref}_n(L(J))| = r^n$, independently of J. Axioms 6.1 and 6.2 are satisfied as a direct consequence of Theorem 6.3. □

6.2.4 Ordering GR(1) Conjunctions with a Common Strongly Connected Invariant

Consider a GR(1) conjunction $\phi = \rho \wedge \bigwedge_{i=1}^{m} J_i$, where ρ is strongly connected, and take two formulae of this kind with the same invariant. We showed in Example 6.3 that even when ρ is strongly connected, in some cases, entropy and Hausdorff dimension may not distinguish between the weaker and the stronger of the two in the implication sense. This problem has been previously pointed out for entropy in the work of [ABD14a] (see also Section 6.4.3).

In general, entropy cannot distinguish between strongly connected languages with the same invariant. In this case, in fact, the automaton has only one SCC, which coincides with the SCC of the invariant, implying that the entropy and Hausdorff dimension are the same for both languages. We have shown this in Example 6.3, where adding any fairness condition to a strongly connected invariant does not change its entropy or Hausdorff dimension.

To capture the addition of any fairness condition to a fixed invariant, we propose using a language decomposition based on negating fairness conditions. In fact, in the previous subsection we have seen that the Hausdorff dimension can distinguish between negated fairness conditions.

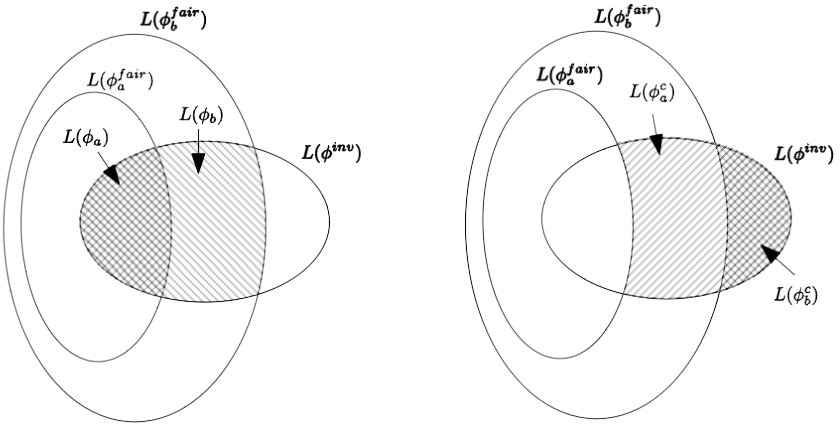


Figure 6.8: Subset relations between $L(\phi_a)$, $L(\phi_b)$, $L(\phi_a^c)$ and $L(\phi_b^c)$ (see Theorem 6.5), $L(\phi_a)$ and $L(\phi_a^c)$ are striped from top left to bottom right and $L(\phi_b)$ and $L(\phi_b^c)$ from top right to bottom left.

Let us consider a simpler case of formulae with the same invariant and a single fairness condition. As before, we exploit the fact that the negation of a fairness condition is a formula of the form $J^c = \mathsf{FG}\neg\beta(\mathcal{V})$ which can be strictly ordered through the Hausdorff dimension in a similar fashion as in Section 6.2.3. When the formula contains an invariant, however, consistency with implication is obtained by considering the Hausdorff dimension of $\rho \wedge J^c$ rather than J^c alone. We define the *fairness complement* in this case as the formula $\phi^c := \rho \wedge J^c$.

The idea is illustrated in Figure 6.8: Let $\phi_a = \rho \wedge J_a$ and $\phi_b = \rho \wedge J_b$ be two formulae having the same invariant, and suppose that ϕ_a implies ϕ_b. Suppose two formulae $\phi_a = \rho \wedge J_a$ and $\phi_b = \rho \wedge J_b$ having the same invariant are such that ϕ_a implies ϕ_b, as in the example above. Then the behaviours removed from ρ by the conjunction with J_a is a superset of the behaviours removed by J_b. The sets of behaviors removed by those conditions are the languages of the formulae $\phi_a^c = \rho \wedge \neg J_a$ and $\phi_b^c = \rho \wedge \neg J_b$. Unlike ϕ_a and ϕ_b, ϕ_a^c and ϕ_b^c can in some cases be discriminated by their Hausdorff dimension, as the following example shows.

Example 6.4. *Consider again the formulae from Example 6.3. Let* $\phi_a^c = \mathsf{G}(a \rightarrow \mathsf{X}b) \wedge \mathsf{FG}\neg a$ *and* $\phi_b^c = \mathsf{G}(a \rightarrow \mathsf{X}b) \wedge \mathsf{FG}\neg b$. *The DMAs of the resulting languages are shown respectively in Figure 6.9. Each of them has just one accepting singleton, so the computation of the Hausdorff dimension is straightforward:* $\mathsf{dim}\,(\phi_a^c) = \frac{1}{2}$ *and* $\mathsf{dim}\,(\phi_b^c) = 0$. *In summary, since* ϕ_a *is more restrictive than* ϕ_b, *the Hausdorff dimension of the* ω*-language cut out of the invariant by* $\mathsf{GF}a$ *is higher than the Hausdorff dimension of the behaviours excluded by* $\mathsf{GF}b$.

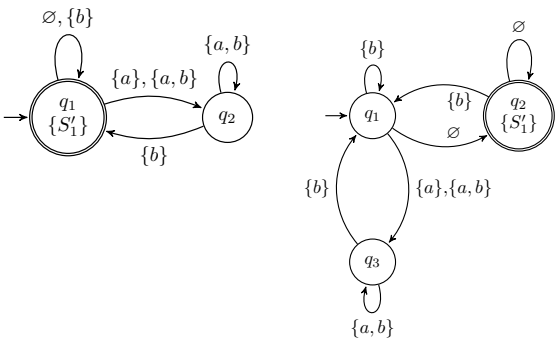


Figure 6.9: DMAs of ϕ_1^c (left) and ϕ_2^c (right) of Example 6.4.

When the GR(1) conjunction contains more than one fairness conditions, we can guarantee consistency with implication by extending the definition of fairness complements. For $\phi = \rho \wedge \bigwedge_{i=1}^{m} J_i$, define $\phi^c := \rho \wedge J^c$ where $J^c := \bigvee_{i=1}^{m} \neg J_i$. The next theorem shows that using this definition of a complement, two formulae can be ordered using the Hausdorff dimension.

Theorem 6.5. *Let $\phi_a = \rho \wedge \bigwedge_{i=1}^{m} J_{a,i}$ and $\phi_b = \rho \wedge \bigwedge_{j=1}^{l} J_{b,j}$, such that $\phi_a \rightarrow \phi_b$ is valid. Let $\phi_a^c = \rho \wedge \bigvee_{i=1}^{m} \neg J_{a,i}$ and $\phi_b^c = \rho \wedge \bigvee_{j=1}^{l} \neg J_{b,j}$. Then $\mathsf{dim}\,(L(\phi_a^c)) \geq \mathsf{dim}\,(L(\phi_b^c))$.*

Proof. Since ϕ_a implies ϕ_b, $L(\phi_a) \subseteq L(\phi_b)$. Furthermore, for $i \in \{a, b\}$, $L(\phi_i) = L(\rho) \cap L(\bigwedge_{j=1}^{m} J_{i,j})$. Therefore, $L(\rho) \setminus L(\bigwedge_{j=1}^{m} J_{a,j}) \supseteq L(\rho) \setminus L(\bigwedge_{j=1}^{l} J_{b,j})$, i.e., $L(\phi_a^c) \supseteq L(\phi_b^c)$. Then, by monotonicity, $\mathsf{dim}\,(L(\phi_a^c)) \geq \mathsf{dim}\,(L(\phi_b^c))$, finishing the proof. □

This theorem provides us with a way to increase the resolution of the weakness measure by adding to the entropy of the formula a second component measuring the size of $\phi^c := \rho \wedge \bigvee_{i=1}^{m} \neg J_i$. This weakness measure provides a partial ordering that applies to sets of GR(1) conjunctions with the same strongly connected invariant.

Definition 6.3 (Weakness of GR(1) conjunctions with strongly connected invariants)**.** *The* weakness *of a GR(1) conjunction $\phi = (\rho \wedge \bigwedge_{i=1}^{m} J_i)$, denoted by $d(\phi)$, is the pair $(d_1(\phi), d_3(\phi))$ such that*

$$d_1(\phi) := H(L(\phi)), \tag{6.3}$$

$$d_3(\phi) := \mathsf{dim}\,(L(\phi^c)) = \mathsf{dim}\left(L\left(\rho \wedge \bigvee_{i=1}^{m} \neg J_i\right)\right), \tag{6.4}$$

where the ω-language $L(\phi^c)$ is called the language of fairness-violating ω-words.

The following partial ordering is defined on the weakness measure: given $d^i = (d^i_1, d^i_3)$, with $i \in a, b$, for two GR(1) conjunctions, $d^a < d^b$ if

1. *$d^a_1 < d^b_1$, or*
2. *$d^a_1 = d^b_1$ and $d^a_3 > d^b_3$.*

The more general Definition 6.6 uses the Hausdorff dimension $\mathsf{dim}\,(\phi)$ as the second component d_2. With strongly connected invariants this is redundant, since a simple application of Theorem 5.1 shows that entropy and Hausdorff dimension are equal.

6.2.5 Ordering GR(1) Conjunctions with Strongly Connected Invariants

We now show that the weakness measure of Definition 6.3 satisfies Axiom 6.1 even when comparing two formulae with different invariants, provided the invariants are strongly connected.

More precisely, next we prove that in this class, if a formula ϕ_a implies a formula ϕ_b then the invariant of ϕ_a implies the invariant of ϕ_b. Therefore, since the invariants are strongly connected, they have the same entropy and Hausdorff dimension if and only if they are equivalent (by Corollary 6.1); finally, in case they are equivalent, the relation induced by d_3 is consistent with implication by the same argument as in Theorem 6.5.

The first step is formalized by the following theorem.

Theorem 6.6. *Let $\phi_a = \rho_a \wedge J_a$ and $\phi_b = \rho_b \wedge J_b$ be two GR(1) conjunctions, and suppose ϕ_a is satisfiable and ρ_a is strongly connected. If $\phi_a \rightarrow \phi_b$ is valid, then $\rho_a \rightarrow \rho_b$ is also valid.*

Notice that just the strong connectedness of ϕ_a (the stronger formula) is required for the theorem to hold.

Proof. By contradiction, suppose that $L(\rho_a) \not\subseteq L(\rho_b)$, where $\rho_a = \mathbf{G}\beta_a^{inv}(\mathcal{V} \cup \mathbf{X}\mathcal{V})$, $\rho_b = \mathbf{G}\beta_b^{inv}(\mathcal{V} \cup \mathbf{X}\mathcal{V})$, $J_a = \mathbf{GF}\beta_a^{fair}(\mathcal{V})$, and $J_b = \mathbf{GF}\beta_b^{fair}(\mathcal{V})$. Then there exists an ω-word w such that $w \models \rho_a$ and $w \not\models \rho_b$.

Since ρ_b does not contain nested $\mathbf{X}$ operators, there is at least one one-step transition in w violating $\beta_b^{inv}(\mathcal{V} \cup \mathbf{X}\mathcal{V})$. Let i be the minimum index such that $w[i]w[i+1] \cdot \Sigma^\omega \not\models \beta_b^{inv}(\mathcal{V} \cup \mathbf{X}\mathcal{V})$.

Now let us consider a word $\bar{w} \models \phi_a$ such that $\bar{w} = \bar{w}[0]\bar{w}[1] \ldots (\bar{w}[f]\bar{v})^\omega$, where for every j, $\bar{w}[j] \in \Sigma$, $\bar{v} \in \Sigma^*$, and $\bar{w}[f] \models \beta_a^{fair}(\mathcal{V})$. This word satisfies ϕ_a, which means that it satisfies both its invariant and fairness condition.

Since ρ_a is strongly connected, we can construct a third ω-word $\bar{\bar{w}}$ satisfying ρ_a: take the first $i+1$ symbols from w, until reaching the violation of b; then concatenate a word s that takes the automaton of ρ_a back to the initial state (s exists because of the strong connectedness), and finally concatenate $\bar{\bar{w}}$, which is strongly connected. That is, $\bar{\bar{w}} = w[0] \ldots w[i]w[i+1]s\bar{w}[0] \ldots (\bar{w}[f]\bar{v})^\omega$.

The word $\bar{\bar{w}}$ satisfies ϕ_a by construction, since it satisfies its invariant and fairness condition. However, it does not satisfy ρ_b because it contains the transition $w[i]w[i+1]$. Therefore it does not satisfy ϕ_b, which is a contradiction, since we supposed $\phi_a \subseteq \phi_b$. Hence, we must conclude that $\rho_a \subseteq \rho_b$. □

Example 6.5. *Let* $\phi_a = \mathbf{G}(a \rightarrow \mathbf{X}(b \wedge c)) \wedge \mathbf{GF}a$ *and* $\phi_b = \mathbf{G}(a \rightarrow \mathbf{X}b) \wedge \mathbf{GF}b$. *It is easy to observe that* ϕ_a *implies* ϕ_b, *and also that the strongly connected invariant of* ϕ_a *alone implies the invariant of* ϕ_b.

By combining this theorem with Corollary 6.1, we obtain the following result:

Corollary 6.2 (Equivalence of strongly connected invariants in GR(1) conjunctions)**.** *Let* ϕ_a *and* ϕ_b *be two GR(1) conjunctions with strongly connected invariants such that* $\phi_a \rightarrow \phi_b$ *is valid and* $\mathsf{dim}(\phi_a) = \mathsf{dim}(\phi_b)$. *Then their invariants are equivalent.*

From this corollary, it immediately follows that the ordering given by the weakness measure in Definition 6.3 satisfies Axiom 6.1 even in this class of GR(1) conjunctions: if ϕ_a and ϕ_b have different entropy, then the ordering induced by it is consistent with implication by Proposition 6.1.

If the formulae have the same entropy, then their invariants are equivalent by Corollary 6.2; we can therefore repeat the reasoning of Theorem 6.5 and illustrated in Figure 6.8.

6.2.6 Ordering Generic GR(1) Conjunctions

In this section, we consider generic GR(1) conjunctions where the invariant may be non-strongly connected. For simplicity of presentation we neglect initial conditions and consider only formulae with one fairness condition; however, our results can easily be extended to formulae with an initial condition and more than one fairness conditions. We start by presenting a simple example which shows that the weakness measure of Definition 6.3 is not consistent with implication (Axiom 6.1) in this general case. The example suggests that a more careful negation of the fairness condition, which we call *fairness complement*, may be helpful in extending our previous weakness measure. We then derive conditions guaranteeing that the new weakness measure is consistent with implication.

Example 6.6. *Consider two GR(1) conjunctions $\phi_1 = \rho_1 \wedge J_1$ and $\phi_2 = \rho_2 \wedge J_2$ over the variables $\mathcal{V} = \{a, b, c\}$, where*

$$\begin{aligned}
\rho_1 &= \mathbf{G}(((a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)) \wedge (a \rightarrow \mathbf{X}(b \vee c)) \\
&\quad \wedge (b \rightarrow \mathbf{X}(a \vee b)) \wedge (c \rightarrow \mathbf{X}c)); \\
\rho_2 &= \mathbf{G}(((\neg b \wedge \neg c) \vee (\neg a \wedge \neg c) \vee (\neg a \wedge \neg b)) \\
&\quad \wedge (a \rightarrow \mathbf{X}(b \vee c)) \wedge (b \rightarrow \mathbf{X}(a \vee b)) \wedge ((\neg a \wedge \neg b) \rightarrow \mathbf{X}(\neg a \wedge \neg b))); \\
J_1 &= \mathbf{GF}a; \\
J_2 &= \mathbf{GF}(a \vee b)\ .
\end{aligned}$$

The DBAs of $L(\phi_1)$ and $L(\phi_2)$ are shown in Figure 6.10. The state s_2 in both automata is redundant, since no run reaching s_2 could be accepted, but it has been depicted since it is part of the automata of the invariants alone.

It is easy to see (from the figure) that the entropy of the two ω-languages is equal and ϕ_1 implies

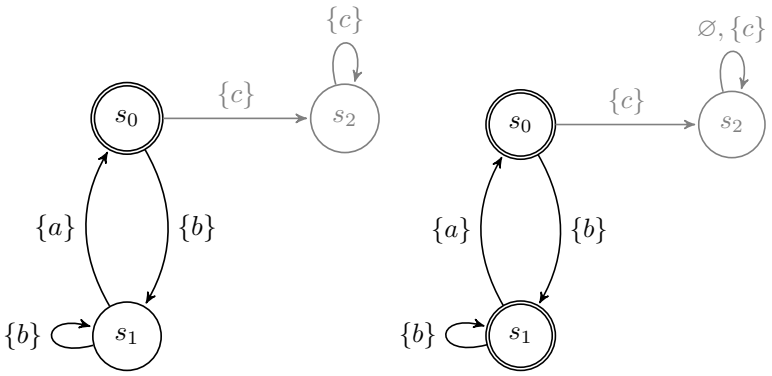


Figure 6.10: DBAs of $L(\phi_1)$ (left) and $L(\phi_2)$ (right). The grayed-out states and edges are parts of the automata of $L(\rho_1)$ and $L(\rho_2)$, but not of $L(\phi_1)$ and $L(\phi_2)$.

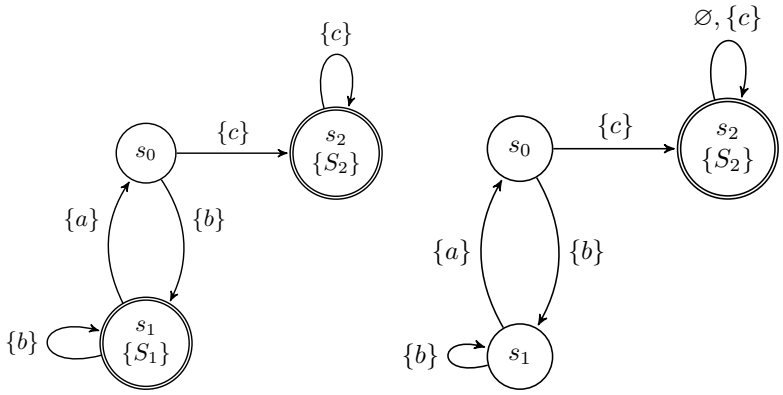


Figure 6.11: DMAs of $L(\rho_1) \setminus L(J_1)$ (left) and $L(\rho_2) \setminus L(J_2)$ (right).

ϕ_2. However, if we compute the weakness $d = (d_1, d_3)$ according to Definition 6.3, ϕ_1 has a higher weakness than ϕ_2. In fact, $L(\phi_1^c)$ is the set of words eventually keeping the automaton always in s_1 or always in s_2, and $L(\phi_2^c)$ is the set of words that eventually end up stuck in s_2 (recall that we defined $\phi^c = \rho \wedge \neg J$ for $\phi = \rho \wedge J$). This defines the DMA of $L(\phi_1^c)$ and $L(\phi_2^c)$ in Figure 6.11, and hence, by (2.5) *(and executing algorithm* HausdorffDMA*), we obtain*

$$d_3(\phi_1) = \mathsf{dim}\,(L(\phi_1^c)) = \mathsf{dim}\,(L(\rho_1) \setminus L(J_1)) = \mathsf{log}_8\, 1 = 0;$$
$$d_3(\phi_2) = \mathsf{dim}\,(L(\phi_2^c)) = \mathsf{dim}\,(L(\rho_2) \setminus L(J_2)) = \mathsf{log}_8\, 2 = 1/3\ .$$

Given the order relation induced by this measure, ϕ_1 incorrectly appears as the weaker formula, since it has smaller d_3 measure. Thus Axiom 6.1 is violated.

The problem in this example stems from using the language of the invariant above as the 'basis' for negating the fairness condition. Next we show that if the definition of the fairness

complement changes (i.e., it is defined differently from $L(\phi^c) = L(\rho) \setminus L(J)$) then the resulting quantity reflects weakness better.

Definition 6.4 (Fairness complement)**.** *Given a GR(1) conjunction $\phi = \rho \wedge J$, the* fairness complement $L^c = L_b \setminus L(J)$ *is the difference between some ω-language L_b, called the* base language*, and the language of J, where the base language L_b is a function of the formula ϕ satisfying the following property: if $\phi_1 \to \phi_2$ is valid and $H(L_b(\phi_1)) = H(L_b(\phi_2))$, then $L_b(\phi_1) = L_b(\phi_2)$.*

This definition allows us to prove consistency with implication for $\mathsf{dim}\,(L^c)$ in the same way as Theorem 6.5: when L_b is a suitable base language and a GR(1) conjunction ϕ_a implies a second one ϕ_b, the sets $L_b(\phi_a)$, $L_b(\phi_b)$, $L(J_a)$, $L(J_b)$, $L(\neg J_a)$, $L(\neg J_b)$ are in the same relation as pictured in Figure 6.8, where $L(\rho)$ is replaced by $L_b(\phi_a) = L_b(\phi_b)$.

Since the Hausdorff dimension of an ω-language is essentially determined by the behavior of its suffixes, we consider the language generated by these suffixes as a candidate of the base language in the above definition. To define the language of suffixes, consider a GR(1) conjunction $\phi = \rho \wedge J$ with $\mathcal{B} = \langle Q, \Sigma, q_0, \delta, F\rangle$. We call an SCC S *accepting* if it has nonempty intersection with F, (i.e., $S \cap F \neq \varnothing$). Given an ω-word $w \in L(\phi)$ whose induced run ends in the accepting SCC S, we call its *accepted index* any index i_S such that for every $i \geq i_S$, $\delta(q_0, w[1..i]) \in S$; we call $w^S = w[i_S, i_S + 1, \ldots]$ an *accepted suffix* of w, and we say that the SCC S *accepts* the suffix w^S. We denote by L_S the set of all suffixes accepted by S, and by $L_{suff} = \bigcup_{S \in \text{accepting SCCs of } \mathcal{B}} L_S$ the set of all suffixes accepted by some SCC of $\mathcal{B}$. L_S and L_{suff} are called the *ω-languages of accepted suffixes* by S and $\mathcal{B}$, respectively.

Example 6.7. *The automata of ϕ_1 and ϕ_2 in Example 6.6 contain only one accepting SCC each. The languages of accepted suffixes are shown in Figure 6.12.*

Note that, by the finiteness of a Büchi automaton, any ω-word accepted by the automaton has a suffix accepted by one of its SCCs. Also notice that the definition of L_S depends on the particular automaton representing the ω-language $L(\phi)$. However, some properties are invariant

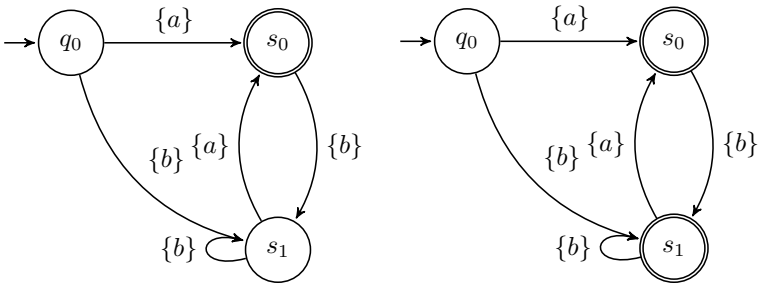


Figure 6.12: DBAs of the ω-languages of accepted suffixes L_{S_1} (left) and L_{S_2} (right).

to the particular automaton of $L(\phi)$ taken into account. In particular, the next theorem shows that the language of suffixes accepted by an SCC is monotonic under implication:

Theorem 6.7. *Let ϕ_a and ϕ_b be two GR(1) conjunctions with corresponding Büchi automata $\mathcal{B}_a$ and $\mathcal{B}_b$. Suppose $\phi_a \rightarrow \phi_b$ is valid. Then for every set of suffixes $W \subset \Sigma^\omega$ accepted by an SCC of $\mathcal{B}_a$ there exists an SCC of $\mathcal{B}_b$ accepting every element of W. In particular, it follows that for every accepting SCC S_a in $\mathcal{B}_a$ there exists an accepting SCC S_b in $\mathcal{B}_b$ such that $L_{S_a} \subseteq L_{S_b}$.*

The proof of the theorem uses the following simple lemma, which allows appending an SCC-accepted prefix to a word that reaches the same SCC:

Lemma 6.1. *Consider a Büchi automaton $\mathcal{B}$ with an SCC S. Let w_1 be an accepted by $\mathcal{B}$ such that the path induced by w_1 reaches S, that is, there is an index $j \in \mathbb{N}$ such that $s_1 = \delta(q_0, w_1[1, \ldots, j]) \in S$. Furthermore, let $w_2 \in \Sigma^\omega$ be a suffix accepted by S. Then there is a $v \in \Sigma^*$ such that $w = w_1[1, \ldots, j]vw_2$ is accepted by $\mathcal{B}$ and the path induced by w remains in S after processing the first j symbols; that is, $\delta(q_0, w[1, \ldots, k]) \in S$ for all $k \geq j$.*

Proof. Since w_2 is accepted by S, there is a prefix $v_2 \in \Sigma^*$ such that v_2w_2 is accepted by $\mathcal{B}$ and its run stays in S after v_2, that is, $s_2 = \delta(q_0, v_2) \in S$ and $\delta(q_0, v_2w_2[1, \ldots, i]) \in S$ for all $i \in \mathbb{N}$. Since S is strongly connected, there is a $v \in \Sigma^*$ that leads from s_1 to s_2 while not leaving S. Therefore, $w_1[1, \ldots, j]vw_2$ is accepted by $\mathcal{B}$ and belongs to S after the first j steps. □

Using this result, we can easily prove Theorem 6.7.

Proof. The proof is done by contradiction. Suppose there is an SCC S_a of $\mathcal{B}_a$ that accepts every suffix in W, but for every SCC S_b in $\mathcal{B}_b$ there is at least one suffix in W that is not accepted

by S_b. Let B denote the number of distinct SCCs of $\mathcal{B}_b$. The contradiction will be achieved by creating an ω-word which belongs to $L(\phi_b)$ but has to visit more than B SCCs of $\mathcal{B}_b$.

Let $w_0^* \in W$. By definition, there exists a finite word $v_0 \in \Sigma^*$ of length $|v_0|$ such that $w_0 = v_0 w_0^* \in L(\phi_a)$ and the automaton remains in the SCC S_a of $\mathcal{B}_a$ after processing the first $|v_0|$ symbols and reaching $s_0 = \delta(q_{0,a}, v_0) \in S_a$. Since $\phi_a \to \phi_b$, $w_0 \in L(\phi_b)$. Let $S_{b,1}$ denote an SCC reached in $\mathcal{B}_b$ while processing the w_0^*-part of this word; that is, there exists $j_1 > |v_0|$ such that $\delta(q_{0,b}, w_0[1, \ldots, j_1]) \in S_{b,1}$. By our assumption, there is a $w_1^* \in W$ that is not accepted by $S_{b,1}$. On the other hand, w_1^* is a suffix accepted by S_a, hence by Lemma 6.1, there is a finite word $v_1 \in \Sigma^*$ such that $w_1 = w_0[1, \ldots, j_1] v_1 w_1^* \in L(\phi_a)$ and its induced path on $\mathcal{B}_a$ belongs to S_a after processing the first $|v_0|$ symbols. Since $\phi_a \to \phi_b$, w_1 also belongs to $L(\phi_b)$. Furthermore, since $S_{b,1}$ does not accept w_1^*, the induced path of w_1 on $\mathcal{B}_b$ must reach an SCC $S_{b,2}$, after reaching $S_{b,1}$, which is distinct from $S_{b,1}$. That is, $\delta(q_{0,b}, w_1[1, \ldots, j_2]) \in S_{b,2}$, and since the first j_1 symbols of w_0 and w_1 coincide, $\delta(q_{0,b}, w_1[1, \ldots, j_1]) \in S_{b,1}$. Continuing with this construction, using $w_2^* \in W$ not accepted by $S_{b,2}$, we can create $w_2 = w_1[1, \ldots, j_2] v_2 w_2^*$ which belongs to $L(\phi_a)$, and hence to $L(\phi_b)$, reaches the SCC $S_{b,1}, S_{b,2}$ after $j_1 < j_2$ steps, respectively, and since $S_{b,2}$ does not accept w_2^*, it also reaches another SCC of $\mathcal{B}_b$, $S_{b,3}$, which is obviously distinct from $S_{b,2}$, but also distinct from $S_{b,1}$, since otherwise $S_{b,1}$ and $S_{b,3}$ would belong to the same SCC. Repeating this construction for B steps, we end up with a word w_B that reaches $B+1$ distinct SCCs of $\mathcal{B}_b$, which is a contradiction, as the automaton only has B SCCs. Therefore, $\mathcal{B}_b$ must have an SCC accepting all suffixes in W. The statement about the language inclusion then follows trivially. □

The base language L_b. Despite being an essential element in constructing the base language of the fairness complement, the ω-language of accepted suffixes is not sufficient. Indeed, Example 6.7 shows a case of two formulae such that one implies the other, they have the same entropy and Hausdorff dimension, but the languages of accepted suffixes are different: it is easy to see from Figure 6.12 that L_{S_1} only contains ω-words that contain the symbol $\{a\}$ infinitely many times, while L_{S_2} also contains sequences .

The problem is that, unlike ρ in Section 6.2.5, the language L_S of suffixes accepted by the SCC S (or L_{suff} in case of multiple accepting SCCs) is not closed. To alleviate this problem, we are going to use its closure $C_S = \mathcal{C}(L_S)$ as the base language. The effect of applying the closure operator to an ω-language accepted by a DBA is producing a DBA with the same graph structure where every co-reachable state (that is, a state from which an accepting state is reachable) becomes accepting itself. Therefore, the closure operation does not change the entropy. Moreover, it can be seen easily that $C_S \cap L(J) = L_S$.

In Example 6.7, it is clear that $C_{S_1} = C_{S_2} = L_{S_2}$, that is, the two closures are equal. The following theorem shows that this connection holds more generally, which is one of the main motivations for choosing $L_b = C_S$.

Theorem 6.8. *Let ϕ_a and ϕ_b be two GR(1) conjunctions. Suppose that $\phi_a \to \phi_b$ is valid and $\mathsf{dim}\,(L(\phi_a)) = \mathsf{dim}\,(L(\phi_b)) = H$. Then, for every accepting SCC S_a of $\mathcal{B}_a$ such that $H(L_{S_a}) = H$, there exists an SCC S_b of $\mathcal{B}_b$ such that $C_{S_a} = C_{S_b}$.*

Proof. By Theorem 6.7 for every SCC S_a in $\mathcal{B}_a$ there exists an SCC S_b in $\mathcal{B}_b$ such that $L_{S_a} \subseteq L_{S_b}$. This implies $\mathcal{C}(L_{S_a}) = C_{S_a} \subseteq \mathcal{C}(L_{S_b}) = C_{S_b}$.

Now assume that the entropy of the language L_{S_a} is equal to H, the Hausdorff dimension of the entire language. Then $H(L_{S_b}) \geq H$. Since $\mathsf{dim}\,(L(\phi_b)) = H$ by assumption and $H(L_{S_b}) \leq \mathsf{dim}\,(L(\phi_b))$, we have $H(L_{S_b}) = H$. Hence, also their respective closures have the same entropy: $H(C_{S_a}) = H(C_{S_b}) = H$.

Now, since C_{S_a} and C_{S_b} are closed and strongly connected, their entropy is strictly monotonic with their subset relation by Theorem 6.1. That is, $C_{S_a} \subseteq C_{S_b}$ and $H(C_{S_a}) = H(C_{S_b})$ imply $C_{S_a} = C_{S_b}$. □

The theorem allows us to define a base language as the union of the closure of the accepted suffix languages under some additional conditions, which will be explained later. To be precise, let $\mathcal{B}$ be a Büchi automaton for a GR(1) conjunction ϕ, and let $\mathcal{S}_H(\phi)$ denote the set of its SCCs whose language of accepted suffixes has entropy $\mathsf{dim}\,(L(\phi))$. Then the base language of

L can be defined as

$$L_b(\phi) = \bigcup_{S \in \mathcal{S}_H(\phi)} C_S \ . \tag{6.5}$$

Theorem 6.8 shows that if ϕ_a and ϕ_b satisfy the conditions of the theorem, $L_b(\phi_a) \subseteq L_b(\phi_b)$. To be able to use L_b as a base language, we need some extra conditions guaranteeing that $L_b(\phi_a) = L_b(\phi_b)$ also holds. Suppose that C_{S_a} and C_{S_b} are unique for $S_a \in \mathcal{S}_H(\phi_a)$ and $S_b \in \mathcal{S}_H(\phi_b)$, respectively (i.e., among the accepting SCCs of $\mathcal{B}_a$ and $\mathcal{B}_b$ whose entropy achieves the Hausdorff dimension, respectively); note that this assumption seems to be satisfied in practice. Assume furthermore that there is a one-to-one correspondence between S_a and S_b in the theorem. Then $L_b(\phi_a) = L_b(\phi_b)$, which is equivalent to

$$\bigcup_{S_a \in \mathcal{S}_H(\phi_a)} C_{S_a} = \bigcup_{S_b \in \mathcal{S}_H(\phi_b)} C_{S_b} \ ,$$

holds if the number of SCCs considered for ϕ_b is not greater than that for ϕ_a. The number of such SCCs is captured in the next definition.

Definition 6.5 (Order of strong connectedness)**.** *Given a formula ϕ and its Büchi automaton DBA $\mathcal{B}$, the* order of strong connectedness *(or simply* order*) of $\mathcal{B}$ is defined as $|\mathcal{S}_H(\phi)|$, that is, the number of accepting SCCs of $\mathcal{B}$ such that the entropy of (the closure of) their accepted suffix language equals* $\mathsf{dim}\,(L(\phi))$.

The next theorem shows that under the above assumptions and definition of L_b, the fairness complement can be used to compare two GR(1) conjunctions of the same order and entropy.

Theorem 6.9. *Let $\phi_a = \rho_a \wedge J_a$ and $\phi_b = \rho_b \wedge J_b$. Suppose $\phi_a \to \phi_b$ is valid, and* $\mathsf{dim}\,(\phi_a) = \mathsf{dim}\,(\phi_b) = H$. *Let $\mathcal{B}_a$ and $\mathcal{B}_b$ be two DBAs of the languages of the respective formulae having the same order, and assume all their accepting SCCs are unique in the sense that their closures are different. Let L_a^c and L_b^c denote, respectively, the fairness complements of ϕ_a and ϕ_b with the base language L_b defined in* (6.5). *Then* $\mathsf{dim}\,(L_a^c) \geq \mathsf{dim}\,(L_b^c)$.

Proof. Theorem 6.7 implies that $\bigcup_{S \in \mathcal{S}_H(\phi_a)} L_S \subseteq \bigcup_{S \in \mathcal{S}_H(\phi_b)} L_S$. Moreover, by the definition of the closure operation, $\bigcup_{S \in \mathcal{S}_H(\phi_a)} L_S = \bigcup_{S \in \mathcal{S}_H(\phi_a)} C_S \cap L(J_a)$ and a similar equation also holds

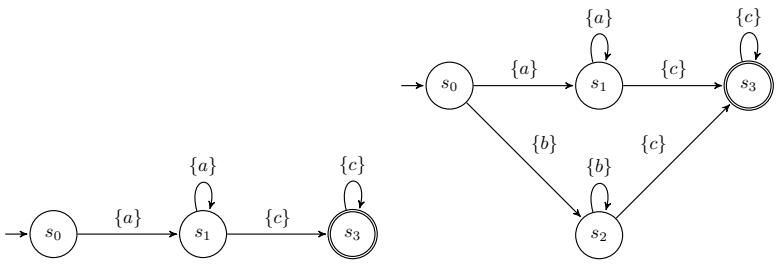


Figure 6.13: DBAs of ϕ_1 (left) and ϕ_2 (right)

for ϕ_b. Furthermore, the argument preceding the theorem and our assumption on the order of the automata and the uniqueness of the SCCs imply that $L_b(\phi_a) = L_b(\phi_b)$, which we denote by L_b. Combining the above we get $L_b \cap L(J_a) \subseteq L_b \cap L(J_b)$, implying $L_b \setminus L(J_a) \supseteq L_b \setminus L(J_b)$. The latter two sets are, respectively, L_a^c and L_b^c by definition. Hence, by the monotonicity of Hausdorff dimension, we conclude that $\mathsf{dim}\,(L_a^c) \geq \mathsf{dim}\,(L_b^c)$. □

Remark 6.3. *Notice that the closure of the language of ϕ, $\mathcal{C}(L(\phi))$ cannot be used as the base language (see Definition 6.4). For instance, consider two GR(1) conjunctions ϕ_1 and ϕ_2 defined by the DBAs in Figure 6.13. It is easy to see that $\phi_1 \to \phi_2$ is valid and ϕ_1 and ϕ_2 have the same entropy (since ϕ_2 has twice as many prefixes as ϕ_1, for any length). At the same time, since the DBA of the closure of ϕ_i can be obtained by making every state accepting, the two formulae have different closures. Hence, they cannot be used as a base language.*

Weakness measure in the general case. This allows us to provide the following general definition of the weakness measure and its order relation, which is only defined among formulae with the same order. The weakness measure defined as such satisfies our original goal of provable consistency with implication.

Definition 6.6 (Weakness of generic GR(1) conjunctions)**.** *Let $\phi = \rho \wedge J$ be a GR(1) formula. Its* weakness *$d(\phi)$ is defined as a tuple $d(\phi) := (d_1(\phi), d_2(\phi), d_3(\phi))$, where*

- $d_1 = H(L(\phi))$
- $d_2 = \mathsf{dim}\,(L(\phi))$
- $d_3 = \mathsf{dim}\,(L^c)$

where L^c is the the fairness complement of ϕ with L_b defined in (6.5).

Given two formulae ϕ_a and ϕ_b with the same order, we say that $d(\phi_a) < d(\phi_b)$ (ϕ_b is weaker *then ϕ_a) if and only if*

- $d_1(\phi_a) < d_1(\phi_b)$, *or*
- $d_1(\phi_a) = d_1(\phi_b)$ *and* $d_2(\phi_a) < d_2(\phi_b)$, *or*
- $d_1(\phi_a) = d_1(\phi_b)$, $d_2(\phi_a) = d_2(\phi_b)$, *and* $d_3(\phi_a) > d_3(\phi_b)$.

Notice that, if ρ is strongly connected, this definition reduces to the first version we have given in the previous subsection. Under the assumption that the closures of the SCCs of the DBA of ϕ_a and ϕ_b are unique, Proposition 6.1 and Theorem 6.9 imply that this weakness measure is consistent with implication (i.e., Axiom 6.1 is satisfied).

Remark 6.4. *The results provided so far assume that the formula contains only one fairness condition. If the formula $\phi = \rho \wedge \bigwedge_j J_j$ contains more than one fairness conditions, the same theorems presented in this section can be proved (as they depend on the structure of the automaton, not on the syntax of the formula). In this case, $L(\phi)$ is represented by a generalized deterministic Büchi automaton (GDBA, see Section 5.2). In a GDBA $\mathcal{G} = \langle Q, \Sigma, q_0, \delta, \{F_{\mathcal{G},1}, \ldots, F_{\mathcal{G},k}\}\rangle$, an SCC S is called* accepting *if it has a non-empty intersection with every accepting set, that is, $S \cap F_{\mathcal{G},j} \neq \varnothing$ for every j. Similarly, the definition of the* fairness complement *L^c can be extended to this case as $L^c = L_b \setminus \bigcap_{j=1}^{k} L(J_j)$, where L_b is the same base language as defined in (6.5).*

With these two small variations, Proposition 6.1 and Theorem 6.9 can be proved with the same statements (where DBAs are replaced with GDBAs) and following the same steps. This leads to proving that the weakness measure given in Definition 6.6 is consistent with implication also when ϕ has multiple fairness conditions.

Notice that in this general case, d_3 is defined as

$$d_3(\phi) := \mathsf{dim}\left(\bigcup_i C_{S_{max,i}} \setminus L\left(\bigwedge_j J_j\right)\right) = \mathsf{dim}\left(\bigcup_i C_{S_{max,i}} \cap \bigcup_j L(\neg J_j)\right),$$

and the relation $d(\phi_a) < d(\phi_b)$ *is defined in the same way as in Definition 6.6. By applying the property of* countable stability *of Hausdorff dimension [Sta15], which states that for any countable set of* ω*-languages* $L_1, L_2, \ldots \subseteq \Sigma^\omega$*,*

$$\mathsf{dim}\left(\bigcup_i L_i\right) = \sup_i \mathsf{dim}\left(L_i\right),$$

d_3 *can be expressed as*

$$d_3(\phi) = \max_{i,j} \mathsf{dim}\left(C_{S_{max,i}} \cap L(\neg J_j)\right),$$

thereby reducing the definition to the base case of a formula with one strongly connected component and one fairness condition only.

6.3 Computing the Hausdorff Dimension of Fairness Complements

In this section, we describe how to compute the component $d_3(\phi) = \mathsf{dim}\left(L^c\right)$ of the weakness measure given in Definition 6.6. We show how to compute the accepting states of the DMA of L^c from the DBA $\mathcal{B}$ of $L(\phi)$. From this, the Hausdorff dimension is computed via Algorithm 2.

We focus on the simple case $\phi = \rho \wedge J$ having a single fairness condition, and such that $\mathcal{B}$ has a single accepting SCC S with maximum entropy (or equivalently, $L(\phi)$ has the order of strong connectedness 1). We saw in Section 6.2.6 that the case of multiple fairness conditions is reduced to the case of a single one. Moreover, when multiple accepting SCCs are present, the procedure we give simply iterates over these SCCs and returns the maximum over the iterations.

As we saw in Section 6.2.6, in order to compute d_3 we need to compute a Muller automaton $\mathcal{M} = \langle Q_\mathcal{M}, \Sigma, q_{\mathcal{M},0}, \delta_\mathcal{M}, T_\mathcal{M} \rangle$ of the fairness complement $L^c = C_S \setminus L(\mathbf{GF}\beta^{fair})$ (see Definition 6.4). Actually, it is sufficient to compute the subgraph of $\mathcal{M}$ induced by its accepting sets of states $S' \in T_\mathcal{M}$, since they are the only components that affect the Hausdorff dimension (see

Algorithm 2).

We, therefore, provide an algorithm to compute the accepting set(s) of the Muller automaton from the Büchi automaton of $L(\phi)$. The procedure is shown in Algorithm 11. The algorithm takes as input the subgraph of $\mathcal{B}$ induced by its accepting SCC S, denoted by $\mathcal{B}[S]$. Lines 1-5 compute the adjacency matrix A_T of the accepting sets $T_\mathcal{M}$ of the DMA $\mathcal{M}$. Line 3 computes the element $[i, j]$ of the adjacency matrix as the number of symbols connecting q_i to q_j in $\mathcal{B}[S]$ that violate β^{fair}.

Algorithm 11: ComputeD3OnSCC

Input: $\mathcal{B}[S] = \langle S \cup \{s_0\}, \Sigma, s_0, \delta^S, S\rangle$: DBA of C_S with adjacency matrix A
Input: $J_j = \mathbf{GF}\beta_j^{fair}(\mathcal{V})$: fairness condition
Output: $d_{3,S,j}$: Hausdorff dimension of $C_S \cap L(\neg J_j)$
1 **for** $i = 1$ *to* $|S|$ **do**
2 **for** $j = 1$ *to* $|S|$ **do**
3 $A_T[i, j] \leftarrow A[i, j] - \left|\left\{\sigma \mid \delta^S(s_i, \sigma) = s_j \text{ and } \sigma \models \beta^{fair}\right\}\right|$;
4 **end**
5 **end**
6 $T_\mathcal{M}^{\max} \leftarrow$ StronglyConnectedComponents(A_T);
7 **foreach** $C_j \in T_\mathcal{M}^{max}$ **do**
8 $\lambda_j \leftarrow$ maximum eigenvalue of $A_T[C_j]$;
9 **end**
10 $\lambda \leftarrow \mathsf{max}_j\, \lambda_j$;
11 $d_{3,S,j} \leftarrow \mathsf{log}_r\, \lambda$;
return $d_{3,S,j}$

Line 6 computes the SCCs of the graph described by A_T. These correspond to the maximal accepting sets of $\mathcal{M}$ (for a definition of maximal accepting sets, see Section 5.2). Hence, lines 7-11 run Algorithm 2 on these accepting sets to compute the desired Hausdorff dimension d_3.

The correctness of this algorithm is based on the following theorem.

Theorem 6.10 (Correctness of Algorithm 11)**.** *Let $\mathcal{B}[S]$ be the subgraph induced on $\mathcal{B}$ by its accepting SCC S. Then there exists a DMA $\mathcal{M}$ of $L^c = C_S \setminus L(\mathbf{GF}\beta^{fair})$, having accepting table $T_\mathcal{M}$, such that the matrix A_T computed by Algorithm 11 is the adjacency matrix of the subgraph induced by the states in $T_\mathcal{M}$.*

Proof. We prove this statement by constructing the DMA $\mathcal{M}$ and showing that its accepting

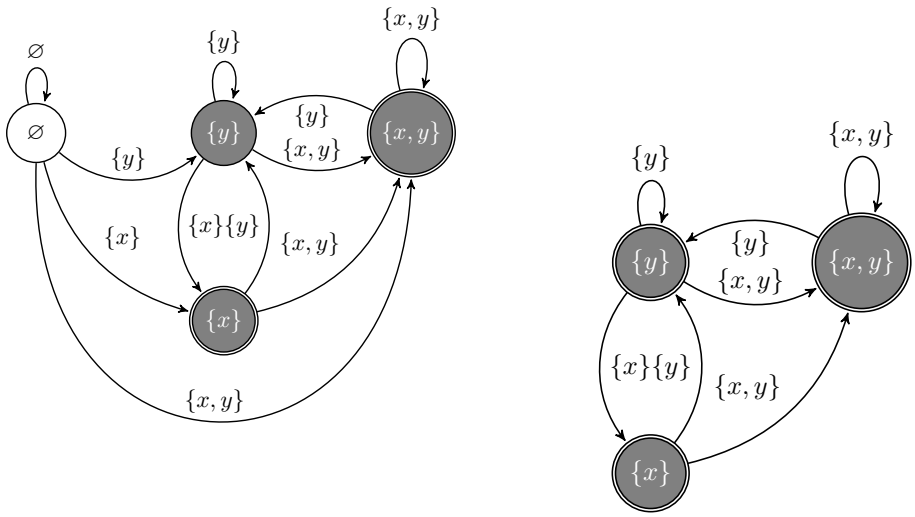


Figure 6.14: DBA $\mathcal{B}$ of the language of $\phi = \mathbf{G}((x \rightarrow \mathbf{X}y) \wedge ((x \vee y) \rightarrow \mathbf{X}(x \vee y)) \wedge \mathbf{GF}x$ (left), and DBA $\mathcal{B}[S]$ of the language C_S (right)

sets of states induce a subgraph described by A_T.

The construction starts with the subgraph induced on the DBA $\mathcal{B}$ of ϕ by the accepting SCC S. Formally, the Büchi automaton of ϕ is defined as $\mathcal{B} = \langle Q, \Sigma, q_0, \delta, F \rangle$. The restriction to S yields the DBA $\mathcal{B}[S] = \langle S \cup \{s_0\}, \Sigma, s_0, \delta^S, S \rangle$, which is, by construction, the automaton of C_S. The initial state s_0 is added artificially to the graph, and does not belong to the SCC S. The function δ^S is the restriction of δ to S.[2] Here, $\delta^S(s, \sigma)$ is the restriction of δ to $S \times \Sigma$.

Let $S_{\mathcal{M}} = S \times \{0, 1\}$. The states of the kind $(s, 0)$ are the ones where the last symbol read does not satisfy the fairness boolean expression β^{fair}, while those of the kind $(s, 1)$ are the ones where the last symbol read satisfies it. Accordingly, the transition function is defined as

$$\delta_{\mathcal{M}}((s, i), \sigma) = \begin{cases} (s', 0) & \text{if } \delta^S(s, \sigma) = s' \text{ and } \sigma \not\models \neg\beta^{fair} \\ (s', 1) & \text{if } \delta^S(s, \sigma) = s' \text{ and } \sigma \models \neg\beta^{fair} \end{cases} \quad \forall s, s' \in S, \forall i \in \{0, 1\} \ .$$

The initial state is $s_{M,0} = (s_0, 1)$. The accepting condition corresponds to any strongly con-

[2]For the sake of precision, the function $\delta^S(s, \sigma)$ is the same as $\delta(s, \sigma)$ for every $s \in S$ and $\sigma \in \Sigma$. For $s = s_0$, $\delta^S(s_0, \sigma)$ is equal to some state $s \in S$ that has an incoming edge labelled by σ from some other state.

In order to ensure that the resulting $\mathcal{B}[S]$ is still deterministic, we need to assume that no two states exist in S having an incoming edge labelled by the same symbol σ. This does not hinder the generality of the results. Indeed, remember that a GR(1) conjunction ϕ cannot have two or more $\mathbf{X}$ operators nested with each other. Therefore, given a DBA $\mathcal{B}$ of ϕ, the accepted suffixes from any state s of $\mathcal{B}$ only depend on the last symbol read before entering s. If two states s and s' have an incoming edge labelled with the same σ, they can be merged into one state and the resulting DBA still accepts $L(\phi)$.

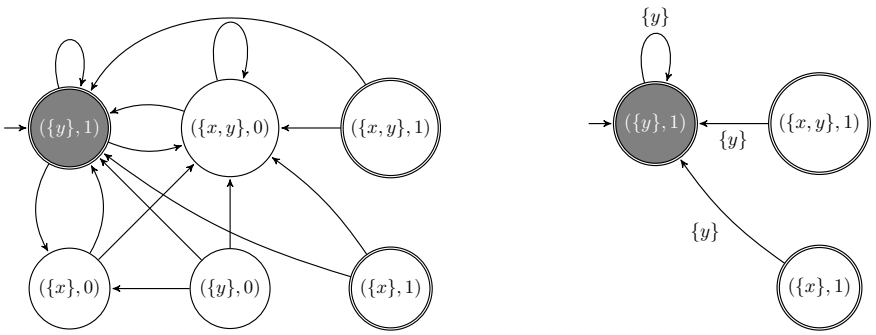


Figure 6.15: DMA of the language $C_S \cap L(\mathbf{FG}\neg x)$ referred to the language in Figure 6.14 (left), and subgraph including the accepting states only (right). In the left-hand automaton, the edge labels are omitted to avoid cluttering: each edge's label corresponds to the valuation labelling the destination state of the edge. The singleton of the state $(\{y\},1)$ in gray is the only one accepting any ω-word.

nected subset of states $(s,1)$, so that any path remaining inside these states is guaranteed to satisfy $\mathbf{G}\neg\beta^{fair}$. Any ω-word ending in these states and eventually remanining there satisfies $\mathbf{FG}\neg\beta^{fair}$, and since the graph obtained is a subgraph of C_S, every such path of this graph is a suffix of an ω-word in C_S. Hence the graph induced by the states $(s,1)$ comprises all the accepted sets of the DMA $\mathcal{M}$ of $C_S \cap L(\mathbf{FG}\neg\beta^{fair})$. By construction, it is easy to see that the adjacency matrix A_T of this graph is derived from $\mathcal{B}[S]$ by subtracting from each element $[i,j]$ the number of edges from state i to j satisfying β^{fair}. Figure 6.15 (left) shows an example of this construction for the language in Figure 6.14.

The Muller automaton $\mathcal{M}$ needs not be computed explicitly, since only the accepting sets contribute to the Hausdorff dimension. It is sufficient to consider the subgraph of $\mathcal{B}[S]$ and remove all the transitions labeled with valuations that do not satisfy $\neg\beta^{fair}$. The maximal accepting subsets of $\mathcal{M}$ (see Section 5.2) coincide with the SCCs of the resulting graph. □

Figures 6.14-6.15 show an example of this construction. The input to the algorithm is the graph of $\mathcal{B}[S]$ shown in Figure 6.14 (right), and the graph described by the adjacency matrix A_T is shown in Figure 6.15 (right).

6.4 Evaluation

We evaluate here the weakness measure in Definition 6.6 by demonstrating its applicability for GR(1) assumption refinement. To demonstrate its generality, we show an additional application to the problem of quantitative model checking.

To this aim, we implemented the weakness measure computation for GR(1) specifications in Python 2.7. Our implementation makes use of the Spot tool [DLLF+16] for LTL-to-automata conversion, and NumPy for eigenvalue computations.

We investigate the use of the weakness measure as a sorting heuristic in the counterstrategy-guided loop of Chapter 3. As described in Section 3.2, the outcome of these approaches is a *refinement tree*, i.e., a tree structure in which each node is associated with a GR(1) conjunction representing environment assumptions. Let ϕ denote a formula associated with a node, the node's children are of the form $\phi \wedge \psi$, where ψ is a single initial condition, invariant, or fairness condition. Since the goal of these approaches is identifying weakest formulae that describe an environment, our weakness measure can be used to provide a preference ranking of the tree nodes.

We conducted experiments on the case studies from Section 4.8, that is two benchmarks for GR(1) assumption refinement, namely the specifications of a lift controller and of the AMBA-AHB protocol for device communications in its versions for two, four and eight master devices [AMT13, BJP+12, LDS11]. The lift controller example specifies a controller for a lift with three floors: the Boolean variable b_i denotes the state of the button on floor i; the Boolean variable f_i is true iff the lift is at floor i. For more details on the initial assumptions $\mathcal{E}$ see Section 4.6. The AMBA-AHB protocol provides signals for requesting access to a bus ($\mathsf{hbusreq_i}$), for granting access ($\mathsf{hgrant_i}$), for signalling the termination of a communication (hready), and for identifying the current owner of the bus ($\mathsf{hmaster}$). Other signals are detailed in [BJP+12]. To our knowledge, the AMBA08 specification is one of the biggest benchmarks available in this field, with 55 binary variables, 28 initial assumptions and 157 guarantees.

We show the use of the weakness measure in two ways. First we compare the use of weakness

with the use of implication when ordering refinements. Specifically, we provide examples of pairs of refinements that can be ordered by both approaches and refinements that can only be ordered by using our weakness measure.

Secondly, we use the weakness measure as a sorting heuristic to drive the search for refinements towards weaker formulae compared with the common breadth-first approach to this search problem used in [CA17, AMT13, MRS19, LDS11]. We show that, despite the fact that this heuristic-based approach produces fewer solutions in the given time, it can reach weaker solutions that are missed by a basic breadth-first search of the refinement tree.

We also show that the computational bottleneck in ordering refinements is the automata computation (i.e., the generation of automata from GR(1) conjunctions [Var96]). Furthermore, implication requires more computation time than weakness to provide an ordering of the entire set of refinements generated during the search process. This part of the evaluation follows a similar methodology to the evaluation presented in [CAG18]. The main difference is that in [CAG18] we used refinements which were generated in [CA17] using an approach that applies breadth-first search and was automated only in part. This in effect slowed down the tree generation process and resulted in smaller trees. Here, instead, we use the larger refinement trees generated by our heuristic search. We conclude the evaluation process by using the comparison data between implication and weakness to validate the correctness of our implementation.

6.4.1 Ordering GR(1) Conjunctions via Weakness and Implication

Consistency between weakness and implication. We first consider the lift controller example. Two refinements computed by the automated approach in [CA17] are: $\phi_1 = \mathbf{G}((\neg b_1 \wedge \neg b_2 \wedge \neg b_3) \rightarrow \mathbf{X}(b_1 \vee b_2 \vee b_3))$; and $\phi_2 = \mathbf{GF}(b_1 \vee b_2 \vee b_3)$. The first forces one of the buttons to be pressed at least every second step in a behaviour. The second forces one of the buttons to be pressed infinitely often in a behaviour. It is clear that ϕ_1 implies ϕ_2. We compare the assumptions obtained by refining the original assumptions with the first one and with the second one: $d(\mathcal{E} \wedge \phi_1) = (0.7746, 0.7746, 0)$ and $d(\mathcal{E} \wedge \phi_2) = (0.7925, 0.7925, 0.5)$. Notice that $d_1(\mathcal{E} \wedge \phi_1) < d_1(\mathcal{E} \wedge \phi_2)$ and this is consistent with the fact that ϕ_1 is stronger than ϕ_2.

Consider now the two fairness refinements: $\phi_2 = \mathbf{GF}(b_1 \vee b_2 \vee b_3)$; and $\phi_3 = \mathbf{GF}b_1$. We have $d(\mathcal{E} \wedge \phi_2) = (0.7925, 0.7925, 0.5)$ and $d(\mathcal{E} \wedge \phi_3) = (0.7925, 0.7925, 0.695)$. Here, d_1 and d_2 are equal for both formulae and $d_3(\mathcal{E} \wedge \phi_2) < d_3(\mathcal{E} \wedge \phi_3)$; this is consistent with the fact that ϕ_2 is weaker than ϕ_3.

Formulae incomparable via implication. Consider ϕ_3 above and $\phi_4 = \mathbf{GF}(b_2 \vee b_3)$. Neither implies the other. However, it is reasonable to argue that ϕ_4 is less restrictive than ϕ_3: while ϕ_3 constrains exactly one button to be pressed infinitely often, ϕ_4 allows the extra choice of which one (out of two) . This intuition is indeed reflected by our computed weakness metric: $d(\mathcal{E} \wedge \phi_3) = (0.7925, 0.7925, 0.695)$ and $d(\mathcal{E} \wedge \phi_4) = (0.7925, 0.7925, 0.5975)$. This confirms that ϕ_4 removes less behaviours from $\mathcal{E}$ than ϕ_3.

Our weakness measure can help in spotting asymmetries between assumptions that are syntactically equal but constrain semantically different variables. Consider an extended version of the lift controller example including the input variable alarm and the output variable stop: whenever alarm is set to high, the lift enters a stop state where it does not move from the floor it is at. Computing the weakness of the two refinements $\phi_5 = \mathbf{G}\neg b_1$ and $\phi_6 = \mathbf{G}\neg\mathsf{alarm}$ yields $d(\mathcal{E} \wedge \mathcal{S} \wedge \phi_5) = (0.3694, 0.3694, 0.3207)$ and $d(\mathcal{E} \wedge \mathcal{S} \wedge \phi_6) = (0.3746, 0.3746, 0.3346)$. This is consistent with the intuition that the former assumption excludes a part of the desirable system behaviors (all the ones that allow it to reach floor 1), while the latter excludes only the error traces ending in the *stop* state, being then a weaker restriction on the combined behaviors of the controller and the environment.

The following two assumption refinements are computed for the AMBA-AHB case study, instead, with two masters: $\psi_1 = \mathbf{G}(\neg\mathsf{hbusreq}_1 \vee \mathbf{X}(\mathsf{hready} \vee \neg\mathsf{hbusreq}_1))$; and $\psi_2 = \mathbf{G}((\neg\mathsf{hgrant}_1 \wedge \mathsf{hready} \wedge \mathsf{hbusreq}_1) \rightarrow \mathbf{X}(\neg\mathsf{hready} \vee \neg\mathsf{hbusreq}_1))$. As in the case of the lift example, neither formula implies the other. The weakness of the resulting assumptions is: $d(\psi^{\mathcal{E}} \wedge \psi_1) = (0.9503, 0.9503, 0.9068)$ and $d(\psi^{\mathcal{E}} \wedge \psi_2) = (0.9607, 0.9607, 0.9172)$. The refinement ψ_2 is weaker than ψ_1. Such insight into their weakness could be used to guide the refinement approach (e.g., [AMT13, CA17]) in choosing to only refine those assumptions that may lead to weaker

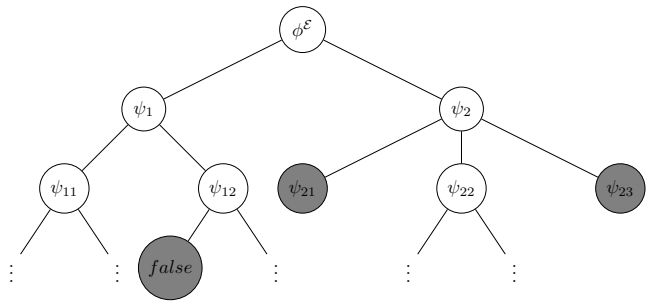


Figure 6.16: Refinement tree. The gray nodes are the leaves: ψ_{21} and ψ_{23} are solutions to the refinement problem.

specifications, for instance further refining ψ_2 rather than ψ_1.

Consistency between equally permissive formulae. Consider the AMBA-AHB protocol with eight masters and the two alternative refinements: $\theta_1 = \mathbf{GF}(\mathsf{hmaster}_0 \vee \neg \mathsf{hbusreq}_1)$; and $\theta_2 = \mathbf{GF}(\mathsf{hmaster}_1 \vee \neg \mathsf{hbusreq}_2)$. Clearly the two alternatives express the same kind of constraint on different masters. Since the two masters do not have priorities over each other, the two refinements have the same weakness as expected: $d(\theta^{\mathcal{E}} \wedge \theta_1) = d(\theta^{\mathcal{E}} \wedge \theta_2) = (0.9396, 0.9396, 0.9214)$.

6.4.2 Sorting Heuristic for GR(1) Assumptions Refinement

To the best of our knowledge, assumption refinement is approached in most cases as a breadth-first search over a refinement tree [AMT13, LDS11, CA17, MRS19]. Given an initial, unrealizable specification $\phi = \mathcal{E} \rightarrow \mathcal{S}$, a counterstrategy $\mathcal{C}$ is computed (see Algorithm 3) and a set of alternative assumptions $\{\psi_1, \ldots, \psi_k\}$ inconsistent with $\mathcal{C}$ is generated. For each ψ_i in turn, the new specification $\mathcal{E} \wedge \psi_i \rightarrow \mathcal{S}$ is checked for realizability and in case it is not realizable, a further set of additional assumptions $\{\psi_{i1}, \ldots \psi_{ik'}\}$ is produced to be added to $\mathcal{E} \wedge \psi_i$. The expansion of one branch of the tree stops either when a solution is reached, or when a refinement made of inconsistent assumptions is generated. An example of a refinement tree is shown in Figure 6.16.

We modify this approach and use our weakness measure to sort the nodes that are expanded by decreasing weakness; therefore, if $\mathcal{E} \wedge \psi_2 \wedge \psi_{22}$ is weaker than $\mathcal{E} \wedge \psi_1$, the former is expanded before the latter. However, our weakness measure leads the search algorithm to expand refinements

consisting only of fairness conditions earlier than any other formula. This happens because fairness conditions typically affect only d_3, which has the lowest priority in the order relation. Therefore, a heuristic search only based on the weakness measure may not converge to any solution in reasonable time, if there exists no solution only formed by fairness conditions. To circumvent this problem, we impose a limit of 8 fairness conditions accepted in any refinement; our approach does not expand further a node of the tree containing more than 8 fairness conditions. This choice is driven by the preliminary observation of the solutions of BFS reported in [CA17]. This approach found one solution with 7 fairness conditions: our choice is such that this solution is not excluded *a priori* from the search.

We run the heuristic search and compare it with the FIFO sorting heuristic in the setting of Section 3.5. The two approaches are run for approximately 4 hours (14,700 seconds) each on the six benchmark case studies of Table 4.5, the three AMBA-AHB cases and three case studies first presented in [KMR17]: *ColorSort*, a robot sorting Lego bricks by color, *GyroAspect*, the specification of a robot with self-balancing capabilities, and *Humanoid*, a robot with humanoid shape. The original AMBA-AHB specification [BJP+12] is realizable: in order to obtain an unrealizable specification, we remove the fairness condition **GF**hready, as done in [AMT13].

We saw in Figure 4.14 that the interpolation bias found more solutions than multivarbias in the three AMBA-AHB and ColorSort, while the latter was more effective in GyroAspect and Humanoid. In this experiment, for each case study we select the bias finding more solutions in it.

Figure 6.17 shows the solutions found by BFS and heuristic search sorted by decreasing weakness. We observe that in every case study except ColorSort the median ranking of BFS solutions is lower than the median ranking of heuristic search: this shows that typically the heuristic approach finds solutions that are weaker than more than half the solutions of BFS. Additionally, in all case studies except ColorSort, the heuristic search finds either a solution as weak as the weakest one found by BFS, or an even weaker solution which was missed by BFS (AMBA08). Notice also that while BFS generates several inconsistent solutions (rightmost point of the plots, labeled FALSE), the heuristic search never produces such a node. This observation corresponds

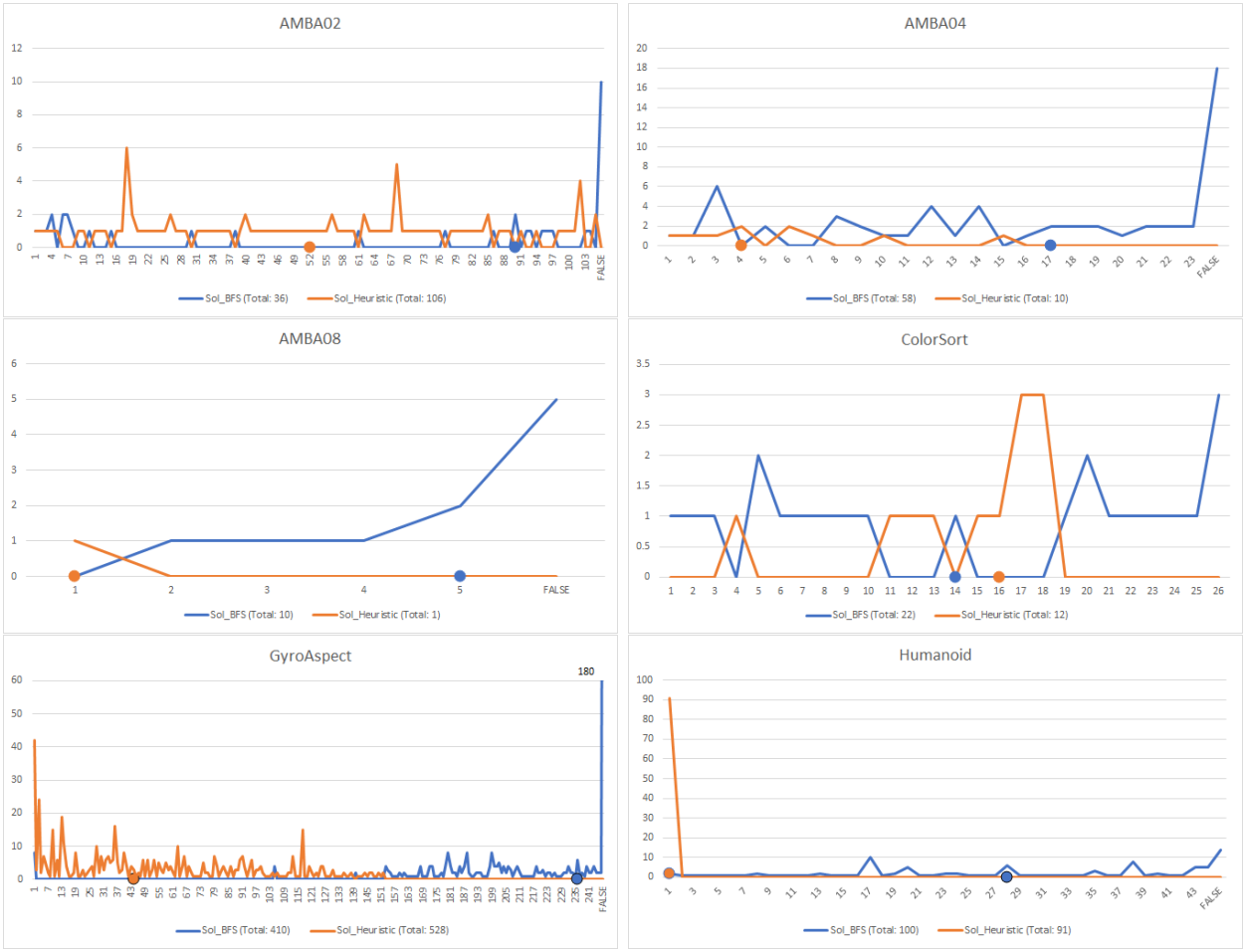


Figure 6.17: Results for BFS and heuristic search. The results of the two approaches are ranked together by decreasing weakness. The x-axis shows the positions of the solutions in the ranking, from the weakest (leftmost) to the strongest (rightmost), with FALSE being the strongest. The y-axis is the number of solutions in each position of the ranking. The filled circles on the x-axis represent the median rankings of BFS and heuristic search solutions.

with the intuition that if at every step the weakest available refinement is picked, the risk of overconstraining the assumptions and producing inconsistent GR(1) conjunctions as a solution is reduced.

The heuristic search works as a conservative search, that attempts to reach a solution by strengthening the current assumptions as little as possible. However, since this may delay reaching a sufficiently constraining set of assumptions to make the specification realizable, fewer solutions are expected to be found in total. However, the reduction in number is not dramatic, and in two case studies (AMBA02 and GyroAspect) the heuristic search even finds more solutions than BFS.

Table 6.2: Results of experiment 2 of heuristic search. The column **# Case studies** reports how many case studies are in each group. The column **Solved** reports how many of them are solved by the FIFO approach only, the heuristic search only, and both. Among the ones solved by both, the column **Weakest** shows in how many case studies the weakest solution is found by FIFO only, the heuristic search only, and both.

Case study group	# Case studies	Solved			Weakest		
		FIFO	Heuristic	Both	FIFO	Heuristic	Both
AMBAs	16	1	0	5	2	0	3
Original SYNTECH	14	1	0	8	0	0	8
Additional SYNTECH	78	23	7	19	3	5	11

The exception of ColorSort is due to the randomness involved in the node expansion procedure. As an intractable number of GR(1) assumptions may be inconsistent with a counterstrategy, randomized assumption generation procedures are employed to generate a subset of these assumptions [CA18]; in this randomization, the weakest refinements found by BFS are missed when running the heuristic search.

The second experiment consists in running the heuristic search on the case studies of Tables 4.1, 4.2 and 4.3 for 10 minutes, and compare the results to the ones in Section 4.8.1. Table 6.2 summarizes the key results. This experiment showcases different outcomes deriving from the use of weakness as a heuristic, including its limitations. Notice that in several case studies there is not much difference between applying a FIFO or a heuristic search approach. These case studies are solved by both approaches, and both find the weakest solutions. In several case studies, this is only due to the randomness of the interpolation bias (see Algorithm 8). This happens for instance in various versions of ColorSort in the Additional SYNTECH group: while in the FIFO search the interpolation bias generates a solution at the first step, in the run with the heuristic search the only refinement explored is the constant false, and therefore no solutions are found. This explains the high number of case studies solved by FIFO only in the Additional SYNTECH group (23 out of 78).

On the other hand, in several versions of GyroAspect in the Additional SYNTECH group, the heuristic search moves earlier towards refinements with six initial conditions, which are weaker than the ones containing invariants or fairness conditions. This allows the search to find several solutions before the 10-minute timeout. FIFO, instead, uses the whole time to search shorter

Table 6.3: Comparison between the discriminative power of implication and weakness

Case study	#Nodes (k)	#Pairs	%Impl/#Pairs	%Weak/#Pairs
AMBA02	318	50,403	35.80%	97.80%
AMBA04	58	1,653	36.78%	97.64%
AMBA08	40	780	32.95%	93.46%

Case study	Time impl	Time weakness
AMBA02	46,535 s	695 s
AMBA04	1,350.241 s	145.67 s
AMBA08	31,735.33 s	20,197.20 s

refinements without reaching any solution. This explains the cases where only weakness finds solutions (7 out of 78).

Performance. We aim to assess the discriminative power and time requirements of the weakness measure compared with implication. To this end we perform an experiment on the refinement trees generated by the heuristic search: each pair of formulae in the refinement trees is compared both via implication and via the weakness measure from Definition 6.6. An implication check for the pair of formulae ϕ_1 and ϕ_2 is performed by computing the nondeterministic transition-based generalized Büchi automata (TGBA) [Lut14] of the formulae $\phi_1 \wedge \neg\phi_2$ and $\phi_2 \wedge \neg\phi_1$, and checking whether any of them is empty [RDLKP13]. In our implementation, the TGBAs are computed via the same SPOT tool used to compute weakness.

Table 6.3 shows the results: the columns show the total number of nodes in the refinement tree (**#Nodes**), the corresponding number of pairs (**#Pairs**), the percentage of pairs that can be ordered strictly via implication (**%Impl/#Pairs**) and via weakness (**%Weak/#Pairs**), the total time to perform all the comparisons via implication (**Time impl**) and via weakness (**Time weakness**).

To assess the discriminative power of the two approaches, we compare the proportion of formulae pairs that have different weakness measure (and thereby can be discriminated via our proposed metric) and the proportion of formulae pairs that can be discriminated via logical implication (either because one strictly implies the other or because the two formulae are logically equivalent). The table shows that, despite weakness does not capture implication strictly in all cases, it still allows for the discrimination of a larger set of assumptions, by virtue of Axiom 6.3.

Regarding the computation time, the following general consideration holds. With implication it is necessary to produce two automata, one for $L(\phi_1) \cap L(\neg\phi_2)$ and one for $L(\phi_2) \cap L(\neg\phi_1)$, and then run an emptiness check on each of them. When comparing k formulae, this operation must be repeated for $O(k^2)$ pairs of formulae. On the other hand, for a set of formulae containing at most m fairness conditions and at most $|\mathcal{S}_H|$ accepting SCCs with maximum entropy (where $|\mathcal{S}_H|$ is the order of strong connectedness, see Definition 6.5), our weakness measure requires $1 + m * |\mathcal{S}_H|$ DBA/DMA computations, yielding $O(|\mathcal{S}_H|mk)$ automata needed for comparing k formulae. In general, in a refinement tree k is larger than m and $|\mathcal{S}_H|$: in the AMBA case studies, the values of k reported in the table are all significantly higher than m (which has been fixed to 8 by design) and the order (which is observed to be 1 in all the refinements explored). In this respect, the advantage of our weakness measure resides in the reduced number of DBA/DMA computations with respect to implication.

The price to pay lies in the complexity of the needed automata: while weakness requires deterministic automata, implication can be checked via nondeterministic ones, which are typically faster to compute [EKS16]. The effect of the complexity can be observed in particular in AMBA08. Since this case study contains several variables, the DBAs of its refinements have a large state space, and require much more time to be computed than the TGBAs used for implication (up to 40 times longer). Thereby, even if the total number of automata computed for the weakness measure is significantly lower than the number of automata for implication, the gain in total time achieved by using the weakness measure is not as significant as in AMBA02 and AMBA04.

Correctness of implementation. We use the data from the comparison between implication and weakness to assess the correctness of our implementation. We simply observe that in no case the weakness measure and the implication checks induce opposite orderings on the pairs of formulae. Despite it being possible that both implementations are defective, the of this being the case is lower than just the weakness computation being defective.

6.4.3 Hausdorff Dimension for Quantitative Model Checking

In this section we provide an application of computing Hausdorff dimension on fairness complements in a quantitative model checking example from [ABD14a].

In the cited work, the goal is to identify an alternative measure to probability in order to quantify the infinite runs of an automaton $\mathcal{B}$ that violate a formula ϕ. The reason adduced by the authors is that under any typical stochastic model used in model checking (such as Markov chains) the probability of an infinite behaviour to satisfy interesting properties like invariants is either 0 or 1. The work proposes to use the entropy $H(L(\mathcal{B}) \cap L(\neg\phi))$ instead of probability measures, and shows that this measure captures the "likelihood" of reaching a violating state when dealing with safety properties (that correspond to invariants in our context). However it points out the limitation of this measure with fairness conditions, which cannot be discriminated by entropy alone (see Section 6.2.3). In particular, if the automaton is strongly connected, one has $H(L(\mathcal{B}) \cap L(\neg\phi)) = H(L(\mathcal{B}))$ if some runs of $\mathcal{B}$ do not satisfy ϕ. Hausdorff dimension, instead, can capture the difference between the two languages.

We apply it to the Dining Philosophers example with three philosophers from [ABD14a]. The model consists of three philosophers and three forks; the philosophers alternately think and eat, but before eating each of the philosophers must grab the fork to his left and the one to his right; if one of the two is taken by another philosopher, the philosopher needs to wait until the fork becomes available; finally, once the philosopher has eaten, he releases his forks for use by everyone.

A deadlock state is any state in which all three philosophers have taken their left fork and no progress can be made from there. Let ϕ^{DP} be the GR(1) formula describing all the philosophers' behaviors that do not reach the deadlock state. This formula uses the three model variables $\mathsf{state1}$, $\mathsf{state2}$ and $\mathsf{state3}$ describing the state of each philosopher, which is any of the values THINK, $\mathsf{LEFTFORK}$ and EAT. It can be expressed in GR(1) as $\phi^{DP} = \mathbf{G}(\neg(\mathsf{state1} = \mathsf{LEFTFORK} \wedge \mathsf{state2} = \mathsf{LEFTFORK} \wedge \mathsf{state3} = \mathsf{LEFTFORK}) \wedge \beta_{\mathsf{T}})$, where ϕ^T expresses the transitions allowed to each philosophers: $\beta_T = (\mathsf{state1} = \mathsf{THINK} \rightarrow \mathbf{X}(\mathsf{state1} = \mathsf{THINK} \vee \mathsf{state1} =$

$\mathsf{LEFTFORK}) \wedge \dots$).

The goal is to assign a measure to the subset of these behaviors such that none of the philosophers starves. This condition is expressed by the fairness formula $\phi_{fair}^{DP} = \mathbf{GF}(\mathsf{state1} = \mathsf{EAT}) \wedge \mathbf{GF}(\mathsf{state2} = \mathsf{EAT}) \wedge \mathbf{GF}(\mathsf{state3} = \mathsf{EAT})$. When using entropy only, $H(L(\phi^{DP})) = H(L(\phi^{DP} \wedge \phi_{fair}^{DP})) = 0.0718$. When using our weakness measure from Definition 6.3, the second component distinguishes the presence of the fairness condition. We use this simplified definition since the invariant ϕ^{DP} is strongly connected. The result is $d_3(\phi^{DP} \wedge \phi_{fair}^{DP}) = \mathsf{dim}\left(L(\phi^{DP}) \setminus L(\phi_{fair}^{DP})\right) = 0.0479 \neq d_3(\phi^{DP}) = 0$. Therefore, contrary to the work in [ABD14a], our measure is able to capture the difference in the permissiveness of the two formulae.

6.5 Related Work

Quantitative Measures for LTL Formulae. The closest notion to our weakness measure is the *entropy* of ω-languages applied by Asarin et al. [ABD14a, ABD$^+$14b] in the context quantitative model checking. This quantity measures how diverse the ω-words contained in the language of an LTL formula are. However, it is not sufficient to distinguish between weaker and stronger fairness conditions [ABD14a]. Our measure based on Hausdorff dimension is capable of making this distinction.

The quality of LTL formulae has also been defined in the context of model verification. Henzinger et al. [Hen10, HO13] define a similarity measure between models of LTL formulae so as to render the model checking output quantitative: instead of returning a true/false response, quantitative model checking computes the distance (*stability radius*) of the model from the boundary of the satisfiability region of an LTL property. The scope of our work is different: the measure we propose can be interpreted as the *extension* of this satisfiability region, rather than its distance from a specific model.

The more recent work by Degiovanni et al. [DCA$^+$18] uses model counting to measure how many behaviours of a system, described by an LTL formula, lead to a conflict between requirements. Since these behaviours are infinite, they propose to fix a boundary k on the time steps and

count all violating finite traces up to the boundary; this provides an estimate of the "likelihood" of reaching a violating state. Our approach takes this notion to the limit for k that grows to infinity, thus replacing the estimate with an exact value.

Quantitative Extensions of Temporal Logics. An alternative way to measure the size of behaviour sets is via probabilities. Probabilistic model checking [HJ94, Kwi07] enhances the syntax and semantics of temporal logics (usually CTL, *computation tree logic*) with probabilities. This allows for expressing properties like "the probability of satisfying a temporal logic formula ϕ by the modelled behaviours is at most p." Further extensions of LTL and/or automata with preference metrics other than probabilities have been proposed in [AAK13, BCHJ09, CDF+06, CHJ08]. The difference between using probabilities/preference metrics and our proposal is that these measures are parameters provided in input in addition to the logical formulae, or are applicable to stochastic settings where models of behaviour are not purely logical (for instance, Markov chains); these parameters, although conceptually related, may not reflect the true weakness of a logical formula. The measure we propose instead quantifies a concept of weakness *intrinsic* to the LTL formula itself.

Weakness and Automata. The problem of identifying weakest assumptions appears in the context of assume-guarantee reasoning [CGP03, LSWW10, NA06] for compositional model checking. In order to perform model checking of large systems, their models are generally broken down to components that can be checked independently for correctness: when checking each component, assume-guarantee reasoning supposes that all other components satisfy their respective specifications and uses this assumption to prove the correctness of the component undergoing the check. In this setting, one of the challenges is to identify the most general (weakest) assumptions over the environment in which each component operates, such that when they are satisfied, the correctness of the entire system is guaranteed. Assumptions in assume-guarantee reasoning are formalized as transition systems (e.g., modal transition systems) rather than declarative LTL specifications, which is where the focus of our work lies.

Quality of GR(1) Assumptions Weakness is just one of the quality criteria for GR(1) assumptions. As discussed in Section 3.3, there are different criteria that can be considered (*interpretability*, *usefulness* [MRS19]). See Section 3.3 for a more extensive discussion of these criteria. They should not be considered as alternatives to weakness, but rather complementary to it.

6.6 Summary

In this chapter, we have introduced a weakness measure for GR(1) conjunctions. This measure is used to provide an order relation consistent with the order given by logical implication, and to extend the order to formulae not related by implication. The definition is based on the entropy and Hausdorff dimension of the formulae's languages, as introduced in Chapter 5. The prospected application of this measure is to provide a sorting heuristic for counterstrategy-guided assumptions refinement.

The main focus of this chapter has been on making the weakness measure at least as discriminative as implication. That is, when implication provides a strict ordering of two formulae, we require the weakness measure not to be equal for the two formulae (Axiom 6.2). To this goal, we have provided increasingly complex definitions of the weakness measure for increasingly complex classes of GR(1) conjunctions.

We have started by defining the weakness measure as the entropy of the language. This measure satisfies Axiom 6.2 in the class of *strongly connected, closed* languages, that is, languages whose automaton is strongly connected and all states are accepting (the latter condition holds for GR(1) conjunctions only containing invariants). When the language is not closed (that is, the GR(1) conjunction can contain fairness conditions), two formulae may have the same entropy even if one strictly implies the other.

To solve this problem, we have adopted a best-effort approach that helps to break these ties in some cases. We have added a second component to the weakness measure, that is, the Hausdorff dimension of the *fairness complement* (Definition 6.3). The fairness complement has

different definitions based on the class of formulae where it is defined. In the case of formulae with strongly connected invariants, it is the difference of the language of the invariant and the language of the fairness conditions. In the general case of non-strongly connected invariants, it is defined as the difference language between the language of the *accepting SCCs* (that is, the SCCs containing at least one accepting state in the automaton) and the language of the fairness conditions. In the latter case, the weakness measure also needs a third component, that is, the Hausdorff dimension of the formula's language (Definition 6.6).

Hence, we have defined the weakness measure to be a triple: the first element is the entropy of the language of the formula, the second is the Hausdorff dimension, and the third is the Hausdorff dimension of the fairness complement. We have defined the order relation induced by the measure to be the lexicographic order of the triples: that is, first two formulae are compared based on their entropy, then, when the entropies are equal, on their Hausdorff dimension, and finally on the Hausdorff dimension of the fairness complements. We prove that this order relation is consistent with the order given by implication (that is, it satisfies Axiom 6.1). This is a relaxed requirement compared to the original goal of consistency with strict implication, which is why we have said that this definition makes a best effort to satisfy Axiom 6.2.

In the experimental section, we have applied the weakness measure as a sorting heuristic to the counterstrategy-guided loop. We have observed that, in this context, pairwise comparisons via the measure require shorter computation time than via implication, and the weakness measure gives a principled criterion to order more pairs than implication. Results show that most solutions to the refinement problem found with this sorting heuristic are indeed weaker than the solutions found with the FIFO heuristic of Section 3.4.3. Therefore, this tackles effectively the problem of weakest assumptions refinement introduced in Definition 3.4.

Chapter 7

Minimality of GR(1) Assumption Refinements

In Chapters 3 and 4, we introduced a set of bias heuristics that can be integrated in the counterstrategy-guided loop of Algorithm 3. All these bias heuristics have a point in common: they take in input a refinement Ψ and a counterstrategy c, and concatenate a new assumption ψ to Ψ such that ψ eliminates c. However, ψ may also eliminate some of the other counterstrategies already eliminated by the assumptions in Ψ, making them "redundant". In this chapter, we formalize this by the concept of minimality of a refinement with respect to a set of , and develop a bias that generates minimal refinements at every step of the search. This helps focusing the search effort onto the region of refinements with fewer assumptions (relative to the ones explored by bias heuristics that just concatenate assumptions), and produce weaker solutions to the assumptions refinement problem.

We motivate the need for minimal refinements on the request-grant example from Chapter 2 (Section 7.1). Then we formally define redundant assumptions and minimal refinements (Section 7.2) and present the algorithm to compute minimal refinements (Section 7.3). Searching minimal refinements only requires more time to find solutions with a large number of assumptions, and in some cases none of these can be found within the time budget. We circumvent this problem by presenting a hybrid search where both minimal and non-minimal refinements

are searched (Section 7.4). Finally, we make an experimental comparison between the standard BFS search strategy, search minimal refinements, and the hybrid search (Section 7.4).

We point out that minimality is orthogonal to the algorithm used to generate new assumptions from counterstrategies, and refinement minimization can be applied in conjunction with any bias analyzed so far (e.g. interpolation or multivarbias).

The work in this chapter is based on our publication in [CAG20b].

7.1 Motivating Example

Existing approaches to assumptions refinement [AMT13, LDS11, CA17] construct candidate solutions by incrementally concatenating assumptions extracted from observed counterstrategies. This, however, can lead to overly constraining refinements, containing more assumptions than needed for realizability.

Let us consider an extension of the request-grant protocol in Example 2.2. The specification we give here corresponds to the version in [AMT13]. There are two input variables that the environment controls to request access to and to clear some resource, req and cl respectively; and two output variables, gr and val, which grants access to the resource and signals whether it is in a valid state respectively.

We specify this protocol in GR(1). The assumption $J^{\mathcal{E}} = \mathbf{GF}\neg\mathsf{req}$ ensures that the environment does not request access continuously. The guarantees $\rho_1^{\mathcal{S}} = \mathbf{G}(\mathsf{req} \rightarrow \mathbf{XF}\mathsf{gr})$, shows how it can be translated into GR(1) using auxiliary variables., $\rho_2^{\mathcal{S}} = \mathbf{G}((\mathsf{cl} \vee \mathsf{gr}) \rightarrow \mathbf{X}\neg\mathsf{gr})$, $\rho_3^{\mathcal{S}} = \mathbf{G}(\mathsf{cl} \rightarrow \neg\mathsf{val})$, $J_1^{\mathcal{S}} = \mathbf{GF}(\mathsf{gr} \wedge \mathsf{val})$ ensure respectively *(1)* a request is eventually followed by a grant; *(2)* whenever a clear or a grant is active, the grant is unset immediately afterwards; *(3)* whenever a clear is issued, the valid flag is unset; and *(4)* the environment can access a valid state of the resource infinitely many times. This specification is unrealizable since an environment that sets cl to true prevents any controller from simultaneously fulfilling the guarantees $\rho_3^{\mathcal{S}}$ and .[1] This

[1] These guarantees constitute an unrealizable core (see Section 2.2.4).

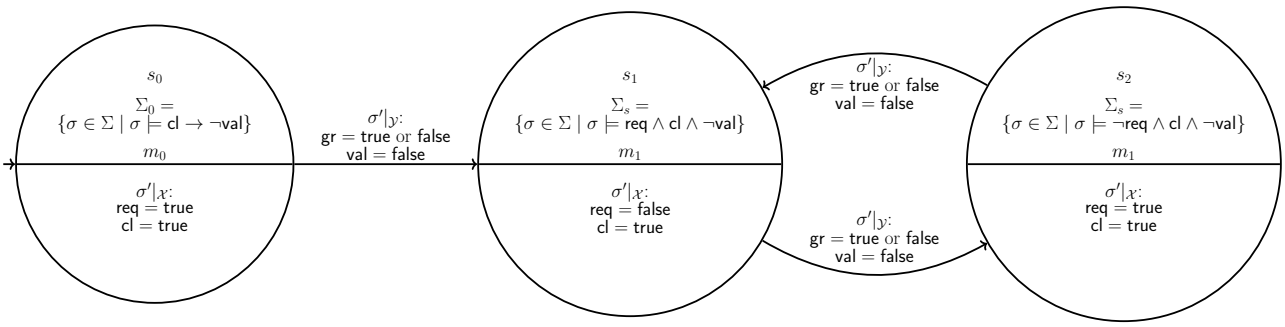


Figure 7.1: Abstract counterstrategy for the request-grant example (see Sections 2.2.3 and 4.3)

environment is represented by the counterstrategy in Fig. 7.1.

Suppose the interpolation bias of Chapter 4 is applied to Fig. 7.1. This produces the assumption $\psi_1 = \mathbf{GF}\neg\mathsf{cl}$, forcing the environment to set cl to false infinitely many times. This is not sufficient to achieve realizability, since this environment can force the violation of the guarantees by setting cl to true every time . This way can never be satisfied, as forces gr to be false at the time step and gr and val can never be true together. , a new counterstrategy is computed and $\psi_2 = \mathbf{G}(\neg\mathsf{val} \rightarrow \mathbf{X}\neg\mathsf{cl})$ is generated. This forces cl to be false at specific time steps (the ones following val being false).

The refinement $\Psi = \{\psi_1, \psi_2\}$ is sufficient, and is returned as a solution. However, the assumption ψ_2 alone would be sufficient, making ψ_1 redundant, and imposing a weaker constraint[2] on the environment compared to ψ. Existing approaches do not support the replacement of older assumptions in such way. Furthermore, since the variable val does not appear in the counterstrategy in Fig. 7.1 (it is a non-influential output variable, see Section 4.3), neither of the biases illustrated in previous chapters could infer this assumption from that counterstrategy.

We propose a criterion to balance between concatenation and replacement of assumptions, based on the notion of minimality with respect to a set of counterstrategies. Notice that our notion does not consider whether the newer assumption is strictly stronger than another (as typically studied in the literature on redundancy, see Section 7.6): in the above example, ψ_2 is not strictly stronger than ψ_1, nor is ψ_1 relative to ψ_2.

[2] *Weaker* can be interpreted either in the implication sense of Definition 3.3 or in the sense of the weakness measure defined in Chapter 6, which, as we proved, is consistent with implication.

7.2 Minimal Assumptions Sets

Here, we formalize the notion of *minimal refinements* which forms the basis of our minimization procedure. We first define the notion of a redundant assumption.

Definition 7.1 (Redundant assumptions)**.** *Given a set of assumptions* $\Psi = \{\psi_1, \ldots, \psi_m\}$ *and a set of counterstrategies* $C = \{c_1, \ldots, c_n\}$*, an assumption* ψ_i *is* redundant with respect to ψ and C *if and only if for all* $c_j \in C$*,* $c_j \not\models \bigwedge_{\psi_h \in \Psi, h \neq i} \psi_h$*.*

In other words, an assumption ψ_i in a refinement Ψ is redundant with respect to a set of counterstrategies C if all the counterstrategies in C are still eliminated if ψ_i is removed from Ψ. So, in order to construct a sufficient solution, given some observed counterstrategies, ψ_i may be safely deleted.

Notice that the definition of redundant assumption is well-posed. A counterstrategy $c \in C$ violates a refinement $\bigwedge \Psi = \bigwedge_i \psi_i$ if and only if it violates at least one ψ_i. It is not the case that two or more assumptions are needed together to eliminate a counterstrategy c, so, when a redundant assumption ψ_i is removed from ψ, there exists an assumption $\psi_h \neq \psi_i$ that eliminates c. In the following, we will just say that an assumption $\psi_i \in \Psi$ is redundant, without specifying the sets Ψ and C when those are clear from the context.

A refinement without a redundant assumption is called minimal.

Definition 7.2 (Minimal refinement)**.** *A set of assumptions* Ψ *is* minimal *with respect to a set of counterstrategies* C *if none of its assumptions is redundant with respect to* Ψ *and* C*.*

7.3 Minimal Refinements Search

In this section, we first present and analyze our algorithm **MinimalRefinement**, which removes redundant assumptions from a refinement. Then we present our changes to the classical FIFO refinement algorithm [AMT13, CA17] to speed up the convergence towards a solution.

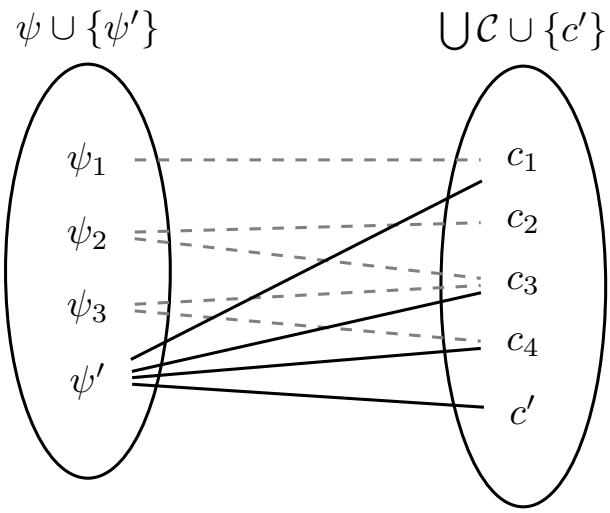


Figure 7.2: Bipartite graph of assumptions and counterstrategies. The continuous lines connect the assumption ψ' generated in the current iteration of the countestrategy-guided loop to the counterstrategies eliminated by it. The dashed lines connect the assumptions of the previous iterations to the respective eliminated counterstrategies.

7.3.1 The Function MinimalRefinement

The core of our minimization procedure is the function **MinimalRefinement**, shown in Algorithm 12. This function receives in input an assumption ψ', an insufficient refinement Ψ, and a set of counterstrategies C, and adds ψ to Ψ ensuring that the resulting refinement is minimal with respect to C. It is designed to be part of the counterstrategy-guided loop in Chapter 3.

Specifically, suppose an unrealizable specification $\langle \mathcal{E}, \mathcal{S} \rangle$ has been extended with the refinement $\Psi = \{\psi_1, \ldots, \psi_m\}$, that was computed incrementally from the counterstrategies $C = \{c_1, \ldots, c_n\}$. Suppose Ψ is not a sufficient refinement. An additional counterstrategy c' is computed by the counterstrategy-guided loop, and a new assumption ψ' is generated to eliminate c'. In general, ψ' may eliminate more than one counterstrategy in $C \cup \{c'\}$, and thus by adding ψ', some ψ_i $(1 \leq i \leq m)$ may become redundant with respect to $C \cup \{c'\}$ and $\Psi \cup \{\psi'\}$.

In order to obtain minimality in the sense of Definition 7.2, the algorithm needs to know the counterstrategies that each assumption eliminates. The relationship between the assumptions in a refinement and the counterstrategies they eliminate can be pictured as a bipartite graph whose vertices are the elements $\psi_i \in \Psi \cup \{\psi'\}$ and $c_j \in C \cup \{c'\}$, such that there is an edge (ψ_i, c_j) if and only if $c_j \not\models \psi_i$.

Example 7.1. *In Fig. 7.2, ψ_1 eliminates c_1 only, ψ_2 removes c_2 and c_3, and ψ_3 removes c_3 and c_4. Suppose ψ'removes c_1, c_3, c_4 and c'. Then, when adding ψ' to Ψ, ψ_1 and ψ_3 become redundant.* □

Let the *degree* of a vertex be the number of edges incident to it, and denote it by $\mathsf{deg}(\psi_i)$ or $\mathsf{deg}(c_i)$, depending on the vertex being an assumption or a counterstrategy respectively. Then an assumption is redundant if and only if all the counterstrategies to which it is connected have a degree at least 2. Indeed, even if such an assumption were eliminated, there would be another assumption eliminating all the counterstrategies connected to it. The algorithm then scans all the ψ_i's and removes all redundant assumptions based on the degree of their connected counterstrategies, resulting in a minimal refinement.

Example 7.2. *All counterstrategies linked to the redundant assumptions ψ_1 and ψ_3 have a degree of 2 or above. So, both can be removed from the refinement.* □

Algorithm 12: MinimalRefinement function

Input: $\Psi = (\psi_1, \ldots, \psi_k)$: insufficient refinement of cardinality k
Input: $\mathcal{C} = (C_1, \ldots, C_k)$: sequence of sets of eliminated counterstrategies such that for every $c \in \bigcup \mathcal{C}$, $c \in C_i$ if and only if $c_i \not\models \psi_i$
Input: ψ': assumption to be added to Ψ
Input: c': a counterstrategy satisfying every assumption in Ψ and eliminated by ψ'
Output: $(\Psi_{new}, \mathcal{C}_{new})$: pair containing a minimal refinement $\Psi_{new} = \Psi \cup \{\psi'\}$ and the corresponding set of eliminated counterstrategies

```
1  C' ← {c'};
2  foreach c ∈ ⋃C do
3      if c ⊭ ψ' then
4          C' ← C' ∪ {c};
5  Ψ_new ← Ψ ∪ {ψ'};
6  C_new ← C ∪ {C'};
7  foreach c ∈ ⋃C_new do
8      deg(c) = CountSets(C_new, c);
9  for i = 1 to k do
10     if ∀c ∈ C_i deg(c) ≥ 2 then
11         Ψ_new.remove(ψ_i);
12         C_new.remove(C_i);
13         foreach c ∈ C_i do
14             deg(c) ← deg(c) − 1;
15 return (Ψ_new, C_new)
```

In order to construct the bipartite graph, the algorithm needs to determine for each pair (ψ_l, c_k) whether $c_k \models \psi_l$. This is a classical model checking problem, and can be solved with standard model checking algorithms [CGH94]. However, in our setting there is no need to model check every pair (ψ_l, c_k). This is because **MinimalRefinement** is called as part of the counterstrategy-guided loop (with the modifications described in Section 7.3.2), so a part of the graph is already constructed at each call.

In what follows, we discuss the details of Algorithm 12. The inputs Ψ and $\mathcal{C}$ correspond to the bipartite graph produced by previous iterations. $\mathcal{C}$ is a collection of counterstrategy sets C_i such that ψ_i eliminates all the counterstrategies in C_i; that is, there is an edge between ψ_i and each of the counterstrategies in C_i. ψ' and c' are the assumption and the counterstrategy produced in the latest iteration of the counterstrategy-guided loop (see Algorithm 13). Hence, by hypothesis $c' \not\models \psi'$.

The function consists of two blocks. In the first (lines 1-4), the algorithm model checks all counterstrategies in the set[3] $\bigcup \mathcal{C}$ against ψ' and constructs the set $C' = \{c \in \bigcup \mathcal{C} \cup \{c'\} \mid c \not\models \psi'\}$. By construction $c' \in C'$ (see line 1). The second block (5-14) analyzes the bipartite graph and identifies the redundant assumptions. First, it builds the new refinement $\Psi_{new} = \Psi \cup \{\psi'\}$ to be minimized and the collection of counterstrategy sets $\mathcal{C}_{new}$ by extending $\mathcal{C}$ accordingly with C' (lines 5-6). Then, for every counterstrategy $c \in \bigcup \mathcal{C}_{new}$, the function **CountSets** counts how many sets in $\mathcal{C}_{new}$ contain c. This corresponds to the degree of the counterstrategy in the bipartite graph. Finally, lines 9-14 remove any ψ_i from ψ_{new} such that every counterstrategy in C_i has a degree greater or equal to 2.

Example 7.3. *Returning to the motivating example of Sec. 7.1,* $\psi_1 = \mathsf{GF}\neg\mathsf{cl}$ *eliminates the first counterstrategy generated, which we denote by* c_1*. So* $C_1 = \{c_1\}$ *and* $\mathcal{C} = \{C_1\}$*. At the first call of* ***MinimalRefinement****, the bipartite graph has a single assumption vertex and a single counterstrategy vertex; so, no change occurs.*

After the second counterstrategy c_2 *is produced,* $\psi_2 = \mathsf{G}(\neg\mathsf{val} \rightarrow \mathsf{X}\neg\mathsf{cl})$ *is generated.* ***MinimalRefinement*** *checks whether* ψ_2 *is violated in* c_1 *and finds that it is. So* $C_2 = \{c_1, c_2\}$*. Now* c_1

[3]Recall that for any sequence (or set) of sets $\mathcal{A} = (A_1, \ldots, A_n)$, $\bigcup \mathcal{A} = \bigcup_{A \in \mathcal{A}} A$.

has degree $\mathsf{deg}(c_1) = 2$ *and* ψ_1 *is only connected to* c_1*. Therefore, the algorithm finds that* ψ_1 *is redundant and it removes it when constructing the new refinement* Ψ_{new}*.* □

The next propositions shows that Algorithm 12 is correct: (1) When the input is a bipartite graph as defined above, the returned refinement Ψ_{new} is minimal with respect to the set containing all the observed counterstrategies $\bigcup \mathcal{C} \cup \{c'\}$; and (2) the output refinement eliminates all counterstrategies; and (3) the output maintains the structure of the pair $(\Psi, \mathcal{C})$.

Proposition 7.1 (Minimality of **MinimalRefinement**)**.** *Let* Ψ *be a refinement,* ψ' *an assumption and* $\mathcal{C} = \{C_1, \ldots, C_n\}$ *a collection of counterstrategy sets. Suppose that (1) for every* $\psi_i \in \Psi$*,* $C_i \in \mathcal{C}$*, and* $c \in \bigcup \mathcal{C}$*,* $c \in C_i$ *if and only if* $c \not\models \psi_i$*; (2)* $c' \not\models \psi'$*; and (3) for every* $\psi_i \in$*,* $c' \models \psi_i$*. Then* Ψ_{new} *computed by Algorithm 12 is minimal with respect to* $\bigcup \mathcal{C}_{new} = \bigcup \mathcal{C} \cup \{c'\}$*.*

For better readability, the longer proof of this proposition (which also includes two lemmas) is deferred to the next subsection. We first state the other propositions.

Proposition 7.2 (Coverage of all counterstrategies)**.** *Under the assumptions of Proposition 7.1,* $\bigcup \mathcal{C}_{new} = \bigcup \mathcal{C} \cup \{c'\}$*.*

Proof. The statement follows the fact that any set $C \in \mathcal{C}_{new}$ is only removed from $\mathcal{C}_{new}$ if every $c \in \mathcal{C}_{new}$ also belongs to at least another set $C_c \in \mathcal{C}_{new}$ (line 12 of Algorithm 12). Therefore, removing C from $\mathcal{C}_{new}$ does not remove c from $\bigcup \mathcal{C}_{new}$. □

Proposition 7.3 (**MinimalRefinement** loop invariant)**.** *Under the assumptions of Proposition 7.1,* $(\psi_{new}, \mathcal{C}_{new})$ *satisfies assumption (1) by replacing* ψ *with* ψ_{new} *and* C *with* $\mathcal{C}_{new}$*.*

Proof. This can be easily concluded by observing that lines 1-4 populate C' with each and every $c \in \bigcup \mathcal{C}_{new}$ such that $c \not\models \psi'$; the subsequent lines that add or remove elements from Ψ_{new} and $\mathcal{C}_{new}$ are always paired. Therefore, the j-th element of $\mathcal{C}_{new}$ contains each and every counterstrategy in $\bigcup \mathcal{C}_{new}$ that is inconsistent with the j-th element of Ψ_{new}. □

Proof of Proposition 7.1. In this subsection, we prove Proposition 7.1. We start with a few simple observations.

Lemma 7.1. *Assume that a refinement Ψ and a collection $\mathcal{C}$ of counterstrategy sets satisfy the assumption (1) of Proposition 7.1. Let the bipartite graph (V, E) be defined such that $V = \Psi \cup \bigcup \mathcal{C}$ and $E = \{(\psi_i, c_j) \in \Psi \times \bigcup \mathcal{C} \mid c_j \not\models \psi_i\}$. Then ψ_i is redundant with respect to Ψ and $\bigcup \mathcal{C}$ if and only if for every c_j such that $(\psi_i, c_j) \in E$, the degree of c_j is greater or equal to 2.*

Proof. Given how the graph is defined, the degree of a counterstrategy is the number of assumptions that are inconsistent with it. So, $\mathsf{deg}(c_j) \geq 2$ if and only if more than two assumptions or more eliminate it. Hence, there exists $\psi_h \neq \psi_i$ such that $\psi_h \not\models c_j$ for any c_j connected to ψ_i. By Definition 7.1 this means ψ_i is redundant. □

Lemma 7.2 (Monotonicity of non-redundancy w.r.t. ψ)**.** *If ψ_i is non-redundant with respect to Ψ and C, it is non-redundant w.r.t. μ and C for any $\mu \subseteq \Psi$.*

Proof. ψ_i is non-redundant, if and only if there is a counterstrategy c_i such that ψ_i is the only assumption removing c_i. Therefore, after removing some of the other assumptions from Ψ to obtain μ, ψ_i will still be the only assumption in μ removing c_i, proving the statement. □

Now we are ready to prove Proposition 7.1.

Proof. (of Proposition 7.1) To prove minimality, we use a loop invariant over the loop on lines 9-14. Let $\Psi_{new}^{[i]}$ be the state of Ψ_{new} at the end of the i-th iteration, with the initial $\Psi_{new}^{[0]} = \Psi \cup \{\psi'\}$ and the output $\Psi_{new} = \Psi_{new}^{[k]}$. We claim the following loop invariant: at the end of iteration i, for all $h \leq i$, $\psi_h \in \Psi_{new}^{[i]}$ if and only if ψ_h is non-redundant with respect to $\Psi_{new}^{[i]}$ and $\bigcup \mathcal{C}_{new}$.

As inductive hypothesis, suppose the loop invariant holds at the end of iteration $i-1$: for all $h \leq i-1$, $\psi_h \in \Psi_{new}$ if and only if ψ_h is non-redundant w.r.t. $\Psi_{new}^{[i-1]}$ and $\bigcup \mathcal{C}_{new}$. During iteration i, ψ_i is removed if and only if the condition in line 10 holds. By Lemma 7.1 this

condition being true is equivalent to ψ_i being redundant w.r.t. $\Psi_{new}^{[i-1]}$. Hence, ψ_i is not removed if and only if it is not redundant w.r.t. $\Psi_{new}^{[i-1]}$.

Moreover, $\Psi_{new}^{[i]} \subseteq \Psi_{new}^{[i-1]}$, since at most one removal occurs during an iteration of the loop. Therefore, by Lemma 7.2, ψ_i is not removed if and only if it is not redundant w.r.t. $\Psi_{new}^{[i]}$. This completes the inductive step: we have proved that for all $h \leq i$, $\psi_h \in \Psi_{new}$ if and only if ψ_h is non-redundant with respect to $\Psi_{new}^{[i]}$.

The property trivially holds for $i = 1$. In this case, ψ_1 is removed from $\Psi_{new}^{[0]} = \Psi$ if and only if it is redundant w.r.t. Ψ. Again by Lemma 7.2, this implies that it is kept in $\Psi_{new}^{[1]}$ if and only if it is non-redundant w.r.t. $\Psi_{new}^{[1]}$.

Hence, the loop invariant holds for $i = k$ and every $\psi_i \in \Psi_{new}$ is non-redundant w.r.t. Ψ_{new} and $\bigcup \mathcal{C}_{new}$. By Definition 7.2, Ψ_{new} is minimal w.r.t. $\bigcup \mathcal{C}_{new}$. □

7.3.2 The Search Algorithm

BFS approaches to assumptions refinement (see Section 3.4.3) keep a FIFO queue of partial refinements—the first element is extracted, and if it is not sufficient a new set of alternative further refinements is generated and appended to the queue and so forth. If a refinement of size n is extracted, the appended refinements have size $n + 1$, and eventually no refinement of size n remains in the queue. Therefore, even if the same refinements appear in the queue more than once, each of them is explored a finite number of times.

On the contrary, we show in the following example that using minimization could lead to the same refinement being added to the queue and explored infinitely many times. We call a refinement that is added more than once to the queue a *duplicate refinement*. Exploring duplicate refinements multiple times can lead to useless, repeated computations, hence delaying the identification of a solution, as the example below shows.

Example 7.4. *Suppose that $\psi_1 \wedge \psi_2'$ is an insufficient refinement and the additional assumption ψ_3 is computed (e.g., using the interpolation bias of Chapter 4), which makes ψ_2' redundant*

w.r.t. the observed counterstrategies. Suppose also that $\psi_1 \wedge \psi_2''$ is insufficient and ψ_3 is computed that makes ψ_2'' redundant. Then $\psi_1 \wedge \psi_3$ is added twice to the queue of candidate refinements. All the further refinements being extensions of $\psi_1 \wedge \psi_3$ will also be explored twice, hence delaying reaching any solution. □

One simple modification to the search procedure is keeping track of the refinements (only) that have already been explored and avoiding their exploration again. However, this may prevent progress towards a solution, as shown the following example.

Example 7.5. *Given an initial specification $\langle \mathcal{E}, \mathcal{S} \rangle$, suppose $\psi_1 \wedge \psi_2 \wedge \psi_3$ is a solution to the realizability problem. The following table describes the first four iterations of the counterstrategy-guided loop where minimization is applied to every refinement generated by the bias heuristic.*

step #	$(\Psi, \mathcal{C})$	c	Ψ'	***MinRef***
1	$(\varnothing, \varnothing)$	c_1	$\{\psi_1, \psi_2\}$	$\{\{\psi_1\}, \{\psi_2\}\}$
2	$(\{\psi_1\}, \{\{c_1\}\})$	c_2	$\{\psi_2\}$	$\{\{\psi_2\}\}$
3	$(\{\psi_2\}, \{\{c_1\}\})$	c_3	$\{\psi_1\}$	$\{\{\psi_1\}\}$
4	$(\{\psi_2\}, \{\{c_1, c_2\}\})$	c_3	$\{\psi_1\}$	$\{\{\psi_2, \psi_1\}\}$

*The column $(\Psi, \mathcal{C})$ represents the head of the FIFO queue at each step. At each search step, the first element in the queue is extracted, a counterstrategy c_i is computed and a set Ψ' of alternative assumptions that eliminate c is generated. The column **MinRef** shows the minimal refinements generated after adding each assumption in Ψ' to the refinement Ψ and minimizing it w.r.t. the counterstrategies in $\bigcup \mathcal{C} \cup \{c\}$ as per Algorithm 12. The queue is empty at start. The counterstrategy c_1 is produced from which ψ_1 and ψ_2 are computed. These are minimized and added to the queue. Then, a tuple $(\Psi, \mathcal{C})$ is extracted from the queue in step 2, from which c_2 is computed, and ψ_2 is generated. After minimizing $\psi_1 \wedge \psi_2$ w.r.t. $\{c_1, c_2\}$, only ψ_2 remains.*

Notice that ψ_2 is generated twice: once in step 1 as a singleton refinement, and once in step 2 as the product of minimization. Therefore, it is popped twice from the queue at steps 3 and 4. At step 4, finally ψ_2 is not redundant and therefore it does not disappear after minimization. If the refinement ψ_2 were not explored twice, the refinement $\psi_2 \wedge \psi_1$ would not have been added to

the queue and the solution above could not be reached. The difference between the first and the second time ψ_2 is explored is in the counterstrategies it eliminates. □

As suggested in this example, a way to circumvent this problem consists in keeping track of the eliminated counterstrategies when checking for duplicate refinements in the queue, rather than just the refinements. Two refinements are actually duplicate if not only do they contain the same assumptions, but they also eliminate the same number of observed counterstrategies.

This is shown in Algorithm 13, which modifies the counterstrategy-guided loop of Algorithm 3 to include the duplicate checks. The first change is in the set candidateRefs, which now contains not only refinements Ψ, but pairs $(\Psi, \mathcal{C})$ representing bipartite graphs. Then, it makes use of a set exploredRefs (line 3) to keep track of the refinements explored together with the number of counterstrategies eliminated by those refinements. This set is updated after every refinement exploration (line 20). If a refinement Ψ has already been explored with the same number of explored counterstrategies, the pair $(\Psi, |\bigcup \mathcal{C}|)$ belongs to exploredRefs and the refinement is discarded as duplicate (line 6).

The main difference between Algorithm 3 and Algoritm 13 is in the way candidate refinements are generated in each iteration. In Algorithm 3, this is done by **GenerateRefinements**, which applies a bias heuristic like varbias (Section 3.4.2) or interpolation (Chapter 4) and returns a set of alternative refinements. In Algorithm 13, each of the assumptions ψ' generated by the bias (line 10) is input to the *MinimalRefinement* function together with the refinement Ψ (line 12) before the resulting minimal refinement is added to candidateRefs. This way, only minimal refinements are explored by the loop.

Lines 14-16, greyed out in the algorithm, also add the non-minimal refinement $\Psi \cup \{\psi'\}$ together with the minimal one. These lines implement the *hybrid* approach we discuss in Section 7.4.

Algorithm 13: RefinementSearch function

Input: $\langle \mathcal{E}, \mathcal{S} \rangle$: pair of sets of initial assumptions and of guarantees
Input: *hybrid*: parameter that controls the type of approach used
Output: solutions $= \{\Psi_i\}$, set of alternative refinements such that $\langle \mathcal{E} \cup \Psi_i, \mathcal{S} \rangle$ is realizable for every i

```
1  solutions ← ∅;
2  candidateRefs ← {∅};
3  exploredRefs ← ∅;
4  repeat
5      (Ψ, C) ← candidateRefs.extract();
6      if (Ψ, |⋃C|) ∉ exploredRefs then
7          E_ref ← E ∪ Ψ;
8          if IsSatisfiable(E_ref) and ¬IsRealizable(⟨E_ref, S⟩) then
9              c' ← ComputeCounterstrategy(E_ref, S);
10             Ψ' ← ApplyBias(E, S, Ψ, c');
11             foreach ψ' ∈ Ψ' do
12                 (Ψ_new, C_new) ← MinimalRefinement(Ψ, C, ψ', c');
13                 candidateRefs.add((Ψ_new, C_new));
14                 if hybrid then
15                     C' ← last set in C_new;
16                     candidateRefs.add((Ψ ∪ {ψ'}, C ∪ {C'}));
17             end
18         else if IsSatisfiable(E_ref) then
19             solutions.add(Ψ);
20         exploredRefs.add((Ψ, |⋃C|));
21     end
22 until candidateRefs = ∅;
23 return solutions
```

7.3.3 Time Complexity

Executing **MinimalRefinement** introduces an additional overhead to the generation of each refinement compared with the state-of-the-art BFS strategy. In the following we argue that this overhead is negligible in one iteration of the main loop of Algorithm 13 (lines 4-20).

Let $m_{\mathcal{C}}$ be the number of counterstrategies in $\bigcup \mathcal{C}$, $m_\Psi = |\Psi|$ the number of assumptions in ψ, and $|Q_c^{max}|$ the maximum number of states in a counterstrategy. Executing lines 1-4 of Algorithm 12 requires a number $m_{\mathcal{C}}$ of LTL model checking operations; each of these involves a counterstrategy and a single GR(1) assumption.All the model checking operations require collectively $O(m_C |Q_c^{max}|^2)$ time (see Section 7.3.4 for details of how this is computed). The

bipartite graph in lines 5-14 has size $O(m_\psi m_\mathcal{C})$: it requires this amount of computation to be built, and asymptotically as many operations to be minimized. Also notice that in the worst case, Algorithm 12 produces as many assumptions as counterstrategies in a refinement: so, $m_\psi \leq m_\mathcal{C}$, yielding an execution time of $O(m_\mathcal{C}^2)$ for the graph operations.

In summary, executing **MinimalRefinement** requires an amount of time going asymptotically to $T_{\mathsf{MinRef}} = O(m_C|Q_c^{\mathsf{max}}|^2 + m_\mathcal{C}^2)$. Let us compare this complexity with the time $O(m_\psi n|Q|^2)$ for realizability checks and counterstrategy computation (lines 8-9 of Algorithm 13), where n is the number of guarantees in the GR(1) formula. Without assumption minimization, $m_\psi = m_\mathcal{C}$ for every refinement, and counterstrategy computation would take $T_{\mathsf{CS}} = O(m_\mathcal{C}|Q|^2)$. By introducing minimization, the time for these checks is $T_{\mathsf{CS,m}} = O(m_\psi|Q|^2)$, and the minimization overhead T_{MinRef} is added. In other words, a part of the time T_{CS} quadratic in $|Q|$ is replaced by the use of **MinimalRefinement**, quadratic in $|Q_c^{max}|$. Counterstrategies often contain significantly fewer states than entire games (which grow exponentially with the number of system variables).

For each explored refinement (i.e., for each iteration of the main loop in Algorithm 13), the minimization function is called as many times as assumptions returned by **ApplyBias**: we denote this number as $|\Psi'|$. Existing methods yield a $|\Psi'|$ in the order of tenths [CA18]. Thus, for small $m_\mathcal{C}$, such that the quadratic term in T_{MinRef} is not dominating, we obtain a speedup factor proportional to

$$\begin{aligned}\frac{T_{\mathsf{CS}}}{T_{\mathsf{CS,m}} + T_{\mathsf{MinRef}}} &= \frac{m_\mathcal{C}|Q|^2}{m_\psi|Q|^2 + |\Psi|m_\mathcal{C}|Q_c^{\mathsf{max}}|^2} \\ &= \frac{m_\mathcal{C}}{m_\psi + |\Psi|m_\mathcal{C}\frac{|Q_c^{\mathsf{max}}|^2}{|Q|^2}} \approx \frac{m_\mathcal{C}}{m_\psi}\end{aligned}$$

for a single refinement exploration. This can be interpreted this way: minimizing refinements yields the same gain in computation time as one would obtain by just exploring a node of length m_ψ instead of a node of length $m_\mathcal{C}$. The overhead introduced by performing the minimization operations is negligible.

These observations are supported experimentally in Sec. 7.5.2. We show that minimization

allows for exploring a larger number of refinements than BFS, meaning that the time per iteration of Algorithm 13 is lowered by the former.

7.3.4 Computation Time of Model Checking in MinimalRefinement

When using the algorithm in [CGH94], LTL model checking is converted into a problem of Computation Tree Logic (CTL) model checking with fairness conditions. The full reduction procedure is described in the cited paper.

In this procedure, a fairness condition is introduced for each **G** and each **F** subformula appearing in the formula to check. In *MinimalRefinement*, the formula to check is a GR(1) assumption ψ' (line 3 of **MinimalRefinement**). Since an assumption contains at most two operators **G** and **F**, there are at most two fairness conditions in the CTL problem. Solving this problem with a symbolic algorithm requires applying two nested alternating fixpoint operations for each fairness condition (chapter 6 of [CGP99]), each taking a number of iterations upper bounded by the number of states in the model to check (see also [BCJ+97] regarding complexity of nested fixpoint operations). In summary, solving one of the model checking problems requires $O(|Q_c^{max}|^2)$, and since m_C model checks are performed, the total computation time of lines 1-4 is $O(m_C|Q_c^{max}|^2)$.

7.4 Hybrid Refinement Generation

The search in Algorithm 13, compared with the BFS approach of Algorithm 3, explores more thoroughly the region of the search space containing shorter refinements, thus finding solutions that BFS would not found otherwise. This however may delay identifying longer refinements that are needed for converging towards a solution. This is a problem, in particular, if the specification of interest does not have short solutions. Moreover, an approach that solely explores minimal refinements might miss shorter solutions which may only be reached by minimizing long candidate refinements, as the example below shows.

Example 7.6. *In the AMBA04 case study (see Section 3.5), a sufficient solution is* $\Psi = \{\mathbf{G}(\neg\mathsf{hready} \rightarrow \mathbf{X}\mathsf{hready})\}$. *Interpolation does not compute this assumption until some partial refinement of length* 3 *is given as input to* ***ApplyBias****. This partial refinement is found by the BFS approach in fewer iterations than Algorithm 13.*

To relax this effect, we propose a hybrid approach that integrates BFS in the search. By allowing refinements explored by BFS to be added to the queue together with the minimal refinements generated by Algorithm 12, as done in lines 14-16 of Algorithm 13 (highlighted in gray). These lines are executed when the input *hybrid* is set to true. In this way, longer candidate refinements are added to candidateRefs in earlier iterations of the search, and longer solutions can be identified in shorter time.

7.5 Experiments

We present the results of experiments conducted on a benchmark of case studies of unrealizable specifications taken from the literature. The experiments are devised to show the impact of embedding **MinimalRefinement** on the number and the length of both the partial refinements explored and the final solutions. Our implementation uses the synthesis tool RATSY [BCG$^+$10] to compute counterstrategies, and our Python implementation builds on the model checking algorithm of [CGH94] to check the satisfaction of assumptions in counterstrategies. All experiments were executed on the same machine (Ubuntu, Intel Core i7 CPU and 16 GiB of RAM).

7.5.1 Methodology

The first experiment uses the experiment in Section 4.8.2 as a baseline. There, we apply a pure BFS approach to the six case studies over 24 hours and record the number of results over time, reporting also statistics of the explored refinement tree like the length (see Table 4.6, rows MinSolDepth and MaxSolDepth). In this experiment, we run our two proposed approaches on the

same case studies. We perform two runs of Algorithm 13: one, labelled MinimalRefinement, in which the *hybrid* parameter is set to false and the search only explores minimal refinements; and one, labelled Hybrid, where the parameter is set to true and the search implements the hybrid approach described in Section 7.4.

A run of the experiment on a single case study involves two phases, replicating the ones of Section 4.8.2. In the first phase, the loop in Algorithm 13 is executed up to the 24-hour timeout, or until no new refinement are contained in candidateRefs. In the second phase, if candidateRefs still contains refinements, a realizability check is performed on them to identify which of those are solutions; they are not expanded in case they are not. This phase lasts 2 hours, and any refinement that has not been checked within this time limit is discarded. The returned solutions undergo two post-processing phases that (1) remove any duplicate solutions, and (2) delete any solution semantically equivalent to another one in the set.

The six case studies are: three versions of the popular AMBA-AHB protocol specification and three developed in [KMR17] for testing Justice Violation Transition Systems, called JVTS case studies for short. The AMBA-AHB specification [BJP+12, AMT13, LDS11, CGS08] describes the requirements of an on-chip communication protocol developed by ARM for high-speed links. The case studies are described in Section 3.5.

In Section 4.8.2, we compare the number of solutions of two bias heuristics, interpolation and multivarbias. For this experiment, we select the bias that produces more solutions in that comparison as the bias for the runs of MinimalRefinement and Hybrid, namely interpolation [CA17] for AMBA02, AMBA04, AMBA08 and ColorSort, and multivarbias for GyroAspect and Humanoid. Indeed, we expect MinimalRefinement to yield fewer solutions than BFS, since it focuses on shorter refinements than BFS. If we tested MinimalRefinement on the bias yielding fewer solutions with BFS, we would presumably observe few or no solutions, and obtain no statistics on the length of the solutions returned by MinimalRefinement. Given the large time allotted per run (26 hours), we make this choice to optimize the quality of the statistics we expect to obtain.

7.5.2 Results and Discussion

Table 7.1 summarizes the results. *ExploredRefs* is the number of non-duplicate refinements explored. Notice that this value is consistently higher for MinimalRefinement, since checking realizability and computing counterstrategies of shorter specifications requires less time. Such gains can be observed in the average computation time despite the length of the minimal refinements typically being 1-3 assumptions fewer than the non-minimal ones. *Min/Max/ModeLen* gives respectively the minimum, maximum, and mode of the length of explored refinements. Minimization effectively reduces the number of assumptions in most explored refinements. This is shown in the row *Min/Max/ModeRedAss*, respectively, giving the minimum, maximum and mode of the number of redundant assumptions eliminated in each MinimalRefinement call. *Sol* is the number of non-equivalent solutions found during the search. These numbers are the results of the post-processing steps described in Section 7.5.1. As expected, our approach finds

Table 7.1: Experiments' results. The rows with * do not consider Hybrid.

	Statistics	AMBA02	AMBA04	AMBA08	ColorSort	GyroAspect	Humanoid
BFS	ExploredRefs	279	315	287	723	1980	2734
	Min/Max/ModeLen	0/5/5	0/5/5	0/4/4	0/6/6	0/3/3	0/3/3
	Sol	73	28	10	23	**883**	116
	Min/Max/ModeSolLen	1/5/4	3/5/4	4/4/4	3/6/5	2/3/3	1/3/3
	SolBFSOnly*	**41**	**17**	**10**	**4**	**843**	**19**
	ExpandedRefs	45	85	49	154	71	102
	False	39	42	10	0	401	19
Minimal Refinement	ExploredRefs	463	658	455	1251	2139	**3558**
	Min/Max/ModeLen	**0/4/2**	**0/3/2**	0/3/1	**0/4/2**	**0/3/2**	**0/2/1**
	Min/Max/ModeRedAss	0/3/1	0/3/1	0/2/0	0/4/0	0/2/1	0/2/1
	Sol	26	13	0	5	449	15
	Min/Max/ModeSolLen	**1/3/2**	**1/3/3**	N/A	**1/1/1**	**1/3/2**	**1/2/1**
	CoveredBFSSol	32	11	0	19	40	97
	SolMinimalOnly*	**21**	**12**	0	**2**	**437**	**6**
	ExpandedRefs	169	372	248	651	817	**1236**
	False	**11**	**2**	0	**0**	**173**	**5**
	DuplicateRefs	455	540	468	774	6117	14495
Hybrid	Explored refs	**1072**	**745**	**791**	**1962**	**2197**	3494
	Min/Max/ModeLen	0/4/2	0/5/4	0/5/2	0/6/6	0/4/2	0/4/2
	Sol	**178**	**54**	**3**	**55**	667	**129**
	Min/Max/ModeSolLen	1/4/3	1/5/4	**1/4/4**	1/6/6	1/4/2	1/4/2
	CoveredBFSSol	**66**	**19**	**10**	**22**	37	**108**
	SolHybridMinimal	5	2	0	5	11	11
	SolHybridOnly	**161**	**45**	**2**	**48**	**610**	**113**
	ExpandedRefs	**248**	**398**	**234**	**667**	**1148**	1139
	False	11	10	3	0	272	20

fewer distinct solutions than BFS (i.e., *Sol* is lower). This has two different explanation. The first is that minimization is a weakening step (it indeed removes assumptions from a refinement, making it weaker). Therefore, MinimalRefinement focuses on weaker refinements than BFS,

and more steps may be required to achieve realizability.

The second explanation is related to the notion of *coverage* we define below, used to compare the solutions of MinimalRefinement and Hybrid with the solutions of BFS.

Definition 7.3 (Coverage between solutions)**.** *We say that the solution refinement* Ψ_1 covers *the solution* Ψ_2 *if* $\Psi_1 \subseteq \Psi_2$.

If the MinimalRefinement solution Ψ_1 covers the BFS solution Ψ_2, then Ψ_2 is bound to contain redundant assumptions not needed for realizability. For instance, in AMBA04 the solution $\mathbf{G}(\neg\mathsf{hready} \rightarrow \mathbf{X}(\neg(\neg\mathsf{hready})))$, identified by the MinimalRefinement approach, covers the longer BFS solution $\{\mathbf{G}(\mathbf{F}(\neg(\mathsf{hbusreq3} \wedge \neg\mathsf{hmaster1}))), \mathbf{G}(\mathbf{F}(\neg(\neg\mathsf{hmaster1} \wedge \mathsf{hbusreq2}) \wedge \neg(\mathsf{hmaster0} \wedge \mathsf{hbusreq2}) \wedge \neg(\mathsf{hbusreq2} \wedge \neg\mathsf{hmaster1}) \wedge \neg(\mathsf{hbusreq2} \wedge \mathsf{hmaster0}))), \mathbf{G}(\mathbf{F}(\neg(\mathsf{hmaster1} \wedge \mathsf{hbusreq1}))), \mathbf{G}((\neg\mathsf{hready}) \rightarrow \mathbf{X}(\neg(\neg\mathsf{hready})))\}$ amongst others.

CoveredBFSSol shows the total number of BFS solutions covered by at least one minimal solution. *SolMinimalOnly* is the number of distinct minimal solutions not covering any BFS solution (thus found by MinimalRefinement only). *SolBFSOnly* is the number of solutions not covered by any solution found by MinimalRefinement. The figures in the table suggest that one solution of MinimalRefinement covers more than one solution of BFS. Hence the lower value of *Sol* for MinimalRefinement is due to the fact that two non-equivalent solutions of BFS, when minimized, return the same minimal solution, which is counted only once in MinimalRefinement. *ExpandedRefs* gives the number of refinements that have undergone expansion, i.e., upon which ComputeCounterstrategy and ApplyBias have been called. This number is higher in our approach, meaning that more counterstrategies could be computed within the same time. *False* contains the number of inconsistent refinements computed during the search (equivalent to false), and *DuplicateRefs* is the number of generated nodes discarded by our search for being duplicates (see Sec. 7.3.2). In summary, although our approach produces fewer solutions than BFS, they are typically shorter and hence less restrictive (also note that our method explores more shorter refinements).

Table 7.1 also shows the effect of hybridization. *SolHybridMinimal* shows the solutions common

to the hybrid approach and MinimalRefinement, and *SolHybridOnly* the ones not covering any BFS solution nor are returned by MinimalRefinement. In all the cases, both the number of refinements explored and the number of solutions increase. The former can be explained since more alternative assumptions (both minimal and non-minimal) are generated at every single iteration of RefinementSearch. The latter is due to the fact that longer solutions are generated earlier in the search. Note that the higher number of solutions allows for covering more BFS solutions, rendering the latter redundant; in AMBA08, one of the solutions covers all the 10 BFS solutions, while we were not able to find any solution by solely exploring minimal refinements. However, some of the solutions returned by Hybrid are not minimal, and therefore the length of those solutions can be higher than the ones of MinimalRefinement.

To assess the weakness of minimal solutions compared to non-minimized ones, we ranked the solutions found by MinimalRefinement and BFS using the weakness measure proposed in Chapter 6. We observe that in AMBA02, the 10 weakest solutions are all found by MinimalRefinement, of which only 2 are found by BFS. In AMBA04, 8 out of the 10 weakest solutions are found by MinimalRefinement; BFS only finds the two other ones (one of which is covered by a shorter minimal solution). With the only exception of GyroAspect, minimal solutions are among the weakest found by the two approaches, and the solutions found by MinimalRefinement cover the majority of the weakest solutions found by BFS. This confirms our hypothesis that MinimalRefinement focuses on weaker refinements and finds weaker solutions than BFS.

7.5.3 Tests with Small Timeout

The second experiment employs a wider set of case studies, and hence less time is allocated to each. This experiment tests the specifications in the original SYNTECH case studies of [MRS19], introduced in Section 3.5 (see Table 4.2). This experiment consists of the same two phases as the previous one, but with a 30 minute timeout. This timeout has been chosen to be longer than the 10-minute timeout of [MRS19] and Section 4.8.1, in order to give MinimalRefinement sufficient time to reach some solution, but short enough to be able to run all the

Table 7.2: 30-minutes experiment results

	BFS			Minimal			Hybrid		
	Explored	Sols	Min/Max/ ModeLen	Explored	Sols	Min/Max/ ModeLen	Explored	Sols	Min/Max/ ModeLen
Run 1	11	4	2/2/2	31	6	2/2/2	**65**	**8**	2/2/2
Run 2	7	1	2/2/2	9	1	2/2/2	**26**	1	2/2/2
Run 3	3	1	1/1/1	3	1	1/1/1	**5**	1	1/1/1
Run 4	30	15	4/4/4	30	15	4/4/4	**146**	**35**	4/4/4
Run 5	764	13	4/4/4	1668	12	**3/4/3**	**1679**	**18**	3/4/4
Run 6	2	1	1/1/1	2	1	1/1/1	**3**	1	1/1/1
Run 7	1177	9	3/3/3	1381	0	N/A	**2763**	**35**	**2/3/2**
Run 8	6	4	1/2/2	6	4	**1/1/1**	**14**	**5**	1/2/1
Run 9	2	1	1/1/1	2	1	1/1/1	**3**	1	1/1/1
Run 10	30	15	4/4/4	30	15	4/4/4	**146**	**35**	4/4/4
Run 11	759	13	4/4/4	**1663**	12	**3/4/3**	1657	**18**	3/4/4
Run 12	30	15	4/4/4	30	15	4/4/4	**135**	**32**	4/4/4
Run 13	658	12	4/4/4	1457	12	**3/4/3**	**1458**	12	3/4/4

Table 7.3: 30-minutes experiment case studies

Run	Case study	Bias
Run 1	GyroUnrealizable_Var2_710_GyroAspect_unrealizable	Interpolation
Run 2	HumanoidLTL_458_Humanoid_fixed_unrealizable	Interpolation
Run 3	HumanoidLTL_531_Humanoid_unrealizable	Multivarbias
Run 4	HumanoidLTL_713_Humanoid_unrealizable	Interpolation
Run 5	HumanoidLTL_713_Humanoid_unrealizable	Multivarbias
Run 6	HumanoidLTL_741_Humanoid_unrealizable	Interpolation
Run 7	HumanoidLTL_741_Humanoid_unrealizable	Multivarbias
Run 8	HumanoidLTL_742_Humanoid_unrealizable	Interpolation
Run 9	HumanoidLTL_742_Humanoid_unrealizable	Multivarbias
Run 10	HumanoidLTL_NotRealizable2_879_Humanoid_unrealizable	Interpolation
Run 11	HumanoidLTL_NotRealizable2_879_Humanoid_unrealizable	Multivarbias
Run 12	HumanoidLTL_NotRealizable2_884_Humanoid_unrealizable	Interpolation
Run 13	HumanoidLTL_NotRealizable2_884_Humanoid_unrealizable	Multivarbias

14 case studies in reasonable time. We run the experiments by using both interpolation and multivarbias implementations of ApplyBias, for a total of 28 runs per approach.

Table 7.2 summarizes the results of this experiment. The table only shows results for the runs in which either MinimalRefinement or BFS found at least one solution. This happened in 13 out of the 28 runs. This is not owing to our approach: the authors of the original work [MRS19] were not able to find solutions to these either. Table 7.3 shows for every run the name of the specification and the bias heuristic used.

In 30 minutes, we are able to observe the same effects as in the first experiments: (1) MinimalRefinement focuses on shorter refinements than BFS and is therefore able to explore more candidates, (2) it finds shorter solutions than BFS but in a smaller number, and (3) Hybrid increases the number of solutions found also compared to BFS. In any case, as predictable, the impact of these advantages is less visible on such a short time scale.

7.6 Related Work

The work in [MRS19] proposes an assumption refinement procedure that makes use of a more efficient alternative to counterstrategies, namely JVTSes [KMR17] (see Section 8.5). A JVTS is an abstraction of counterstrategies in which cycle states and transient states are merged into "macro-states". Although the functions **ComputeCounterstrategy** and **ApplyBias** in Algorithm 13 receive counterstrategies in input, the notions of redundancy and minimality can an be seamlessly applied to JVTSes. This is because our **MinimalRefinement** procedure treats counterstrategies as abstract objects, only exploiting their satisfaction relation with assumptions. What is assumed on the input is that the question whether or not it satisfies an LTL formula is well-defined. This assumption is fulfilled by JVTSes.

The same work in [MRS19] also presents the notion of *refinement cores*: given a solution to a refinement problem, its core is defined as a minimal subset of those assumptions that is still a solution to the problem. They apply delta debugging [Zel99] to the solutions of their BFS search in order to find such cores and hence produce shorter solutions. The solutions produced by MinimalRefinement are refinement cores by construction. The main difference is that our minimization algorithm affects the entire search rather than the final solutions. By focusing the search on shorter refinements, we find solutions that cannot be found by reducing BFS's solutions (row SolMinimalOnly in Table 7.1). Another difference lies in the technique used for minimization, model-checking the observed counterstrategies versus delta debugging. While the former can be used at any point of the search, the latter can only be applied to solutions, as it requires a realizable refinement as input.

The definition of minimal assumptions sets is inspired by the problem of minimum set cover [Vaz03, BMPS05]. Given a set of elements C and a collection S of subsets of C, the minimum set cover is a subcollection of subsets $S' \subseteq S$ such that $\bigcup S' = C$ and S' contains the least number of subsets needed to cover C. Notice that our problem of finding minimal refinements is a relaxation of this NP-hard problem, as we are not interested in minimizing the number of subsets; rather, our notion of minimality corresponds to that of non-redundancy in [BMPS05]. Accordingly, Algorithm 12 is a simplification of the greedy algorithm SCGreedy described

in [BMPS05]: our algorithm does not select the assumption that removes the most counterstrategies at each step (as in line 2 of SCGreedy). This step could be easily included in our approach. However, it would not produce a sensible improvement in our case studies, since all explored nodes are already quite small (at most of size 3, see Section 7.5).

Redundancy and minimization in logical formulae have been studied extensively, in work such as [GF93, GST17, Lib05, NKWF14]. The notion of redundancy we present here is *substantially* different. Redundancy in logic is typically defined in terms of implication and entailment: a formula is redundant if it is implied/entailed by another. In our case, redundancy is defined with respect to the goal of eliminating a sample of counterstrategies: one assumption may not be entailed by any other in a refinement and still be redundant if all the observed counterstrategies are eliminated by another assumptions in the refinement.

7.7 Summary

In this chapter, we introduce a notion of minimality of refinements with respect to a set of counterstrategies. The goal is to reduce the number of assumptions contained in the refinements explored by the counterstrategy-guided loop. The notion of minimality is based on the observation that each assumption generated in one iteration of the loop has the objective of eliminating a counterstrategy. If some assumption eliminates counterstrategies that are also eliminated by other assumptions in the refinement, the assumption becomes redundant and should be removed.

To this goal, we provide a minimization algorithm that extends any bias heuristic to eliminate all redundant assumptions from a refinement. The minimization algorithm is structured to reuse the information from previous iterations of the counterstrategy-guided loop in the form of a bipartite graph, hence reducing the overhead imposed on the bias on each iteration. We prove the correctness of the algorithm and discuss the impact of its overhead.

The effect of enhancing the bias with minimization is that the counterstrategy-guided loop focuses on shorter refinements than approaches that do not use minimization. This results in

a shorter time needed to explore each refinement and a larger number of refinements being explored in a given time budget. However, since the generated refinements contain fewer assumptions, they are also weaker and the time to reach the first solution typically increases.

In particular, we observe that in the experiments, no solutions can be found on the largest case study (AMBA08) if only minimal refinements are searched. To soften this effect of minimization, we also propose a hybrid search approach in which at every iteration of the counterstrategy-guided loop, both the minimal and the non-minimal refinements produced by the bias are added to the set of candidates. The experiments show that this has the effect of combining the advantages of both extremes: finding a large number of solutions while exploring a larger number of refinements and discovering minimal solutions.

Chapter 8

Related Work

In this chapter we survey additional work that deals with assumptions refinement in different contexts than reactive synthesis, or use effectively some of the ideas presented in this thesis for other objectives.

8.1 Automata-Based Assumptions Refinement

Additional work on assumptions refinement makes use of non-declarative models for assumptions, such as game structures or labeled transition systems. With regards to the parity game models used for controller synthesis (such as defined in [SS13, SSR08]), the work in [CHJ08] defines the concept of safety assumptions as sets of edges that have to be avoided by the environment, and the concept of fairness assumptions as sets of edges that have to be traversed by the environment infinitely often. The work devises an algorithm for finding minimal edge sets in order to ensure that the controller has a winning strategy. Our approach instead focuses on synthesizing general declarative temporal assertions whose inclusion has the effect of removing edges from the game structure.

The problem of synthesizing environment constraints has also been tackled from the perspective of model checking. Assume-guarantee reasoning [Var95, PDH99] is a model checking paradigm

used to enable verification of complex systems. In these approaches, the system is decomposed into several modules that are checked separately; in checking each module, the rest of the system is considered as an environment that has to satisfy a set of assumptions; checking the module consists in proving that the property to check is also true, provided that the assumptions are true. To complete the proof, the environment is finally checked against its assumptions.

Several approaches have been proposed to automatically identify a sufficient set of assumptions for this paradigm [GPG08, AMT15, LSWW10, NA06, GPB02, BGP03, CGP03]. A first difference from our work lies in the final goal (verification in their case, synthesis in ours). Moreover, much work in the area models assumptions as labeled transition systems rather than declarative specifications. The work in [CGP03, BGP03] uses the popular algorithm L^* [Ang87] to learn automata from membership and equivalence queries. Equivalence queries are used to test candidate assumptions for sufficiency and membership queries to check whether additional traces should be included in the assumptions, in order to weaken them when possible. The approach is targeted for safety properties, which, when converted to LTL, are a superset of GR(1) invariants but do not include fairness conditions. The work in [GPG08] uses instead abstraction refinement for generating assumptions: the goal is to allow nondeterministic automata to be output, resulting in smaller state spaces and faster verification. Nam et al. [NA06] proposes an approach to find a good system decomposition automatically, together with learning the assumptions. The only work in this field that uses declarative specifications is [AMT15]: here assumptions and properties to check are GR(1) formulae, and the approach of [AMT13] is applied to find the assumptions. This constitutes an additional application of assumptions refinement other than controller synthesis, where our interpolation-based approach can be applied as well.

8.2 Declarative Assumptions Refinement

A further approach to assumptions refinement is presented in [KFG+18] as part of a reactive synthesis algorithm. This work focuses on specifications expressed as $\forall\exists$ formulae over the

states of the game and the next input and output variables. A specification is in the form of an initial guarantee $G_I(s)$ that is satisfied by a subset of game states, a transition constraint $G_T(s, i, s')$ that dictates which states s' are reachable when the input i is read from state s, and an assumption $A(s, i)$ that constrains the input i in the state s. Such kinds of formulae are as expressive as GR(1) invariants, and therefore are not able to express fairness conditions (although they are able to express a bounded-time notion of fairness, namely k-liveness [CS12]). This reduction in expressiveness comes together with a more efficient synthesis algorithm only involving one fixpoint operation rather than three nested ones as in [BJP$^+$12].

Part of the synthesis process consists in refining A with an additional assumption $R_n(s, i)$, a disjunctive formula identifying *regions of validity*: this is the set of states and inputs from which there exists an output state s' satisfying the specification. The procedure AE-VAL used to compute R_n makes use of SMT-solving rather than the symbolic algorithm in [KHB09]. In our context, AE-VAL could replace together the counterstrategy computation and bias heuristic used to compute assumptions (lines 6-7 of Algorithm 3). However, in spite of a potentially faster technique to compute assumptions, this procedure would return one solution only, and it would be limited to searching invariants.

The notion of error resilience outlined in Section 8.3 has also been studied in [NWZ20] for the class of parity games, that extends GR(1) games. The games are extended with *disturbance edges*, actions that the environment can take instead of the expected actions. The resilience notion is related to how many disturbances in the environment's behaviour can be tolerated by the controller while still winning the game. The work proposes an algorithm to optimize the resilience of the resulting controllers in such games.

8.3 Reactive Synthesis

Our work falls under the same category as [LDS11, AMT13, MRS19], which focus on refining GR(1) specifications. In general, assumptions refinement is a preliminary step for reactive synthesis: given a GR(1) specification, the goal of synthesis is to identify a controller strategy

that wins the game defined in Section 2.2.2. Our setting of reference is the classical formulation of reactive synthesis, where the specification is a pure GR(1) formula as defined in Chapter 8. The algorithm to solve this problem is presented in [BJP+12, PP06], and assumes a realizable specification. Several applications of reactive synthesis can be found in [BDP+13, DBPU13, GKW+15]. Recently, the work in [MS20] proposes just-in-time synthesis, where the controller is not computed at once at synthesis time, but the next state is dynamically computed at execution time.

There has been interest in reactive synthesis with quantitative extensions of GR(1) specifications, allowing more expressive power to formulate preferences between assumptions/guarantees and/or finite-time constraints. The work in [ET14] proposes modifications to the synthesis algorithm in [BJP+12] to enable synthesis with the possibility of violating the assumptions (a notion known as *error resilience*): the controller is guaranteed to satisfy its guarantees even if up to k assumptions violations occur within a time frame of d steps.

There has been recent work on adapting the concept of *energy games* (games with quantitative goals) to reactive synthesis. Energy games were introduced in [CdAHS03] to model the behaviour of multi-component systems in which two components compete on the access of a shared resource, such as battery energy, for the satisfaction of some goals. The work in [AMPR20] defines an extension of μ-calculus (the language to express equations containing fixpoint operators, in which it is possible to define the set of accepting states of a GR(1) game [BJP+12]) to multi-valued logic. This extension allows to define symbolic algorithms to solve these games, based on extensions of Binary Decision Diagrams (BDDs, structures used to represent Boolean functions efficiently, popular in model checking) to Algebraic Decision Diagrams (ADDs, structures that represent functions of Boolean variables with values in any finite set of constants [MPR16]).

Reactive synthesis has also been tackled through different models than the GR(1) games presented in Chapter 2. The work in [DBPU10] introduces the SGR(1) LTS control model. This model merges the computational advantages of GR(1) synthesis (that is, the existence of a polynomial-time synthesis algorithm) with a richer model to describe a reactive system, where

the variables describing its states (the so-called *fluents*) are disjoint from the alphabet of *actions* that determine transitions between states. The work in [DBPU14] extends SGR(1) synthesis to cases in which models are nondeterministic, that is, there is uncertainty on the behaviour of the environment and/or the implemented controller. The work in [DBK+14] proposes a multi-tier specification model, where different sets of guarantees is specified for different sets of assumptions. The controller dynamically adapts its goals to the assumptions satisfied by the environment at runtime, allowing for graceful degradation of its goals. In a similar setting where fluents and action symbols are distinguished, the work in [TBN20] proposes an approach to identifying security vulnerabilities by weakening assumptions on the behaviour of the user and enforcing the synthesis of a controller that still satisfies the security requirements under the weakened assumptions.

Given these variations on the original synthesis algorithm, the resulting controllers may be different, and quality criteria to compare them are needed. The work in [FMR20] presents a set of heuristic to reduce the running time of a synthesis algorithm and potentially reduce it. These heuristics include: *(i)* modifying the fixpoint algorithm for synthesis, described in [BJP+12], to skip iterations that would repeat computations already observed in previous iterations; *(ii)* speeding up the computation of the number of unrealizable cores (see Section 2.2.4) in the specification, by reusing partial results of the multiple realizability checks involved in unrealizable core identification if they are repeated across the checks; *(iii)* replacing the GR(1) game with a Rabin(1) game, that can be faster to solve.

In all this work, the approaches require realizable specifications. We see our contribution as complementary, to ensure the existence of such specifications.

8.4 Specification Mining

A common step of all works in assumptions refinement is the use of a bias heuristic that extracts temporal properties from behavioral models (see Section 3.2). This is a common practice in the field of specification mining [GS08, YEB+06, LPB15, NG18, GNMR16]. In this work, the input

is a set of execution traces of a software system and one or more templates of formulae. The goal is to identify a set of formulae that are satisfied by the traces and describe the behavior of the entire system concisely, for verification and debugging purposes. The patterns employed in these works are the ones defined by Dwyer et al. in [DAC99].

The difference to our setting is in the application context and the kind of information used. In our context, assumptions refinement is done before any implementation of the controller is available. Specification mining, instead, refers to existing systems, and the goal is finding specifications of their behaviour by observing it, e.g. legacy systems. While GR(1) assumptions refinement is based on some form of CEGIS approach (see Section 3.2), using negative examples only, specification mining uses both positive and negative examples for inducing good specifications.

8.5 Counterstrategy Alternative: Justice Violations Transition Systems

Recently Justice Violations Transition Systems (JVTS) [KMR17] have been proposed as alternatives to counterstrategies for debugging unrealizable specifications. The authors show that these structures are more concise and their computation is more time-efficient than counterstrategies, while retaining the same expressive power. JVTSes merge the looping states and the non-branching sequences of states of a counterstrategy into single states. The recent work in [MRS19] proposes to replace counterstrategies with JVTSes in the counterstrategy-guided loop of Chapter 3. In our work, we focus on counterstrategies in order to provide a fairer comparison with the state of the art.

The heuristics of weakness (Chapter 6) and minimality (Chapter 7) can be seamlessly employed in a search algorithm based on JVTSes rather than counterstrategies. Indeed, computing the weakness of a refinement Ψ is independent of whether the refinement is generated from counterstrategies or JVTSes. Likewise, the notion of minimality with respect to a set of counter-

strategies does not depend on the internal structure of counterstrategies. The only assumption is that a counterstrategy is an "object" for which one can tell whether or not it satisfies an assumption, or equivalently, an object for which a model checking algorithm is defined. This property holds for JVTSes too.

The interpolation-based bias, instead, relies heavily on the structure of counterstrategies (see Section 4.3). Replacing counterstrategies with JVTSes would require modifications in the translation scheme of our approach (see Section 4.4.2), and in the templates used in the baseline multivarbias approach (see Section 4.8). Investigation on how interpolation can be integrated with JVTSes is matter for future work.

8.6 Weakening Guarantees

Whilst our work, and those of [LDS11, AMT13], seek to strengthen assumptions, the work in [DBK+14] takes a complementary perspective, that of weakening guarantees. In [DBK+14], the authors propose a multi-tier control framework that assumes a set of ranked environment models and their associated guarantees (from weak to strong) to be given by a user. Each pair is assumed to form a realizable specification and defines an associated controller synthesis problem to be solved. Transition between the tiers is triggered by the detection of inconsistency between the environment model of an activated tier and the environment behaviour exhibited at runtime. Contrary to their work, we do not assume a realizable specification is given and instead try to compute environment specifications automatically that make the specification realizable. Furthermore, our work produces alternative environment assumptions for guaranteeing realisability whereas their work assumes one model per tier.

8.7 LTL Specification Revision

GR(1) assumptions refinement approaches can be seen an instance of the general problem of LTL specification revision. This has applications in many contexts, like robot motion planning,

e.g., [Fai11, KFS15], and requirements engineering, e.g. [AvLK+16]. In [Fai11], an approach is proposed for revising specifications that are violated by a given system model represented as Büchi automata. In this work a lattice is constructed over the class of semantically equivalent LTL formulae and traversed to find weaker specifications that are satisfied in the given model. Our work takes on a different perspective in several respects. First of all, while [Fai11] focuses on weakening violated properties, we focus on strengthening initial specifications of environments. Moreover, we do not assume we are given a model to check against the initial formula; our work aims at achieving realizability, that is the existence of at least an automaton that satisfies the given GR(1) property. Thirdly, the model employed in [Fai11] is monolithic: the Büchi automaton provided in input defines the allowed behavior in terms of the entire set of variables; in our context, strategies and counterstrategies model a component that has control over a subset of the variables, receiving the complement subset as an input; this is in accordance with the assumption we make of having two agents, the environment and the controller, interacting with each other adversarially. Similar observations can be made when comparing our work with [KFS15]: in this paper, automata specifications are employed instead of LTL specifications, and the revision process works as a search in the automata space.

In [AvLK+16], the authors propose a method for revising goal specifications when unanticipated environment conditions (called obstacles) are detected. Their approach uses a logic-based learning technique for modifying a class of specifications expressed in Metric-LTL for a given a set of traces in which the specifications are violated. The modification operations include deletion and addition of Boolean expressions within a temporal formulae, as well as the addition and deletion of temporal formulae. The approach however does not guarantee that any revision produced is a refinement of the existing specification nor realizable. The approach instead focuses on ensuring that the specification is consistent with observed violation traces.

8.8 Oracle-Guided Inductive Synthesis

Countstrategy-guided assumptions refinement may be viewed as a particular instantiation of Oracle-Guided Inductive Synthesis (OGIS) [Ses12, JS17]. OGIS describes a class of solutions consisting of an iterative interaction between a learner and an oracle: the learner tries to find a solution concept by induction over a set of examples, and the oracle answers the learner's queries by providing meaningful examples for the learning process. In our context, the solution concept to learn is a refinement that makes a GR(1) specification realizable; the learner is the refinement inductive synthesis procedure (line 7 of Algorithm 3 described in Chapter 3); and the oracle is the realizability checking engine (invoked in line 6 of Algorithm 3). The learner queries the oracle about the correctness of the proposed refinement as a solution, and the oracle either answers yes if this is the case or provides an example in the form of a counterexample if not. Other instances of OGIS are counterexample-guided inductive synthesis of programs [JGST10, ABJ$^+$13, JS14], of abstraction refinement [Cla00, EKS06], and refining LTL requirements using model checking and inductive learning [AKRU13].

Chapter 9

Conclusions and Future Work

We finally summarize the main achievements of this thesis and outline some directions for future work.

9.1 Summary of Thesis Achievements

Giving formal specifications of a system behaviour is a crucial step in enabling formal verification and/or correct synthesis of the system. However, characterizing the assumptions underlying a specification is an error-prone task and if done incorrectly may lead to specifications that are unrealizable. Automated procedures for discovering assumptions are then needed to help identifying under which assumptions some specification can be realized.

In this thesis, we analyse formally the heuristics used for inferring assumptions from characterizations of undesired environment behaviours called counterstrategies, and introduce new heuristics in order to discover better assumptions. In Chapter 3, we categorize these heuristics into bias heuristics and sorting heuristics, and identify the limitations of existing ones. As shown in Chapter 3, the effectiveness of these heuristics in finding solutions depends largely on the parameters provided by the user, and therefore on their *a priori* knowledge of the candidate solutions. Moreover, these heuristics do not account for the quality of the final solution, expressed in terms of its weakness.

In Chapter 4, we introduced a novel bias based on Craig interpolation. It addresses the first limitation by automating some of the parameter choices typically done manually. We observed experimentally that interpolation often converges to a solution in shorter time than the baseline multivarbias heuristic, and discovers a bigger proportion of solutions over all the explored candidate refinements. We called this metric *effectiveness* of the search and proposed it as a novel quality criterion for assumptions refinement approaches.

In Chapters 5 and 6, we provide a quantitative definition of the weakness of GR(1) conjunctions based on the entropy and the Hausdorff dimension of their languages. In doing so, we provide extensions to the algorithm for computing the Hausdorff dimension of ω-languages (which the literature provides for Muller automata only) to the cases where the ω-languages are represented by Büchi automata and generalized Büchi automata. With respect to the ω-languages of GR(1) conjunctions, we discover some property of interest about the language of SCC-accepted suffixes of an ω-language. These suffixes are the portions of an ω-language that run inside a single accepting SCC of a Büchi automaton of the ω-language. In particular, we prove that if $L_1 \subseteq L_2$, then also the SCC-accepted suffixes of L_1 are a subset of the SCC-accepted suffixes of L_2. By leveraging this property, we are able to define a quantitative weakness measure for GR(1) conjunctions consistent with implication that distinguishes conjunctions related to each other via strict implication. We used this measure as a sorting heuristic to the effect of discovering weaker solutions than the ones discovered by a FIFO search.

Chapter 7 presents a second bias we propose which tackles weakness from a different perspective. We notice that typical refinement procedures that concatenate the discovered assumptions progressively only based on the latest counterstrategy produced may generate overly constrained refinements. We therefore define a new notion of redundancy of assumptions and minimality of refinements with respect to a set of counterstrategies in order to produce refinements containing just the assumptions needed to achieve realizability. Through minimality, we observe that the search for refinements focuses on refinements containing fewer assumptions than typical BFS approaches on refinement trees and produce shorter solutions. The drawback is that the number of solutions within a given time budget is smaller, and hybrid approaches searching both minimal and non-minimal refinements perform better.

9.2 Future Work

In this section, we sketch possible directions for future work.

Further heuristics. The heuristics we develop address effectively several of the quality parameters we listed in Section 3.3, some of them are not addressed. In particular, we explored the setting of CEGIS, in which no positive examples of environment behaviour are given, but only counterstrategies (see Section 3.2). Therefore, our heuristics do not account for the usefulness of a refinement (that is, its adherence to the real environment where the controller will operate [MRS19]). Current approaches that use positive examples add a post-processing phase to the counterstrategy-guided loop, where all the solutions are checked against the available positive examples, and removed if they are not satisfied. Future work needs to be directed into finding heuristics to include positive examples in the most effective way, in order to drive the search towards "useful" refinements (in the sense of Section 3.3) and avoid wasting computation into generating non-useful solutions.

In this thesis, we propose a sorting heuristic that selects refinements based on their weakness (Chapter 6), with the aim of identifying the weakest solutions to the refinement problem. However, this heuristic is in contrast with the efficiency of the search. Indeed, when the assumptions added to a refinement are weaker, there often need to be more assumptions in order to achieve realizability (Section 6.4.2). This delays the time to reach a solution and reduces the number of solutions found in a given budget of explored refinements (the effectiveness as defined in Section 3.3). A future line of research may be identifying sorting heuristics able to reduce provably the number of steps needed to reach some solution from a given refinement.

To this goal, a notion of *distance of a refinement from realizability* can be defined as a function of the number of iterations of the counterstrategy-guided loop needed to reach some solution in the refinement tree from the given refinement. In machine learning settings, this notion is known as the *teaching dimension* [Ang04]. Using directly the minimum number of steps to some solution has two drawbacks: *(i)* it is an *a-posteriori* measure that can be computed only after the search is completed, and therefore it cannot be used as a sorting heuristic; *(ii)* from

our preliminary experiments on the case studies of Section 3.5, several refinements have at least one solution as a direct child in the refinement tree. Therefore this definition of distance would yield 1 for almost every refinement and would not discriminate between them.

Another candidate for the distance from realizability is the number of unrealizable cores in a refined specification. Given that any assumption in a refinement must eliminate at least one unrealizable core in the original specification [CA17], it is reasonable to assume that a refined specification with fewer unrealizable cores requires fewer additional assumptions in order to achieve realizability. However, the computation of this quantity is hindered by the exponential complexity in finding the number of cores [dlBSW03]. Future work can be directed to estimating this quantity, or finding efficient algorithms to compute this number.

A third candidate for the distance from realizability may be related to the structure of the counterstrategy. The intuition is that the more "complex" a counterstrategy is, the more environment behaviours can lead to the violation of a guarantee and therefore the more assumptions need to be added to the specification in order to reach realizability. In this case, a quantity like entropy or Hausdorff dimension could be effective for quantifying the complexity of a counterstrategy, with the caveat of adapting their definitions from Büchi automata to counterstrategy graphs.

Search with proactive restarts. The counterstrategy-guided loop keeps exploring refinements until the set of candidate refinements is empty or some timeout is reached. In some case studies (Chapter 4), none of the heuristics we study in this thesis is able to find any solution before the timeout. In order to overcome this limitation, techniques to reduce the expected time to reach the first solution can be applied. One of these is applicable in case a randomized heuristic (such as multivarbias) is used.

This technique consists in proactively restarting the search after a certain amount of refinements has been explored. This amount is determined by *Luby sequences* [LSZ93, OLLW18], sequences of budgets to assign to the search after each restart. It has been proven that if the search algorithm is a Las Vegas algorithm (that is, a randomized algorithm that is guaranteed to find

some solution after a random amount of time), then the expected time to reach a solution is sensibly reduced by applying restarts, independent of the distribution of the termination time. Experimental and theoretical studies can be conducted to investigate the effect of proactive restarts to counterstrategy-guided assumptions refinement.

Approximating entropy and Hausdorff dimension. In Chapter 5, we give exact algorithms to compute the entropy and Hausdorff dimension of an ω-language. These algorithms require deterministic ω-automata to be computed. Computing a deterministic automaton requires an exponential blowup with respect to the non-deterministic counterparts [KR10], and therefore the time required for computing entropy and Hausdorff dimension may be significantly reduced if they could be computed on non-deterministic ω-automata. It is not clear, however, how to do so, since in non-deterministic automata there is no one-to-one correspondence between runs on the automaton and ω-words in the language; hence, adjacency matrices give limited insight in the structure of the language. Further research should be directed to exploring the problem of computing (or approximating with controllable precision) the entropy and Hausdorff dimension of ω-languages using non-deterministic automata.

Bibliography

[AAK13] Shaull Almagor, Guy Avni, and Orna Kupferman. Automatic Generation of Quality Specifications. In *International Conference on Computer Aided Verification (CAV)*, pages 479–494, 2013.

[ABD14a] Eugene Asarin, Michel Blockelet, and Aldric Degorre. Entropy model checking. In *12th Workshop on Quantitative Aspects of Programming Languages (QAPL) - Joint with European Joint Conference On Theory and Practice of Software (ETAPS)*, 2014.

[ABD+14b] Eugene Asarin, Michel Blockelet, Aldric Degorre, Cătălin Dima, and Chunyan Mu. Asymptotic Behaviour in Temporal Logic. In *Joint Meeting of the Annual Conference on Computer Science Logic and the Symposium on Logic in Computer Science (CSL-LICS)*, pages 1–9, 2014.

[ABJ+13] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-Guided Synthesis. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–8, 2013.

[ADG16] Aws Albarghouthi, Isil Dillig, and Arie Gurfinkel. Maximal Specification Synthesis. *ACM SIGPLAN Notices*, 51(1):789–801, 2016.

[AKRU13] Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastian Uchitel. Elaborating Requirements Using Model Checking and Inductive Learning. *IEEE Transactions on Software Engineering*, 39(3):361–383, 2013.

[AL91] Martín Abadi and Leslie Lamport. The Existence of Refinement Mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Della Rocca, editors, *Automata, Languages and Programming*, pages 1–17, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[AMPR20] Gal Amram, Shahar Maoz, Or Pistiner, and Jan Oliver Ringert. Energy mu-Calculus: Symbolic Fixed-Point Algorithms for omega-Regular Energy Games. *arXiv preprint arXiv:2005.00641*, 2020.

[AMT13] Rajeev Alur, Salar Moarref, and Ufuk Topcu. Counter-Strategy Guided Refinement of GR(1) Temporal Logic Specifications. In *Formal Methods in Computer Aided Design (FMCAD)*, pages 26–33. IEEE, 2013.

[AMT15] Rajeev Alur, Salar Moarref, and Ufuk Topcu. Pattern-Based Refinement of Assume-Guarantee Specifications in Reactive Synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 501–516. Springer, 2015.

[Ang87] Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[Ang04] Dana Angluin. Queries Revisited. *Theoretical Computer Science*, 313(2):175–194, 2004.

[ASB19] Chetan Arora, Mehrdad Sabetzadeh, and Lionel C. Briand. An Empirical Study on the Potential Usefulness of Domain Models for Completeness Checking of Requirements. *Empirical Software Engineering*, 2019.

[AvLK+16] Dalal Alrajeh, Axel van Lamsweerde, Jeff Kramer, Alessandra Russo, and Sebastian Uchitel. Risk-Driven Revision of Requirements Models. In *International Conference on Software Engineering (ICSE)*, pages 855–865, 2016.

[BBDER01] Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient Detection of Vacuity in Temporal Model Checking. *Formal Methods in System Design*, 18(2):141–163, 2001.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded Model Checking. *Advances in Computers*, 58, 2003.

[BCF+08] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT Solver. In *Computer Aided Verification (CAV)*, pages 299–303. Springer Berlin Heidelberg, 2008.

[BCG+10] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. RATSY – A New Requirements Analysis Tool with Synthesis. In *Computer Aided Verification (CAV)*, pages 425–429. Springer Berlin Heidelberg, 2010.

[BCHJ09] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Better Quality in Synthesis through Quantitative Objectives. In *Computer Aided Verification (CAV)*, pages 140–156, 2009.

[BCJ+97] Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo Marrero. An Improved Algorithm for the Evaluation of Fixpoint Expressions. *Theoretical Computer Science*, 178(1-2):237–255, 1997.

[BDP+13] Victor Braberman, Nicolas D'Ippolito, Nir Piterman, Daniel Sykes, and Sebastian Uchitel. Controller Synthesis: From Modelling to Enactment. In *International Conference on Software Engineering (ICSE)*, pages 1347–1350. IEEE, 2013.

[BGJ+07] Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Specify, Compile, Run: Hardware from PSL. *Electronic Notes in Theoretical Computer Science*, 190(4):3–16, 2007.

[BGP03] Howard Barringer, Dimitra Giannakopoulou, and Corina S. Pasareanu. Proof Rules for Automated Compositional Verification through Learning. In *SAVCBS*

Workshop on Specification And Verification of Component-Based Systems at ESEC/FSE, 2003.

[BJP+12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'Ar. Synthesis of Reactive(1) Designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

[BKeS12] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. LTL to Büchi Automata Translation: Fast and More Deterministic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 612, pages 95–109. Springer Berlin Heidelberg, 2012.

[BMPS05] Cristina Bazgan, Jérôme Monnot, Vangelis Th Paschos, and Fabrice Serrière. On the Differential Approximation of MIN SET COVER. *Theoretical Computer Science*, 332(1-3):497–513, 2005.

[Bou13] Jean-Louis Boulanger. *Static Analysis of Software: The Abstract Interpretation*. Wiley, 2013.

[BP94] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Society for Industrial and Applied Mathematics, 1994.

[Büc66] Julius Richard Büchi. Symposium on Decision Problems: On a Decision Method in Restricted Second Order Arithmetic. *Studies in Logic and the Foundations of Mathematics*, 44:1–11, 1966.

[CA17] Davide G. Cavezza and Dalal Alrajeh. Interpolation-Based GR(1) Assumptions Refinement. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 281–297. Springer Berlin Heidelberg, 2017.

[CA18] Davide G. Cavezza and Dalal Alrajeh. Interpolation-based GR(1) assumptions refinement. *CoRR*, abs/1611.07803, 2018.

[CAG18] Davide G. Cavezza, Dalal Alrajeh, and András György. A Weakness Measure for GR(1) Formulae. In *International Symposium on Formal Methods (FM)*, pages 110–128. 2018.

[CAG20a] Davide G. Cavezza, Dalal Alrajeh, and András György. A Weakness Measure for GR(1) Formulae. 2020. To appear in Formal Aspects of Computing.

[CAG20b] Davide G. Cavezza, Dalal Alrajeh, and András György. Minimal Assumptions Refinement for Realizable Specifications. In *International Conference on Formal Methods in Software Engineering (FormaliSE)*, page 66–76. ACM, 2020.

[Cav16] Davide G. Cavezza. Interpolation-Based GR(1) Assumptions Refinement, 2016. Master of Research thesis.

[CdAHS03] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource Interfaces. In *Embedded Software*, pages 117–133, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[CDF+06] Krishnendu Chatterjee, Luca De Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. Compositional Quantitative Reasoning. In *International Conference on the Quantitative Evaluation of Systems, QEST 2006*, pages 179–188, 2006.

[CGH94] Edmund Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. Another Look at LTL Model Checking. In *International Conference on Computer Aided Verification (CAV)*, volume 71, pages 415–427, 1994.

[CGP99] Edmund M Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, London, Cambridge, 1999.

[CGP03] Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning Assumptions for Compositional Verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 331–346. Springer Berlin Heidelberg, 2003.

[CGS08] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 397–412, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[CHJ08] Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment Assumptions for Synthesis. In *International Conference on Concurrency Theory (CONCUR)*, pages 147–161. Springer Berlin Heidelberg, 2008.

[Chu57] Alonzo Church. Application of Recursive Arithmetic to the Problem of Circuit Synthesis. *Summaries of talks presented at the Summer Institute of Symbolic Logic*, 1:3–50, 1957.

[Chu62] Alonzo Church. Logic, Arithmetic and Automata. *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.

[CJS09] Cristian S. Calude, Helmut Jürgensen, and Ludwig Staiger. Topology on Words. *Theoretical Computer Science*, 410(24-25):2323–2335, 2009.

[Cla00] Edmund Clarke. Counterexample-Guided Abstraction Refinement. In *10th International Symposium on Temporal Representation and Reasoning and Fourth International Conference on Temporal Logic*, number iii, pages 7–8. IEEE Comput. Soc, 2000.

[Cra57] William Craig. Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. *The Journal of Symbolic Logic*, 22(03):269–285, sep 1957.

[CRST08] Alessandro Cimatti, Marco Roveri, Viktor Schuppan, and Andrei Tchaltsev. Diagnostic Information for Realizability. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, Lecture Notes in Computer Science, pages 52–67, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[CS12] Koen Claessen and Niklas Sorensson. A Liveness Checking Algorithm that Counts. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 52–59. IEEE, 2012.

[CS13] Cristian Cadar and Koushik Sen. Symbolic Execution. In *Runtime Verification*, pages 2–2. Springer Berlin Heidelberg, 2013.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, New York, NY, 2nd edition, 2006.

[DAAU14] Renzo Degiovanni, Dalal Alrajeh, Nazareno Aguirre, and Sebastian Uchitel. Automated Goal Operationalisation Based on Interpolation and SAT Solving. In *International Conference on Software Engineering (ICSE)*, pages 129–139. ACM Press, 2014.

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *International Conference on Software Engineering (ICSE)*, pages 411–420, 1999.

[DBK+14] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. Hope for the Best, Prepare for the Worst: Multi-tier Control for Adaptive Systems. *International Conference on Software Engineering (ICSE)*, pages 688–699, 2014.

[DBPU10] Nicolás D'Ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesis of Live Behaviour Models. In *International Symposium on Foundations of software engineering (FSE)*, page 77. ACM Press, 2010.

[DBPU13] Nicolás D'Ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesizing Nonanomalous Event-Based Controllers for Liveness Goals. *ACM Transactions on Software Engineering and Methodology*, 22(1):1–36, 2013.

[DBPU14] Nicolas D'Ippolito, Victor Braberman, Nir Piterman, and Sebastian Uchitel. Controllability in Partial and Uncertain Environments. In *International Conference on Application of Concurrency to System Design*, pages 52–61. IEEE, 2014.

[DCA+18] Renzo Degiovanni, Pablo Castro, Marcelo Arroyo, Marcelo Ruiz, Nazareno Aguirre, and Marcelo Frias. Goal-Conflict Likelihood Assessment Based on Model Counting. In *International Conference on Software Engineering (ICSE)*, pages 1125–1135. ACM Press, 2018.

[DKW08] Vijay D'Silva, Daniel Kroening, and Georg Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, jul 2008.

[dlBSW03] Maria Garcia de la Banda, Peter J Stuckey, and Jeremy Wazny. Finding all Minimal Unsatisfiable Subsets. In *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declarative programming (PPDP)*, pages 32–43. ACM Press, 2003.

[DLLF+16] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Étienne Renault, and Laurent Xu. Spot 2.0 — A Framework for LTL and ω-Automata Manipulation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9938 LNCS:122–129, 2016.

[DLvL06] Christophe Damas, Bernard Lambeau, and Axel van Lamsweerde. Scenarios, Goals, and State Machines: A Win-win Partnership for Model Synthesis. *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 197–207, 2006.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, page 337–340. Springer-Verlag, 2008.

[DW90] Frank Dederichs and Rainer Weber. Safety and Liveness from a Methodological Point of View. *Information Processing Letters*, 36(1):25–30, oct 1990.

[EKS06] Javier Esparza, Stefan Kiefer, and Stefan Schwoon. Abstraction Refinement with Craig Interpolation and Symbolic Pushdown Systems. In *International Conference*

on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pages 489–503. Springer Berlin Heidelberg, 2006.

[EKS16] Javier Esparza, Jan Křetínský, and Salomon Sickert. From LTL to Deterministic Automata. *Formal Methods in System Design*, 49(3):219–271, dec 2016.

[ELTV17] Rüdiger Ehlers, Stéphane Lafortune, Stavros Tripakis, and Moshe Y. Vardi. Supervisory Control and Reactive Synthesis: a Comparative Introduction. *Discrete Event Dynamic Systems: Theory and Applications*, 27(2):209–260, 2017.

[ET14] Rüdiger Ehlers and Ufuk Topcu. Resilience to Intermittent Assumption Violations in Reactive Synthesis. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, number 1, pages 203–212. ACM Press, 2014.

[Fai11] Georgios E. Fainekos. Revising Temporal Logic Specifications for Motion Planning. *International Conference on Robotics and Automation (ICRA)*, pages 40–45, 2011.

[Fal04] Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley & Sons, 2004.

[Fin16] Bernd Finkbeiner. Synthesis of Reactive Systems. *Dependable Software Systems Engineering*, 45:72–98, 2016.

[FMH16] Ioannis Filippidis, Richard M. Murray, and Gerard J. Holzmann. A Multi-Paradigm Language for Reactive Synthesis. *Electronic Proceedings in Theoretical Computer Science*, 202:73–97, 2016.

[FMR20] Elizabeth Firman, Shahar Maoz, and Jan Oliver Ringert. Performance Heuristics for GR(1) Synthesis and Related Algorithms. *Acta Informatica*, 57(1):37–79, 2020.

[GF93] Georg Gottlob and Christian G. Fermüller. Removing Redundancy from a Clause. *Artificial Intelligence*, 61(2):263–289, jun 1993.

[GHN+04] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast Decision Procedures. In *Computer Aided Verification (CAV)*, pages 175–188, 2004.

[GKW+15] Andrew Gacek, Andreas Katis, Michael W. Whalen, John Backes, and Darren Cofer. Towards Realizability Checking of Contracts Using Theories. In *NASA Formal Methods Symposium*, volume 9058, pages 173–187. Springer, Cham, 2015.

[GL02] Dimitra Giannakopoulou and Flavio Lerda. From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata. *22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Sytems (FORTE)*, pages 308–326, 2002.

[GM03] Dimitra Giannakopoulou and Jeff Magee. Fluent Model Checking for Event-Based Systems. *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE)*, 28(5):257, 2003.

[GNMR16] Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth. Learning Invariants Using Decision Trees and Implication Counterexamples. *ACM SIGPLAN Notices*, 51(1):499–512, 2016.

[GPB02] Dimitra Giannakopoulou, Corina S. Păsăreanu, and Howard Barringer. Assumption Generation for Software Component Verification. In *International Conference on Automated Software Engineering (ASE)*, number i, pages 3–12, 2002.

[GPB05] Dimitra Giannakopoulou, Corina S. Păsăreanu, and Howard Barringer. Component Verification with Automatically Generated Assumptions. *International Conference in Automated Software Engineering (ASE)*, 2005.

[GPG08] Mihaela Gheorghiu Bobaru, Corina S. Păsăreanu, and Dimitra Giannakopoulou. Automated Assume-Guarantee Reasoning by Abstraction Refinement. In *Computer Aided Verification (CAV)*, volume 2, pages 135–148. Springer Berlin Heidelberg, 2008.

[GS08] Mark Gabel and Zhendong Su. Symbolic Mining of Temporal Specifications. In *International Conference on Software Engineering (ICSE)*, pages 51–60, 2008.

[GSN+16] Shromona Ghosh, Dorsa Sadigh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donzé, Alberto L. Sangiovanni-Vincentelli, S. Shankar Sastry, and Sanjit A. Seshia. Diagnosis and Repair for Synthesis from Signal Temporal Logic Specifications. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 31–40. ACM Press, 2016.

[GST17] Ofer Guthmann, Ofer Strichman, and Anna Trostanetski. Minimal Unsatisfiable Core Extraction for SMT. *Formal Methods in Computer-Aided Design (FMCAD)*, pages 57–64, 2017.

[GtC09] Amélie Gheerbrant and Balder ten Cate. Craig Interpolation for Linear Temporal Languages. In *Computer Science Logic - 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings*, pages 287–301. Springer Berlin Heidelberg, 2009.

[GTW02] Erich Gradel, Wolfgang Thomas, and Thomas Wilke. *Automata Logics, and Infinite Games*, volume 2500. Springer Berlin Heidelberg, 2002.

[Hen10] Thomas Henzinger. From Boolean to Quantitative Notions of Correctness. *ACM SIGPLAN Notices*, 45(1):157, 2010.

[HJ86] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1986.

[HJ94] Hans Hansson and Bengt Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, sep 1994.

[HJ12] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. 2012.

[HO13] Thomas A. Henzinger and Jan Otop. From Model Checking to Model Measuring. In *International Conference on Concurrency Theory (CONCUR)*, pages 273–287. 2013.

[Hua95] Guoxiang Huang. Constructing Craig Interpolation Formulas. In *International Computing and Combinatorics Conference*, pages 181—-190, 1995.

[JGST10] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. Oracle-Guided Component-Based Program Synthesis. In *International Conference on Software Engineering (ICSE)*, volume 1, page 215. ACM Press, 2010.

[JS14] Susmit Jha and Sanjit A. Seshia. Are There Good Mistakes? A Theoretical Analysis of CEGIS. *Electronic Proceedings in Theoretical Computer Science*, 157(Synt):84–99, jul 2014.

[JS17] Susmit Jha and Sanjit A. Seshia. A Theory of Formal Synthesis via Inductive Learning. *Acta Informatica*, 54(7):693–726, 2017.

[JZ93] Michael Jackson and Pamela1 Zave. Domain Descriptions. *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 56–64, 1993.

[Kam68] Hans Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Ucla, 1968.

[Kam15] Norihiro Kamide. Interpolation Theorems for Some Variants of LTL. *Reports on Mathematical Logic*, 50:3–30, 2015.

[KE12] Jan Kretínský and Javier Esparza. Deterministic Automata for the (F, G)-fragment of LTL. In *Computer Aided Verification (CAV)*, pages 7–22, 2012.

[KFG+18] Andreas Katis, Grigory Fedyukovich, Huajun Guo, Andrew Gacek, John Backes, Arie Gurfinkel, and Michael W. Whalen. Validity-Guided Synthesis of Reactive Systems from Assume-Guarantee Contracts. In W. Rance Cleaveland, editor, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 176–193. Springer Berlin Heidelberg, 2018.

[KFS15] Kangjin Kim, Georgios E. Fainekos, and Sriram Sankaranarayanan. On the Minimal Revision Problem of Specification Automata. *International Journal of Robotics Research*, 34(12):1515–1535, 2015.

[KGFP09] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[KHB09] Robert Konighofer, Georg Hofferek, and Roderick Bloem. Debugging Formal Specifications Using Simple Counterstrategies. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 152–159. IEEE, 2009.

[KMR17] Aviv Kuvent, Shahar Maoz, and Jan Oliver Ringert. A Symbolic Justice Violations Transition System for Unrealizable GR(1) Specifications. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, number 1, pages 362–372. ACM Press, 2017.

[Kon10] Savas Konur. A Survey on Temporal Logics. 2010.

[KR10] Orna Kupferman and Adin Rosenberg. The Blow-Up in Translating LTL to Deterministic Automata. In *International Workshop on Model Checking and Artificial Intelligence (MoChArt)*, pages 85–94, 2010.

[Kra97] Jan Krajíček. Interpolation Theorems, Lower Bounds for Proof Systems, and Independence Results for Bounded Arithmetic. *The Journal of Symbolic Logic*, 62(02):457–486, 1997.

[Kwi07] Marta Kwiatkowska. Quantitative Verification: Models, Techniques and Tools. In *Joint meeting of the European Software Engineering Conference and the Symposium on The Foundations of Software Engineering (ESEC/FSE)*, page 449. ACM Press, 2007.

[Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[LDS11] Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. Mining Assumptions for Synthesis. In *International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 43–50, 2011.

[Li14] Wenchao Li. *Specification Mining : New Formalisms , Algorithms and Applications*. PhD thesis, 2014.

[Lib05] Paolo Liberatore. Redundancy in Logic I: CNF Propositional Formulae. *Artificial Intelligence*, 163(2):203–232, 2005.

[LKMU08] Emmanuel Letier, Jeff Kramer, Jeff Magee, and Sebastian Uchitel. Deriving Event-Based Transition Systems from Goal-Oriented Requirements Models. *International Conference on Automated Software Engineering (ASE)*, 15(2):175–206, 2008.

[LM95] Douglas Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, nov 1995.

[LMN16] Christof Löding, P. Madhusudan, and Daniel Neider. Abstract Learning Frameworks for Synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9636, pages 167–185. Springer Berlin Heidelberg, 2016.

[LPB15] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL Specification Mining. In *International Conference on Automated Software Engineering (ASE)*, number 1, pages 81–92. IEEE, nov 2015.

[LSSS14] Wenchao Li, Dorsa Sadigh, S. Shankar Sastry, and Sanjit A. Seshia. Synthesis for Human-in-the-Loop Control Systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 8413 LNCS, pages 470–484. Springer Berlin Heidelberg, 2014.

[LSWW10] Alessio Lomuscio, Ben Strulo, Nigel Walker, and Peng Wu. Assume-Guarantee Reasoning with Local Specifications. In *International Conference on Formal Engineering Methods*, pages 204–219, 2010.

[LSZ93] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal Speedup of Las Vegas Algorithms. *Information Processing Letters*, 47(4):173–180, sep 1993.

[Lut14] Alexandre Duret Lutz. LTL Translation Improvements in Spot 1.0. *International Journal of Critical Computer-Based Systems*, 5(1/2):31, 2014.

[LV02] Emmanuel Letier and Axel Van Lamsweerde. Deriving Operational Software Specifications from System Goals. *ACM SIGSOFT Software Engineering Notes*, 27(6):119–128, 2002.

[LZP+13] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. LTL Satisfiability Checking Revisited. *Proceedings of the International Workshop on Temporal Representation and Reasoning*, pages 91–98, 2013.

[McM03] Kenneth L. McMillan. Interpolation and SAT-Based Model Checking. In *Computer Aided Verification (CAV)*, volume 2742, pages 1–13, 2003.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition, 1997.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Number 1977. Springer New York, New York, NY, 1992.

[MPR16] Shahar Maoz, Or Pistiner, and Jan Oliver Ringert. Symbolic BDD and ADD Algorithms for Energy Games. *arXiv preprint arXiv:1611.07622*, 2016.

[MR15] Shahar Maoz and Jan Oliver Ringert. GR(1) Synthesis for LTL Specification Patterns. In *European Software Engineering Conference held jointly with International Symposium on Foundations of Software Engineering (ESEC/FSE)*, number 1, pages 96–106. ACM Press, 2015.

[MR18] Shahar Maoz and Jan Oliver Ringert. On the Software Engineering Challenges of Applying Reactive Synthesis to Robotics. In *International Workshop on Robotics Software Engineering (RoSE)*, volume 82, pages 17–22. ACM Press, 2018.

[MRS19] Shahar Maoz, Jan Oliver Ringert, and Rafi Shalom. Symbolic Repairs for GR(1) Specifications. In *International Conference on Software Engineering (ICSE)*, pages 1016–1026. IEEE, 2019.

[MS94] Wolfgang Merzenich and Ludwig Staiger. Fractals, Dimension, and Formal Languages. *Informatique théorique et applications*, 28(3-4):361–386, 1994.

[MS20] Shahar Maoz and Ilia Shevrin. Just-In-Time Reactive Synthesis. 2020. To appear in the Proceedings of ASE 2020.

[Mul63] David E. Muller. Infinite Sequences and Finite Machines. *Fourth Annual Symposium on Switching Circuit Theory and Logical Design (SWCT)*, pages 3–16, 1963.

[NA06] Wonhong Nam and Rajeev Alur. Learning-Based Symbolic Assume-Guarantee Reasoning with Automatic Decomposition. *Automated Technology for Verification and Analysis (ATVA)*, pages 170–185, 2006.

[NG18] Daniel Neider and Ivan Gavran. Learning Linear Temporal Properties. In *Formal Methods in Computer Aided Design (FMCAD)*, pages 1–10. IEEE, oct 2018.

[NKWF14] ThanhVu Nguyen, Deepak Kapur, Westley Weimer, and Stephanie Forrest. Using Dynamic Analysis to Generate Disjunctive Invariants. pages 608–619, 2014.

[NMM+19] Alexandr Naumchev, Bertrand Meyer, Manuel Mazzara, Florian Galinier, Jean Michel Bruel, and Sophie Ebersold. AutoReq: Expressing and verifying requirements for control systems. *Journal of Computer Languages*, 51(September 2018):131–142, 2019.

[NWZ20] Daniel Neider, Alexander Weinert, and Martin Zimmermann. Synthesizing Optimally Resilient Controllers. *Acta Informatica*, 57(1-2):195–221, 2020.

[OLLW18] Laurent Orseau, Tor Lattimore, Levi H.S. Lelis, and Théophane Weber. Single-Agent Policy Tree Search with Guarantees. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 2018-Decem, pages 3201–3211, 2018.

[PDH99] Corina Păsăreanu, Matthew Dwyer, and Michael Huth. Assume-Guarantee Model Checking of Software: A Comparative Case Study. *Theoretical and Practical Aspects of SPIN Model Checking*, pages 168–183, 1999.

[PLG13] Fabio Patrizi, Nir Lipovetzky, and Hector Geffner. Fair LTL Synthesis for Non-Deterministic Systems Using Strong Cyclic Planners. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2343–2349, 2013.

[PMAB19] Guillaume Plassan, Katell Morin-Allory, and Dominique Borrione. Mining Missing Assumptions from Counter-Examples. *ACM Transactions on Embedded Computing Systems*, 18(1):1–25, 2019.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, volume 1977-Octob, pages 46–57. IEEE, 1977.

[PP06] Nir Piterman and Amir Pnueli. Faster Solutions of Rabin and Streett Games. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 275–284. IEEE, 2006.

[PPS05] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 364–380, 2005.

[PR89] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL)*, pages 179–190. ACM Press, 1989.

[Pri62] Arthur N. Prior. Tense-Logic and the Continuity of Time. *Studia Logica*, 13:133–151, 1962.

[Pud97] Pavel Pudlák. Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997.

[RDLKP13] Etienne Renault, Alexandre Duret-Lutz, Fabrice Kordon, and Denis Poitrenaud. Three SCC-Based Emptiness Checks for Generalized Büchi Automata. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, pages 668–682, 2013.

[Res20] Grand View Research. Microcontroller market size, share and trends analysis report by product (8-bit, 16-bit, 32-bit), by application (automotive, consumer electronics, industrial, medical devices, military and defense), and segment forecasts, 2020 - 2027. https://www.grandviewresearch.com/industry-analysis/microcontroller-market, 2020. Accessed: June 2020.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Buchi Automata from LTL Formulae. *Computer Aided Verification (CAV)*, (1855):1–17, 2000.

[Sen06] Eugene Seneta. *Non-negative Matrices and Markov Chains*. Springer, 2006.

[Ses12] Sanjit A. Seshia. Sciduction: Combining Induction, Deduction, and Structure for Verification and Synthesis. In *Design Automation Conference (DAC)*, volume 103, page 356. ACM Press, 2012.

[Ses15] Sanjit A. Seshia. Combining Induction, Deduction, and Structure for Verification and Synthesis. *Proceedings of the IEEE*, 103(11):2036–2051, nov 2015.

[SL08] Armando Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, University of California, Berkeley, 2008.

[SL13] Armando Solar-Lezama. Program Sketching. *International Journal on Software Tools for Technology Transfer*, 15(5-6):475–495, oct 2013.

[SS13] Saqib Sohail and Fabio Somenzi. Safety First: A Two-Stage Algorithm for the Synthesis of Reactive Systems. *International Journal on Software Tools for Technology Transfer*, 15(5-6):433–454, oct 2013.

[SSR08] Saqib Sohail, Fabio Somenzi, and Kavita Ravi. A Hybrid Algorithm for LTL Games. In *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 309–323, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Sta98] Ludwig Staiger. The Hausdorff Measure of Regular ω-languages is Computable. Technical Report August, Martin-Luther-Universitat, 1998.

[Sta15] Ludwig Staiger. On the Hausdorff Measure of Regular omega-Languages in Cantor Space. Technical Report 1, 2015.

[SYN] The reactive synthesis competition. http://www.syntcomp.org/. Accessed: June 2020.

[Tar72] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[TBN20] Thein T. Tun, Amel Bennaceur, and Bashar Nuseibeh. OASIS: Weakening User Obligations for Security-critical Systems. 2020. To appear in the Proceedings of RE 2020.

[TDDO17] Cong Tian, Zhao Duan, Zhenhua Duan, and C. H.Luke Ong. More Effective Interpolations in Software Model Checking. In *International Conference on Automated Software Engineering (ASE)*, pages 183–193, 2017.

[Var95] Moshe Y. Vardi. On the Complexity of Modular Model Checking. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 101–111. IEEE, 1995.

[Var96] Moshe Y. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic. *Logics for concurrency*, pages 238 – 266, 1996.

[Vaz03] Vijay V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[VL00] Axel Van Lamsweerde and Emmanuel Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about Infinite Computations. *Information and Computation*, 115(1):1–37, nov 1994.

[VW98] Axel Van Lamsweerde and Laurent Willemet. Inferring Declarative Requirements Specifications from Operational Scenarios. *IEEE Transactions on Software Engineering*, 24(12):1089–1114, 1998.

[YEB+06] Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. Perracotta: Mining Temporal API Rules from Imperfect Traces. In *International Conference on Software engineering (ICSE)*, page 282. ACM Press, 2006.

[YP05] Michal Young and Mauro Pezze. *Software Testing and Analysis: Process, Principles and Techniques*. Wiley, 2005.

[Zel99] Andreas Zeller. Yesterday, my program worked. Today, it does not. Why? *ACM SIGSOFT Software Engineering Notes*, 24(6):253–267, nov 1999.