

Imperial College of Science, Technology and Medicine
Department of Computing

**Preferential Description Logics:
Reasoning in the presence of inconsistencies**

Graham Deane

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, June 2016

Abstract

The Web Ontology Languages define a rich machine readable language for knowledge representation on the world wide web. The current generation are based on Description Logics, a family of decidable logics that offer low complexity of reasoning. Due to the principle of explosion, from a contradiction anything follows, inconsistencies prevent meaningful inference from classical reasoners. However, locating and repairing inconsistencies can be resource intensive.

This thesis presents an *inconsistency tolerant* semantics for the Description Logic $\mathcal{ALC}$ called *Preferential $\mathcal{ALC}$* ($p\text{-}\mathcal{ALC}$). A $p\text{-}\mathcal{ALC}$ knowledge base is comprised of *defeasible* and *non-defeasible* axioms. The defeasible ABox and TBox are labelled with *weights* that reflect a relative measure of confidence in each axiom and provide a mechanism to “arbitrate” between inconsistencies. Entailment is defined through the notion of *preferred interpretations* which minimise the total weight of the unsatisfied axioms. We introduce a tableau algorithm for $p\text{-}\mathcal{ALC}$ in which the open branches correspond to preferred interpretations. We prove that the algorithm is sound, complete and terminates for any input knowledge base and show how it can be used to compute $p\text{-}\mathcal{ALC}$ entailment by refutation. The proposed $p\text{-}\mathcal{ALC}$ differs from existing semantics that obtain inferences from inconsistent knowledge bases designed for classical reasoners. For instance: the *para-consistent* and the *repair* semantics, lack a mechanism for “arbitration” of inconsistency; and the mechanism included in *possibilistic logic* results in a logic with a weak consequence relation.

We present an implementation of the algorithm using answer set programming (ASP) that is solved incrementally and exploits the *optimisation* of answer sets to identify preferred interpretations of $p\text{-}\mathcal{ALC}$. The well defined semantics of ASP enabled us to prove that the implementation is correct. The implementation is also verified empirically and the performance is evaluated against a set of synthetically generated inconsistent knowledge bases in which inconsistency is introduced.

Statement of Originality

I, Graham Deane, declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

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

Acknowledgements

I would like to express my deepest gratitude to my co-supervisors Dr Krysia Broda and Professor Alessandra Russo who have provided their continuous expert guidance, encouragement and support during the work presented in, and the writing of, this thesis.

I would also like to thank my PhD examiners Professor Marek Sergot (internal) and Dr Antonis Bikakis (external) from University of London for their very constructive feedback and detailed comments during the finalisation of this thesis.

I am most grateful to all the staff and the researchers in the Department of Computing at Imperial College for their advice and support during my time in the department.

I would like to thank Nicolas Matentzoglou for his assistance in searching for knowledge bases within the corpora at Manchester and Kody Moodley for sharing his views on the generation of synthetic knowledge bases.

The work presented in this thesis was funded by the Engineering and Physical Sciences Research Council.

Contents

Acknowledgements	v
1 Introduction	1
1.1 Summary of Contributions	6
1.2 Thesis Overview	7
2 An introduction to Description Logics	9
2.1 The Description Logic $\mathcal{ALC}$	11
2.1.1 Syntax	11
2.1.2 Semantics	14
2.1.3 Reasoning tasks	16
2.2 The extended family of Description Logics	17
2.2.1 The $\mathcal{EL}$ family	21
2.2.2 The <i>DL-Lite</i> family	22
2.3 Complexity	24
2.3.1 Complexity of Description Logic reasoning tasks	27
2.4 Inconsistency in Description Logic	29

2.4.1	A survey of existing approaches for handling inconsistencies	30
2.5	Summary	36
3	Preferential $\mathcal{ALC}$	39
3.1	Syntax	40
3.2	Semantics	42
3.3	Properties of $p\text{-}\mathcal{ALC}$	48
3.3.1	Existence of preferred interpretations	48
3.3.2	Refutation-style proofs can be used to show $p\text{-}\mathcal{ALC}$ entailment	53
3.4	Reasoning in the presence of inconsistencies	55
3.5	Summary	59
4	A Tableau Algorithm for $p\text{-}\mathcal{ALC}$	61
4.1	Background	61
4.2	The tableau algorithm for $p\text{-}\mathcal{ALC}$	72
4.3	General properties of the $p\text{-}\mathcal{ALC}$ tableau algorithm	82
4.3.1	Static blocking	82
4.3.2	Termination	86
4.3.3	All branches close	88
4.3.4	Saturation	88
4.4	Soundness	94
4.5	Completeness	98

4.6	Correctness	106
4.7	Summary	108
5	Implementation	109
5.1	Background	110
5.1.1	Syntax and Semantics	110
5.1.2	Reasoning tasks	118
5.1.3	Features specific to clingo	118
5.1.4	Properties of Answer Set Programs	121
5.2	Encoding the $p\text{-}\mathcal{ALC}$ tableau algorithm in ASP	127
5.2.1	The representation of a $p\text{-}\mathcal{ALC}$ knowledge base	128
5.2.2	Encoding the expansion rules	131
5.3	Properties of the translation	138
5.4	Complexity	164
5.5	Summary	165
6	Evaluation	166
6.1	Introduction and Methodology	166
6.2	Verification of correctness	168
6.3	Knowledge bases for performance evaluation	171
6.3.1	Inconsistent knowledge bases	172
6.3.2	Corpora of knowledge bases	173

6.4	Synthetic knowledge bases	174
6.4.1	Generating a consistent knowledge base	175
6.4.2	Generating an inconsistent knowledge base	182
6.4.3	Generating queries	186
6.5	Experiments	188
6.5.1	Experiment 1: Defeasible axioms	189
6.5.2	Experiment 2: Inconsistency	193
6.5.3	Experiment 3: Multi-threaded solving	196
6.5.4	Experiment 4: Non-uniform weights	199
6.6	Summary	200
7	Related Work	203
7.1	Repair Semantics	203
7.2	Encoding Description Logic in ASP	209
7.3	Generation of inconsistent knowledge bases	210
7.4	Summary	211
8	Conclusions and Future Work	212
8.1	Future development of the existing implementation	213
8.1.1	Optimising the tableau	213
8.1.2	Optimising the ASP implementation	217
8.1.3	Weighted priorities in $p\text{-}\mathcal{ALC}$	218

8.2	Extending the approach for languages beyond $\mathcal{ALC}$	219
8.2.1	Languages with the finite-model property	219
8.2.2	Languages without the finite-model property	220
8.2.3	Synthetic knowledge bases	221
8.3	Alternative methods of implementation	221
8.4	Summary	223
A	OWL 2 Functional Style Syntax	224
B	Knowledge bases found in the corpora	226
	Bibliography	235

List of Tables

2.1	$\mathcal{A}_{fam}$, an ABox representing the knowledge from Example 2.1	13
2.2	$\mathcal{T}_{fam}$, a TBox representing the knowledge from Example 2.1	13
2.3	Language features with associated identifiers ($\mathcal{L}$) for OWL 2 DL	19
2.4	Members of the <i>DL-Lite</i> family of languages	23
2.5	The combined complexities for Description Logic knowledge base consistency checking.	28
2.6	A summary of publications that permit planned exceptions through non-classical Description Logics.	34
2.7	A summary of publications on inconsistency tolerant Description Logics.	37
4.1	The $\mathcal{ALC}$ tableau expansion rules $\mathcal{R}$	63
4.2	The $\mathcal{ALC}$ blocking tableau rules $\mathcal{R}$	67
4.3	The $\rightarrow_{\exists\forall}$ blocking tableau rule	69
4.4	The $p\text{-}\mathcal{ALC}$ tableau rules include two new rules, the $\rightarrow_{\mathcal{T}_d}$ -rule and $\rightarrow_Q$ -rule.	77
4.5	Five open branches developed for $\mathcal{K}_{inc}$	98
5.1	The Complexity of Brave Reasoning for fragments of ASP as quoted in [LPF ⁺ 06].	127

5.2	The mappings used to translate an ordered concept C^o to an ASP term C^τ . . .	130
5.3	The mappings used to translate a $p\text{-}\mathcal{ALC}$ axiom Z to an ASP term Z^τ	131
5.4	For each rule $r \in P_b$, each term (t) of an argument (a) occurring in head(r) satisfies at least one condition C	139
5.5	For each rule $r \in P_{cum}$, each term (t) of an argument (a) occurring in head(r) satisfies at least one condition C	139
6.1	A summary of research on inconsistent and non-classical knowledge bases.	172
6.2	The characteristics of $\mathcal{ALC}$ knowledge bases found in the corpora.	173
6.3	Characteristics of the knowledge bases $\mathcal{K}_{sma}$ and $\mathcal{K}_{med}$	188
8.1	The tableau rules for lazy unfolding	214
A.1	The representation of DL concepts and axioms in FSS	224

List of Figures

2.1	A visualisation of the groups, the individuals and the relationships between individuals in Example 2.1	11
3.1	A visualisation of original knowledge modelled by $\mathcal{K}_{pat}$ and modified knowledge modelled by $\mathcal{K}_{\overline{pat}}$	47
4.1	The beginning of an infinite branch generated by Algorithm 1 for $\mathcal{K}_{cyc1}$	66
4.2	A completed $\mathcal{ALC}$ blocking tableau for example $\mathcal{K}_{cyc2}$	68
4.3	An illustration of unblocking when using a dynamic blocking strategy.	69
4.4	An illustration of static blocking.	71
4.5	An illustration of the construction of a Herbrand interpretation based on an open branch.	71
4.6	The complete $\mathcal{ALC}$ tableau for $\mathcal{K}_{qcc}$ in which no blocking occurs.	74
4.7	The complete $\mathcal{ALC}$ tableau for $\mathcal{K}_{qcs}$ in which blocking occurs.	74
4.8	The complete $\mathcal{ALC}$ tableau for $\mathcal{K}_{miss}$ in which no blocking occurs.	75
4.9	The complete $\mathcal{ALC}$ tableau for $\mathcal{K}_{miss}$ in which the concept E is split.	76
4.10	A set of fully developed branches generated by Algorithm 2 for $\mathcal{K}_{inc}$	79
4.11	A 2-distant branch generated by Algorithm 2 for $\mathcal{K}_{\overline{pat}}$	80

4.12	A $p\text{-}\mathcal{ALC}$ tableau for $\mathcal{K}_{inc}$ in which the concept E is split.	81
4.13	The proof outline for soundness of Algorithm 2, Theorem 4.26 and its corollary, Corollary 4.27.	95
4.14	The proof outline for completeness of Algorithm 2, Theorem 4.30.	99
4.15	A branch developed by Algorithm 2 for $\mathcal{K}_{inc}$	104
4.16	The proof outline for correctness of Algorithm 2, Theorem 4.31.	107
5.1	The argument graph $\mathcal{G}^A(\Pi_{5.5})$	121
5.2	An illustration of the iterative construction of an answer set of Π^i through re- peated applications of the Splitting Set Theorem.	146
6.1	The hierarchical structure of a TBox in a synthetic knowledge base.	178
6.2	Experiment 1: The average times to find n , prove queries and disprove queries for consistent defeasible knowledge bases generated from $\mathcal{K}_{sma}$	190
6.3	Experiment 2: The average percentages of defeasibility at each n found in the n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$	194
6.4	Experiment 2: The times taken to find n for each of the n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$	195
6.5	Experiment 2: The average times taken to find n for n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$	195
6.6	Experiment 2: The average times taken to find n , prove queries and disprove queries for n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$	196
6.7	Experiment 2: A comparison of defeasible axiom labelling when using Algorithm 5 and Algorithm 6	197

6.8	Experiment 3: Average lapsed times using a single threaded solver for knowledge bases generated from $\mathcal{K}_{sma}$	198
6.9	Experiment 3: Average lapsed times using a 8-threaded solver for knowledge bases generated from $\mathcal{K}_{sma}$	198
6.10	Experiment 3: Parallel efficiency.	199
6.11	Experiment 4: A comparison of reasoning performance for uniform and non-uniform weights.	201
8.1	The average times taken to find n , prove queries and disprove queries for n -inconsistent knowledge bases generated from $\mathcal{K}_{med}$ using lazy unfolding rules. . .	215

Chapter 1

Introduction

The Web Ontology Languages [BFH⁺09] (abbreviated as OWL) were designed specifically to underpin knowledge representation on the world wide web [BL03]. The adoption of Description Logics [BCM⁺07] as the underlying logic for these languages was to meet the challenge of scaling to represent large volumes of diverse knowledge by exploiting their decidable nature and relatively low complexity of reasoning. However, the underlying semantics of these languages assume model theoretic semantics [HPS09] and thus require a consistent Knowledge Base (KB) due to the principle of explosion, “ex contradictione sequitur quodlibet” - from a contradiction anything follows. Consistency is particularly difficult to guarantee in the very environment for which they were designed, on the web, where knowledge is inherently distributed and frequently maintained by independent groups. The physical separation of resources during editing, evolution and versioning can impede communication and lead to changes that inadvertently introduce inconsistencies. These may emerge immediately or at some later time when subsequent changes are made. Combining and merging information from independent sources provides an opportunity to enrich the knowledge by inferring additional consequences. However, even starting from independently consistent knowledge bases can lead to problems when the knowledge is combined. It is only once the knowledge is combined that the derivable consequences expose inconsistencies. Locating the origin of inconsistencies and then effecting the necessary *repairs* to restore consistency is a not a trivial task, see [FMK⁺08] for an overview of approaches and

the associated challenges. We pursue instead, an *inconsistency tolerant* approach, where the aim is to draw meaningful conclusions from inconsistent knowledge.

Existing work in this area includes *repair semantics* [LLR⁺10] and [EL16] in which inconsistency is tolerated within the logic $DL-Lite_{\mathcal{A}}$ [CDL⁺07]. However, these approaches have not been extended to more expressive logics. A number of paraconsistent logics (for example [HW09], [MMH13] and [ZXLV14]) have been developed for more expressive logics, but the resulting logics are very weak. This situation was improved by Quasi-Classical Paraconsistent logics (for example [ZXLV14]). However, in the logics thus far, there is no control over the *arbitration* of inconsistencies, deciding which axioms to favour when conflict occurs. This was addressed by the incorporation of Possibilistic logic (for example [ZQS13] and [QZ13]) which allows axioms to be labelled with a measure of necessity (or possibility) of the truth of the axiom. The arbitration mechanism relies on establishing a threshold that captures the overall confidence level in a knowledge base that determines which axioms may be defeated. This leads to rather coarse control over the arbitration. The absence of a solution that permits inconsistency tolerant reasoning in expressive logics, and that also offers precise control over the arbitration of inconsistency, has led to the work in this thesis.

Our focus is on scenarios that begin with a consistent knowledge base designed for reasoning under the classical model theoretic semantics, in which, due to addition or modification of the knowledge, inconsistencies emerge. Such inconsistencies occur in three distinct forms:

- Direct conflicts between facts (called *ABox axioms*). For example, a knowledge base that includes both $Woman(alice)$ and $\neg Woman(alice)$ is clearly inconsistent;
- Contradictions between rules (called *TBox axioms*) and ABox axioms. For example, $Woman(alice)$, $\neg Female(alice)$ and $Woman \sqsubseteq Female$ where $\sqsubseteq$ corresponds to material implication;
- Potentially conflicting TBox axioms. For example a knowledge base that includes the TBox axioms $Woman \sqsubseteq Human$, $Human \sqsubseteq Animal$, $Woman \sqsubseteq \neg Animal$ becomes inconsistent once an individual belonging to the concept $Woman$ is added to it.

The last case leads to the notion of *incoherence* within a knowledge base, the presence of concepts that admit no individuals. In its most extreme form the TBox admits no models at all; for example a TBox that includes $\top \sqsubseteq \textit{Human}$, $\textit{Human} \sqsubseteq \textit{Animal}$ and $\textit{Animal} \sqsubseteq \neg \textit{Human}$ where $\top$ denotes the “top” concept to which all individuals belong. Such a TBox is said to be *inconsistent*. Incoherence (and therefore an inconsistent TBox) is relatively simple to detect by checking the satisfiability of each concept and tools such as the Protégé authoring tool [NSD⁺01] can be used to find and repair the TBox in a knowledge base. In contrast, conflicts involving facts may be much harder to identify. Such conflicts can be due to interactions between combinations of inferred consequences and involve many different ABox and TBox axioms.

When dealing with inconsistency, humans seem to naturally adopt certain strategies. The first is to ignore inconsistencies that are irrelevant to steps in the inference process and the second is to try to arbitrate inconsistency by identifying the statements in which one has the greatest confidence and then to draw inferences from scenarios that *minimise* the falsification of statements. We have incorporated these strategies into the well known Description Logic $\mathcal{ALC}$ to form *preferential-ALC* ($p\text{-}\mathcal{ALC}$), a logic that allows reasoning in the presence of inconsistency. The attribute *preferential* acknowledges that the resultant logic belongs to the general family of preferential non-monotonic logics identified in [Sho87]. We work on the assumption that the TBox is satisfiable, since, as we have already noted, inconsistencies in the TBox are straightforward to repair.

Example 1.1. *Consider the classically consistent knowledge base expressed by the TBox axioms:*

$$\left\{ \textit{Reliant} \sqsubseteq \neg \textit{Fast} \text{ (1)}, \quad \textit{BMW} \sqsubseteq \textit{Fast} \text{ (2)}, \quad \textit{Blue} \sqsubseteq \neg \textit{Yellow} \text{ (3)} \right\}$$

together with facts concerning individuals Del, Alice and their respective cars:

$$\left\{ \begin{array}{ll} \textit{Drives}(\textit{del}, c1) & \text{(4)}, \quad \textit{Drives}(\textit{alice}, c2) & \text{(7)}, \\ \textit{Reliant}(c1) & \text{(5)}, \quad \textit{BMW}(c2) & \text{(8)}, \\ \textit{Yellow}(c1) & \text{(6)}, \quad \textit{Blue}(c2) & \text{(9)} \end{array} \right\}$$

Intuitively, from this knowledge base we can infer:

$$\begin{array}{lll} \neg \text{Fast}(c1) & \neg \text{BMW}(c1) & \neg \text{Blue}(c1) \\ \text{Fast}(c2) & \neg \text{Reliant}(c2) & \neg \text{Yellow}(c2) \end{array}$$

However, adding the axiom $\text{Yellow}(c2)$ (10) leads to an inconsistency involving axioms (9), (10) and (3). Given such an inconsistent knowledge base, and treating the inferences about Dels's car $c1$ independently, we might expect to retain the inferences $\neg \text{Fast}(c1)$, $\neg \text{BMW}(c1)$ and $\neg \text{Blue}(c1)$. Similarly, ignoring the inconsistency relating to the colour of $c2$, we could infer $\text{Fast}(c2)$ and $\neg \text{Reliant}(c2)$.

In both cases, no conclusions can be drawn about the colours of the cars because we have no reason to prefer either Blue or Yellow.

Informally, the arbitration of conflicting axioms is achieved in our preferential logic by identifying the particular axioms that can be falsified (*defeated*) and by including a measure of the relative confidence in those axioms by assigning each *defeasible* axiom an integer weight. Inferences are then drawn from the knowledge base by considering scenarios that falsify a minimal (summed by weight) set of defeasible axioms.

Example 1.1 (Continued). For instance, by assigning weights (indicated by $^{[w]}$) to (9) and (10) to form (9') and (10') as follows:

$$\text{Blue}(c2)^{[1]} \quad (9') \quad \text{Yellow}(c2)^{[1]} \quad (10')$$

we express that axioms (9') and (10') are defeasible and that we have equal confidence in (9') and (10'). We cannot infer the colour of $c2$ because (9') and (10') were assigned equal weights of 1. However, these are lower than the weight 2 assigned to (3) and we therefore infer $(\neg(\text{Blue} \sqcap \text{Yellow}))(c2)$. Any scenario that made $c2$ blue and yellow would falsify (3).

Now suppose that Bill drives a blue and yellow police car, so we add:

$$\text{Drives}(\text{bill}, c3) \quad (11) \quad (\text{Blue} \sqcap \text{Yellow})(c3) \quad (12) \quad \text{BMW}(c3) \quad (13)$$

where $\sqcap$ denotes the intersection of concepts, and make axiom (3) defeasible to permit a blue and yellow car: $Blue \sqsubseteq \neg Yellow^{[2]} \quad (3')$. By a similar argument to that given above, we expect to conclude $Fast(c3)$ and $\neg Reliant(c3)$ but not to infer the colour of $c3$.

Axiom (12) is not defeasible. We therefore assume (3') is defeated and conclude $Blue(c3)$ and $Yellow(c3)$. However, we do not block all inferences involving axiom (3') if only the instance of axiom (3') for $c3$ needs to be falsified. If this were the case, and all inferences involving axiom (3) were prevented, then we would lose the inference $\neg Blue(c1)$. The defeat of a TBox axiom instance is treated independently of other instances of the TBox axiom.

By reassigning $Yellow(c2)^{[3]}$ we can obtain the additional inferences of $Yellow(c3)$ and $\neg Blue(c3)$. Here, the assumption is that the weakest axiom $Blue(c3)^{[1]}$ is now falsified.

We have developed “TINference”, a sound and complete system based on $p\text{-}\mathcal{ALC}$ that can be used to obtain meaningful inferences from an inconsistent knowledge base. Axioms are made defeasible by assigning each an integer weight, the simplest choice being to make all axioms defeasible with an assigned weight of 1. However, since reasoning performance is sensitive to the number of axioms that are made defeasible, existing tools such as the OWL-API Explantation library [HPS08] are used to identify axioms that are involved in inconsistency. The weights are chosen to reflect the confidence in the axioms based on their provenance. This approach provides control over the arbitration of inconsistency and hence less cautious inferences can be made. TINference is based on a tableau algorithm for proof by refutation and is implemented using answer set programming (ASP). We chose ASP for the implementation as modern solvers are capable of tackling computationally hard problems and they support a notion of preference which allows the identification of optimal solutions. The well defined semantics of ASP has allowed us to verify that the implementation is correct.

We have conducted an evaluation of the system that demonstrates the correctness of the implementation empirically and gains insight into its performance. As part of this evaluation, we created *OWLGen*, a tool to create synthetic inconsistent $\mathcal{ALC}$ knowledge bases that broadly reflect the structures found in the corpora of existing knowledge bases.

Our approach is applicable to situations where our background knowledge of how the world should work, expressed as a TBox, does not perfectly fit the facts, which are expressed as an ABox. This is of increasing importance for the field of Ontology Based Data Access (OBDA) [CDL⁺07] where knowledge in a TBox is used to enrich an ABox of facts stored in a database. Recent developments (for example [BCS⁺16]), are pushing the technique ever towards support for more expressive Description Logics.

1.1 Summary of Contributions

This thesis presents three contributions to the field of inconsistency tolerant reasoning for Description Logic. These are:

1. $p\text{-}\mathcal{ALC}$, a novel logic that allows meaningful inferences to be drawn from an inconsistent knowledge base. Compared to the related *ABox Repair semantics*, which uses the $DL\text{-}Lite_{\mathcal{A}}$ language, $p\text{-}\mathcal{ALC}$ addresses a more expressive Description Logic, $p\text{-}\mathcal{ALC}$ consequences are less cautious and $p\text{-}\mathcal{ALC}$ has the additional benefit of control over the arbitration of conflict resolution by varying the values of weights.
2. The TINference system which computes the preferred consequences, constructing proofs by refutation based on a tableau algorithm which is implemented in ASP. We have shown that the tableau algorithm is terminating, sound and complete for all finite $p\text{-}\mathcal{ALC}$ knowledge bases and that the tableau algorithm has been correctly implemented in ASP. This leads to a terminating, sound and complete system for computing preferred consequences. We have conducted an evaluation of the system that demonstrates the system is correctly implemented and have gained insight into the scalability of the approach.
3. OWLGen, a tool that can be used to create synthetic inconsistent $\mathcal{ALC}$ knowledge bases that broadly reflect the structures found in the corpora of existing knowledge bases. The OWLGen tool allows parametric control over a wide range of variables to create knowledge bases of varying dimensions, complexity and level of inconsistency. The tool was used to

generate knowledge bases for our evaluation and will be made available for use by the wider community.

Preliminary results from the research work leading to this thesis have been published in:

Graham Deane, Krysia Broda and Alessandra Russo. Reasoning in the presence of inconsistency through Preferential ALC. *LPAR-20. 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations*, volume 35 of EPiC Series in Computing, pages 67-80. EasyChair, 2015.

1.2 Thesis Overview

Chapter 2 provides the background on Description Logics. The basic notions are formalised through the foundational logic called $\mathcal{ALC}$ and then the more expressive logics that underpin the OWL 2 languages [BFH⁺09]. The nature of inconsistency in Description Logics is discussed together with a survey of the approaches that have been proposed in the literature to handle inconsistency.

Chapter 3 introduces the inconsistency tolerant logic Preferential $\mathcal{ALC}$ ($p\text{-}\mathcal{ALC}$). After the formal syntax and semantics have been introduced, key properties of the logic are proved and used to show that inferences under the $p\text{-}\mathcal{ALC}$ semantics, called the *preferred consequences*, can be achieved by refutation style proofs performed in two stages.

Chapter 4 presents a tableau algorithm for $p\text{-}\mathcal{ALC}$. Background information on tableau methods are introduced through an existing tableau algorithm [BHS08] that decides satisfiability of an $\mathcal{ALC}$ knowledge base. We show how the algorithm is adapted for $p\text{-}\mathcal{ALC}$ to create a sound, complete and terminating algorithm which can be used to compute the preferred consequences of a $p\text{-}\mathcal{ALC}$ knowledge base.

Chapter 5 introduces our implementation of the $p\text{-}\mathcal{ALC}$ tableau algorithm in ASP. Background information is provided on ASP and clingo, the answer set solver chosen for the implementation. Important properties of answer set programs that are used to demonstrate correctness of the

implementation are then reviewed. We show how a $p\text{-}\mathcal{ALC}$ knowledge base can be translated to an answer set program such that, when it is solved incrementally, the program has answer sets that capture the information in the tableau branches. Exploiting the optimisation features of ASP, we demonstrate correspondence between the optimal answer sets and minimal branches obtained by the tableau and explain how the implementation is used to compute preferred consequences of a $p\text{-}\mathcal{ALC}$ knowledge base.

Chapter 6 provides an evaluation of our ASP implementation. We demonstrate through a series of experiments that the translation to an ASP program correctly implements the tableau algorithm of chapter 3 correctly, and provide insight onto its practicality by measuring its performance when applied to inconsistent knowledge bases. This chapter also includes details on the generation of the synthetic knowledge bases that are used in the evaluation.

Chapter 7 extends our survey of related work given in Chapter 2 and addresses the relationship between our work and the work that is most closely related.

Chapter 8 summarises the conclusions that can be drawn from the thesis and outlines our plans for future work.

Chapter 2

An introduction to Description Logics

This chapter provides an overview of Description Logics. First of all, the basic notions of a Description Logic are introduced. These notions are then formalised through the foundational logic called $\mathcal{ALC}$. More expressive logics are then introduced, focussing on those that underpin the OWL 2 languages [BFH⁺09]. Finally, the notion of inconsistency in Description Logics is discussed together with a survey of the approaches that have been proposed in the literature to handle inconsistency.

Description logics are the result of several decades of research into finding decidable fragments of first order logic that are expressive enough to represent real world problems and also have favourable complexity characteristics. They have been designed for efficient representation of and reasoning with hierarchical knowledge. Various Description Logics have been proposed. They are distinguished by an acronym that reflects the set of *features* that are available in the logic. For instance, $\mathcal{ALC}$ refers to an *attributive language* ($\mathcal{AL}$) augmented with constructors that are used to denote concept complements $\mathcal{C}$ [BCM⁺07].

In a Description Logic a *domain of interest* is represented in a language formed from sets of *names* (the *signature*) and a set of *connectives*. *Individual names* denote individuals within the domain, *concept names* denote sets of individuals and *role names* denote the relationships between individuals.

Concepts are used to describe a subset of individuals within the domain. They are built using names from the signature and connectives called *constructors*. For example, the concept *Man* might be used to describe the set of individuals who are men. $Man \sqcup Woman$, describes the set of individuals who are “either a man or a woman”, where the constructor $\sqcup$ denotes the union of concepts. $Human \sqcap Male$ describes “human males” where the constructor $\sqcap$ denotes the intersection of concepts. Concepts may include characterisations based on quantified roles (the numbers of relationships between the individuals). For example, $\forall hasChild. \neg Male$ describes the set of individuals having *only* ($\forall$) children that are not ($\neg$) male. Similarly, $Man \sqcap \exists hasChild. \top$, describes the set of individuals who are men and have at least one child. $\top$ denotes the set of all individuals in the domain, so $\exists hasChild. \top$ is read as the set of individuals having at least one ($\exists$) *hasChild* role relationship.

A description of a domain of interest is called a *knowledge base*. It is divided into an *ABox*, facts about individuals, called *assertional knowledge*, and a *TBox*, structural knowledge that captures the relationships between concepts and roles, called *terminological knowledge*. The facts in an ABox are called *assertion axioms*. For example, the *concept assertion* $Man(bob)$ expresses that the individual *bob* belongs to the concept *Man*. Similarly, the *role assertion* $hasChild(alice, charlie)$ expresses the existence of a *hasChild* role linking the individual *alice* to the individual *charlie*. The *TBox* is a set of axioms that define a set of relations between concepts (or roles), where the connective $\sqsubseteq$ is used to denote a subsumption relation. For example, to express that humans are *subsumed* by the (possibly broader) concept of animals, we write $Human \sqsubseteq Animal$. $hasChild \sqsubseteq hasDescendant$ denotes that each pair of individuals connected by a *hasChild* role is a subset of the pairs of individuals connected by a *hasDescendant* role. The connective $\equiv$ is used to denote concept equivalence. To express that men are exactly those individuals who are both human and male, one can write $Man \equiv Human \sqcap Male$.

In the next section, the syntax and semantics for the logic $\mathcal{ALC}$ are presented. Other logics that include different features and that target particular applications are discussed in Section 2.2. In these logics the same underlying basic notions of $\mathcal{ALC}$ apply, but with definitions sometimes modified to accommodate features that are not available in $\mathcal{ALC}$.

2.1 The Description Logic $\mathcal{ALC}$

The syntax and the semantics presented in this section are standard for Description Logics and are based on those given in [BCM⁺07]. $\mathcal{ALC}$ is a basic Description Logic but it is expressive enough to represent many problems. Example 2.1 below will be used to provide a running example to illustrate knowledge representation in $\mathcal{ALC}$.

Example 2.1. *The knowledge of individuals is: Alice is a woman, she is married to a man called Bob, has a child called Charlie and a pet dog. The structural knowledge is: a woman is a female human and a man is a male human; humans and dogs are animals but represent distinct groupings; only humans have pets, all pets are animals and all children are human. The groups, the individuals and the relationships between individuals are visualised in Figure 2.1.*

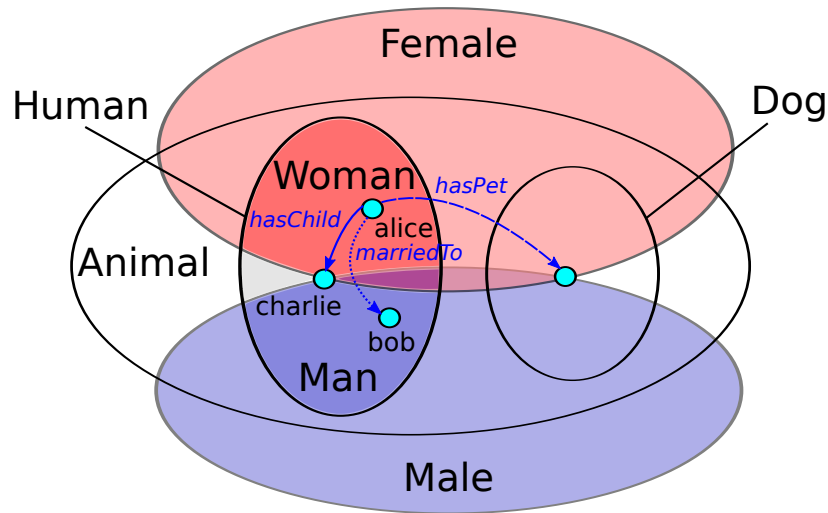


Figure 2.1: A visualisation of the groups, the individuals and the relationships between individuals in Example 2.1

2.1.1 Syntax

Definition 2.1 (Signature). *An $\mathcal{ALC}$ signature is a tuple $\langle N_I, N_C, N_R \rangle$ where N_I , N_C and N_R are finite sets of names that refer, respectively, to individuals, named concepts and roles.*

A suitable signature for Example 2.1 is

$$\begin{aligned} &\langle \{alice, bob, charlie\} \\ &\quad \{Man, Woman, Male, Female, Human, Dog, Animal\}, \\ &\quad \{hasPet, hasChild, marriedTo\} \rangle \end{aligned}$$

Definition 2.2 (Concept). Let $\langle N_I, N_C, N_R \rangle$ be an $\mathcal{ALC}$ signature. $\mathcal{ALC}$ concepts C, D are defined inductively from the names in the signature and the following set of symbols called constructors:

$$C, D = \top | \perp | A | \neg C | C \sqcap D | C \sqcup D | \exists R.C | \forall R.C$$

where $A \in N_C$; $R \in N_R$; $\top$ and $\perp$ denote the top and bottom concepts respectively; $\neg C$ denotes concept negation; $C \sqcap D$ and $C \sqcup D$ denote concept intersections and unions; $\exists R.C$ and $\forall R.C$ denote existential and universal role restrictions; and the concept C within a quantified role restriction is called the *filler* for the restriction.

In Example 2.1, the concept $\exists hasPet.\top$ captures the description of individuals having a pet, an existential role restriction with the top concept as the filler.

Definition 2.3 (Knowledge Base). Let $\langle N_I, N_C, N_R \rangle$ be an $\mathcal{ALC}$ signature. $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ is an $\mathcal{ALC}$ knowledge base, where $\mathcal{A}$ is called the *assertion box* (*ABox*) and $\mathcal{T}$ is called the *terminological box* (*TBox*). $\mathcal{A}$ is a (possibly empty) finite set of concept assertion axioms of the form $C(x)$ and role assertion axioms of the form $R(x, y)$ where C is an $\mathcal{ALC}$ concept, $R \in N_R$ and $x, y \in N_I$. $\mathcal{T}$ is a (possibly empty) finite set of concept inclusions of the form $C \sqsubseteq D$ and concept equivalence axioms of the form $C \equiv D$, where C and D are $\mathcal{ALC}$ concepts. $C \equiv D$ abbreviates $C \sqsubseteq D$ and $D \sqsubseteq C$. $sig(\mathcal{K})$ denotes the signature of $\mathcal{K}$. $\mathcal{L}(\mathcal{K})$ denotes the language of $\mathcal{K}$, the set of all possible concepts and axioms that may be constructed from the signature.

The knowledge base $\mathcal{K}_{fam} = \langle \mathcal{A}_{fam}, \mathcal{T}_{fam} \rangle$ represents Example 2.1 where the axioms of $\mathcal{A}_{fam}$ and $\mathcal{T}_{fam}$ are given by Tables 2.1 and 2.2 respectively.

Axiom	Expresses
$Woman(alice)$	Alice is a Woman
$Man(bob)$	Bob is a Man
$marriedTo(alice, bob)$	Alice is married to Bob
$hasChild(alice, charlie)$	Alice has a child called Charlie
$(\exists hasPet.Dog)(alice)$	Alice has a pet dog

Table 2.1: $\mathcal{A}_{fam}$, an ABox representing the knowledge from Example 2.1

Axiom	Expresses
$Man \equiv Human \sqcap Male$	Men are male humans
$Woman \equiv Human \sqcap Female$	Women are female humans
$Human \sqsubseteq Animal$	Humans are animals
$Dog \sqsubseteq Animal$	Dogs are animals
$Human \sqsubseteq \neg Dog$	Humans and Dogs are disjoint concepts
$\exists hasPet.\top \sqsubseteq Human$	Only humans have pets (domain constraint)
$\top \sqsubseteq \forall hasPet.Animal$	All pets are animals (range constraint)
$\top \sqsubseteq \forall hasChild.Human$	All children are human (range constraint)

Table 2.2: $\mathcal{T}_{fam}$, a TBox representing the knowledge from Example 2.1

Definition 2.4 (Sub-formulae). *Let C be an $\mathcal{ALC}$ concept. The sub-formulae of C , denoted $\mathcal{F}(C)$, are defined recursively:*

- $C \in \mathcal{F}(C)$ is a sub-formula;
- if $\neg C \in \mathcal{F}(C)$ is a sub-formula then C is a sub-formula;
- if $C \sqcap D \in \mathcal{F}(C)$ is a sub-formula then C and D are sub-formulae;
- if $C \sqcup D \in \mathcal{F}(C)$ is a sub-formula then C and D are sub-formulae;
- if $\exists R.C \in \mathcal{F}(C)$ is a sub-formula then C is a sub-formula;
- if $\forall R.C \in \mathcal{F}(C)$ is a sub-formula then C is a sub-formula;

Additional nomenclature is provided to classify terminological axioms based on their form and their relationships to other axioms in the knowledge base.

Definition 2.5 (Axiom nomenclature). *Let Z be a terminological axiom of an $\mathcal{ALC}$ knowledge base $\mathcal{K}$. If Z is a concept inclusion of the form $C \sqsubseteq D$ where $C \in N_C$ (a concept name) and D is a concept then Z is called a primitive definition of C . If Z is a concept equivalence of the*

form $C \equiv D$ where $C \in N_C$ and D is a concept then Z is called a non-primitive definition of C . A concept inclusion that is not a definition is called a general concept inclusion (GCI).

A definition is said to be unique if there is no other definition of C in $\mathcal{K}$; and it is acyclic if C does not occur in D or indirectly in D (within definitions of the sub-formulae of D).

2.1.2 Semantics

The semantics are provided through model theoretic semantics.

Definition 2.6 (Interpretation). Let $\mathcal{K}$ be an $\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$. An interpretation $\mathcal{I}$ of $\mathcal{K}$ is the pair $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}}$ is a non-empty (possibly countably infinite) set called the domain of the interpretation and $\cdot^{\mathcal{I}}$ is an interpretation function. The function $\cdot^{\mathcal{I}}$ interprets the names from the signature of $\mathcal{K}$ s.t. each individual name $x \in N_I$ is interpreted as a domain element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$; each concept name $A \in N_C$ is interpreted as a unary relation $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$; and each role name $R \in N_R$ is interpreted as a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended inductively to any concept C that can be formed from the signature through the following definitions:

$$\begin{aligned}
 (\top)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
 (\perp)^{\mathcal{I}} &= \emptyset \\
 (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
 (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
 (C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
 (\forall R.C)^{\mathcal{I}} &= \{u \in \Delta^{\mathcal{I}} \mid \forall v [(u, v) \in R^{\mathcal{I}} \rightarrow v \in C^{\mathcal{I}}]\} \\
 (\exists R.C)^{\mathcal{I}} &= \{u \in \Delta^{\mathcal{I}} \mid \exists v [(u, v) \in R^{\mathcal{I}} \wedge v \in C^{\mathcal{I}}]\}
 \end{aligned}$$

Historically, Description Logic semantics have required that every individual name from the signature represents a unique individual within the domain. This is referred to as the unique names assumption (UNA).

Definition 2.7 (Unique names assumption). *Let $\mathcal{I}$ be an interpretation of an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ with signature $\langle N_I, N_C, N_R \rangle$. $\mathcal{I}$ satisfies the unique names assumption iff for each pair of distinct names x and y in N_I , $x^{\mathcal{I}} \neq y^{\mathcal{I}}$.*

However, to maximise flexibility, the OWL 2 specification allows the author to choose whether or not to apply the UNA. The UNA only becomes important in logics that are more expressive than $\mathcal{ALC}$, where counting (or otherwise distinguishing between) individuals is allowed. It should be noted that $\Delta^{\mathcal{I}}$, the domain of the interpretation, can be any (possibly countably infinite) non-empty set. Under the unique names assumption, $|\Delta^{\mathcal{I}}| \geq |N_I|$, since each individual name must be interpreted as a different domain element. However, there is no upper bound on $|\Delta^{\mathcal{I}}|$ and an interpretation may include domain elements that are not assigned names. Such an interpretation reflects the existence of *unnamed individuals* within the domain.

Definition 2.8 (Unnamed individual). *Let $\mathcal{I}$ be an interpretation of an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ with signature $\langle N_I, N_C, N_R \rangle$. A domain element, $u \in \Delta^{\mathcal{I}}$, is an unnamed individual iff for each name $x \in N_I$, $x^{\mathcal{I}} \neq u$.*

It is often convenient to consider *Herbrand* interpretations of a knowledge base.

Definition 2.9 (Herbrand Interpretation). *Let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be an interpretation of an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ with signature $\langle N_I, N_C, N_R \rangle$ and N_U be a set of names that are not in N_I . $\Delta_b = N_I \cup N_U$ is the Herbrand domain based on $N_I \cup N_U$ and $\mathcal{I}$ is a Herbrand interpretation of $\mathcal{K}$ based on Δ_b iff $\Delta^{\mathcal{I}} = \Delta_b$ and for each name x in $N_I \cup N_U$, $x^{\mathcal{I}} = x$.*

To lighten the notation, where no domain is specified for a Herbrand interpretation, N_U is assumed to be empty and $\Delta_b = N_I$. Models of a knowledge base and notions of logical entailment are defined classically.

Definition 2.10 (Model). *Let $\mathcal{I}$ be an interpretation of an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ with signature $\langle N_I, N_C, N_R \rangle$. An axiom Z , in $\mathcal{K}$, is satisfied by $\mathcal{I}$, written $\mathcal{I} \models Z$, iff either (i), (ii) or (iii) holds:*

- (i) if Z is of the form $C(x)$, $x^{\mathcal{I}} \in C^{\mathcal{I}}$;

- (ii) if Z is of the form $R(x, y)$, $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$;
- (iii) if Z is of the form $C \sqsubseteq D$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

A concept C is said to be *satisfiable* in $\mathcal{K}$ iff there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}}$ is non-empty. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K}$ iff every axiom is satisfied by $\mathcal{I}$. $\mathcal{K}$ is said to be *consistent* if it has a model. If there is no model of $\mathcal{K}$ then the knowledge base is said to be *inconsistent*. Let Z be an axiom written in the signature of $\mathcal{K}$. $\mathcal{K}$ *entails* Z , written $\mathcal{K} \models Z$, iff Z is satisfied in every model of $\mathcal{K}$. Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox. Then $\mathcal{T}$ is said to be *consistent* if there is a model of the knowledge base $\langle \emptyset, \mathcal{T} \rangle$, and *inconsistent* otherwise.

The model theoretic semantics for Description Logics leads to an open world assumption (OWA): no assumptions are made about the truth of axioms that are not entailed.

In the running example, $\mathcal{K}_{fam} \models \neg \text{Dog}(\text{charlie})$. From $\text{hasChild}(\text{alice}, \text{charlie})$ and $\top \sqsubseteq \forall \text{hasChild}. \text{Human}$ we can conclude $\mathcal{K}_{fam} \models \text{Human}(\text{charlie})$ and now from $\text{Human} \sqsubseteq \neg \text{Dog}$ we conclude $\mathcal{K}_{fam} \models \neg \text{Dog}(\text{charlie})$. However, $\mathcal{K}_{fam} \not\models \text{Man}(\text{charlie})$. Under the open world assumption, Charlie could be a man, a woman, both or neither.

Having set out the formal syntax and semantics of $\mathcal{ALC}$, we next outline important reasoning tasks in Description Logics.

2.1.3 Reasoning tasks

A fundamental reasoning task for a Description Logic is deciding *knowledge base satisfiability* (also called a *consistency check*). A consistency check is generally made before performing any other reasoning tasks since any axiom is entailed from an inconsistent knowledge base.

Retrieval of information about the individuals stored in the knowledge base are known as *ABox reasoning tasks*. For example, deciding if individual x is an instance of a concept C is called *instance checking*. In the Definitions 2.11-2.14 it is assumed that $\mathcal{K}$ is an $\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$, $x \in N_C$, $R \in N_R$ and C, D are concepts.

Definition 2.11 (Instance checking). *An instance check to decide if x is an instance of concept C returns true iff $\mathcal{K} \models C(x)$. An instance check to decide if (x, y) is an instance of role R returns true iff $\mathcal{K} \models R(x, y)$.*

Closely related to instance checking tasks are *query tasks*. A query task is used to find a set of individuals that satisfy some specified search criteria. The *concept retrieval* task is the most basic example. It returns all the named individuals that are instances of a given concept. Concept retrieval is equivalent to performing a concept instance check for each named individual in N_I .

Definition 2.12 (Concept retrieval). *The concept retrieval task for C returns the set $\{x | x \in N_I \text{ and } \mathcal{K} \models C(x)\}$.*

TBox reasoning tasks involve making inferences about the structure of the knowledge. The *concept satisfiability* task establishes if a concept admits a non empty model with respect to a TBox.

Definition 2.13 (Concept satisfiability). *Concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ iff there exists an interpretation $\mathcal{I}$ of the knowledge base $\langle \emptyset, \mathcal{T} \rangle$ such that $C^{\mathcal{I}} \neq \emptyset$.*

The *concept subsumption* task decides if a concept C is subsumed by a second concept D .

Definition 2.14 (Concept subsumption). *Concept D subsumes concept C iff $\mathcal{K} \models C \sqsubseteq D$.*

Having introduced the formalism through $\mathcal{ALC}$, we now consider the range of logics available in OWL 2.

2.2 The extended family of Description Logics

$\mathcal{ALC}$ is not sufficiently expressive for all problems. In this section we detail the additional features that are available within the *OWL 2 DL* languages [BFH⁺09]. Here the annotation *DL* indicates that each language corresponds to a Description Logic and that model theoretic

semantics are applicable¹. These features are combined to construct more (and sometimes less) expressive Description Logics to suit a variety of applications. Table 2.3 summarizes the features available up to the expressive logic $\mathcal{SROIQ}(D)$, indicating the identifier associated with a feature and an example of the feature syntax.

Entity Relationship models (ER-models) define an abstract representation for data models [Che76] and may include *cardinality constraints*, for instance, to capture that a *Human* has at most two *hasParent* relations. These can be modelled by enriching a Description Logic with *number restrictions*. Feature $\mathcal{Q}$ generalises the existential role restrictions to permit the description of having *at most*, *at least* or *exactly* some number of relations. For example, $\leq 2 \text{ hasParent.Human}$ describes individuals that have at most 2 human parents. Feature $\mathcal{N}$ restricts feature $\mathcal{Q}$ such that the filler must be $\top$ (in such languages $\top$ is omitted). Thus $\leq 2 \text{ hasParent}$ describes humans with at most 2 parents. Restricting a feature in this way reduces the computational complexity, and examples of languages that exploit this are discussed further in Section 2.5. Feature $\mathcal{F}$ indicates that cardinality constraints of *exactly 1* are permitted; these are used to express role functionality. Feature $\mathcal{I}$ allows an inverse role relationship to be described. For example, the inverse of the *hasParent* relationship (written hasParent^-) can be used to capture the set of individuals having children by writing $\exists \text{hasParent}^-. \top$. In the Description Logics discussed so far, it is not possible to represent domain closure or that a concept represents a specified set of individuals, which is important when capturing information from closed world systems such as databases. This is addressed by feature $\mathcal{O}$ denoting *nominals*. The syntax $\{o\}$ defines the *nominal concept* that includes only the individual named o . Given $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ an interpretation of a knowledge base, $\{o\}^{\mathcal{I}} = \{o^{\mathcal{I}}\}$, that is to say the set containing the element in $\Delta^{\mathcal{I}}$ that o is mapped to. In the running example: $\top \sqsubseteq \{alice\} \sqcup \{bob\} \sqcup \{charlie\} \sqcup \{fido\}$ restricts the domain of the interpretation such that every element is mapped to at least one of the four names. Under the UNA, every model of a knowledge base that includes this axiom has $|\Delta^{\mathcal{I}}| = 4$.

Roles may also have a natural hierarchical structure, for example the existence of a *hasChild* relationship between individuals implies the existence of a *hasDescendant* relationship. Feature

¹[BFH⁺09] also specifies languages that are not expressible in any known Description Logic.

$\mathcal{H}$ allows the expression of role subsumptions, for instance: $hasChild \sqsubseteq hasDescendant$. $\mathcal{S}$ designates a language that is based on the attributive language $\mathcal{AL}$ and includes both concept negation $\mathcal{C}$ and transitive roles ($\mathcal{AL}$ and $\mathcal{C}$ are omitted from the name). Feature $\mathcal{R}$ further enriches $\mathcal{H}$ with the following: role (ir)reflexivity, disjoint roles, a universal role, role chains, the concept *Self* and negative role assertions. A role chain, written $\circ$, allows role composition; for example, the notion of the “has uncle relationship” is captured by $hasParent \circ hasBrother$. To capture the rich data types found in databases the signature may be extended to include *concrete domains* and *attributes*. A concrete domain is a set of constants that denote possible values for a data type, for instance: sets of integers, real numbers or strings. An attribute is a data role relation that allows individuals to be linked to concrete data. For example, $hasAge(alice, 30)$ and $hasFamilyName(alice, \text{“Smith”})$ capture Alice’s age is 30 and her family name is Smith. The data feature is denoted (D).

$SR\mathcal{OIQ}(D)$ denotes the Description Logic that includes the maximal set of features available in OWL 2 DL.

$\mathcal{L}$	Feature	Syntax	Example
$\mathcal{C}$	Negated concepts	$\neg C$	$\neg \exists hasChild.Female$
$\mathcal{E}$	Existential restriction to a concept	$\exists R.C$	$\exists hasChild.Female$
$\mathcal{U}$	Disjunctive concepts	$C \sqcup D$	$Man \sqcup Woman$
$\mathcal{I}$	Inverse roles within concepts	R^-	$\exists hasChild^-.Female$
$\mathcal{N}$	Number restrictions within concepts	$\geq n R, \leq n R$ and $= n R$	$\leq 2 hasParent$ $= 2 hasChild$
$\mathcal{Q}$	Qualified number restrictions within concepts	$\geq n R.C, \leq n R.C$ and $= n R.C$	$\geq 3 hasChild.Male$ $= 4 hasChild.Human$
$\mathcal{O}$	Nominals within concepts	$\{o\}$	$\exists hasChild.\{charlie\}$
$\mathcal{F}$	Functional relation axioms	$Func(R)$	$Func(marriedTo)$
$\mathcal{S}$	$\mathcal{ALC}$ + transitive role axioms	$Tra(R)$	$Tra(hasDescendant)$
$\mathcal{H}$	Role hierarchy axioms with $\mathcal{I}$ allows symmetric roles	$R \sqsubseteq S$ $Sym(R)$	$hasChild \sqsubseteq hasDescendant$ $Sym(marriedTo)$
$\mathcal{R}$	$\mathcal{H}$ + disjoint roles, reflexive and irreflexive roles, negated role assertions, role chains, the universal role and the role <i>Self</i>	$Dis(R)$ $Ref(R), Irr(R)$ $\neg R(a, b)$ $R \circ S$ U <i>Self</i>	$Dis(childOf, marriedTo)$ $Irr(marriedTo)$ $\neg marriedTo(alice, charlie)$ $hasParent \circ hasBrother$ $\exists U.\{alice\}$ $\exists knows.Self$
(D)	Data (concrete domains accessed using data attributes)	Integers, real numbers, strings etc. <i>hasValue, hasPrice, hasName</i> etc.	

Table 2.3: Language features with associated identifiers ($\mathcal{L}$) for OWL 2 DL

In recognition that the characteristics of knowledge varies depending on the application, three sub-languages called *profiles* are specified in [CCD⁺12]. The profiles incorporate state-of-the-art research on families of Description Logics and are designed to “trade some expressive power for the efficiency of reasoning” [CCD⁺12].

A number of so called *computational cliffs* [BCM⁺07] have been identified in Description Logics. These are features (or combinations of features) that, when added to a simpler logic, lead to a significant increase in complexity. These cliffs can be understood by considering the underlying *sources of complexity* of reasoning. Within $\mathcal{ALC}$, there are three underlying sources of complexity that impact on the task of checking if a concept is satisfiable:

1. *OR-branching*: The requirement to consider different candidate models. For example, the inclusion of a concept union constructor $\sqcup$ within a concept may require that more than one candidate model has to be checked. If no model can be found that satisfies the concept on one side of the union, then it is necessary to check instead for a model that satisfies the concept on the other side.
2. *AND-branching*: The presence of existentially quantified role restrictions within a concept implies that a model of the concept may involve more than one domain element. Here, the notion of a “branch” refers to checking concept satisfiability of distinct domain elements within the same model. For example, the concept $(C \sqcap (\exists R. \neg C))$ is only satisfiable in a domain that includes at least 2 elements. One element must satisfy C and another, linked from the first by the role R , must satisfy $\neg C$. $\mathcal{ALC}$ allows existentially and universally quantified role restrictions to be freely nested, and the size of the model that needs to be checked can grow exponentially with respect to the nesting.
3. *Unrestricted TBoxes*: If a TBox is acyclic, then reasoning algorithms can be designed to exploit this property by rewriting complex concepts in terms of concept definitions, a process called *unfolding* [BCM⁺07]. Such algorithms have a lower complexity of reasoning than algorithms for an unrestricted TBox.

Of particular importance are the OWL 2 EL profile and the OWL 2 QL profile [CCD⁺12] which

are used when a system is required to scale for large TBoxes or large ABoxes respectively. The EL profile is underpinned by the $\mathcal{EL}$ family of Description Logics and the QL profile by the *DL-Lite* family. The characteristics of these families are given next.

2.2.1 The $\mathcal{EL}$ family

$\mathcal{EL}$ [BKM99] was developed to target problems that require large TBoxes. It has been particularly successful in modelling medical terminologies such as the GALEN [RNG93] and the SNOMED [CRP⁺93] projects. $\mathcal{EL}$ is obtained from $\mathcal{ALC}$ by eliminating negation (and the bottom concept), disjunction and universally quantified restrictions. Thus, the syntax for an $\mathcal{EL}$ concept C is:

$$C := \top | A | C \sqcap D | \exists R.C$$

The removal of both concept unions and universal quantifiers eliminates or-branching and and-branching, two of the aforementioned underlying sources of complexity.

$\mathcal{EL}^{++}$ [BBL05] which underpins the OWL 2-EL profile squeezes into $\mathcal{EL}$: the bottom concept, nominals, role inclusions, role chains and concrete domains without increasing the complexity of reasoning. The resultant language allows complex terminologies to be expressed, for example [HRG96]:

$$(\text{ulcer} \sqcap \exists \text{hasLoc.stomach}) \sqsubseteq (\text{ulcer} \sqcap \exists \text{hasLoc} . (\text{lining} \sqcap \exists \text{ispartof.stomach}))$$

A knowledge base satisfiability check must test every individual against a (potentially large) TBox. Hence, the $\mathcal{EL}$ family of logics are good for large TBoxes and small ABoxes. Consequently, $\mathcal{EL}$ family knowledge bases typically have no (or very few) individuals. Reasoning tasks are focussed on concept satisfiability and subsumption checking. Where a large number of individuals is anticipated the *DL-Lite* family of logics is more appropriate.

2.2.2 The *DL-Lite* family

The *DL-Lite* family [CDL⁺07] were developed to specifically target problems with large ABoxes. In particular, they were designed with inter-operability with relational database systems. In the *DL-Lite* family: there are no universally quantified restrictions; all existential restrictions have $\top$ as the filler (which are omitted) and are written $\exists R$ (instead of $\exists R.\top$); negation and disjunction appear within concepts as syntactic sugar (and are only permitted in locations that do not introduce additional non-deterministic steps in reasoning tasks); and inverse roles are included.

The most basic language in the family is called *DL-Lite_{core}*. In *DL-Lite_{core}*, concepts C, D and roles P are constructed from a signature $\langle N_I, N_C, N_R \rangle$ using the following syntax:

$$C := A \mid \exists P \quad D := C \mid \neg C \quad P := R \mid R^{-}$$

where $A \in N_C$ and $R \in N_R$. A *DL-Lite_{core}* TBox is a set of axioms of the form $C \sqsubseteq D$.

The $\sqcup$ and $\forall$ constructors are not included, eliminating both or-branching and and-branching. However, negation is permitted on the right hand side of concept inclusions, allowing the disjointness of concepts to be expressed. This does not introduce non-determinism as the concept being negated is either a concept name, an existentially quantified role or an existentially quantified inverse role. For the existential cases, the concept $\neg \exists R$ mandates an individual has no R -successor and $\neg \exists R^{-}$ mandates an individual has no R -ancestor. The provision of the inverse role permits domain and range constraints (important in database design) to be expressed using $\exists R^{-} \sqsubseteq C$ and $\exists R \sqsubseteq C$ respectively. In the *DL-Lite* languages, checking knowledge base satisfiability can be performed very efficiently for knowledge bases with large numbers of individuals and a relatively small TBox. The ABox contains only positive concept (name) assertions and positive role assertions, and thus is satisfiable in the absence of a TBox. The complexity of reasoning is driven by the need to check each individual's satisfiability with respect to the derived concepts imposed by the TBox. This check must also take into account the existence of inferred (un-named) successors that may be required to satisfy these concepts. The strict

syntactic limitations imposed on the TBox structure allows this to be performed efficiently.

$DL-Lite_{core}$ forms the basis of several important DL languages. $DL-Lite_{\mathcal{R}}$ adds role inclusion axioms that permit negated roles on the right hand side. This enhancement is important as it permits disjoint roles and negated role assertions to be expressed. The negative role assertion $\neg R(a, b)$ is obtained by defining $R' \sqsubseteq \neg R$ and then asserting $R'(a, b)$. The OWL 2 QL profile [CCD⁺12] adds concrete data and attributes to the language $DL-Lite_{\mathcal{R}}$ without increasing the complexity. Cardinality constraints are useful to express role cardinality constraints commonly found in database schemas. $DL-Lite_{\mathcal{F}}$ adds functional roles to $DL-Lite_{core}$, and $DL-Lite_{bool}^{\mathcal{N}}$ adds number restrictions ($\mathcal{N}$) and the freedom to place negation on either side of concept inclusions (enabling the full range of Boolean operators to be expressed within concepts). There is a well known interaction [BCM⁺07] between existential role restrictions ($\exists R.C$) and number restrictions ($\leq n R, \geq n R, = n R$) that makes the identification of candidate models more complex. For example [BCM⁺07], consider the concept:

$$(\exists R.A) \sqcap (\exists R.(\neg A \sqcap \neg B)) \sqcap (\exists R.B) \sqcap (\leq 2 R) \quad (2.1)$$

An algorithm cannot simply choose three distinct R -successors (one for each existential) for an individual that satisfies concept (2.1). A model requires that there are exactly two R -successors: one R -successor satisfying A and B and the other satisfying $(\neg A \sqcap \neg B)$. This interaction cannot occur in $DL-Lite_{bool}^{\mathcal{N}}$ concepts since the filler in any existentially quantified restrictions is always $\top$.

Table 2.4 summarises the features available in the different variants of the $DL-Lite$ languages.

$\mathcal{L}$	Adds to $DL-Lite_{core}$
$DL-Lite_{core}$	
$DL-Lite_{\mathcal{R}}$	Role Box axioms: $P \sqsubseteq Q$ where $Q := P \mid \neg P$
OWL 2 QL	Role Box + Attributes (data roles)
$DL-Lite_{\mathcal{F}}$	Functional Roles: $\text{func}(R)$ where $R \in N_R$
$DL-Lite_{\mathcal{A}}$	Role Box + Attributes (data roles) + Functional Roles
$DL-Lite_{bool}^{\mathcal{N}}$	Boolean concepts: $D_1 \sqsubseteq D_2$ and number restrictions

Table 2.4: Members of the $DL-Lite$ family of languages

The Description Logics that have been introduced in this section illustrate the diversity of the

logics and how the combinations of features included in a language influences the expressivity and hence their computational complexity. In the next section we review a system for measuring computational complexity.

2.3 Complexity

In this section we review the approach to analysing computational complexity using complexity classes and use it to classify the complexity of reasoning tasks for the Description Logics presented in the previous section. The overview and definitions in this section are based on [AB09].

Informally, *computational complexity* indicates how difficult a problem is to solve. This is expressed in terms of the usage of the resources of *time* and *space* as functions of the input problem size. We focus on the class of *decision problems*. These are problems that can be answered by an algorithm that returns a “yes” or “no” answer. Deterministic and non-deterministic Turing machines (DTMs and NDTMs) are used to benchmark the utilisation of time and space. Such machines are *universal* and can be used to encode any algorithm. A machine is said to be *terminating* iff for any finite input problem it halts after a finite number of *steps*. A machine that encodes an algorithm for a decision problem can terminate (it may not) in one of two halting states that distinguish the answer “yes” from the answer “no”. The class of *decidable* problems are those problems that can be encoded by a terminating Turing machine. The resources consumed by such a machine define **TIME** as the time taken, and **SPACE** as the space used, expressed as a function of the size of the input problem.

Definition 2.15 (Decision Problem). *Let P denote a set of problems and $M \subseteq P$. The decision problem $D(M, P)$ is to check whether an input $p \in P$ is in M or not. If an answer for the input p is returned, then the answer is “yes” if $p \in M$ and “no” if $p \notin M$.*

For example, assume P to be the set of all $\mathcal{ALC}$ knowledge bases and M to be the set of all satisfiable $\mathcal{ALC}$ knowledge bases. The knowledge base satisfiability checking task for an input $p \in P$ is the problem of deciding if p is in M .

A complexity class $\mathcal{C}$ defines the set of decidable decision problems that can be answered by a Deterministic Turing machine (DTM) or a Non-deterministic Turing machine (NDTM) consuming no more than the resource $\mathcal{C}$ specified as **TIME** or **SPACE**. For example, the complexity classes **P** (resp. **NP**) denotes the set of decidable decision problems that can be answered by a DTM (resp. NDTM) in a time that is polynomial in the size of the input. Similarly, the class **PSPACE** denotes the set of decidable decision problems that can be answered by a DTM requiring space that is polynomial in the size of the input.

Having established a benchmark for computational complexity, we need a mechanism to establish which complexity classes a particular problem belongs to. Membership in a complexity class $\mathcal{C}$ implies the existence of an algorithm that consumes no more resources than $\mathcal{C}$. It establishes an upper bound on the complexity of a problem.

Definition 2.16 (Complexity class membership). *Let $D(M, P)$ be a decision problem. $D(M, P)$ is in a complexity class $\mathcal{C}$ iff there exists a sound, complete and terminating algorithm for $D(M, P)$ that consumes at most $\mathcal{C}$ resources for any input $p \in P$.*

A direct approach to demonstrate membership in a complexity class $\mathcal{C}$ would be to try to construct a suitable Turing machine and analyse time and space usage. However, this is generally difficult and therefore the notion of a *reduction* from one problem to another problem is introduced.

Definition 2.17 (Polynomial reduction). *Let $D(M_1, P_1)$ and $D(M_2, P_2)$ be two decision problems. The problem $D(M_1, P_1)$ is polynomially reducible to the problem $D(M_2, P_2)$ written $D(M_1, P_1) \leq D(M_2, P_2)$ if there is a map $f : P_1 \rightarrow P_2$ such that for all $p \in P_1$:*

$$p \in M_1 \text{ iff } f(p) \in M_2$$

The intuition here is that as long as constructing the map f is efficient, in the sense it is insignificant when compared to the complexity of the problems, a reduction can be used to establish bounds on complexity class membership by establishing reductions to (or from) problems with known complexity.

For example, suppose that there exists a reduction from the problem $d_1 = D(M_1, P_1)$ to the problem $d_2 = D(M_2, P_2)$ and d_2 is known to be in a complexity class $\mathcal{C}$. We can decide any problem from d_1 by reducing it to a problem in d_2 and then applying the decision procedure for d_2 . This reduction shows that the complexity class $\mathcal{C}$ is an upper bound on the complexity class for d_1 .

To understand the lower bound of complexity we use the notion of a problem being hard for a class. Informally, a problem is *hard* for a class $\mathcal{C}$, written $\mathcal{C}$ -hard, if the problem is “at least as hard as deciding the hardest problems in $\mathcal{C}$ ”. We can show that a problem d is hard for a class $\mathcal{C}$, if we can find a reduction for every problem in the class $\mathcal{C}$ to the problem d . Now, suppose that $d_2 = D(M_2, P_2)$ is known to be $\mathcal{C}$ -hard and d_2 can be reduced to $d_1 = D(M_1, P_1)$. Since every problem in d_2 can be answered by a reduction to a problem in d_1 , d_1 is at least $\mathcal{C}$ -hard because every problem in $\mathcal{C}$ can be reduced to a problem in d_2 and then to to a problem in d_1 . This reduction provides a lower bound on the complexity for d_1 .

A problem is said to be $\mathcal{C}$ -complete for the class $\mathcal{C}$ if it is a member of $\mathcal{C}$ and also $\mathcal{C}$ -hard. $\mathcal{C}$ -complete decision problems represent the hardest possible decision problems for $\mathcal{C}$. These notions are formalised in Definition 2.18

Definition 2.18 (Complexity Bounds). *Let d_1 and d_2 be decision problems and $\mathcal{C}$ be a complexity class.*

- *if $d_1 \leq d_2$ and d_2 is a member of $\mathcal{C}$ then d_1 is a member of $\mathcal{C}$ (upper bound)*
- *d_1 is $\mathcal{C}$ -hard iff for every decision problem $d_2 \in \mathcal{C}$ there exists a reduction $d_2 \leq d_1$.*
- *if $d_2 \leq d_1$ and d_2 is $\mathcal{C}$ -hard then d_1 is $\mathcal{C}$ -hard (lower bound)*
- *if d_1 is in $\mathcal{C}$ and is $\mathcal{C}$ -hard then d_1 is $\mathcal{C}$ -complete.*

The following additional complexity classes will be used in the comparison of the complexity of reasoning tasks in the next section: **EXP** (resp. **NEXP**) is the set of decidable decision problems that can be answered by a DTM (resp. NDTM) in a time that is exponential in the

size of the input. The classes LOGSPACE^2 (resp. EXPSPACE) denote problems that can be answered by a DTM using a space that is a logarithmic (exponential) in the size of the input. NLOGSPACE corresponds to sets of problems that can be answered by a NDTM in logarithmic space. The complexity classes discussed thus far can be expressed as a hierarchy of increasing computational complexity [AB09]:

$$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP} \subseteq \text{NEXP}$$

At the present time it is believed that all the inclusions are strict. Problems that belong to P are said to be computationally *tractable* [AB09].

2.3.1 Complexity of Description Logic reasoning tasks

We will use the complexity of deciding knowledge base satisfiability (with respect to the size of the ABox and the TBox) as a benchmark to compare the logics. Other reasoning tasks which can be re-expressed in terms of deciding knowledge base satisfiability inherit the same complexity. These include:

1. *Instance checking.* a is an instance of concept C in $\mathcal{K}$ iff $\mathcal{K} \cup \neg C(a)$ is unsatisfiable.
2. *Concept satisfiability.* C is satisfiable w.r.t. a TBox $\mathcal{T}$ iff $\langle C(a), \mathcal{T} \rangle$ is satisfiable.
3. *Concept subsumption.* $C \sqsubseteq D$ iff $\mathcal{K} \cup (C \sqcap \neg D)(a)$ is unsatisfiable.

On the other hand, algorithms for answering query tasks are generally harder than those for satisfiability. For example, concept retrieval can be achieved through repeated instance checks. However, in [CDL⁺07] the authors demonstrated how satisfiability and query answering in *DL-Lite* can be implemented through a *FOL-reduction*. The approach, known as Ontology Based Data Access (OBDA), allows the ABox to be stored in a relational database and answers are obtained by translating the problem expressed in terms of the TBox into a (first order) database

²The LOGSPACE and NLOGSPACE problems cannot be encoded on a 1-tape machine. A 2-tape or Random Access machine is required (see [AB09]).

query. Since the translation is efficient when the TBox is small, the complexity of reasoning in the presence of large ABoxes is dominated by the complexity of answering the database query. To distinguish the complexity characteristics that are most relevant for a particular application, the complexity results for a reasoning task are quoted in three different forms [Var82]. *Data complexity* defines the complexity with respect to the input size of the ABox, *expression complexity* defines the complexity with respect to the input size of the TBox, and the *combined complexity* defines the complexity with respect to the input size of the ABox and the TBox. Table 2.5 shows the combined complexity of knowledge base satisfiability checking

$\mathcal{L}$	Satisfiability
$DL-Lite_{core}$	$\leq \text{NLOGSPACE}$
$DL-Lite_{\mathcal{R}}$	$\leq \text{NLOGSPACE}$
OWL 2 QL	NLOGSPACE-complete
$DL-Lite_{\mathcal{F}}(\text{UNA})$	NLOGSPACE-complete
$DL-Lite_{\mathcal{A}}(\text{UNA})$	NLOGSPACE-complete
$DL-Lite_{bool}^{\mathcal{N}}(\text{UNA})$	NLOGSPACE-complete
$DL-Lite_{\mathcal{F}}$	P-complete
$DL-Lite_{\mathcal{A}}$	P-complete
$DL-Lite_{bool}^{\mathcal{N}}$	$\leq \text{NP}$
$\mathcal{EL}$, OWL 2 EL	P-complete
$\mathcal{ALC}$ acyclic (or absent) TBox	PSPACE-complete
$\mathcal{ALC}$	EXP-complete
$\mathcal{SROIQ}(D)$, OWL 2 DL	NEXP-complete

Table 2.5: The combined complexity for knowledge base consistency checking. The complexities of the $DL-Lite$ logics are taken from [ACKZ09] Table 2; for the the OWL languages from [CCD⁺12]; and for $\mathcal{ALC}$ from the Description Logic Complexity Navigator [Zol].

for the Description Logics introduced in this section. For each of the $DL-Lite$ languages, the data-complexity for instance checking is in AC^0 [ACKZ09]. The precise definition of AC^0 is not important here, other than to state that $\text{AC}^0 \subset \text{LOGSPACE}$ and is therefore highly tractable. The data complexity of query checking for OWL 2 QL (and some of the other DL-Lite languages) is in AC^0 [ACKZ09] highlighting its suitability for problems with large ABoxes.

2.4 Inconsistency in Description Logic

In many applications the knowledge is collated from multiple sources and inconsistency inevitably arises. As noted in [Kal06], pinpointing the cause of inconsistency within a knowledge base intuitively corresponds to identifying minimal subsets of axioms that lead to a conflict. We formalise the notion of inconsistency in terms of *justifications* as given in [Kal06]. The original definition has been modified to reflect the notation used in this thesis.

Definition 2.19 (Justification [Kal06]). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a knowledge base and Z be an axiom such that $\mathcal{K} \models Z$. A set of axioms $\mathcal{J}$ is a justification for Z in $\mathcal{K}$ if $\mathcal{J} \subseteq \mathcal{T} \cup \mathcal{A}$, $\mathcal{J} \models Z$ and if $\mathcal{J}' \subset \mathcal{J}$ then $\mathcal{J}' \not\models Z$.*

A convenient technique for understanding the *inconsistencies in a knowledge base* $\mathcal{K}$ is to use the set of justifications for $\top \sqsubseteq \perp$ [HPS08]. This axiom is independent of the knowledge base signature and expressible in all Description Logics that allow $\perp$. The (justifications for) inconsistencies may be classified into three distinct types: those that involve both TBox and ABox axioms, those that involve only ABox axioms and those that involve only TBox axioms. Recalling the knowledge from Example 1.1, let $\mathcal{K}_{cars} =$

$$\left\langle \left\{ \begin{array}{lll} Drives(del, c1), & Drives(alice, c2), & Drives(bill, c3), \\ Yellow(c1), & Blue(c2), & (Blue \sqcap Yellow)(c3), \\ Reliant(c1), & Yellow(c2) & BMW(c3), \\ & BMW(c2) \end{array} \right\}, \left\{ \begin{array}{l} Reliant \sqsubseteq \neg Fast \\ BMW \sqsubseteq Fast \\ Blue \sqsubseteq \neg Yellow \end{array} \right\} \right\rangle$$

$\mathcal{K}_{cars}$ includes the following inconsistencies involving only ABox axioms, the first about Alice's car $c2$: the set $\{Blue(c2), Yellow(c2), Blue \sqsubseteq \neg Yellow\}$ and the second about Bill's car $c3$: the set $\{(Blue \sqcap Yellow)(c3), Blue \sqsubseteq \neg Yellow\}$.

In languages that support the negation of concepts it is possible to introduce inconsistencies that involve only ABox axioms. For example, adding the axiom $\neg Yellow(c3)$ or $(\forall Drives. \neg BMW)(alice)$ to $\mathcal{K}_{cars}$.

Inconsistencies involving only TBox axioms occur iff the TBox is unsatisfiable. For example, adding the axioms $\top \sqsubseteq BMW$ and $\neg Reliant \sqsubseteq \neg Fast$ to the TBox of $\mathcal{K}_{cars}$ makes the TBox unsatisfiable. No individual can satisfy each axiom in the set $\{\top \sqsubseteq BMW, BMW \sqsubseteq Fast, BMW \sqsubseteq \neg Reliant, \neg Reliant \sqsubseteq \neg Fast\}$. Related to inconsistency is the notion of *incoherence*:

Definition 2.20 (Incoherence). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a knowledge base with signature $\langle N_I, N_C, N_R \rangle$. $\mathcal{K}$ is incoherent iff there exists an unsatisfiable concept $A \in N_C$.*

A consistent knowledge base can be incoherent. For example, a knowledge base that includes the unsatisfiable class *Blue* and in which no individuals are asserted to be, or can be inferred to be, *Blue* may be consistent (provided there are no other sources of inconsistency). Such incoherence is undesirable since any later revision to the knowledge base that introduces as a consequence that an individual is *Blue* also renders the knowledge base inconsistent.

2.4.1 A survey of existing approaches for handling inconsistencies

We have conducted a survey of the literature and have identified many different approaches to handling inconsistency in Description Logics. Rather than simply grouping such approaches by the logic used, we group the approaches by the underlying problem being addressed. We refer to a knowledge base as a *CMT knowledge base* if it is written in the standard Description Logic syntax and was designed for the classical model theoretic semantics as described in sections 2.1 and 2.2. In contrast, a *non-CMT knowledge base* may include non-standard syntax and is designed specifically for some non-classical logic. We group the approaches according to following problems:

- 1) *Knowledge base repair*. Revise a CMT knowledge base such that consistency is restored and reasoning can be performed using a classical reasoner;
- 2) *Reasoning with planned exceptions*. Build and then draw conclusions from a non-CMT knowledge base;

- 3) *Inconsistency-tolerant reasoning*. Draw meaningful conclusions from a (possibly inconsistent) *CMT*-knowledge base.

Adopting 1) requires that the origin of the inconsistencies is established by computing justifications for the inconsistency and then the knowledge base is modified to effect the repair. Once repaired, inferences may be performed on the consistent knowledge base using classical reasoners. By contrast, 2) and 3) draw conclusions directly from the unmodified knowledge base and require the use of a bespoke reasoner. The key distinction between approaches 2) and 3) is the nature of the input knowledge base. The non-CMT knowledge bases used in 2) are designed specifically for the chosen non-classical logic, whereas 3) draws conclusions from a CMT knowledge base that may or may not be inconsistent.

Knowledge base repair

The aim of a repair strategy is to make (typically minimal) changes to an inconsistent knowledge base such that it is made consistent. In [FMK⁺08] the authors conduct a comprehensive survey of the origin of inconsistency. They concluded that inconsistencies could be characterised by understanding the nature of the changes that were made to the knowledge base and led to inconsistency. They grouped the reasons for making changes to a knowledge base into categories, and posit that the approach needed to make the necessary repairs varies by category. For instance, resolving heterogeneous knowledge bases (combining knowledge bases with extensive overlap but in a way that uses different terminology or different representation formats) may include strategies for identifying correspondence between different names used for the same entity or identifying logical transformations that relate one concept to another. In contrast, resolving inconsistencies that have been introduced when modifying a knowledge base (including authoring, editing, evolution and versioning) requires pinpointing the set of axioms that are involved in the inconsistency and making a (minimal) repair. We focus on this latter case as it is more closely related to our work.

Reiter's theory of diagnosis [Rei87] the notion of minimal repair is defined as a *minimal hitting set*, the smallest set of elements that when removed, consistency is restored. In [Kal06], the

hitting set algorithm is applied to justifications of a Description Logic knowledge base and used to identify sets of minimal repairs (axioms to be removed). Further granularity of repairs is obtained in [HPS08] by considering *precise* and *laconic justifications*. These provide insight into the sub-formulae within the axioms that contribute to inconsistency. Tools based on [HPS08] allow automation or semi-automation of repair. However, identifying a particular repair that resolves the inconsistency and leads to meaningful results is non-trivial. This problem becomes more acute as the number of different, and possibly interlinked, justifications increases. In some cases, repair may not be a viable option, for instance where access to revise a knowledge base is restricted. These challenges have stimulated research on semantics that draw conclusions from the knowledge base without repairing it.

Reasoning with planned exceptions

Inconsistency within a knowledge base can be “planned for” and incorporated as part of the design, for instance by introducing a notion of “typicality” to the TBox under the assumption that concept inclusion axioms are typically true but admit exceptions. Non-classical conclusions are then drawn from solutions that “minimise” the exceptions. Consider the scenario outlined in Example 2.2 below.

Example 2.2. *Tweety is a penguin. Birds are penguins, typically birds fly and typically penguins don't fly.*

We will use the symbol $\sqsubset$ to denote a *defeasible* concept inclusion, one that permits exceptions.

Example 2.2 can now be represented by the non-classical knowledge base

$$\mathcal{K}_{pen} = \langle \{Penguin(tweety)\}, \{Penguin \sqsubseteq Bird, Bird \sqsubset Fly, Penguin \sqsubset \neg Fly\} \rangle$$

Intuitively, $\mathcal{K}_{pen}$ has two minimal solutions: one in which Tweety flies and the other in which he does not. The TBox of $\mathcal{K}_{pen}$ is incoherent when viewed classically, by eliminating the defeasible axioms and replacing $\sqsubset$ by $\sqsubseteq$. The concept *Penguin* is unsatisfiable.

Early work on non-classical Description Logics in [BH95] and [KPK06] focussed on *Default Logic* [Rei80] but this approach does not accommodate knowledge bases with unnamed individuals, which was partially addressed in [SHJ15]. In [DNR02], epistemic operators were added to Description Logic following the approach of Minimal Knowledge and Negation as Failure (MKNF) [Lif91]. Combining *Defeasible Logic* [Nut87] with Description Logic was explored in [Gov04] but the language fragment supported, $\mathcal{ALC}^-$, was limited by the exclusion of support for existentially quantified concepts. More recently, logics based on *Preferential Logics* exploiting the KLM approach [KLM90] have been proposed. In [CMMV13b] and [CMMN14] implementations were evaluated against a range of synthetic knowledge bases. Non-monotonic Description Logics based on *Circumscription* [McC86] have been proposed in [BLW06], [GH08] and [BFPS15] but, to our knowledge, no implementations based on this work have been published to date. In [BFPS15] a *semantics for overriding* called $\mathcal{DL}^N$ is proposed. An implementation was developed that integrates existing classical Description Logic reasoners and was evaluated against medical $\mathcal{EL}^{++}$ knowledge bases in which defeasibility had been injected.

Table 2.6 summarises publications that permit planned exceptions through non-classical Description Logics.

Most of the work summarised in Table 2.6 focusses on accommodating exceptions in the TBox. In [BCMV13] and [CMMV13b] their technique was extended to allow inferences with respect to a supplied ABox. However, the facts asserted within the ABox are not permitted to be defeasible. In languages that include $\mathcal{O}$ (nominals) it is possible to eliminate the ABox by replacing each concept assertion $C(a)$ with $\{a\} \sqsubseteq C$ and each role assertion $R(a, b)$ with $\{a\} \sqsubseteq \exists R.\{b\}$. Once the ABox is expressed as part of the TBox, it is then possible to express these axioms defeasibly. This technique was explored in [BFPS15] and also in [GGOP13]. In [DNR02] the epistemic modal operators **K** and **A** may appear within concepts. Informally, **K** C can be read “known to be C ” and **A** C as “assumed to be C ” where the notion of assumption can be used to express defeasible knowledge. However, we are not aware of implementations of these approaches.

Reference	Basis	Language	Def.	Imp.
Default Logic				
[BH95]	Default Logic	$\mathcal{ALCF}$	$\mathcal{T}$	No
[KPK06]	Default Logic + weights	$\mathcal{SHOIN}$	$\mathcal{T}$	Yes
[DtEK09]	Default Logic + rules (Hybrid)	$\mathcal{ALC}$	$\mathcal{T}$	Yes
[SHJ15]	Default Logic + Free defaults	$\mathcal{ALC}$	$\mathcal{T}$	No
Auto Epistemic Logic				
[DNR02]	MKNF	$\mathcal{ALC}$	$\mathcal{A}, \mathcal{T}$	No
Defeasible Logic				
[Gov04]	Defeasible Logic	$\mathcal{ALC}^-$	$\mathcal{T}$	No
Preferential Logic				
[GGOP09]	KLM: Preferential, order objects ($\mathcal{ALC} + \mathbf{T}$)	$\mathcal{ALC}$	$\mathcal{T}$	No
[GGOP13]	KLM: Preferential, order objects ($\mathcal{ALC} + \mathbf{T}_{min}$)	$\mathcal{ALC}$	$\mathcal{A}, \mathcal{T}$	No
[BHM08]	KLM: Preferential, order objects	$\mathcal{ALC}$	$\bar{\mathcal{T}}$	No
[BMV11]	KLM: Preferential + Rational, order relations	$\mathcal{ALC}$	$\mathcal{T}$	No
[MMV12]	KLM: Rational + Protégé prototype	$\mathcal{ALC}$	$\mathcal{T}$	No
[BCMM13]	KLM: Rational + defeasible role restrictions	$\mathcal{ALC}$	$\mathcal{T}$	No
[BCMV13]	KLM: Rational + extended to the ABox	$\mathcal{ALC}$	$\mathcal{T}$	No
[CS10]	KLM: Rational, order individuals	$\mathcal{ALC}$	$\bar{\mathcal{T}}$	No
[CS12]	KLM: Lexicographic Closure	$\mathcal{ALC}$	$\mathcal{T}$	No
[CMMV13a]	KLM: Rational	$\mathcal{ALC}$	$\mathcal{T}$	Yes
[CMMV13b]	KLM: Rational + extended to the ABox	$\mathcal{ALC}$	$\mathcal{T}$	No
[CS13]	KLM: Rational + boolean inheritance nets	$\mathcal{ALC}$	$\mathcal{T}$	Yes
[CMMN14]	KLM: Relevant Closure	$\mathcal{ALC}$	$\mathcal{T}$	Yes
Circumscription				
[BLW06]	Circumscription	$\mathcal{ALCIO},$ $\mathcal{ALCQO}$	$\mathcal{T}$	No
[GH08]	Circumscription with tableau algorithm	$\mathcal{ALCO}$	$\mathcal{T}$	No
[BFPS15]	Circumscription with decidability results	$\mathcal{ALCF},$ $DL-Lite_F,$ $DL-Lite_R$	$\mathcal{T}$	No
Overriding				
[BFPS15]	$\mathcal{DL}^N$	$\mathcal{SROIQ}(D)$	$\mathcal{T}$	Yes

Table 2.6: A summary of publications that permit planned exceptions through non-classical Description Logics, indicating the language targeted, where defeasibility is accommodated and whether or not an implementation was developed. Entries are grouped by the underlying non-classical logic, sub-grouped by author and then ordered by date.

Inconsistency tolerant semantics

Under inconsistency tolerant semantics (also known as error tolerant semantics), the starting point is a CMT knowledge base that was designed for a classical reasoner and the goal is to draw meaningful conclusions from a classically inconsistent knowledge base. Conclusions are drawn by permitting some subset of the original axioms to be falsified. To provide a degree of

control over this process a syntactic provision may be included that identifies which axioms may be falsified. This may be further enhanced by defining a mechanism for ordering the selection.

Paraconsistency via multi-valued logics has been used to develop error-tolerant logics, for example in [PS89, Str97, MHL08, MH09] and [MMH13]. The last of these includes a review of these approaches. The system allows inconsistency to be accommodated within both the ABox and the TBox but the resulting logics are weak. For example, disjunctive syllogism fails: from $(A \sqcup B)(x)$ and $\neg A(x)$ we cannot conclude $B(x)$. To address this weakness quasi-classical logics (QC) have been proposed [ZXL09, HW09, ZXLV14]. Two distinct semantics are proposed: *weak-QC* and *strong-QC*. Weak-QC coincides with the logic $\mathcal{SHIQ4}$ proposed in [MHL08]. The strong variant of the QC semantics changes the semantics for conjunction and disjunction such that disjunctive syllogism is possible. A system based on strong-QC semantics was implemented for a range of Description Logics in [ZXLV14].

The *repair semantics* [LLR⁺10] addresses inconsistency tolerance for the *DL-Lite* family of logics. Entailment under the *closed ABox repair semantics* (ICAR) is defined in terms of maximally consistent consequences derivable from subsets of ABox axioms that are consistent with TBox axioms. This approach was later extended to consider TBox repairs in [LP14].

The methods discussed thus far do not offer precise control over the “arbitration” between conflicting consequences. This limits the set of consequences that can be derived from an inconsistent knowledge base in cases where alternative solutions cannot be distinguished. Enriching the semantics to differentiate solutions was explored through *argumentation theory* in [ZXL09] and by *distance based measures* in [ZWWQ15]. In contrast, arbitration can be controlled syntactically by augmenting the language with information that indicates measures of confidence in the axioms.

Embedding Possibilistic Logic [DLP94] within Description Logics permits each axiom to be labelled with a real number in the range $(0, 1]$. These labels indicate a measure of the necessity (or possibility) of the truth of the axiom. This approach has been used to accommodate inconsistency, for example in [Hol94, QP08, ZXL09, QJPD11, ZQS13, QZ13]. However, when Possibilistic Logic is applied to inconsistency tolerance a rather weak logic results, as noted by

Dubois and Prade in [DLP94]: “The handling of inconsistency in possibilistic logic is rather coarse: all formulas in $\mathcal{K}$ whose certainty level is equal to or less than the inconsistency level, are inhibited.”

Early work on probabilistic Description Logics in [KLP97] and [DPPY04] focussed on Bayesian Networks. More recently, in [Luk08] probabilistic logic is incorporated into the expressive Description Logic $\mathcal{SHOIN}(D)$. A probabilistic knowledge base is composed of classical TBox, a probabilistic TBox and one probabilistic ABox for each individual. The probabilistic TBox and ABoxes are expressed as *conditional constraints*. Informally, a conditional constraint written $(C|D)[l, u]$ states that the probability of an individual being in concept C given they are in concept D lies between l and u . The semantics are based on a probabilistic variant of default reasoning that utilises the rule of maximum specificity, preferring more specific information over less specific information, to choose between conflicting default statements. Lexicographic entailment is expressed in terms of conditional constraints in which the lower (upper) bound is established by ordering possible worlds by their respective probabilities and identifying the minimum (resp. maximum) probability that the condition holds. Issues that are not directly addressed in this work are the mechanisms for establishing the values of bounds assigned to the constraints and the impact of interdependence between conditional constraints.

Table 2.7 summarises recent work in inconsistency tolerant Description Logics. Our own work is most closely related to the inconsistency tolerant approaches summarised in Table 2.7 and, in particular, those based on the *Repair* semantics. In Chapter 7, we examine the nature of the relationship between our work and the various Repair semantics in greater depth.

2.5 Summary

In this section we have introduced the family of Description Logics relevant to OWL 2 and seen how the careful selection of language features can be used to construct logics that target particular applications. The inevitable presence of inconsistencies in real world environments, together with the pre-requisite for a knowledge base to be consistent under model theoretic

Who	Semantics	Language	False.	Imp.
Paraconsistent				
[PS89]	Paraconsistent	$\mathcal{ALCN}$	$\mathcal{A}, \mathcal{T}$	No
[Str97]	Paraconsistent	$\mathcal{ALC}$	$\mathcal{A}, \mathcal{T}$	No
[HW09]	Paraconsistent	$\mathcal{SHIQ}$	$\mathcal{A}, \mathcal{T}$	No
[MHL08]	Paraconsistent	$\mathcal{SHIQ}$	$\mathcal{A}, \mathcal{T}$	No
[MH09]	Paraconsistent	$\mathcal{SROIQ}$	$\mathcal{A}, \mathcal{T}$	No
[MMH13]	Paraconsistent	$\mathcal{SROIQ}$	$\mathcal{A}, \mathcal{T}$	Yes
[ZXL09]	QC Paraconsistent	$\mathcal{ALCNQ}$	$\mathcal{A}, \mathcal{T}$	No
[HW09]	QC Paraconsistent	$\mathcal{SHIQ}$	$\mathcal{A}, \mathcal{T}$	No
[ZXLV14]	QC Paraconsistent	$\mathcal{SHOIN}(D)$	$\mathcal{A}, \mathcal{T}$	Yes
Repair				
[LLR ⁺ 10]	Repair $AR, IAR, CAR, ICAR$	$DL-Lite_{\mathcal{A}}$	$\mathcal{A}$	No
[LP14]	Repair IAR	$\mathcal{EL}$	$\mathcal{T}$	Yes
[EL16]	Repair GR and LGR	$DL-Lite$	$\mathcal{A}, \mathcal{T}$	No
Argumentation				
[ZXL09]	Argumentation	$DL-Lite_{core}$	$\mathcal{A}, \mathcal{T}$	No
Distance measures				
[ZWWQ15]	Distance based (features)	$DL-Lite_{bool}^{\mathcal{N}}$	$\mathcal{A}, \mathcal{T}$	No
Possibilistic				
[Hol94]	Possibilistic	$\mathcal{ALCN}$	$\mathcal{A}, \mathcal{T}$	No
[QP08]	Possibilistic	$\mathcal{ALC}$	$\mathcal{A}, \mathcal{T}$	No
[QJPD11]	Possibilistic with implementation	$\mathcal{ALC}$	$\mathcal{A}, \mathcal{T}$	Yes
[ZQS13]	Possibilistic with Tableau	$\mathcal{ALCI}_{\mathcal{R}+}$	$\mathcal{A}, \mathcal{T}$	No
[QZ13]	Possibilistic (KLM preferential)	$\mathcal{ALC}$	$\mathcal{T}$	No
Probabilistic				
[KLP97]	Baysian Network	$\mathcal{ALN}$	$\mathcal{T}$	No
[DPY04]	Baysian Network	$\mathcal{ALN}$	$\mathcal{T}$	No
[Luk08]	Probabilistic with lexicographic entailment	$\mathcal{SHOIN}(D)$	$\mathcal{A}, \mathcal{T}$	No

Table 2.7: A summary of publications on inconsistency tolerant Description Logics, indicating the language targeted, which axioms may be falsified and whether or not an implementation was developed. Entries are grouped by the underlying non-classical logic, sub-grouped by author and then ordered by date.

semantics, has led to the development of three strategies for accommodating inconsistency. Eliminating inconsistency through repair is effective but it is challenging to select repairs that lead to meaningful results. Where the nature of inconsistency is well understood before a knowledge base is constructed, a non classical logic can be chosen and then a non-CMT knowledge constructed that takes into account the planned exceptions. If the starting point is an inconsistent CMT knowledge base then inconsistency-tolerant semantics can be employed to draw meaningful conclusions from the knowledge base. However, research into such semantics has exposed a number of challenges. The logics may be too weak for useful inference, they may

restrict where inconsistency can be admitted (ABox vs. TBox) and offer little or no control over how the arbitration of inconsistencies is managed. These challenges has driven the development of our $p\text{-}\mathcal{ALC}$.

Chapter 3

Preferential $\mathcal{ALC}$

This chapter introduces our inconsistency tolerant *Preferential $\mathcal{ALC}$* ($p\text{-}\mathcal{ALC}$). We begin with justifications for the key choices that were made during the development of this logic. The formal syntax and semantics for $p\text{-}\mathcal{ALC}$ are given next. Finally, the important properties of the logic are presented and we show that inferences under the $p\text{-}\mathcal{ALC}$ semantics can be performed using refutation style proofs.

As discussed in Chapter 2, different Description Logics exist. In this thesis, we have selected $\mathcal{ALC}$ as our base logic, for two reasons. Firstly, it provides the full range of boolean concepts constructors together with unrestricted use of both existential and universal quantifiers. As such, a broad range of problems can be modelled and it subsumes the important logic $\mathcal{EL}$. Secondly, many reasoning techniques for more expressive logics are based on extensions to the reasoning algorithms developed for $\mathcal{ALC}$. The existence of documented pathways to extend these algorithms suggests a research pathway for the development of inconsistency tolerant reasoning in more expressive logics.

In Section 2.4 we saw that inconsistency in a knowledge base may originate from sets of TBox and/or ABox axioms that cannot be satisfied in any interpretation. Intuitively, to accommodate inconsistency and permit meaningful inference to be performed we must be prepared to consider falsifying some axioms during the reasoning process. As noted in Chapter 1, inconsistency of the TBox is straightforward to repair and we focus on the harder problem of inconsistencies

relating to the ABox. Under the assumption that the TBox is consistent, a knowledge base could include inconsistencies involving TBox and ABox axioms or inconsistencies involving only ABox axioms. In $p\text{-}\mathcal{ALC}$, inconsistency can be accommodated through a notion of *defeasible axioms*. By allowing both ABox axioms and TBox axioms to be defeasible, $p\text{-}\mathcal{ALC}$ offers flexibility in how inconsistency tolerance is achieved. A defeasible axiom is defined as an $\mathcal{ALC}$ axiom associated with a positive integer weight w that indicates a measure of confidence in the truth of the axiom, the higher the value the greater the confidence. In the sequel, we will see how the weights are used to provide control over conflict arbitration.

3.1 Syntax

The ABox and the TBox in a $p\text{-}\mathcal{ALC}$ knowledge base are each divided into two parts, the non-defeasible part and the defeasible part.

Definition 3.1 (Knowledge base). *A $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$ is a tuple $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$, where $\mathcal{A}$ and $\mathcal{T}$ are, respectively, finite (possibly empty) sets of non-defeasible ABox and TBox axioms, and $\mathcal{A}_d$ and $\mathcal{T}_d$ are, respectively, finite (possibly empty) sets of defeasible ABox and TBox axioms such that $\mathcal{A}$ and $\mathcal{A}_d$ are disjoint and $\mathcal{T}$ and $\mathcal{T}_d$ are also disjoint. If $\langle \mathcal{A}, \mathcal{T} \rangle$ is satisfiable over the signature of $\mathcal{K}$, then $\mathcal{K}$ is said to be credible.*

The notation $Z^{[w]}$ denotes a defeasible axiom where Z is an $\mathcal{ALC}$ axiom and w is the weight. Given a set $\mathcal{S}$ of defeasible axioms, the notation $\mathcal{S}^{-W}$ will be used to refer to $\{Z | Z^{[w]} \in \mathcal{S}\}$, the axioms without weights. In general, weights of the defeasible axioms do not have to be equal, but in the case where they are all equal, a $p\text{-}\mathcal{ALC}$ knowledge base is said to be *uniform*.

The notion of credibility is used to exclude any knowledge base that contains a set of non-defeasible axioms that is inconsistent. Such a set has no mechanism to accommodate the inconsistency because it includes no defeasible axioms. In what follows, a $p\text{-}\mathcal{ALC}$ knowledge base will be assumed to be credible unless otherwise stated.

The syntax of $p\text{-}\mathcal{ALC}$ is illustrated through Example 3.1:

Example 3.1. *The knowledge base $\mathcal{K}_{cars}$ from Example 1.1 can be re-expressed as the $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}_{\overline{cars}} = \langle \mathcal{A}_{3.1}, \mathcal{T}_{3.1}, \mathcal{A}_{d\ 3.1}, \mathcal{T}_{d\ 3.1} \rangle$ where:*

$$\begin{aligned} \mathcal{A}_{3.1} &= \left\{ \begin{array}{l} Drives(del, c1), Yellow(c1), Reliant(c1), \\ Drives(alice, c2), BMW(c2), \\ Drives(bill, c3), (Blue \sqcap Yellow)(c3), BMW(c3) \end{array} \right\} \\ \mathcal{T}_{3.1} &= \left\{ Reliant \sqsubseteq \neg Fast, BMW \sqsubseteq Fast \right\} \\ \mathcal{A}_{d\ 3.1} &= \left\{ Blue(c2)^{[1]}, Yellow(c2)^{[1]} \right\} \\ \mathcal{T}_{d\ 3.1} &= \left\{ Blue \sqsubseteq \neg Yellow^{[2]} \right\} \end{aligned}$$

$\mathcal{K}_{\overline{cars}}$ is credible because $\langle \mathcal{A}_{3.1}, \mathcal{T}_{3.1} \rangle$ is consistent. Informally, $\mathcal{K}_{\overline{cars}}$ indicates that our confidence that no object can be both blue and yellow (assigned 2) is stronger than the confidence that we know the colour of Alice's car (each is assigned 1). In the absence of any other information, we might conclude that Alice's car $c2$ is either blue or yellow but not both; whereas Bill's car $c3$ is both blue and yellow.

The sub-formulae in a $p\text{-}\mathcal{ALC}$ knowledge base are all the sub-formulae of the $p\text{-}\mathcal{ALC}$ concepts that appear within the axioms of the knowledge base.

Definition 3.2 (Sub-formulae of a $p\text{-}\mathcal{ALC}$ knowledge base). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a $p\text{-}\mathcal{ALC}$ knowledge base. The sub-formulae of $\mathcal{K}$, denoted $\mathcal{F}(\mathcal{K})$, are defined recursively:*

- If $C(a) \in \mathcal{A}$ or $C(a)^{[w]} \in \mathcal{A}_d$ then C is a sub-formula of $\mathcal{K}$;
- If $C \sqsubseteq D \in \mathcal{T}$ or $C \sqsubseteq D^{[w]} \in \mathcal{T}_d$ then $\neg C \sqcup D$ is a sub-formula of $\mathcal{K}$;
- if C is a sub-formula in $\mathcal{K}$ and D is a sub-formula of C then D is a sub-formula in $\mathcal{K}$.

The semantics for $p\text{-}\mathcal{ALC}$ are given next.

3.2 Semantics

The semantics of $p\text{-}\mathcal{ALC}$ extends the notion of $\mathcal{ALC}$ interpretations to the defeasible axioms and introduces a notion of “distance of an interpretation”. Informally, the distance of the interpretation quantifies the penalties paid for falsifying defeasible axioms in the interpretation.

Definition 3.3 (Distance of an Interpretation). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a $p\text{-}\mathcal{ALC}$ knowledge base. A ($p\text{-}\mathcal{ALC}$) interpretation of $\mathcal{K}$ is an $\mathcal{ALC}$ interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ of $\langle \mathcal{A} \cup \mathcal{A}_d^{-W}, \mathcal{T} \cup \mathcal{T}_d^{-W} \rangle$. The set of unsatisfied instances of a defeasible axiom $Z^{[w]} \in \mathcal{A}_d \cup \mathcal{T}_d$ with respect to $\mathcal{I}$, denoted $U(Z^{[w]}, \mathcal{I})$, is defined as follows:*

$$U(Z^{[w]}, \mathcal{I}) = \begin{cases} \{ \langle C(x)^{[w]}, x^{\mathcal{I}} \rangle \}, & \text{if } \mathcal{I} \not\models C(x) & \text{where } Z^{[w]} \text{ is } C(x)^{[w]} \\ \{ \langle R(x, y)^{[w]}, x^{\mathcal{I}} \rangle \}, & \text{if } \mathcal{I} \not\models R(x, y) & \text{where } Z^{[w]} \text{ is } R(x, y)^{[w]} \\ \{ \langle C \sqsubseteq D^{[w]}, u \rangle \mid (u \in C^{\mathcal{I}}) \wedge (u \notin D^{\mathcal{I}}) \} & & \text{where } Z^{[w]} \text{ is } C \sqsubseteq D^{[w]} \end{cases}$$

The set of unsatisfied instances of defeasible axioms in $\mathcal{K}$ with respect to $\mathcal{I}$, denoted $\mathcal{U}(\mathcal{K}, \mathcal{I})$, is defined by

$$\mathcal{U}(\mathcal{K}, \mathcal{I}) = \bigcup_{Z^{[w]} \in \mathcal{A}_d \cup \mathcal{T}_d} U(Z^{[w]}, \mathcal{I}).$$

The distance of the interpretation $\mathcal{I}$, denoted $d(\mathcal{U}(\mathcal{K}, \mathcal{I}))$, is given by

$$d(\mathcal{U}(\mathcal{K}, \mathcal{I})) = \sum_{\langle Z^{[w]}, u \rangle \in \mathcal{U}(\mathcal{K}, \mathcal{I})} w.$$

Defeasible ABox axioms and instances of defeasible TBox axioms that are falsified by an interpretation are said to be defeated. An interpretation $\mathcal{I}$ of $\mathcal{K}$ is said to be n -distant if $n = d(\mathcal{U}(\mathcal{K}, \mathcal{I}))$.

Informally, the distance of an interpretation $\mathcal{I}$ is the sum of the weights of all defeasible axioms of $\mathcal{A}_d$ and all defeasible axiom instances of $\mathcal{T}_d$ that are not satisfied by $\mathcal{I}$. ABox axioms are treated *atomically*, since each axiom is either satisfied or unsatisfied in an interpretation.

Different is the case of defeasible TBox axioms. If TBox axioms were treated atomically, a single individual falsifying the axiom in a given interpretation would lead to all other instances of the TBox to be defeated even though the interpretation would not necessarily falsify them. In our notion of distance we consider falsification of *instances* of defeasible TBox axioms. Each defeasible TBox axiom $C \sqsubseteq D^{[w]}$ is treated as if it were a set of independent axioms of the form $(\neg C \sqcup D)(a)^{[w]}$. Here, $a \in N_I \cup N_U$ where N_U denotes a set of unique names for unnamed domain elements in $\Delta^{\mathcal{I}}$.

Example 3.1 (continued). Let $\mathcal{I}_1$ be a Herbrand interpretation of $\mathcal{K}_{\overline{cars}}$ where $Blue^{\mathcal{I}_1} = \{c2, c3\}$, $Yellow^{\mathcal{I}_1} = \{c1, c3\}$, $Reliant^{\mathcal{I}_1} = \{c1\}$, $BMW^{\mathcal{I}_1} = \{c2, c3\}$, $Fast^{\mathcal{I}_1} = \{c2, c3\}$ and $Drives^{\mathcal{I}_1} = \{(del, c1), (alice, c2), (bill, c3)\}$. $\mathcal{I}_1$ clearly satisfies the non-defeasible axioms in $\mathcal{A}$ and $\mathcal{T}$. Similarly, the following defeasible axioms and instances are satisfied:

- $Blue(c2)^{[1]}$ because $\mathcal{I}_1 \models Blue(c2)$;
- the instances of $Blue \sqsubseteq \neg Yellow^{[2]}$ for $c1, c2, del, alice$ and $bill$:
 $\mathcal{I}_1 \models \neg Blue(c1)$ and $\mathcal{I}_1 \models Yellow(c1)$; now $\mathcal{I}_1 \models (\neg Blue \sqcup \neg Yellow)(c1)$;
 $\mathcal{I}_1 \models Blue(c2)$ and $\mathcal{I}_1 \models \neg Yellow(c2)$; now $\mathcal{I}_1 \models (\neg Blue \sqcup \neg Yellow)(c2)$;
 and each of $del, alice$ and $bill$ are neither blue nor yellow and the corresponding instance is satisfied.

However:

- $Yellow(c2)^{[1]}$ is falsified because $\mathcal{I}_1 \models \neg Yellow(c2)$ and
- the instance of $Blue \sqsubseteq \neg Yellow^{[2]}$ for $c3$ is falsified:
 $\mathcal{I}_1 \models Blue(c3)$ and $\mathcal{I}_1 \models Yellow(c3)$; now $\mathcal{I}_1 \not\models (\neg Blue \sqcup \neg Yellow)(c3)$.

We conclude that $\mathcal{I}_1$ is 3-distant.

Using the notion of distance of interpretations, a partial ordering relation, $\prec$, can be defined over the set of all interpretations of a $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$. Preferred interpretations

are those interpretations with minimal distance value. We will show later that every credible knowledge base has at least one preferred interpretation.

Definition 3.4 (Preferred interpretation). *Let $\mathcal{I}$ and $\mathcal{I}'$ be interpretations of a credible $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$. $\mathcal{I} \prec \mathcal{I}'$ if and only if $d(\mathcal{U}(\mathcal{K}, \mathcal{I})) < d(\mathcal{U}(\mathcal{K}, \mathcal{I}'))$. $\mathcal{I}$ is said to be a preferred interpretation of $\mathcal{K}$ if and only if (i) $\mathcal{I}$ satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ and (ii) there is no other interpretation $\mathcal{I}'$ of $\mathcal{K}$ that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ such that $\mathcal{I}' \prec \mathcal{I}$. $\mathcal{K}$ is said to be n -inconsistent if the preferred interpretations of $\mathcal{K}$ are n -distant. $\mathcal{K}$ is said to be consistent iff $n = 0$.*

The n -inconsistency of a $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$ provides a measure of the inconsistency in $\mathcal{K}$. The entailment relation of $p\text{-}\mathcal{ALC}$ is based on all preferred interpretations.

Example 3.1 (continued). $\mathcal{I}_1$ is a preferred interpretation of $\mathcal{K}_{\overline{\text{cars}}}$. That is, there are no 2-distant, 1-distant or 0-distant interpretations of $\mathcal{K}_{\overline{\text{cars}}}$. In Chapter 4 we will provide an algorithm that can be used to check this. There are preferred (3-distant) interpretations of $\mathcal{K}$ in which Alice drives a yellow car, $\text{Blue}(c2)^{[1]}$ is falsified and $\text{Yellow}(c2)^{[1]}$ is satisfied. Any interpretation that satisfies both $\text{Blue}(c2)^{[1]}$ and $\text{Yellow}(c2)^{[1]}$ also falsifies $\text{Blue} \sqsubseteq \neg \text{Yellow}^{[2]}$ for $c2$ and is therefore not a preferred interpretation of $\mathcal{K}$.

Definition 3.5 (Preferred consequence). *Let $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base and let Z be a non-defeasible axiom written in the language of $\mathcal{K}$. Z is a preferred consequence of $\mathcal{K}$, written $\mathcal{K} \models_p Z$, if and only if Z is satisfied in every preferred interpretation $\mathcal{I}$ of $\mathcal{K}$.*

For example, $\mathcal{K}_{\overline{\text{cars}}} \models_p (\text{Blue} \sqcup \text{Yellow})(c2)$. Role assertions can also accommodate inconsistency as illustrated by Example 3.2.

Example 3.2. *Let $\mathcal{K}_{dr1} = \langle \emptyset, \emptyset, \{R(a, b)^{[1]}, (\forall R.C)(a)^{[1]}, \neg C(b)^{[1]}\}, \emptyset \rangle$. Every interpretation of $\mathcal{K}_{dr1}$ fails to satisfy at least one defeasible ABox axiom. If we order all its interpretations according to their distance value, there are no 0-distant interpretations but there are 1-distant interpretations. $\mathcal{K}_{dr1}$ is therefore 1-inconsistent.*

Now consider the result of adding to $\mathcal{K}_{dr1}$ the defeasible assertion $R(a, a)^{[1]}$ to form $\mathcal{K}_{dr2} = \langle \emptyset, \emptyset, \{R(a, a)^{[1]}, R(a, b)^{[1]}, (\forall R.C)(a)^{[1]}, \neg C(b)^{[1]}\}, \emptyset \rangle$. Every interpretation of $\mathcal{K}_{dr2}$ must still

fail to satisfy at least one defeasible ABox axiom from the inconsistent set $\{R(a, b)^{[1]}, (\forall R.C)(a)^{[1]}, \neg C(b)^{[1]}\}$. The preferred interpretations, those that minimise the number of falsified axioms, are again 1-distant and we conclude that $\mathcal{K}_{dr2}$ is 1-inconsistent. In each preferred interpretation $R(a, a)^{[1]}$ is satisfied whereas in some preferred interpretations $R(a, b)^{[1]}$ is satisfied and in others $R(a, b)^{[1]}$ is falsified. We conclude that $\mathcal{K}_{dr2} \models R(a, a)$ and $\mathcal{K}_{dr2} \not\models R(a, b)$.

The atomic nature in which inconsistencies are accommodated by $\mathcal{A}_d$ leads to $p\text{-}\mathcal{ALC}$ being sensitive to the syntactic form of the assertional knowledge.

Example 3.3. Let $\mathcal{K}_{syn1} = \langle \emptyset, \{D \sqsubseteq E\}, \{C(a)^{[1]}, D(a)^{[1]}, \neg E(a)^{[1]}\}, \emptyset \rangle$ and

$$\mathcal{K}_{syn2} = \langle \emptyset, \{D \sqsubseteq E\}, \{(C \sqcap D)(a)^{[1]}, \neg E(a)^{[1]}\}, \emptyset \rangle.$$

$\mathcal{K}_{syn1}$ and $\mathcal{K}_{syn2}$ are both 1-inconsistent. However, $\mathcal{K}_{syn1} \models C(a)$ and $\mathcal{K}_{syn2} \not\models C(a)$. For $\mathcal{K}_{syn1}$ the inconsistent set is $\{D \sqsubseteq E, D(a)^{[1]}, \neg E(a)^{[1]}\}$ and in every preferred interpretation $\mathcal{I}$ of $\mathcal{K}_{syn1}$, $\mathcal{I} \models C(a)$. In contrast, the inconsistent set for $\mathcal{K}_{syn2}$ contains all its axioms. There are 1-distant interpretations of $\mathcal{K}_{syn2}$ that falsify $(C \sqcap D)(a)^{[1]}$ and in some of these interpretations, the interpretation $\mathcal{I}$ makes $\mathcal{I} \models \neg C(a)$.

The behaviour illustrated in Example 3.3 leaves the language flexible: axioms can be written in the form that best reflects the knowledge being represented.

The advantage of defining the distance of an interpretation through the notion of unsatisfied TBox axiom instances (as opposed to atomic TBox axioms) is illustrated in Example 3.4.

Example 3.4. Let $\mathcal{K}_{inc} = \langle \{C(a), \neg D(a)\}, \emptyset, \{C(b)^{[1]}\}, \{C \sqsubseteq D^{[1]}\} \rangle$. $\mathcal{K}_{inc}$ includes an inconsistency associated with a , the set $\{C(a), \neg D(a), C \sqsubseteq D^{[1]}\}$. The distance of a $p\text{-}\mathcal{ALC}$ interpretation takes into account each domain element for which a TBox axiom is defeated. For instance, the interpretation $\mathcal{I}_1$ where $C^{\mathcal{I}_1} = \{a^{\mathcal{I}_1}, b^{\mathcal{I}_1}\}$ and $D^{\mathcal{I}_1} = \{b^{\mathcal{I}_1}\}$ would be a 1-distant interpretation of $\mathcal{K}_{inc}$, whereas the interpretation $\mathcal{I}_2$, where $C^{\mathcal{I}_2} = \{a^{\mathcal{I}_2}\}$ and $D^{\mathcal{I}_2} = \emptyset$, would be a 2-distant interpretation. $\mathcal{I}_1$ would therefore be a preferred interpretation for which $D(b)$ would be true. $\mathcal{K}_{inc}$ is 1-inconsistent. $\mathcal{K}_{inc} \models C(b)$ and $\mathcal{K}_{inc} \models D(b)$. The consequence $D(b)$ due to the defeasible assertion $C(b)^{[1]}$ and (an instance of) the defeasible concept inclusion $C \sqsubseteq D^{[1]}$ is unaffected by the defeat of an instance of $C \sqsubseteq D^{[1]}$ for a .

To conclude this section, Example 3.5 illustrates a scenario concerning a patient support group in which a classically designed knowledge base is augmented with new knowledge that leads to inconsistency. The inconsistency is resolved by adding weights to selected axioms indicating the penalty applied if they are falsified, yielding a credible knowledge base. Once credible, consequences can be drawn from the knowledge base.

Example 3.5. *Let a and b be individuals. Consider the following assertional knowledge: “ a belongs to a group of patients”, $G(a)$; “ a is not sick”, $\neg S(a)$; “everyone that a refers to the group is not sick”, $(\forall R.\neg S)(a)$; “ b is sick”, $S(b)$; and “ b refers at least one patient to the group”, $(\exists R.G)(b)$. In addition, the terminological knowledge: “Healthy and sick are disjoint concepts”, $H \sqsubseteq \neg S$ and “patients in the group are healthy”, $G \sqsubseteq H$. The knowledge is represented by the classical $\mathcal{ALC}$ knowledge base*

$$\mathcal{K}_{pat} = \left\langle \left\{ \begin{array}{l} G(a), \neg S(a), (\forall R.\neg S)(a), \\ S(b), (\exists R.G)(b) \end{array} \right\}, \left\{ \begin{array}{l} H \sqsubseteq \neg S, \\ G \sqsubseteq H \end{array} \right\} \right\rangle$$

The following knowledge is added to the knowledge base $\mathcal{K}_{pat}$: “everybody that b refers is sick” $(\forall R.S)(b)$ and “ a refers b ”, $R(a, b)$. The resultant knowledge base is inconsistent because it includes two sets of axioms leading to inconsistencies: $\{(\forall R.\neg S)(a), S(b), R(a, b)\}$ which contains only ABox axioms; and $\{(\forall R.S)(b), (\exists R.G)(b), H \sqsubseteq \neg S, G \sqsubseteq H\}$ which includes both ABox and TBox axioms. To accommodate these inconsistencies the axioms $R(a, b)^{[1]}$ and $G \sqsubseteq H^{[1]}$ are made defeasible with equal weight [1]. Now the knowledge is represented by the credible $p\text{-}\mathcal{ALC}$ knowledge base

$$\mathcal{K}_{\overline{pat}} = \left\langle \left\{ \begin{array}{l} G(a), \neg S(a), (\forall R.\neg S)(a), \\ S(b), (\exists R.G)(b), (\forall R.S)(b) \end{array} \right\}, \{H \sqsubseteq \neg S\}, \{R(a, b)^{[1]}\}, \{G \sqsubseteq H^{[1]}\} \right\rangle$$

The original knowledge modelled by $\mathcal{K}_{pat}$ and the modified knowledge modelled by $\mathcal{K}_{\overline{pat}}$ are shown in Figure 3.1.

In $\mathcal{K}_{\overline{pat}}$, every preferred interpretation must satisfy the non defeasible axioms. Any preferred interpretation of $\mathcal{K}_{\overline{pat}}$, that satisfies $(\forall R.\neg S)(a)$ and $S(b)$ must falsify, and therefore defeat,

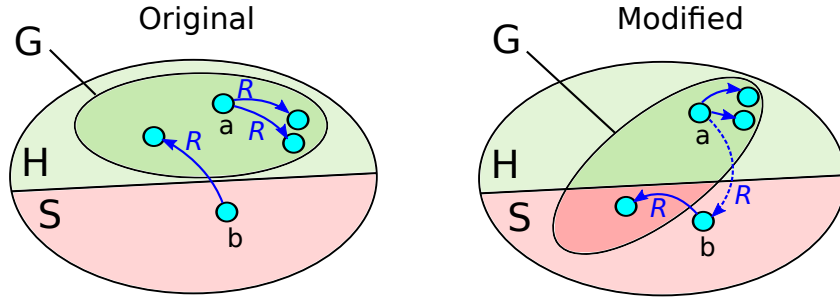


Figure 3.1: A visualisation of original knowledge modelled by $\mathcal{K}_{pat}$ and modified knowledge modelled by $\mathcal{K}_{\overline{pat}}$.

$R(a, b)^{[1]}$. But any such preferred interpretation must also satisfy $(\exists R.G)(b)$. So there is some (named or unnamed) individual x in the domain for which $R(b, x)$ and $G(x)$ are satisfied. By $(\forall R.S)(b)$, $S(x)$ is also satisfied and since $(\neg H \sqcup \neg S)(x)$ is satisfied, so is $\neg H(x)$. Thus, $(\neg G \sqcup H)(x)$ is not satisfied (i.e. $G \sqsubseteq H^{[1]}$ is defeated). Hence, each preferred interpretation must be at least 2-distant. Hence, $\mathcal{K}_{\overline{pat}}$ is 2-inconsistent, and since in every preferred interpretation $G \sqsubseteq H^{[1]}$ is satisfied for a , we have that $\mathcal{K}_{\overline{pat}} \models H(a)$. From the above argument we have $\mathcal{K}_{pat} \models S(b)$ and $\mathcal{K}_{pat} \models (\exists R.(G \sqcap S))(b)$. The latter follows because each preferred interpretation includes some individual reified as x for which $G(x)$ and $S(x)$ are satisfied. We can infer that a is healthy, because he/she belongs to the group, but there is at least one unhealthy individual within this group who was referred by b .

Considering instead $S(b)$ to be defeasible would allow for the possibility that b might not be sick. Let $\mathcal{K}'_{pat}$ be the knowledge base formed by replacing $S(b)$ in $\mathcal{K}_{pat}$ by the defeasible axiom $S(b)^{[1]}$. Now there are 2-distant preferred interpretations in which b is sick and not referred by a and others in which b is healthy and was referred by a . With these changes $\mathcal{K}'_{pat} \not\models H(b)$ and $\mathcal{K}'_{pat} \not\models S(b)$. The conflict can be arbitrated by choosing a higher weight for axiom $S(b)^{[1]}$ or $R(a, b)^{[1]}$. For example, consider $\mathcal{K}''_{pat}$ obtained from $\mathcal{K}'_{pat}$ by replacing $R(a, b)^{[1]}$ with $R(a, b)^{[2]}$. Now, $\mathcal{K}''_{pat} \models \neg S(b)$. In an interpretation satisfying $S(b)^{[1]}$, $\neg H(b)$ is also satisfied from $H \sqsubseteq \neg S$, and $R(a, b)^{[2]}$ is defeated. But defeating $R(a, b)^{[2]}$ adds a weight of 2 and any such interpretation would be at least 3-distant and therefore not preferred.

3.3 Properties of $p\text{-}\mathcal{ALC}$

With the syntax and semantics of $p\text{-}\mathcal{ALC}$ now formalised, we introduce some core properties of the logic that will be used to underpin the development of reasoning algorithms for $p\text{-}\mathcal{ALC}$. From the intuition that inconsistent information should be accommodated by the defeasible axioms, we introduced the notion of a “credible” knowledge base, which required that the non-defeasible axioms are satisfiable. We show that for every credible $p\text{-}\mathcal{ALC}$ knowledge base there is an n -distant preferred interpretation of the knowledge base, for some non-negative integer n , from which it follows that a credible $p\text{-}\mathcal{ALC}$ knowledge base is n -inconsistent for some non-negative integer n . Finally, we show that if the n -inconsistency of a $p\text{-}\mathcal{ALC}$ knowledge base is known then proofs by refutation can be used to derive preferred consequences from the knowledge base.

3.3.1 Existence of preferred interpretations

In this section we prove the existence of preferred interpretations. We do this by showing that there is an upper bound on the size of the domains of the interpretations that need to be considered to identify a preferred interpretation. In particular, we first show that the finite model property of $\mathcal{ALC}$ concepts [Cal96] extends to $p\text{-}\mathcal{ALC}$ concepts (Lemma 3.6). Hence a credible $p\text{-}\mathcal{ALC}$ knowledge base has finite domain interpretations which satisfy the non-defeasible axioms, falsify a finite number of defeasible axiom instances and have a finite distance. We next introduce the notion of a *contraction* of an interpretation (Definition 3.7), which leads to confirming an upper bound on the size of the domain of interpretations that need to be considered to identify a preferred interpretation of a $p\text{-}\mathcal{ALC}$ knowledge base (Lemma 3.10).

Recall, the finite model property of $\mathcal{ALC}$ [Cal96], which states that “if a schema (or concept expression) admits a model, then it also admits one with a finite domain, and therefore reasoning with respect to unrestricted models amounts to reasoning with respect to finite ones” [Cal96]. $p\text{-}\mathcal{ALC}$ concepts also exhibit the finite model property, as stated in Lemma 3.6. In the following, we refer to an interpretation (model) having a finite domain as a finite interpretation (resp.

model).

Lemma 3.6 (Every satisfiable $p\text{-}\mathcal{ALC}$ concept is satisfiable in a finite interpretation). *Let $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$, and C be a satisfiable $p\text{-}\mathcal{ALC}$ concept written in the signature of $\mathcal{K}$. Then there exists a finite interpretation that satisfies C .*

Proof. The proof follows directly from the finite model property of $\mathcal{ALC}$ concepts and that the interpretation of (and therefore satisfaction of) a $p\text{-}\mathcal{ALC}$ concept is defined classically. $\square$

Definition 3.7 (Contraction of a $p\text{-}\mathcal{ALC}$ interpretation). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible knowledge base and $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be an m -distant finite interpretation of $\mathcal{K}$ that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ for some $m \geq 0$. Let $u \in \Delta^{\mathcal{I}}$ be an element and $v \in \Delta^{\mathcal{I}}$ be an unnamed element. $\mathcal{I}'$, a v -contraction of $\mathcal{I}$ w.r.t. u , is constructed as follows:*

- $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}} \setminus v$ and;
- for each element $w \in \Delta^{\mathcal{I}'}$:
 - (i) for each individual name $x \in N_I$: $x^{\mathcal{I}'} = w$ iff $x^{\mathcal{I}} = w$
 - (ii) for each $A \in N_C$: $A^{\mathcal{I}'} = A^{\mathcal{I}} \setminus \{v\}$
 - (iii) for each $R \in N_R$: $R^{\mathcal{I}'} = (R^{\mathcal{I}} \setminus \{(w, w') \mid w = v \text{ or } w' = v\}) \cup \{(w, u) \mid (w, v) \in R^{\mathcal{I}}\}$

Definition 3.7 is “asymmetric” in the sense that when we remove element v , the “inbound” relations $(w, v) \in R^{\mathcal{I}}$ are replaced by $(w, u) \in R^{\mathcal{I}}$ whereas the “outbound” relations $(v, w) \in R^{\mathcal{I}}$ are not. In Lemma 3.8, we prove the condition under which unnamed domain elements can be removed from an m -distant interpretation of a credible knowledge base without increasing the distance of the interpretation. Informally, these conditions capture where u satisfies the same concepts as v and therefore will agree on the existence of “outbound” relations. This accounts for the asymmetry in Definition 3.7.

Given an interpretation $\mathcal{I}$ of $\mathcal{K}$ and $w \in \Delta^{\mathcal{I}}$, the notation $\mathcal{S}(\mathcal{I}, w)$ denotes the subset of subformulae $\mathcal{F}(\mathcal{K})$ that are satisfied by w .

Lemma 3.8 (A p - $\mathcal{ALC}$ contraction preserves the satisfiability of axioms). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible knowledge base and $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be an m -distant finite interpretation of $\mathcal{K}$ that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ for some $m \geq 0$. Let $u \in \Delta^{\mathcal{I}}$ be an element and $v \in \Delta^{\mathcal{I}}$ be an unnamed element. Let $\mathcal{I}'$ be the v -contraction of $\mathcal{I}$ w.r.t. u . If $\mathcal{S}(\mathcal{I}, v) = \mathcal{S}(\mathcal{I}, u)$ then $\mathcal{I}'$ satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ and is m' -distant for some $m' \leq m$.*

Proof. Assume $\mathcal{S}(\mathcal{I}, v) = \mathcal{S}(\mathcal{I}, u)$. By Definition 3.7 $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}} \setminus v$. Let $w \in \Delta^{\mathcal{I}'}$. We show $\mathcal{S}(\mathcal{I}', w) = \mathcal{S}(\mathcal{I}, w)$ using structural induction over $C \in \mathcal{F}(\mathcal{K})$:

- Base Case. $C = A$ or $C = \neg A$, where $A \in N_C$. By Definition 3.7 (ii). $A \in \mathcal{S}(\mathcal{I}', w)$ iff $A \in \mathcal{S}(\mathcal{I}, w)$ and $\neg A \in \mathcal{S}(\mathcal{I}', w)$ iff $\neg A \in \mathcal{S}(\mathcal{I}, w)$.
- Inductive step. Assume as inductive hypothesis for concepts $C, D \in \mathcal{F}(\mathcal{K})$ that $C \in \mathcal{S}(\mathcal{I}', w)$ iff $C \in \mathcal{S}(\mathcal{I}, w)$ and $D \in \mathcal{S}(\mathcal{I}', w)$ iff $D \in \mathcal{S}(\mathcal{I}, w)$. The proof then goes by cases:
 - $C \sqcap D$. $C \sqcap D \in \mathcal{S}(\mathcal{I}', w) \leftrightarrow (C \in \mathcal{S}(\mathcal{I}', w) \text{ and } D \in \mathcal{S}(\mathcal{I}', w)) \leftrightarrow C \in \mathcal{S}(\mathcal{I}, w) \text{ and } D \in \mathcal{S}(\mathcal{I}, w)$, by the inductive hypothesis, $\leftrightarrow C \sqcap D \in \mathcal{S}(\mathcal{I}, w)$.
 - $C \sqcup D$. $C \sqcup D \in \mathcal{S}(\mathcal{I}', w) \leftrightarrow (C \in \mathcal{S}(\mathcal{I}', w) \text{ or } D \in \mathcal{S}(\mathcal{I}', w)) \leftrightarrow C \in \mathcal{S}(\mathcal{I}, w) \text{ or } D \in \mathcal{S}(\mathcal{I}, w)$, by the inductive hypothesis, $\leftrightarrow C \sqcup D \in \mathcal{S}(\mathcal{I}, w)$.
 - $\exists R.C$. $\exists R.C \in \mathcal{S}(\mathcal{I}', w) \leftrightarrow \exists R.C \in \mathcal{S}(\mathcal{I}, w)$:

The “only if” case:

Assume $\exists R.C \in \mathcal{S}(\mathcal{I}', w)$. By this assumption, there exists $w_2 \in \Delta^{\mathcal{I}'}$ s.t. $(w, w_2) \in R^{\mathcal{I}'}$ and $C \in \mathcal{S}(\mathcal{I}', w_2)$. By the induction hypothesis $C \in \mathcal{S}(\mathcal{I}, w_2)$. There are two subcases:

1. $w_2 \neq u$. Since $(w, w_2) \in R^{\mathcal{I}'}$, by Definition 3.7 (iii) $(w, w_2) \in R^{\mathcal{I}}$. $(w, w_2) \in R^{\mathcal{I}} \wedge C \in \mathcal{S}(\mathcal{I}, w_2) \Rightarrow \exists R.C \in \mathcal{S}(\mathcal{I}, w)$.
2. $w_2 = u$. Since $(w, u) \in R^{\mathcal{I}'}$, by Definition 3.7 (iii) $(w, u) \in R^{\mathcal{I}}$ or $(w, v) \in R^{\mathcal{I}}$.

Taking each case:

- * $(w, u) \in R^{\mathcal{I}}$. $(w, u) \in R^{\mathcal{I}} \wedge C \in \mathcal{S}(\mathcal{I}, u) \Rightarrow \exists R.C \in \mathcal{S}(\mathcal{I}, w)$.
- * $(w, v) \in R^{\mathcal{I}}$. Since $C \in \mathcal{S}(\mathcal{I}, u)$ and by assumption $\mathcal{S}(\mathcal{I}, u) = \mathcal{S}(\mathcal{I}, v)$, $C \in \mathcal{S}(\mathcal{I}, v)$. $(w, v) \in R^{\mathcal{I}} \wedge C \in \mathcal{S}(\mathcal{I}, v) \Rightarrow \exists R.C \in \mathcal{S}(\mathcal{I}, w)$.

The “if” case:

Assume $\exists R.C \in \mathcal{S}(\mathcal{I}, w)$. By this assumption, there exists $w_2 \in \Delta^{\mathcal{I}}$ s.t. $(w, w_2) \in R^{\mathcal{I}}$ and $C \in \mathcal{S}(\mathcal{I}, w_2)$. There are two subcases:

1. $w_2 \neq v$. Since $(w, w_2) \in R^{\mathcal{I}}$, by Definition 3.7 (iii) $(w, w_2) \in R^{\mathcal{I}'}$. By the induction hypothesis $C \in \mathcal{S}(\mathcal{I}', w_2)$. $(w, w_2) \in R^{\mathcal{I}'} \wedge C \in \mathcal{S}(\mathcal{I}', w_2) \Rightarrow \exists R.C \in \mathcal{S}(\mathcal{I}', w)$.
 2. $w_2 = v$. Since $(w, v) \in R^{\mathcal{I}}$, by Definition 3.7 (iii) $(w, u) \in R^{\mathcal{I}'}$. By assumption $\mathcal{S}(\mathcal{I}, u) = \mathcal{S}(\mathcal{I}, v)$ therefore $C \in \mathcal{S}(\mathcal{I}, u)$. By the induction hypothesis $C \in \mathcal{S}(\mathcal{I}', u)$. $(w, u) \in R^{\mathcal{I}'} \wedge C \in \mathcal{S}(\mathcal{I}', u) \Rightarrow \exists R.C \in \mathcal{S}(\mathcal{I}', w)$.
- $\forall R.C$. We show $\forall R.C \in \mathcal{S}(\mathcal{I}', w) \leftrightarrow \forall R.C \in \mathcal{S}(\mathcal{I}, w)$ by contrapositive:

The “only if” case:

Assume $\forall R.C \notin \mathcal{S}(\mathcal{I}, w)$. There exists some $w_2 \in \Delta^{\mathcal{I}}$ s.t. $(w, w_2) \in R^{\mathcal{I}}$ and $C \notin \mathcal{S}(\mathcal{I}, w_2)$. There are two subcases:

1. $w_2 \neq v$. Since $(w, w_2) \in R^{\mathcal{I}}$, by Definition 3.7 (iii) $(w, w_2) \in R^{\mathcal{I}'}$. By the induction hypothesis $C \notin \mathcal{S}(\mathcal{I}', w_2)$. $(w, w_2) \in R^{\mathcal{I}'} \wedge C \notin \mathcal{S}(\mathcal{I}', w_2) \Rightarrow \forall R.C \notin \mathcal{S}(\mathcal{I}', w)$.
2. $w_2 = v$. Since $(w, v) \in R^{\mathcal{I}}$, by Definition 3.7 (iii) $(w, u) \in R^{\mathcal{I}'}$. By assumption $\mathcal{S}(\mathcal{I}, u) = \mathcal{S}(\mathcal{I}, v)$ therefore $C \notin \mathcal{S}(\mathcal{I}, u)$. By the induction hypothesis $C \notin \mathcal{S}(\mathcal{I}', u)$. $(w, u) \in R^{\mathcal{I}'} \wedge C \notin \mathcal{S}(\mathcal{I}', u) \Rightarrow \forall R.C \notin \mathcal{S}(\mathcal{I}', w)$.

The “if” case:

Assume $\forall R.C \notin \mathcal{S}(\mathcal{I}', w)$. There exists some $w_2 \in \Delta^{\mathcal{I}'}$ s.t. $(w, w_2) \in R^{\mathcal{I}'}$ and $C \notin \mathcal{S}(\mathcal{I}', w_2)$. By the induction hypothesis $C \notin \mathcal{S}(\mathcal{I}, w_2)$. There are two subcases:

1. $w_2 \neq u$. Since $(w, w_2) \in R^{\mathcal{I}'}$, by Definition 3.7 (iii) $(w, w_2) \in R^{\mathcal{I}}$. $(w, w_2) \in R^{\mathcal{I}} \wedge C \notin \mathcal{S}(\mathcal{I}, w_2) \Rightarrow \forall R.C \notin \mathcal{S}(\mathcal{I}, w)$.
2. $w_2 = u$. Since $(w, u) \in R^{\mathcal{I}'}$, by Definition 3.7 (iii) $(w, u) \in R^{\mathcal{I}}$ or $(w, v) \in R^{\mathcal{I}}$.

Taking each case:

- * $(w, u) \in R^{\mathcal{I}}$. $(w, u) \in R^{\mathcal{I}} \wedge C \notin \mathcal{S}(\mathcal{I}, u) \Rightarrow \forall R.C \notin \mathcal{S}(\mathcal{I}, w)$.
- * $(w, v) \in R^{\mathcal{I}}$. Since $C \notin \mathcal{S}(\mathcal{I}, u)$ and by assumption $\mathcal{S}(\mathcal{I}, u) = \mathcal{S}(\mathcal{I}, v)$,

$$C \notin \mathcal{S}(\mathcal{I}, v). (w, v) \in R^{\mathcal{I}} \wedge C \notin \mathcal{S}(\mathcal{I}, v) \Rightarrow \forall R. C \notin \mathcal{S}(\mathcal{I}, w).$$

Every domain element in $\Delta^{\mathcal{I}'}$ satisfies the same sub-formulae as in $\Delta^{\mathcal{I}}$ and by Definition 3.7 (i) every individual name is mapped to a domain element in $\mathcal{I}'$ iff only it is mapped to that element in $\mathcal{I}$. We conclude $\langle \mathcal{A}, \mathcal{T} \rangle$ is satisfied in $\mathcal{I}'$. Suppose element v falsifies f axiom instances of $\mathcal{A}_d \cup \mathcal{T}_d$ in $\mathcal{I}$. These instances correspond to some overall distance d of $\mathcal{I}$ such that $d \geq f$. Since $\mathcal{I}'$ falsifies the same instances at all other elements ($\neq v$) $m' = m - f$. We conclude $\mathcal{I}'$ is m' distant for some $m' \leq m$. $\square$

Clearly, a finite interpretation of a credible knowledge base falsifies some finite number, m , of axiom instances. In fact, there is a finite upper bound on the size of domains that must be considered to establish a minimum value of m as we now show. The intuition is that since $\mathcal{K}$ has a finite number of sub-formulae, the number of possible permutations of the sub-formulae that are satisfied at any given domain element is also finite. We use the notion of contraction (Definition 3.7) to show the existence of the upper bound.

Definition 3.9. Let $\mathcal{I}$ be an m -distant interpretation of a credible knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ and $|\Delta^{\mathcal{I}}| = k$. $\mathcal{I}$ is a k - m -interpretation of $\mathcal{K}$ if $\mathcal{I}$ is a model of $\langle \mathcal{A}, \mathcal{T} \rangle$. $\mathcal{I}$ is a smallest k - m -interpretation of $\mathcal{K}$ if there does not exist $\mathcal{I}'$, a k - m' -interpretation of $\mathcal{K}$, where $m' < m$.

Lemma 3.10 (Upper bound on the size of the domain). Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible p - $\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$. Then, there exists $\mathcal{I}, k, m_{\mathcal{I}}$, such that $\mathcal{I}$ is a smallest k - $m_{\mathcal{I}}$ -interpretation of $\mathcal{K}$ for which there does not exist any l - $m_{\mathcal{J}}$ -interpretation of $\mathcal{K}$ where $l > k$ and $m_{\mathcal{J}} < m_{\mathcal{I}}$.

Proof. Assume by contradiction that there exists $\mathcal{J}, l, m_{\mathcal{J}}$, such that $\mathcal{J}$ is an l - $m_{\mathcal{J}}$ -interpretation of $\mathcal{K}$, $l > k$ and $m_{\mathcal{J}} < m_{\mathcal{I}}$. (*)

Now consider a very special $k = \max(2^{|\mathcal{F}(\mathcal{K})|}, |N_I|)$, called $k_{\max}$, and some interpretation $\mathcal{I}_{\max}$ which is $m_{\mathcal{I}}$ -distant, where $m_{\mathcal{I}}$ is the smallest distance of all interpretations with $|\Delta^{\mathcal{I}}| = k_{\max}$. From (*) there exists $\mathcal{J}, l, m_{\mathcal{J}}$, such that $\mathcal{J}$ is an l - $m_{\mathcal{J}}$ -interpretation of $\mathcal{K}$, $l > k_{\max}$ and $m_{\mathcal{J}} < m_{\mathcal{I}}$.

Suppose such a $\mathcal{J}$ has a domain size $k_{big} > k_{max}$. By the unique names assumption every individual name from N_I is assigned to a unique element in $\Delta^{\mathcal{J}}$. Every unnamed element $w \in \Delta^{\mathcal{J}}$ satisfies some subset of the sub-formulae of $\mathcal{K}$. The maximum possible number of such subsets is $2^{|\mathcal{F}(K)|}$. Hence, in any interpretation with a domain size $k_{big} > k_{max} = \max(2^{|\mathcal{F}(K)|}, |N_I|)$ there is at least one pair of elements $u, v \in \Delta^{\mathcal{J}}$ where v is an unnamed element, that satisfy the same sub-formulae in $\mathcal{J}$. Let $\mathcal{J}'$ be the v -contraction of $\mathcal{J}$ w.r.t u . By Lemma 3.8 $\mathcal{J}'$ is $m_{\mathcal{J}'}$ -distant where $m_{\mathcal{J}'} \leq m_{\mathcal{J}} < m_{\mathcal{I}}$. This process may be repeated until the size of $\mathcal{J}'$ is reduced to k_{max} giving a $\mathcal{J}''$ where $m_{\mathcal{J}''} < m_{\mathcal{I}}$. But by assumption, the least distant interpretation with a domain of size k_{max} is $m_{\mathcal{I}}$, a contradiction. $\square$

Proposition 3.11 (A credible $p\text{-}\mathcal{ALC}$ knowledge base has a preferred interpretation). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible $p\text{-}\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$. There exists a non-negative integer n and a preferred interpretation $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I}$ is n -distant.*

Proof. By assumption $\mathcal{K}$ is credible and there exist one or more finite interpretations of $\mathcal{K}$ that satisfy $\langle \mathcal{A}, \mathcal{T} \rangle$ and each with an associated (finite) distance. By the credibility of $\mathcal{K}$: $|\mathcal{F}(K)|$, $|N_I|$ and k_{max} are finite. By Lemma 3.10, there is an upper bound $k_{max} = \max(2^{|\mathcal{F}(K)|}, |N_I|)$ on the size of interpretations that need to be considered to locate any interpretation that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ and has the smallest distance. Let m denote the smallest such distance in interpretations up to k_{max} . By Definition 3.4 the interpretations that satisfy $\langle \mathcal{A}, \mathcal{T} \rangle$ and have the smallest distance are the preferred interpretations. Hence, $n = m$ and is a non-negative integer. $\square$

From Proposition 3.11 and Definition 3.4 it follows directly that any given credible knowledge base $\mathcal{K}$ is n -inconsistent, for some non negative integer n .

3.3.2 Refutation-style proofs can be used to show $p\text{-}\mathcal{ALC}$ entailment

We show next that if the n -inconsistency of a $p\text{-}\mathcal{ALC}$ knowledge base is determined, then *proof by refutation* can be used to derive the preferred consequences of such a knowledge base.

Many common inference tasks for description logics can be reexpressed as deciding knowledge base satisfiability. For instance, $\mathcal{K} \models C(x)$ can be proved by showing that $\mathcal{K} \cup \neg C(x)$ is unsatisfiable. Most modern description logic reasoners, for instance [GHM⁺14] and [TH06], are underpinned by a *knowledge base satisfiability tableau algorithm*, a proof procedure that, when applied to a knowledge base, generates some partial model of the knowledge base if this is consistent, or a closed tableau otherwise.

To prove that refutation can be used to show $p\text{-}\mathcal{ALC}$ entailment, we show first that the $p\text{-}\mathcal{ALC}$ preferred consequences include the consequences of $\langle \mathcal{A}, \mathcal{T} \rangle$, as captured by Lemma 3.12.

Lemma 3.12. *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a $p\text{-}\mathcal{ALC}$ knowledge base and $C(x)$ be a concept assertion written in the language of $\mathcal{K}$. If $\langle \mathcal{A}, \mathcal{T} \rangle \models C(x)$ then $\mathcal{K} \approx C(x)$.*

Proof. Assume $\langle \mathcal{A}, \mathcal{T} \rangle \models C(x)$. By standard $\mathcal{ALC}$ entailment every model of $\langle \mathcal{A}, \mathcal{T} \rangle$ satisfies $C(x)$. Let $\mathcal{I}$ be an arbitrary preferred interpretation of $\mathcal{K}$. Since $\mathcal{I}$ is a model of $\langle \mathcal{A}, \mathcal{T} \rangle$, $\mathcal{I}$ must satisfy $C(x)$. $\square$

Theorem 3.13 ($p\text{-}\mathcal{ALC}$ consequence by refutation). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be an n -inconsistent $p\text{-}\mathcal{ALC}$ knowledge base and let $C(x)$ be a concept assertion in the language of $\mathcal{K}$. Then $\mathcal{K} \approx C(x)$ if and only if either (i) $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ is inconsistent or, (ii) $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ is m -inconsistent for some $m > n$.*

Proof. Let S_1 be the set of interpretations of $\mathcal{K}$ that satisfy $\langle \mathcal{A}, \mathcal{T} \rangle$ and S_0 be the set of interpretations of $\mathcal{K}$ that satisfy $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T} \rangle$. Clearly, $S_0 \subseteq S_1$.

“**only if**”: Assume $\mathcal{K} \approx C(x)$. Certainly, either $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T} \rangle$ is consistent or it is not. Taking each case in turn:

1. $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T} \rangle$ is not consistent. Then (i) holds.
2. $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T} \rangle$ is consistent. Then, by definition, S_0 is non-empty. We show that $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ is m -inconsistent, for some $m > n$ by showing that every preferred interpretation of $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ is m -distant. By Definition 3.4, it is sufficient

to show that every interpretation in S_0 is m' -distant for $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$, for any $m' > n$. Assume by contradiction, that there exists an interpretation $\mathcal{I}$ in S_0 that is m' -distant, for some $m' \leq n$. In case $m' = n$, since $\mathcal{I}$ satisfies also $\langle \mathcal{A}, \mathcal{T} \rangle$, by the assumption that $\mathcal{K}$ is n -inconsistent we would have that $\mathcal{I}$ is a preferred interpretation of $\mathcal{K}$ and therefore $\mathcal{I}$ satisfies $C(x)$. But $\mathcal{I}$ satisfies $\neg C(x)$ too, hence contradiction. Let's assume now that $m' < n$. Then since $S_0 \subseteq S_1$, $\mathcal{I} \in S_1$. But this contradicts the assumption that $\mathcal{K}$ is n -inconsistent. Therefore every interpretation in S_0 is $> n$ -distant and $m > n$.

“if”: The proof is by cases.

1. Assume that $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T} \rangle$ is inconsistent. By assumption and definition of entailment $\langle \mathcal{A}, \mathcal{T} \rangle \models C(x)$. Hence, by Lemma 3.12, $\mathcal{K} \approx C(x)$.
2. Assume that $\langle \mathcal{A} \cup \{\neg C(x)\}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ is m -inconsistent for some $m > n$. By Definition 3.4 every interpretation in S_0 is at least m -distant. We need to show that $\mathcal{K} \approx C(x)$, i.e. show that every preferred interpretation of $\mathcal{K}$ satisfies $C(x)$. Since $\mathcal{K}$ is n -inconsistent, every preferred interpretation of $\mathcal{K}$ is n -distant. Let $\mathcal{I}$ be such an n -distant interpretation of $\mathcal{K}$. Since $n < m$, $\mathcal{I} \notin S_0$. Hence $\mathcal{I}$ does not satisfy $\neg C(x)$, which means it satisfies $C(x)$.

□

3.4 Reasoning in the presence of inconsistencies

We now illustrate the use of $p\text{-}\mathcal{ALC}$ through some simple examples. We consider entailments derived from inconsistent knowledge bases and highlight how the distribution and choice of values for the weights influences the entailments. We assume that each knowledge base (inconsistent or otherwise) is *coherent*.

We begin by showing that given a consistent $\mathcal{ALC}$ knowledge base $\mathcal{K}$, the classical consequences

of $\mathcal{K}$ coincide with the $p\text{-}\mathcal{ALC}$ consequences of $\mathcal{K}'$ obtained by making every axiom in $\mathcal{K}$ defeasible.

Proposition 3.14. *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a consistent $\mathcal{ALC}$ knowledge base and $\mathcal{K}' = \langle \emptyset, \emptyset, \mathcal{A}^W, \mathcal{T}^{W'} \rangle$ where $\mathcal{A}^W$ (resp $\mathcal{T}^{W'}$) denotes the assignment of arbitrary integer weights in the range $[1, W]$, $W \geq 1$ to each axiom in $\mathcal{A}$ (resp. $\mathcal{T}$). Let $C(x)$ be a concept assertion written in the language of $\mathcal{K}$. Then $(\mathcal{K}' \text{ is 0-inconsistent and } \mathcal{K}' \models C(x))$ iff $\mathcal{K} \models C(x)$.*

Proof. “**if**”: Assume $\mathcal{K} \models C(x)$. By standard $\mathcal{ALC}$ entailment every model of $\mathcal{K}$ satisfies $C(x)$. By assumption $\mathcal{K}$ is consistent and therefore has a model. Let $\mathcal{I}$ be one such model of $\mathcal{K}$. Since $\mathcal{I}$ is a model of $\mathcal{K}$, $\mathcal{I}$ is a 0-distant interpretation of $\mathcal{K}'$ (no axioms instances are falsified). $\mathcal{I}$ is a preferred interpretation of $\mathcal{K}'$ (no interpretation has a smaller distance) and $\mathcal{I}$ satisfies $C(x)$. Hence $\mathcal{K}' \models C$.

“**only if**”: Assume $\mathcal{K}'$ is 0-inconsistent and $\mathcal{K}' \models C(x)$. As $\mathcal{K}'$ is 0-inconsistent then every preferred interpretation $\mathcal{I}'$ does not falsify any instance of the defeasible Abox or Tbox (must be at least one) and makes $C(x)$ true. Let $\mathcal{I}$ be a model of $\mathcal{K}$, then it is a preferred interpretation of $\mathcal{K}'$ and hence makes $C(x)$ true. Hence $\mathcal{K} \models C(x)$. $\square$

Informally, the weights chosen for the defeasible axioms have no effect on entailment when a knowledge base is consistent. For an inconsistent knowledge base, we could make all the axioms defeasible to obtain consequences under the $p\text{-}\mathcal{ALC}$ semantics. However, it is sufficient to make any subset defeasible such that the remaining non-defeasible axioms are consistent. Making an axiom defeasible which does not appear anywhere in the justifications of $\mathcal{K} \models \top \sqsubseteq \perp$ also has no effect on $p\text{-}\mathcal{ALC}$ entailment.

As a simple starting point, choosing uniform weights for the defeasible axioms leads to entailments that are based on minimising the number of falsified axioms (and axiom instances).

Example 3.6. *Consider the set of facts $\mathcal{A}_{3.6} = \{C(a), E(a), F(a), C(b)\}$ and a classical TBox $\mathcal{T}_{3.6} = \{C \sqsubseteq D, D \sqsubseteq E, E \sqsubseteq \neg F, E \sqsubseteq \neg G\}$. Let $\mathcal{K}_{3.6} = \langle \mathcal{A}_{3.6}, \mathcal{T}_{3.6} \rangle$. $\mathcal{K}_{3.6}$ is inconsistent due to the axioms $\{C(a), E(a), F(a), C \sqsubseteq D, D \sqsubseteq E, E \sqsubseteq \neg F\}$.*

Suppose that we are confident that $\mathcal{T}_{3.6}$ is an accurate model of the world and choose to accommodate the inconsistency by making each ABox axiom defeasible with a weight of 1. Let $\mathcal{K}'_{3.6} = \{\emptyset, \mathcal{T}_{3.6}, \mathcal{A}_{3.6}^{[1]}, \emptyset\}$. Now any interpretation of $\mathcal{K}'_{3.6}$ that satisfies $\mathcal{T}_{3.6}$ must falsify either $F(a)^{[1]}$ (making it 1-distant) or $C(a)^{[1]}$ and $E(a)^{[1]}$ (making it 2-distant). Hence, $\mathcal{K}'_{3.6}$ is 1-inconsistent and $\mathcal{K}'_{3.6} \models C(a), D(a), E(a), \neg F(a), \neg G(a)$, and $\mathcal{K}'_{3.6} \models C(b), D(b), E(b), \neg F(b), \neg G(b)$. Informally, the summing of weights reflects that there is more evidence that supports the conclusion $\neg F(a)$ than $F(a)$. In contrast, reassigning a weight of 3 (or greater) to $F(a)$ would reverse the situation, leading to the 2-inconsistent knowledge base $\mathcal{K}''_{3.6}$. $\mathcal{K}''_{3.6} \models \neg C(a), \neg D(a), \neg E(a), F(a)$, and $\mathcal{K}''_{3.6} \models C(b), D(b), E(b), \neg F(b), \neg G(b)$. Notice that axioms $E \sqsubseteq \neg G$ and $C(b)$ do not appear in justifications for $\mathcal{K}_{3.6} \models \top \sqsubseteq \perp$. Hence, $C(b)$ could have been left as a non-defeasible axiom.

A similar pattern can be observed where role assertions and quantified role restrictions interact.

Example 3.7. Consider the set of facts $\mathcal{A}_{3.7} = \{(\forall R.\neg E)(a), R(a, b), R(a, c), E(b), E(c)\}$. Let $\mathcal{K}_{3.7} = \{\emptyset, \emptyset, \mathcal{A}_{3.7}^{[1]}, \emptyset\}$. Now any interpretation of $\mathcal{K}_{3.7}$ that satisfies $(\forall R.\neg E)(a)^{[1]}$ must falsify either $R(a, b)^{[1]}$ or $E(b)^{[1]}$; and either $R(a, c)^{[1]}$ or $E(c)^{[1]}$. Hence, such an interpretation is 2-distant. In contrast, there are 1-distant interpretations that satisfy all axioms except $(\forall R.\neg E)(a)^{[1]}$. $\mathcal{K}_{3.7}$ is therefore 1-inconsistent, $\mathcal{K}_{3.7} \models R(a, b), R(a, c), E(b), E(c), (\exists R.E)(a)$ and $\mathcal{K}_{3.7} \not\models (\forall R.\neg E)(a)$. Reassigning a weight of 3 (or greater) to $(\forall R.\neg E)(a)$ would reverse the situation, leading to the 2-inconsistent knowledge base $\mathcal{K}''_{3.7}$. $\mathcal{K}''_{3.7} \models (\forall R.\neg E)(a)$ and $\mathcal{K}''_{3.7} \not\models R(a, b), R(a, c), E(b), E(c), (\exists R.E)(a)$.

Adding non-uniform weights arbitrates inconsistencies and generally leads to more consequences being derivable. However, this is not universally true, as illustrated by assigning the weight of 3 to $(\forall R.\neg E)(a)$ in $\mathcal{K}''_{3.7}$ of Example 3.7.

Extending the defeasibility to the TBox provides an extra degree of flexibility.

Example 3.6 (continued). Returning to $\mathcal{A}_{3.6} = \{C(a), E(a), F(a), C(b)\}$ and $\mathcal{T}_{3.6} = \{C \sqsubseteq D, D \sqsubseteq E, E \sqsubseteq \neg F, E \sqsubseteq \neg G\}$, we might be confident about the facts but not that the dis-

joint relationships always hold. Let $\mathcal{K}_{3.6}''' = \langle \mathcal{A}_{3.6}, \{C \sqsubseteq D, D \sqsubseteq E\}, \emptyset, \{E \sqsubseteq \neg F^{[1]}, E \sqsubseteq \neg G^{[1]}\} \rangle$. Now instances of $E \sqsubseteq \neg F^{[1]}$ may be falsified, in particular allowing for 1-distant interpretations of $\mathcal{K}_{3.6}'''$ in which $E(a)$ and $F(a)$ are made true. $\mathcal{K}_{3.6}'''$ is 1-inconsistent. $\mathcal{K}_{3.6}''' \approx C(a), D(a), E(a), F(a), \neg G(a)$, and $\mathcal{K}_{3.6}''' \approx C(b), D(b), E(b), \neg F(b), \neg G(b)$. Notice that the axiom $E \sqsubseteq \neg G$ does not appear in justifications for $\mathcal{K}_{3.6} \models \top \sqsubseteq \perp$. Hence, $E \sqsubseteq \neg G$ could have been left as a non-defeasible axiom.

We have already noted the sensitivity of the logic to syntactic form in ABox axioms. A similar issue exists for TBox axioms in which the “double counting” of falsified instances may occur where one defeasible TBox axiom subsumes another. For example, axioms $C \sqsubseteq D^{[1]}$ and $C \sqsubseteq D \sqcap E^{[2]}$. In an interpretation $\mathcal{I}$ such that $\mathcal{I} \models C(x)$ and $\mathcal{I} \models \neg D(x)$, $\mathcal{I}$ falsifies both axiom instances which introduces a distance of 3 to the interpretation. Such structures are symptomatic of a sub-optimally modelled TBox and may also lead to inefficient reasoning. Ideally, these redundant structures should be identified and then eliminated by using techniques such as those proposed in [NS13].

The use of weights to identify preferred interpretations may lead to un-intuitive consequences if the TBox does not accurately model the real world relationships. In particular, where inter-dependencies are not correctly reflected.

Example 3.6 (continued). Suppose $\mathcal{T}_{3.6}$ is augmented with three additional axioms: $C_1 \sqsubseteq C$, $C_2 \sqsubseteq C$ and $C_3 \sqsubseteq C$. Our assumption is that concepts C_1 , C_2 and C_3 are entirely independent. However, if this were not the case, and (non)membership of one of these three concepts necessitates (non)membership of all of them, then falsification of a set of defeasible assertions $\{C_1(a)^{[1]}, C_2(a)^{[1]}, C_3(a)^{[1]}\}$ may result in “over-counting” evidence.

Our approach identifies preferred interpretations by summing integer weights attached to defeasible axioms. Of potential relevance is the long-standing body of work in non-monotonic and defeasible reasoning on the use of priority/preference orderings to resolve conflicts between rules. This has a long history and has led to a wide variety of approaches and methods. It remains an active area of research. Evaluating the perceived quality of results obtained by any

given method is somewhat subjective and there is no general agreement either on the source of differences between results or how to choose between them. There have been attempts to formulate some abstract general principles that any such strategy should exhibit but this work is still at a preliminary stage. A recent example is a study of preference orderings in logic programming [Šim14] that investigates how a range of strategies display varying strengths and weaknesses when evaluated against a proposed set of such principles. The aggregation of priorities/preferences, treated in our approach by summing weights, touches on another challenging set of open questions traditionally studied in the logic of preference, decision theory, argumentation, moral and ethical reasoning, among other areas.

For our own work, further research is required to investigate how the selection of uniform vs. non-uniform weights and their distribution over TBox and ABox axioms are best tailored to arbitrate inconsistency and obtain intuitive results from simple hierarchical descriptions of objects and their established relationships. Further work would also be required to determine to what extent our approach could be refined or informed by findings from research on priority/preference orderings in defeasible reasoning and logic programming.

In Chapter 8 we consider an extension of our implementation that introduces a notion of priority. The priorities express an ordering on axioms where conflicts are arbitrated by allowing higher priority axioms to dominate (unconditionally) over lower priority axioms. The formalisation with priorities provides more precise control over the arbitration of inconsistencies. Again substantial further work would be needed to clarify the relationships with potentially relevant work on priority/preference orderings.

3.5 Summary

In this section we have formalised the syntax and the semantics for $p\text{-}\mathcal{ALC}$. We showed that every $p\text{-}\mathcal{ALC}$ knowledge base is n -inconsistent for some non-negative integer n and that when n is established it is possible to derive our notion of preferred consequences from a knowledge base using refutation-style proofs. Inspired by this and the existing tableau techniques for $\mathcal{ALC}$

we have developed a modified tableau algorithm that can be used to derive the preferential consequences of a $p\text{-}\mathcal{ALC}$ knowledge base.

Chapter 4

A Tableau Algorithm for $p\text{-}\mathcal{ALC}$

In this chapter we present a tableau algorithm for $p\text{-}\mathcal{ALC}$. We begin by providing background information on tableau methods and introduce an existing tableau algorithm [BHS08] that can be used to decide satisfiability of an $\mathcal{ALC}$ knowledge base. The algorithm incorporates a *blocking strategy* that ensures termination by limiting the size of the domains considered during reasoning. Next, we show how this algorithm has been adapted to accommodate our notion of defeasible axioms and used to develop branches that represent preferred interpretations of a credible knowledge base. We show the correctness of our algorithm by proving termination, soundness and completeness properties. Finally, we show how to use the algorithm to compute preferred consequences of a credible $p\text{-}\mathcal{ALC}$ knowledge base through proof by refutation.

4.1 Background

Tableau algorithms underpin many modern description logic reasoners including Hermit [GHM⁺14] and Fact++ [TH06]. The tableau algorithm establishes the satisfiability of a knowledge base, and generally implements reasoning tasks through *refutation-style* proofs. A knowledge base $\mathcal{K} \models Z$, where Z is an axiom in the logic of $\mathcal{K}$, iff $\mathcal{K} \cup \{\neg Z\}$ is unsatisfiable. For example, to show $\mathcal{K} \models C(a)$, a tableau algorithm may be used to test if $\mathcal{K} \cup \{\neg C(a)\}$ is unsatisfiable.

The $\mathcal{ALC}$ knowledge base satisfiability tableau algorithm presented below is based on the algorithm introduced in Chapter 3 of [BHS08]¹. Without loss of generality, we assume that all axioms are given in *negation normal form*, indicating that negation is restricted to appear only in front of concept names. For a concept C , we use $\neg C$ to denote $\neg C$ written in negation normal form. We introduce the algorithm in steps to emphasise how the challenges of creating a terminating, sound and complete algorithm were overcome.

Informally, given a knowledge base $\mathcal{K}$, the algorithm attempts to show that $\mathcal{K}$ has no model, or returns a consistent ABox that can be used to construct a model of $\mathcal{K}$. The assertions given in the ABox are successively decomposed into simpler assertions using a set of expansion rules that reflect consistency preserving rules of inference. The expansion of concept unions leads to non-determinism and requires that different possible expansions must be checked. The expansion rules for existentially quantified role restrictions introduce new names, called *parameters*, that do not appear in the knowledge base signature. These individuals serve as *witnesses* to the existential expansions. Each name introduced is required to be *fresh* to the ABox being expanded, meaning it does not already appear within this ABox. The constraints imposed by the TBox are accommodated by including suitable ABox axioms. This process continues until either: (a) each possible expansion leads to a contradiction, called a *clash*, indicating the knowledge base is unsatisfiable; or (b) we generate a consistent ABox where no further expansion is possible, representing a (partial) model of $\mathcal{K}$.

The valid (tableau) expansions of an ABox with respect to a TBox are formalised by Definition 4.1 and Table 4.1. These are adapted from Definition 3 of [BS01] and Figure 3.1 of [BHS08] respectively.

Definition 4.1 (Valid expansion). *Let $\mathcal{A}_b$ be an ABox and $\mathcal{T}$ be a TBox. $\mathcal{A}_e$ is a valid expansion of $\mathcal{A}_b$ w.r.t. $\mathcal{T}$ iff $\mathcal{A}_e$ is generated from $\mathcal{A}_b$ by applying an instance of an $\mathcal{ALC}$ tableau rule $\mathcal{R}$ from Table 4.1. $\mathcal{A}_b$ is said to be completed iff there are no valid expansions of $\mathcal{A}_b$. $\mathcal{A}_b$ is said to include a clash if $\{A(x), \neg A(x)\} \subseteq \mathcal{A}_b$ or $\perp(x) \in \mathcal{A}_b$. $\mathcal{A}_b$ is said to be closed if it contains a clash and said to be open otherwise.*

¹Following [BS01], we record each branch as an ABox rather than constructing a *completion tree*, the technique used in [BHS08].

Rules $\mathcal{R}$	Valid expansions of $\mathcal{A}_b$
$\rightarrow_{\sqcap}$ -rule	If $(C \sqcap D)(x) \in \mathcal{A}_b$ and $(C(x) \notin \mathcal{A}_b \text{ or } D(x) \notin \mathcal{A}_b)$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(x), D(x)\}$
$\rightarrow_{\sqcup}$ -rule	If $(C \sqcup D)(x) \in \mathcal{A}_b$ and $(C(x) \notin \mathcal{A}_b \text{ and } D(x) \notin \mathcal{A}_b)$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(x)\}$ or $\mathcal{A}_e = \mathcal{A}_b \cup \{D(x)\}$
$\rightarrow_{\forall}$ -rule	If $(\forall R.C)(x) \in \mathcal{A}_b$ and $R(x, y) \in \mathcal{A}_b$ and $C(y) \notin \mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(y)\}$
$\rightarrow_{\exists}$ -rule	If $(\exists R.C)(x) \in \mathcal{A}_b$ and $\neg \exists y [R(x, y) \in \mathcal{A}_b \text{ and } C(y) \in \mathcal{A}_b]$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{R(x, z), C(z)\}$ where z is fresh for $\mathcal{A}_b$
$\rightarrow_{\mathcal{T}}$ -rule	If $(C \sqsubseteq D) \in \mathcal{T}$ and $\exists$ an individual x in $\mathcal{A}_b$ and $(\neg C \sqcup D)(x) \notin \mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{(\neg C \sqcup D)(x)\}$

Table 4.1: The $\mathcal{ALC}$ tableau expansion rules $\mathcal{R}$

Algorithm 1 is a simple algorithm that can be used to construct a tableau. The algorithm does not enforce any ordering on the rule applications nor any preference over choices taken at each non-deterministic step. Consequently, more than one possible tableau can be generated from a knowledge base.

Algorithm 1: The basic $\mathcal{ALC}$ knowledge base satisfiability tableau algorithm

Input: $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$, an $\mathcal{ALC}$ knowledge base

Output: $\mathcal{E}$, a set of expanded ABoxes

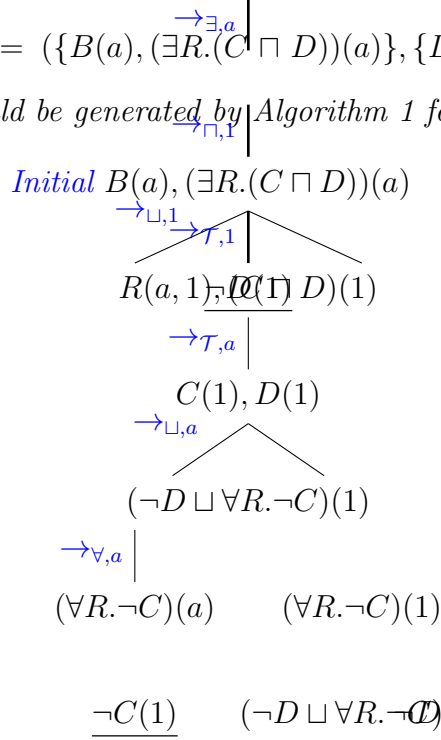
```

1  $\mathcal{E} := \{\mathcal{A}\};$ 
2 repeat
3   if  $\mathcal{A}_b \in \mathcal{E}$  is open and an instance of a rule  $R$  applies to  $\mathcal{A}_b$  then
4     Remove  $\mathcal{A}_b$  from  $\mathcal{E}$ ;
5     foreach valid expansion  $\mathcal{A}_e$  of  $\mathcal{A}_b$  by the instance of  $R$  do
6       Add  $\mathcal{A}_e$  to  $\mathcal{E}$ ;
7     end
8   end
9 until all ABoxes in  $\mathcal{E}$  are closed or find a completed open ABox;
10 return  $\mathcal{E}$ 
```

Since the application of a rule instance can result in more than one expanded ABox at line 5, it is convenient to represent the development of a tableau using a tree structure. The top node is labelled with the assertions given in the initial ABox. Each edge represents a valid expansion

by a rule instance and each node is labelled with the assertions added by the expansion. An ABox may be read from a *branch* of the tree by constructing the union of all labels in the nodes of the branch from the leaf node to the root of the tree. A branch is said to be completed, open or closed based on the ABox it represents. When a clash is detected a branch is marked as closed by underlining the assertion that leads to the conflict. A consequence of the assumption that concepts are in negation normal form is that for any clash $A(x)$ with $\neg A(x)$, A is a named concept.

Example 4.1. Let $\mathcal{K}_{sat} = (\{B(a), (\exists R.(C \sqcap D))(a)\}, \{D \sqsubseteq \forall R.\neg C\})$. One possible open completed tableau that could be generated by Algorithm 1 for $\mathcal{K}_{sat}$ is represented by the tree:



This tree illustrates a run in which the algorithm closes two branches before finding an open completed branch. The open branch represents the ABox $\mathcal{A}_{4.1}$:

$$\begin{aligned}
 \mathcal{A}_{4.1} = \{ & B(a), (\exists R.(C \sqcap D))(a), R(a, 1), (C \sqcap D)(1), C(1), D(1), (\neg D \sqcup \forall R.\neg C)(1), \\
 & (\forall R.\neg C)(1), (\neg D \sqcup \forall R.\neg C)(a), \neg D(a) \}
 \end{aligned}$$

The Abox obtained from an open branch returned by Algorithm 1 is completed, and can be

used to construct a Herbrand model of the knowledge base. In the following, an open branch obtained from the algorithm is assumed to refer to an open completed branch.

Definition 4.2 (Interpretation based on an open branch). *Let $\mathcal{A}_b$ be an ABox obtained from an open completed branch b generated by Algorithm 1 for $\mathcal{K}$ an $\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$. Let $P(b)$ be the set of parameters introduced in the branch and Δ_b be the Herbrand domain based on N_I augmented by $P(b)$. The Herbrand interpretation of $\mathcal{K}$ based on $\mathcal{A}_b$ is constructed as follows: For every $x, y \in \Delta_b$, $A \in N_C$ and $R \in N_R$: $A(x)$ is true iff $A(x) \in \mathcal{A}_b$ and $R(x, y)$ is true iff $R(x, y) \in \mathcal{A}_b$.*

The idea here is that the Herbrand interpretation based on an open branch satisfies the (clash free) set of simple concept assertions and role assertions. Since these assertions were obtained by applying consistency preserving rules to the axioms of $\mathcal{A}$ and $\mathcal{T}$, the interpretation also satisfies each axiom of $\mathcal{A}$ and $\mathcal{T}$. The notion of the *soundness* of Algorithm 1 captures that the Herbrand interpretation based on a clash free and fully expanded branch is a model of $\mathcal{K}$ (see Theorem 3 [BHS08]).

Example 4.1 (continued). *Applying Definition 4.2 to the open branch $\mathcal{A}_{4.1}$ obtained for $\mathcal{K}_{sat}$ we can construct the Herbrand interpretation $\mathcal{I}_1$, where $\Delta^{\mathcal{I}_1} = \{a, 1\}$, $B^{\mathcal{I}_1} = \{a\}$, $C^{\mathcal{I}_1} = \{1\}$, $D^{\mathcal{I}_1} = \{1\}$, and $R^{\mathcal{I}_1} = \{(a, 1)\}$. Hence, by Theorem 3 [BHS08], we conclude that this is a model of $\mathcal{K}_{sat}$ and $\mathcal{K}_{sat}$ is consistent.*

A completed open ABox $\mathcal{A}_b$ generated for a knowledge base $\mathcal{K}$ defines a partial interpretation of $\mathcal{K}$. An interpretation $\mathcal{I}$ obtained from Definition 4.2 chooses false for $A(x)^{\mathcal{I}}$ if $A(x) \notin \mathcal{A}_b$. This choice is arbitrary, since either true or false may be chosen to form a model. However, Definition 4.2 chooses false for $R(x, y)^{\mathcal{I}}$ if $R(x, y) \notin \mathcal{A}_b$ to guarantee that $\mathcal{I}$ is a model of $\mathcal{K}$ because the $\rightarrow_{\forall}$ -rule takes advantage of an asymmetry of $\mathcal{ALC}$: negated role assertions of the form $\neg R(x, y)$ are not part of the language. Recall from the semantics of universally quantified role restriction that $(\forall R.C)(x)$ will be satisfied by a Herbrand interpretation $\mathcal{I}$ iff $\forall y \in \Delta^{\mathcal{I}} [R(x, y) \rightarrow C(y)]$ or equivalently $\forall y \in \Delta^{\mathcal{I}} [\neg R(x, y) \vee C(y)]$. The rule does not need to apply a disjunction to every individual because $\neg R(x, y)$ can never be in $\mathcal{A}_b$. It is sufficient to introduce $C(y)$ when $R(x, y) \in \mathcal{A}_b$.

Example 4.2. Let $\mathcal{K}_{rsat} = \langle \{\neg C(a), (\forall R.C)(a)\}, \emptyset \rangle$. There are no valid expansions of the $ABox$. The Herbrand interpretation $\mathcal{I}_2$ constructed from the $ABox$ by Definition 4.2 is $\Delta^{\mathcal{I}_2} = \{a\}$, $C^{\mathcal{I}_2} = \emptyset$, and $R^{\mathcal{I}_2} = \emptyset$ and is a model of $\mathcal{K}_{rsat}$. In contrast, any interpretation $\mathcal{I}'_2$ which chooses $C^{\mathcal{I}'_2} = \emptyset$ and $R^{\mathcal{I}'_2} = \{(a, a)\}$ is not a model of $\mathcal{K}_{rsat}$.

Remark 4.3. Algorithm 1 may not terminate if the $TBox$ includes cyclic definitions.

Example 4.3 illustrates how cycles in the terminology of a knowledge base lead to non-termination.

Example 4.3. Let $\mathcal{K}_{cyc1} = \langle \{C(a)\}, \{C \sqsubseteq \exists R.C\} \rangle$. Applying Algorithm 1 to $\mathcal{K}_{cyc1}$ leads to the development of an infinite branch, the beginning of which is shown in Figure 4.1.

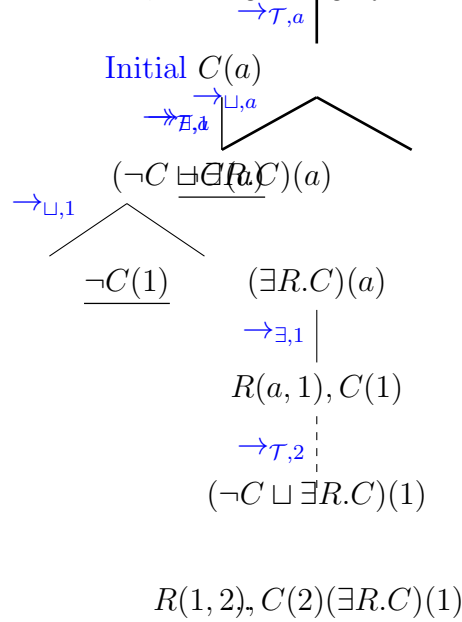


Figure 4.1: The beginning of an infinite branch generated by Algorithm 1 for $\mathcal{K}_{cyc1}$

To ensure termination in the presence of cyclic definitions, a *blocking* strategy can be used. A number of different blocking strategies have been developed (see [BS01] for a review). Informally, the idea is to detect when expanding an assertion would provide no additional information because we have already expanded an assertion that required satisfying the same set of concepts. Definition 4.4 given below is an example of *subset blocking* [BBH96].

Definition 4.4 (Blocking conditions). *Let $\mathcal{A}_0 \subseteq \mathcal{A}_1 \dots \subseteq \mathcal{A}_n$ where $n > 0$ denote the sequence of (not closed) ABoxes of a branch in a sequence of n applications of tableau rules. An individual x is older than an individual y if x is introduced in $\mathcal{A}_i$ and y is introduced in $\mathcal{A}_j$ where $0 \leq i < j \leq n$. An individual y is blocked by individual x , written $y \blacktriangleleft x$, at step $\mathcal{A}_j$ if*

(i) x is older than y and

(ii) $\{C \mid C(y) \in \mathcal{A}_j\} \subseteq \{C \mid C(x) \in \mathcal{A}_j\}$.

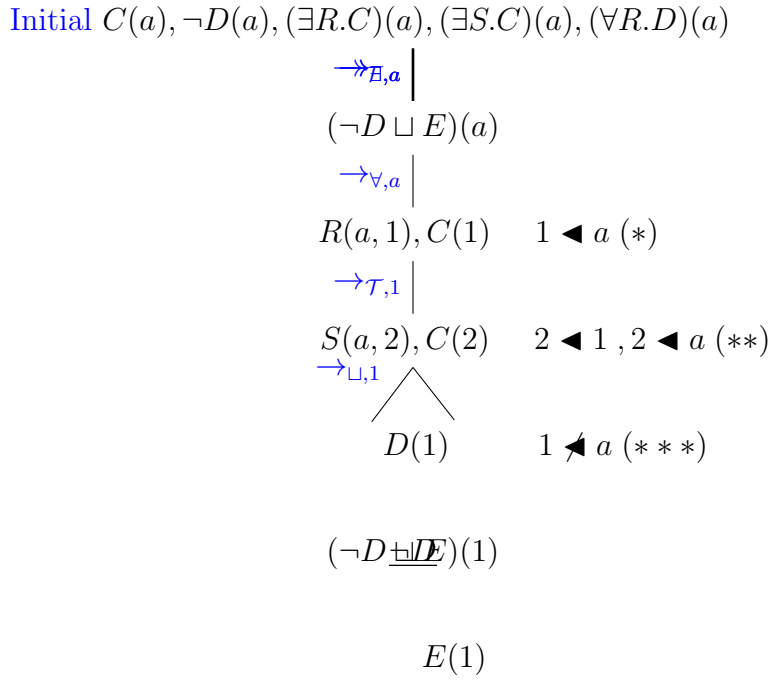
If y is blocked by x we say y is blocked.

Remark 4.5. *Definition 4.4 includes the condition that the blocking parameter must be older than the blocked parameter. This condition prevents mutual blocking, where two individuals x and y that are required to satisfy exactly the same set of concepts leads to the situation that x blocks y and y blocks x . The deadlock is broken by asserting precedence to the older individual.*

Table 4.2 shows how the rules are revised to incorporate the blocking strategy of Definition 4.4. When applying the modified rules Algorithm 1 satisfies two properties: (1) it is guaranteed to terminate for $\mathcal{ALC}$ knowledge bases; and (2) it is complete, that is it returns an open branch if and only if the knowledge base is satisfiable (Theorem 3 [BHS08]). Hence, by soundness, the algorithm is sufficient to check for the existence of a model.

Rules $\mathcal{R}$	Valid expansions of $\mathcal{A}_b$
$\rightarrow_{\sqcap}$ -rule	If $(C \sqcap D)(x) \in \mathcal{A}_b$, x is not blocked and $(C(x) \notin \mathcal{A}_b \text{ or } D(x) \notin \mathcal{A}_b)$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(x), D(x)\}$
$\rightarrow_{\sqcup}$ -rule	If $(C \sqcup D)(x) \in \mathcal{A}_b$, x is not blocked and $(C(x) \notin \mathcal{A}_b \text{ and } D(x) \notin \mathcal{A}_b)$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(x)\}$ or $\mathcal{A}_e = \mathcal{A}_b \cup \{D(x)\}$
$\rightarrow_{\forall}$ -rule	If $(\forall R.C)(x) \in \mathcal{A}_b$ and x is not blocked and $R(x, y) \in \mathcal{A}_b$ and $C(y) \notin \mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(y)\}$
$\rightarrow_{\exists}$ -rule	If $(\exists R.C)(x) \in \mathcal{A}_b$, x is not blocked and $\neg \exists y [R(x, y) \in \mathcal{A}_b \text{ and } C(y) \in \mathcal{A}_b]$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{R(x, z), C(z)\}$ where z is fresh for $\mathcal{A}_b$
$\rightarrow_{\mathcal{T}}$ -rule	If $(C \sqsubseteq D) \in \mathcal{T}$ and $\exists$ an unblocked individual x in $\mathcal{A}_b$ and $(\neg C \sqcup D)(x) \notin \mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{(\neg C \sqcup D)(x)\}$

Table 4.2: The $\mathcal{ALC}$ blocking tableau rules $\mathcal{R}$



- (*) $1 \blacktriangleleft a$ because $\{C\}_1 \subseteq \{C, \neg D, \exists R.C, \exists S.C, \forall R.D, \neg D \sqcup E\}_a$
(**) $2 \blacktriangleleft 1$ because $\{C\}_2 \subseteq \{C\}_1$ and $2 \blacktriangleleft a$ because $\{C\}_2 \subseteq \{C, \neg D, \exists R.C, \exists S.C, \forall R.D, \neg D \sqcup E\}_a$
(***) $1 \blacktriangleleft a$ because $\{C, D\}_1 \not\subseteq \{C, \neg D, \exists R.C, \exists S.C, \forall R.D, \neg D \sqcup E\}_a$

Figure 4.3: An illustration of unblocking when using a dynamic blocking strategy. Algorithm 1 is applied to $\mathcal{K}_{blk} = \langle \{C(a), \neg D(a), (\exists R.C)(a), (\forall R.D)(a)\}, \{D \sqsubseteq E\} \rangle$ using the blocking tableau rules in Table 4.2. The parameter 1 becomes blocked at (*) and unblocked at (**).

A dynamic blocking strategy necessitates that the algorithm must continuously recheck the blocking conditions after each step. However, this can be avoided by using a *static* blocking strategy [BS01], whereby, once an individual is blocked in a branch it will never become unblocked. Replacing the blocking $\rightarrow_{\exists}$ -rule from Table 4.2 by the combined $\rightarrow_{\exists\forall}$ -rule shown in Table 4.3 leads to static blocking for Algorithm 1.

Rule	Valid expansion of $\mathcal{A}_b$
$\rightarrow_{\exists\forall}$	If $(\exists R.C)(x) \in \mathcal{A}_b$, x is not blocked and $\neg \exists y [R(x, y) \in \mathcal{A}_b \text{ and } C(y) \in \mathcal{A}_b]$ and no other rule applies to $\mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{R(x, z), C(z)\} \cup \{D(z) (\forall R.D)(x) \in \mathcal{A}_b\}$ where z is fresh

Table 4.3: The $\rightarrow_{\exists\forall}$ blocking tableau rule

Informally, no individuals are introduced before the named individuals. Hence named individuals are never blocked. The application of the $\rightarrow_{\exists\forall}$ -rule is delayed until no other rules are applicable. Hence, the other rules are applied to all the named individuals until no more rules apply. The named individuals are said to be *fully expanded* and no further rules are applied

to them from this point forward. The application of a $\rightarrow_{\exists\forall}$ -rule instance to an individual x introduces a fresh parameter p , a role assertion $R(x, p)$ and all the concept assertions due to the quantified role restrictions of x in a single step. When a fresh parameter p is introduced, there are two cases:

1. p is unblocked. The applicable rule instances excluding the $\rightarrow_{\exists\forall}$ -rule are applied to p which introduce (and never remove) further concept assertions for p . No rules are applied to the older individuals and therefore p remains unblocked. The $\rightarrow_{\forall}$ -rule is never applied to a parameter p because (1) the $\rightarrow_{\exists\forall}$ introduces $R(x, p)$ and concept assertions due to universal role restrictions of x and (2) every parameter is fresh; hence, no rule adds a role assertion $R(y, p)$ where $y \neq x$. When p is fully expanded, any $\rightarrow_{\exists\forall}$ -rule that is applied to p (or another individual) will introduce a parameter which will be younger than p and hence will not influence the blocking of p .
2. p is blocked. The precondition of each rule requires that p is not blocked to be applicable to p . Hence, no rules are applied to p and p remains blocked. As in 1), any fresh parameter introduced later due to the application of a $\rightarrow_{\exists\forall}$ -rule will be younger than p and will not influence the blocking of p .

Example 4.5 (continued). *Figure 4.4 illustrates the static blocking that results when Algorithm 1 is applied to $\mathcal{K}_{blk}$ using the combined $\rightarrow_{\exists\forall}$ -rule.*

The construction of an interpretation based on an open branch in Definition 4.2 is modified to take into account the presence of blocked individuals. The intuition behind the revised construction given in Definition 4.6 is that each blocked individual is replaced by an unblocked individual which blocks it. The resultant interpretation includes only unblocked (and therefore fully expanded) individuals.

Definition 4.6 (Interpretation based on an open branch with blocking). *Let $\mathcal{K}$ be an $\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$, $\mathcal{A}_b$ be an open branch generated for $\mathcal{K}$ by Algorithm 1 using the static blocking tableau rules. Let $P(b)$ denote the set of unblocked parameters introduced in the branch and $\mathcal{A}'_b$ be the set of axioms in $\mathcal{A}_b$ not involving blocked individuals.*

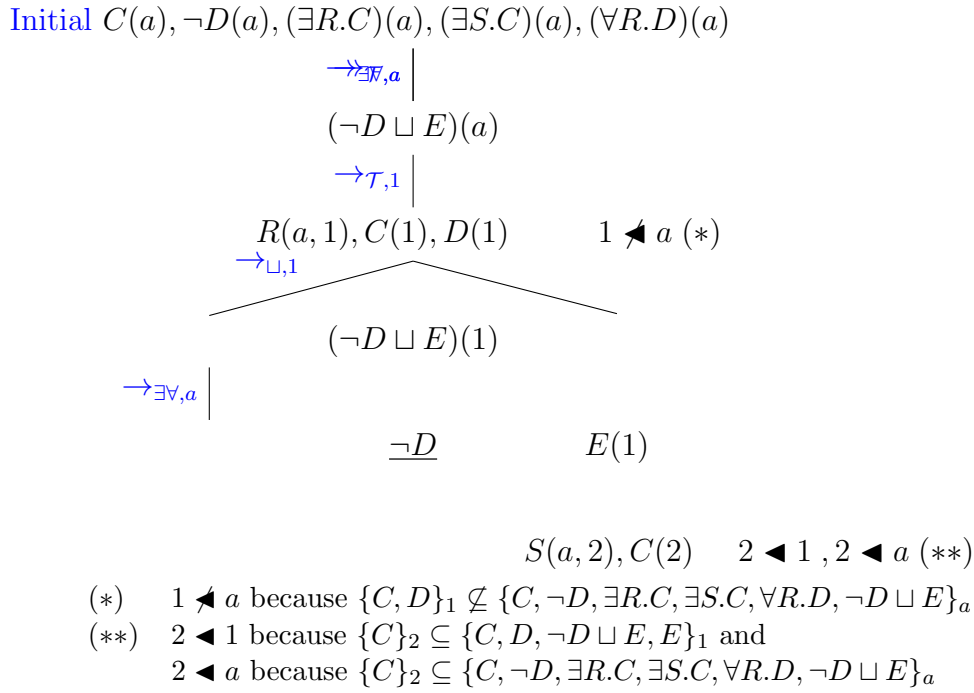


Figure 4.4: An illustration of static blocking. Algorithm 1 is applied to $\mathcal{K}_{blk} = \langle \{C(a), \neg D(a), (\exists R.C)(a), (\forall R.D)(a)\}, \{D \sqsubseteq E\} \rangle$ with blocking tableau rules where the $\rightarrow_{\exists}$ -rule is replaced by the $\rightarrow_{\exists\forall}$ -rule.

Let Δ_b be the Herbrand domain based on the signature of $\mathcal{K}$ in which N_I is augmented by the unblocked parameters $P(b)$. The Herbrand interpretation $\mathcal{I}_b$ of $\mathcal{K}$ based on Δ_b is defined as follows. For every $x, y \in \Delta_b$, $A \in N_C$ and $R \in N_R$: $A(x)$ is true iff $A(x) \in \mathcal{A}'_b$; $R(x, y)$ is true iff either $R(x, y) \in \mathcal{A}'_b$ or $R(x, z) \in \mathcal{A}_b$ and z is blocked by y .

Example 4.5 (continued). Let $\mathcal{I}_5$ denote the Herbrand interpretation of $\mathcal{K}_{blk}$ based on the open branch shown in Figure 4.4. Individual 2 is blocked by both the (unblocked) individuals a and 1. In $\mathcal{I}_5$: $C(a), R(a, 1), C(1), D(1), E(1), S(a, a)$ and $S(a, 1)$ are made true. The interpretation is illustrated in Figure 4.5.

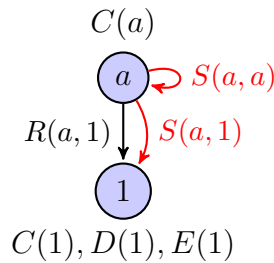


Figure 4.5: $\mathcal{I}_5$ is the Herbrand interpretation of $\mathcal{K}_{blk}$ that is constructed from the open branch $\{C(a), \neg D(a), (\exists R.C)(a), (\exists S.C)(a), (\forall R.D)(a), (\neg D \sqcup E)(a), R(a, 1), C(1), D(1), (\neg D \sqcup E)(1), E(1), S(a, 2), C(2)\}$ shown in Figure 4.4.

4.2 The tableau algorithm for $p\text{-}\mathcal{ALC}$

The basic concepts of $\mathcal{ALC}$ static subset blocking tableaux have now been explained. We will next present a variant of Algorithm 1 and show how to apply it to $p\text{-}\mathcal{ALC}$.

To exploit Theorem 3.13 and perform proofs by refutation requires that we find the n -distance of a credible knowledge base. In the $p\text{-}\mathcal{ALC}$ tableau algorithm, the $\mathcal{ALC}$ static subset blocking tableau rules are augmented with new rules such that the notion of a branch is extended to record both the ABox under expansion and a set of axiom instances called the *omitted set*. To obtain n , we consider all possible branches, generated from valid expansions of the non-defeasible axioms and a subset of the defeasible ABox axioms where we omit a subset of the TBox axioms from defeasible TBox rule applications. The intuition is that the interpretation based on an open branch which minimises the omitted set coincides with a preferred interpretation. Minimisation is based on the notion of n -distance, the sum of the weights of the axiom instances in the omitted set. The n -distance of such a branch that minimises the omitted set gives the n -inconsistency of the given initial knowledge base.

In Section 4.1 we saw that an interpretation based on an open branch from an $\mathcal{ALC}$ satisfiability tableau algorithm applied to $\mathcal{K}$ is a model of $\mathcal{K}$. However, the precise form of this model is not important to demonstrate satisfiability. In contrast, under the $p\text{-}\mathcal{ALC}$ semantics and when inconsistencies are present, the form of an interpretation may impact the distance of the interpretation.

Example 4.6. Consider the $p\text{-}\mathcal{ALC}$ knowledge base:

$$\mathcal{K}_{inc} = \left\langle \left\{ \begin{array}{l} C(a) \\ D(a) \\ (\exists R.E)(a) \end{array} \right\}, \left\{ E \sqsubseteq C \sqcap D \right\}, \emptyset, \left\{ C \sqsubseteq \neg D^{[1]} \right\} \right\rangle$$

$\mathcal{K}_{inc}$ is 1-inconsistent, as illustrated by the 1-distant Herbrand interpretation $\mathcal{I}_{6a}$ where $C^{\mathcal{I}_{6a}} = D^{\mathcal{I}_{6a}} = E^{\mathcal{I}_{6a}} = \{a\}$ and $R^{\mathcal{I}_{6a}} = \{(a, a)\}$. In contrast, the larger Herbrand interpretation $\mathcal{I}_{6b}$ where $C^{\mathcal{I}_{6b}} = D^{\mathcal{I}_{6b}} = \{a, 1\}$, $E^{\mathcal{I}_{6b}} = \{1\}$ and $R^{\mathcal{I}_{6b}} = \{(a, 1)\}$ is 2-distant.

To ensure the discovery of interpretations that omit the least number of axiom instances the conditions under which blocking occurs need to be refined. Informally, the idea is to enable blocking to occur as early as possible in a branch and prevent the introduction of unnecessary parameters. Two techniques are used. Firstly, without loss of generality, the input language for the modified tableau algorithm is restricted such that only *simple* concepts, a concept name A or the negation of a concept name $\neg A$, appear within quantified role restrictions. We refer to a concept appearing within a role restriction as a *quantified concept*. The restriction to simple concepts ensures that blocking conditions are not obfuscated by the presence of complex concepts. A pre-processing step ensures that all the quantified concepts are simple by introducing equivalent concepts to replace them. This step involves a straightforward recursive replacement starting from the ‘innermost’ non-simple concepts within axioms and can be computed in polynomial time. For instance, the concept $\exists R.(C \sqcap \forall R.(D \sqcup E))$ may be simplified as $\exists R.X$ with the definitions $X \equiv C \sqcap \forall R.Y$ and $Y \equiv D \sqcup E$.

Example 4.7. Consider the following $\mathcal{ALC}$ knowledge bases:

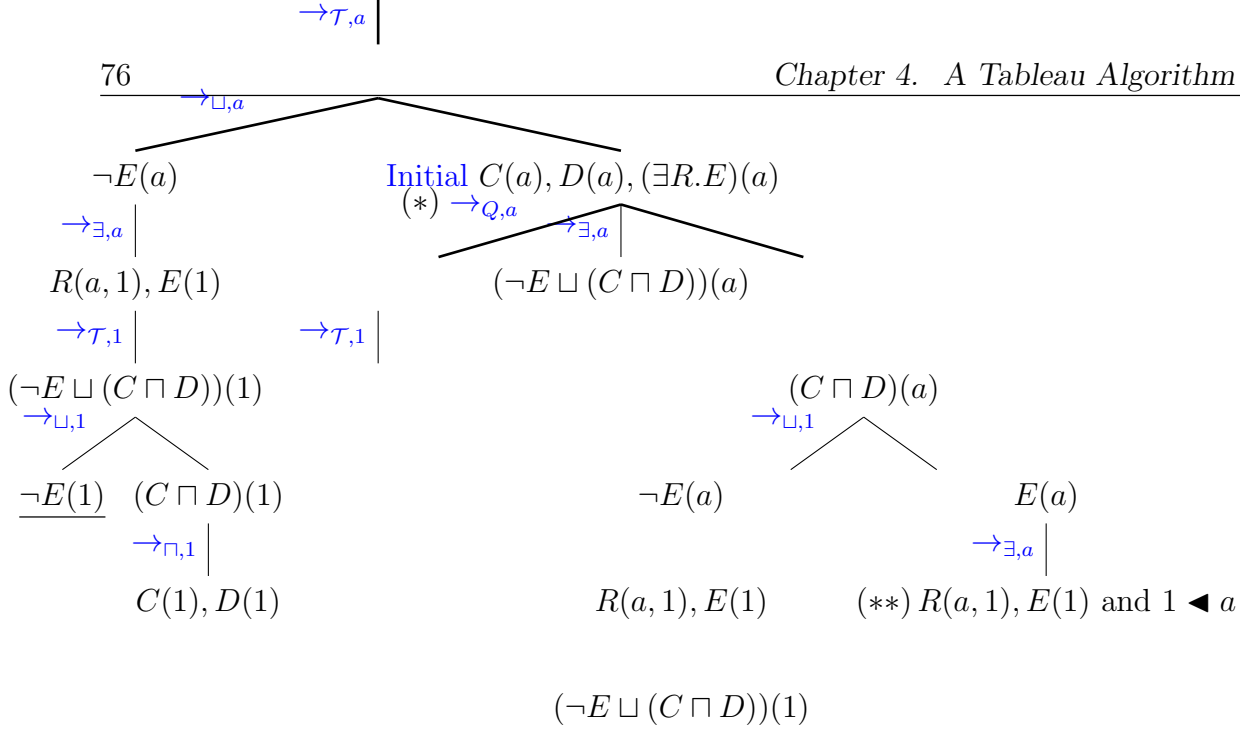
$$\mathcal{K}_{qcc} = \left\langle \left\{ \begin{array}{c} C(a) \\ D(a) \\ (\exists R.(C \sqcap D))(a) \end{array} \right\}, \emptyset \right\rangle \quad \mathcal{K}_{qcs} = \left\langle \left\{ \begin{array}{c} C(a) \\ D(a) \\ (\exists R.X)(a) \end{array} \right\}, \left\{ \begin{array}{l} X \sqsubseteq C \sqcap D \\ C \sqcap D \sqsubseteq X \end{array} \right\} \right\rangle$$

In $\mathcal{K}_{qcs}$ the quantified concept $C \sqcap D$ is replaced by the simple concept X and the definition $X \equiv C \sqcap D$ is added to the non-defeasible TBox of the knowledge base. The $\mathcal{ALC}$ static blocking tableau developed for $\mathcal{K}_{qcc}$ and $\mathcal{K}_{qcs}$ are shown in Figures 4.6 and 4.7 respectively. The parameter in the tableau for $\mathcal{K}_{qcs}$ is blocked and the interpretation based on the branch has a smaller domain size than for the interpretation based on the branch developed for $\mathcal{K}_{qcc}$ in which the parameter is unblocked.

$$\mathcal{K}_{miss} = \left\langle \left\{ \begin{array}{c} C(a) \\ D(a) \\ (\exists R.E)(a) \end{array} \right\}, \left\{ E \sqsubseteq C \sqcap D \right\} \right\rangle$$
$$\begin{array}{c}
\text{Initial } C(a), D(a), (\exists R.E)(a) \\
\rightarrow_{\sqcup, a} \quad \rightarrow_{\exists, a} \\
\swarrow \quad \downarrow \quad \searrow \\
(\neg E \sqcup (\neg E(\bar{a}D)))(a) \\
\rightarrow_{\tau, 1} \mid \quad \rightarrow_{\exists, a} \mid \\
R(a, 1), E(1)(C \sqcap D)(a) \\
\rightarrow_{\tau, 1} \mid \quad \rightarrow_{\sqcup, 1} \\
(\neg E \sqcup (C \sqcap D)) \mid R(1), E(1) \\
\rightarrow_{\sqcup, 1} \mid \quad \rightarrow_{\sqcap, 1} \mid \\
\underline{\neg E(1)} \quad (C \sqcap D) \mid (C \sqcap D)(1) \\
\rightarrow_{\sqcap, 1} \mid \\
\mathcal{C}(\mathbb{1}), \mathcal{D}(\mathbb{1})(C \sqcap D)(1)
\end{array}$$

hand open branch of the tableau the individual a is not specifically assigned to be either $E(a)$ or $\neg E(a)$ and the interpretation based on this branch will assign $\neg E(a)$. However, $\mathcal{K}_{miss}$ can also be satisfied in an interpretation with a domain of size 1 when the interpretation satisfies $E(a)$. Figure 4.9 shows a tableau for $\mathcal{K}_{miss}$ in which we introduce a $\rightarrow_Q$ -rule which “splits” on the quantified concepts. In this case the rule forms two branches, one with $E(a)$ and the other with $\neg E(a)$.

Algorithm 2 incorporates this approach and it is used to construct a single p - $\mathcal{ALC}$ tableau branch from valid expansions formalised in Definition 4.8 using the p - $\mathcal{ALC}$ tableau rules shown in Table 4.4 and the blocking conditions of Definition 4.4. The rules include the static blocking



4.9: The complete $\mathcal{ALC}$ tableau for $\mathcal{K}_{miss}$ in which the concept E is split at $(*)$. 1 is identified by a in the open branch $(**)$. In the Herbrand interpretation $\mathcal{I}_8$ based on the open branch $(**)$, $C^{\mathcal{I}_8} = D^{\mathcal{I}_8} = E^{\mathcal{I}_8} = \{a\}$, $R^{\mathcal{I}_8} = \{(a, a)\}$ and $|\Delta^{\mathcal{I}_8}| = 1$.

rules and two new rules ($\rightarrow_{\mathcal{T}_d}$ and $\rightarrow_Q$). The underlying idea is that the consistency of $\mathcal{ALC}$ knowledge base $\mathcal{K}$ is obtained by checking for the existence of an open branch returned by Algorithm 2 and the n -inconsistency of $\mathcal{K}$ is obtained by searching for an open branch returned by Algorithm 2 that “minimises the omitted set”.

Definition 4.8 ($p\text{-}\mathcal{ALC}$ valid expansion). Let $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ denote a current (open) branch started starting from a given knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ where $\mathcal{A}_b$ is an $\mathcal{ALC}$ ABox and $\mathcal{O}_b$ is called the omitted set. $\mathcal{A}_e$ is a valid expansion of $\mathcal{A}_b$ with respect to $\mathcal{T}$, $\mathcal{T}_d$ and $\mathcal{O}_b$ if and only if $\mathcal{A}_e$ is generated from $\mathcal{A}_b$ by applying an instance of a $p\text{-}\mathcal{ALC}$ tableau rule in Tables 4.2-4.4, with blocking conditions in Definition 4.4. $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$ is said to be closed if there are no valid expansions of $\mathcal{A}_b$ under the set of rules $\mathcal{R}$; $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$ is said to be open if $\mathcal{A}_b$ includes $C(x)$ and $\neg C(x)$ for some concept C , and open otherwise.

Initially, Algorithm 2 starts (in line 1) by choosing $\mathcal{A}_o$, a subset of the defeasible ABox that will be omitted. These are recorded (in line 2) in the omitted set $\mathcal{O}_b$. The branch

Algorithm 2: A $p\text{-}\mathcal{ALC}$ tableau branch generating algorithm

Input: $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$, a finite $p\text{-}\mathcal{ALC}$ knowledge base
Output: $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$, where $\mathcal{A}_b$ is a clash free expanded ABox and $\mathcal{O}_b$ is the *omitted set*
Output: $\perp$ when the branch closes

- 1 Choose $\mathcal{A}_o$ a subset of $\mathcal{A}_d$;
- 2 $\mathcal{O}_b = \{ \langle C(x)^{[w]}, x \rangle \mid C(x)^{[w]} \in \mathcal{A}_o \} \cup \{ \langle R(x, y)^{[w]}, x \rangle \mid R(x, y)^{[w]} \in \mathcal{A}_o \}$;
- 3 $\mathcal{A}_b := \mathcal{A} \cup (\mathcal{A}_d - \mathcal{A}_o)^{-W}$;
- 4 **while** $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$ is open and not completed **do**
 - 5 Choose r an instance of a rule that applies to $\mathcal{A}_b$;
 - 6 $\mathcal{A}_b :=$ Choose an expansion of $\mathcal{A}_b$ by r ;
 - 7 **if** r is $\rightarrow_{\mathcal{T}_d}$ for unblocked x , $C \sqsubseteq D^{[w]}$ and $(\neg C \sqcup D)(x) \notin \mathcal{A}_b$ **then**
 $\mathcal{O}_b := \mathcal{O}_b \cup \{ \langle C \sqsubseteq D^{[w]}, x \rangle \}$;
- 8 **end**
- 9 **if** $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$ is open **then return** $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$;
- 10 **else return** $\perp$;

Rules $\mathcal{R}$	Valid expansions of $\mathcal{A}_b$
$\rightarrow_{\sqcap}$ -rule	If $(C \sqcap D)(x) \in \mathcal{A}_b$, x is not blocked and $(C(x) \notin \mathcal{A}_b \text{ or } D(x) \notin \mathcal{A}_b)$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(x), D(x)\}$
$\rightarrow_{\sqcup}$ -rule	If $(C \sqcup D)(x) \in \mathcal{A}_b$, x is not blocked and $(C(x) \notin \mathcal{A}_b \text{ and } D(x) \notin \mathcal{A}_b)$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(x)\}$ or $\mathcal{A}_e = \mathcal{A}_b \cup \{D(x)\}$
$\rightarrow_{\forall}$ -rule	If $(\forall R.C)(x) \in \mathcal{A}_b$ and x is not blocked and $R(x, y) \in \mathcal{A}_b$ and $C(y) \notin \mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C(y)\}$
$\rightarrow_{\mathcal{T}}$ -rule	If $(C \sqsubseteq D) \in \mathcal{T}$ and $\exists$ an unblocked individual x in $\mathcal{A}_b$ and $(\neg C \sqcup D)(x) \notin \mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{(\neg C \sqcup D)(x)\}$
$\rightarrow_{\exists\forall}$ -rule	If $(\exists R.C)(x) \in \mathcal{A}_b$, x is not blocked and $\neg \exists y [R(x, y) \in \mathcal{A}_b \text{ and } C(y) \in \mathcal{A}_b]$ and no other rule applies to $\mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{R(x, z), C(z)\} \cup \{D(z) \mid (\forall R.D)(x) \in \mathcal{A}_b\}$ where z is fresh
$\rightarrow_{\mathcal{T}_d}$ -rule	If $C \sqsubseteq D^{[w]} \in \mathcal{T}_d$ and there exists an unblocked individual x in $\mathcal{A}_b$ and $((\neg C \sqcup D)(x) \notin \mathcal{A}_b \text{ and } \langle C \sqsubseteq D^{[w]}, x \rangle \notin \mathcal{O}_b)$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{(\neg C \sqcup D)(x)\}$ or $\mathcal{A}_e = \mathcal{A}_b$
$\rightarrow_Q$ -rule	If $C_q \in \mathcal{Q}(\mathcal{K})$ and there exists an unblocked individual x in $\mathcal{A}_b$ and $C_q(x) \notin \mathcal{A}_b$ and $\neg C_q(x) \notin \mathcal{A}_b$ then $\mathcal{A}_e = \mathcal{A}_b \cup \{C_q(x)\}$ or $\mathcal{A}_e = \mathcal{A}_b \cup \{\neg C_q(x)\}$

Table 4.4: The $p\text{-}\mathcal{ALC}$ tableau rules include two new rules, the $\rightarrow_{\mathcal{T}_d}$ -rule and $\rightarrow_Q$ -rule.

ABox $\mathcal{A}_b$ is initialised (in line 3) with the non-defeasible ABox axioms and the remaining (those not chosen to be omitted) defeasible ABox axioms. ABox axioms are “expanded”, at lines 5-7 within the while loop starting at line 4. The application of the $\rightarrow_{\mathcal{T}_d}$ rule to defeasible TBox axioms may or may not lead to the addition of a concept assertion to the ABox of the branch. The omitted set $\mathcal{O}_b$ records (in line 7) the defeasible TBox axiom instances for the $\rightarrow_{\mathcal{T}_d}$ rule applications that do not add a concept assertion. The Algorithm continues until (line 4) the

branch closes because a clash is found or the branch is completed because there are no more valid expansions of the ABox. Finally, the Algorithm either returns an open branch (line 9) or $\perp$ (line 10) when the branch is closed. We will show later that given a finite $p\text{-}\mathcal{ALC}$ knowledge base Algorithm 2 always terminates. Note that by changing the initial choice of defeasible ABox axioms and/or choosing different additions to a branch of defeasible TBox related axiom instances, Algorithm 2 can generate many possible branches.

We now formalise the notation used for branches developed by Algorithm 2. Given a finite $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$, the application of Algorithm 2 to $\mathcal{K}$ generates an open branch $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$ or it returns $\perp$ indicating that the developing branch closed. The *distance* of an open branch refers to the sum of the weights of the defeasible axiom instances recorded in the omitted set $\mathcal{O}_b$. An open branch is called an *m-minimal branch* if there is no other open branch whose distance is smaller than m .

A $p\text{-}\mathcal{ALC}$ tableau formed from branches generated by our algorithm is visualised as a set of branches leading from the root node. The root node is labelled with the non-defeasible ABox axioms which are common to all branches. In each branch, a subset of the non-defeasible ABox $\mathcal{A}$ is chosen. There are $2^{|\mathcal{A}_d|}$ different possible omissions from $\mathcal{A}_d$, and these are represented by edges (labelled $\subseteq \mathcal{A}_d$) from the root node. The elements added to the omitted set $\mathcal{O}_b$ are recoded at each node. These are easily distinguished from the axioms of $\mathcal{A}_b$ as they are tuples of the form $\langle Z^{[w]}, x \rangle$. The non-deterministic results of applying the $\rightarrow_{\mathcal{T}_d}$ -rule are represented by two edges, one in which the instance is included and the other in which it is omitted. The branch Abox $\mathcal{A}_b$ (resp. omitted set $\mathcal{O}_b$) is easily constructed from the tree by taking the union of axiom labels (omitted instance labels) in the nodes of the path from the leaf node to the root of the tree.

Example 4.9. Recall, $\mathcal{K}_{inc} = \langle \{C(a), \neg D(a)\}, \emptyset, \{C(b)^{[1]}\}, \{C \sqsubseteq D^{[1]}\} \rangle$ from Example 3.4. The knowledge base illustrates how the defeasibility of the TBox axiom $C \sqsubseteq D^{[1]}$ accommodates the inconsistency w.r.t. $C(a), \neg D(a)$ but allows the inference of $D(b)$ from $C(b)^{[1]}$. Figure 4.10 shows a set of completed branches developed by Algorithm 2 for $\mathcal{K}_{inc}$.

Initial $G(a), \neg S(a), (\forall R.\neg S)(a), S(b), (\exists R.G)(b), (\forall R.S)(b)$

$\rightarrow_{\exists\forall, b}$ |

$\rightarrow_{\mathcal{T}_{a,1}}$ $\langle R(a, b)^{[1]}, a \rangle$
 $(\neg H \sqcup \neg S)(b)$

$\neg H(b)$

$\neg S(b)$

$\langle G \sqsubseteq H^{[1]}, a \rangle$

$\rightarrow_{\sqcup, b}$

$\rightarrow_{\mathcal{T}_{a,b}}$
 $\rightarrow_{\sqcup, \mathcal{T}_{a,1}}$
 $\rightarrow_{\mathcal{T}_1}$
 $(\neg G \sqcup H)(b)$
 $\vdots$
 $\neg \underline{G(b)}$ $\underline{H(b)}$ $H(a)$

$\langle G, \underline{H}^{[1]}, b \rangle$

$R(b, 1), G(\langle G, \underline{H}^{[1]}, 1 \rangle)$
2-distant

[illegible]

Figure 4.12: A $p\text{-}\mathcal{ALC}$ tableau for $\mathcal{K}_{inc}$ in which the concept E is split at $(*)$. 1 is blocked by a in the open branch $(**)$. In the Herbrand interpretation $\mathcal{I}_9$ based on the 1-distant open branch $(**)$, $C^{\mathcal{I}_9} = D^{\mathcal{I}_9} = E^{\mathcal{I}_9} = \{a\}$, $R^{\mathcal{I}_9} = \{(a, a)\}$, $|\Delta^{\mathcal{I}_9}| = 1$, $\mathcal{U}(\mathcal{K}, \mathcal{I}_9) = \langle C \sqsubseteq D^{[1]}, a \rangle$ and $\mathcal{I}_9$ is 1-distant.

4.3 General properties of the $p\text{-}\mathcal{ALC}$ tableau algorithm

In this section, we prove properties of Algorithm 2 and properties of the branches that it generates. We begin by showing that the modified algorithm preserves the static blocking strategy and hence that the algorithm terminates for any (finite) $p\text{-}\mathcal{ALC}$ knowledge base. Next, we show the algorithm returns no open branches if, and only if, the knowledge base is not credible (the non-defeasible axioms are inconsistent). This property is required in refutation proofs when the negated query is inconsistent with the non-defeasible axioms. We then show six properties of an open branch that collectively define the *saturation* of a branch that are used to demonstrate the *soundness*, *completeness* and *correctness* of the algorithm in Sections 4.4-4.6.

4.3.1 Static blocking

We begin by introducing some notation to describe the detailed operation of blocking in Algorithm 2 and show that it still leads to a static strategy.

We recall from Definition 4.4 that the development of an ABox by a tableau algorithm can be represented as a finite sequence of ABoxes. Let $\mathcal{S}$ denote the sequence of ABoxes $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \dots \subseteq \mathcal{A}_l$ developed by Algorithm 2 where $l \geq 0$. $\mathcal{A}_0$ denotes the initial assignment of axioms in line 3 of Algorithm 2 and each $\mathcal{A}_k$, where $1 \leq k \leq l$ denotes the axioms in a branch after the k -th tableau rule instance is applied in line 6. The named individuals in the sequence $\mathcal{S}$ are introduced in $\mathcal{A}_0$, and fresh parameters in $\mathcal{S}$ are introduced at each application of the $\rightarrow_{\exists\forall}$ -rule.

Observation 1. *Named individuals are never blocked because Definition 4.4 condition (i) requires blocking by an older individual and no individual is older than those introduced at $\mathcal{A}_0$.*

In contrast, a parameter introduced in $\mathcal{A}_k$ may be either blocked or unblocked in $\mathcal{A}_k$ according to Definition 4.4 condition (ii). Recall from Definition 4.8 that the precondition of the $\rightarrow_{\exists\forall}$ rule mandates that all other applicable rules have been applied before it is applied. Therefore, a

sequence $\mathcal{S}$ representing a branch can be seen as partitioned into *subsequences* where the start of each subsequence corresponds to the application an $\rightarrow_{\exists\forall}$ -rule instance.

Definition 4.9 (A p - $\mathcal{ALC}$ tableau branch subsequence). *Let $\mathcal{S}$ be the sequence of ABoxes, $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \dots \subseteq \mathcal{A}_l$ where $l \geq 0$ constituting the branch b . $Sub_{i,j} = \mathcal{A}_i \subseteq \dots \subseteq \mathcal{A}_j$ where $0 \leq i \leq j \leq l$ is a subsequence of $\mathcal{S}$ iff*

- (1) $i = 0$ or the rule applied at $\mathcal{A}_i$, where $i > 0$, is a $\rightarrow_{\exists\forall}$ rule instance;
- (2) each rule applied at $\mathcal{A}_k$ where $i < k \leq j$ is not a $\rightarrow_{\exists\forall}$ rule instance and;
- (3) $j = l$ or the rule applied at $\mathcal{A}_{j+1}$ is a $\rightarrow_{\exists\forall}$ rule instance.

Lemma 4.10 and its corollary given next capture the properties that underpin the *static* blocking strategy used by the algorithm.

Given a sequence $\mathcal{A}_0 \subseteq \dots \subseteq \mathcal{A}_l$, we use the notation $\mathcal{C}_x^i$ to denote $\{C \mid C(x) \in \mathcal{A}_i\}$ where $0 \leq i \leq l$. Informally, $\mathcal{C}_x^i$ denotes the *concepts of x at $\mathcal{A}_i$* , the set of concepts that the ABox asserts for the individual x at the i -th step in the sequence. We show in Lemma 4.10 that a subsequence which introduces a new parameter p adds concept assertions for p only, and no other name. Given a sequence $\mathcal{S}$, a subsequence $Sub_{i,j}$ of $\mathcal{S}$ is said to be *earlier* (later) than subsequence $Sub_{i',j'}$ of $\mathcal{S}$ iff $j < i'$ (resp. $i > j'$).

Lemma 4.10. *Let $Sub_{i,j}$ be a subsequence of $\mathcal{A}_0 \subseteq \dots \subseteq \mathcal{A}_l$, the sequence of a branch b , where $0 < i \leq j \leq l$, and p be the parameter introduced at $\mathcal{A}_i$ as a witness for an (older) individual x . Then $\mathcal{A}_j = \mathcal{A}_i \cup \{R(x, p)\} \cup \mathcal{Z}$, where $\mathcal{Z}$ is a set of a concept assertions of the form $C(p)$.*

Proof. The axioms added to the branch at $\mathcal{A}_i$ by the $\rightarrow_{\exists\forall}$ rule instance are assertions of the form $R(x, p)$ and assertions of the form $D(p)$, where D are simple concepts. For each $\mathcal{A}_{i+1} \subseteq \dots \subseteq \mathcal{A}_j$, the only applicable rules are instances of the $\rightarrow_{\sqcap}$, $\rightarrow_{\sqcup}$, $\rightarrow_{\tau}$, $\rightarrow_{\tau_d}$ or $\rightarrow_Q$ rule for the new parameter p , which introduce a (possibly empty) set of concept assertions of the form $C(p)$. □

As a corollary, a subsequence does not alter the concepts of older individuals.

Corollary 4.11. *Let $\text{Sub}_{i,j}$ be a subsequence of $\mathcal{A}_0 \subseteq \dots \subseteq \mathcal{A}_l$, the sequence of a branch b , where $0 < i \leq j \leq l$, and p be the parameter introduced at $\mathcal{A}_i$. Let y be an individual older than p in b . Then $\mathcal{C}_y^j = \mathcal{C}_y^i$.*

Proposition 4.12 (Algorithm 2 results in a static blocking strategy). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a $p\text{-}\mathcal{ALC}$ knowledge base and $\mathcal{S} = \mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \dots \subseteq \mathcal{A}_l$, $l \geq 0$ denote the sequence of a branch b developed by Algorithm 2 applied to $\mathcal{K}$. Let $\text{Sub}_{i,j}$ be a subsequence of $\mathcal{S}$ and y be an individual introduced at $\mathcal{A}_i$. Then the following properties hold:*

- (a) *For all $k > j$, $\mathcal{C}_y^k = \mathcal{C}_y^j$;*
- (b) *For all $k > i$, y is blocked in $\mathcal{A}_k$ iff y is blocked in $\mathcal{A}_i$.*

Proof. Each property is justified by considering the operation of Algorithm 2.

- (a) By assumption $k > j$ and every step after j belongs to some later subsequence. y is older than any parameter p introduced in a later subsequence and by Corollary 4.11 we conclude $\mathcal{C}_y^j$ are unchanged by the sequence. We conclude $\mathcal{C}_y^k = \mathcal{C}_y^j$.

(b) **The “if” case.**

Take an arbitrary $k > i$ and assume y is blocked in $\mathcal{A}_i$. By Observation 1, y is a parameter. From Definition 4.4 there exists some older individual x introduced in an earlier sequence $\mathcal{A}_g \subseteq \dots \subseteq \mathcal{A}_h$, $h \leq g < i$, that blocks y s.t. $\mathcal{C}_y^i \subseteq \mathcal{C}_x^i$. The concepts in $\mathcal{C}_y^i$ are exactly those concepts introduced by the $\rightarrow_{\exists\forall}$ -rule at step i and no others. By Definition 4.8 blocked individuals are not expanded and, since y is blocked in $\mathcal{A}_i$, no further rules apply in this sequence. We conclude $j = i$ and $\mathcal{C}_y^j = \mathcal{C}_y^i$. By (a), the concepts of y are unchanged after $\mathcal{A}_j$ and since $k > j = i$ we can conclude $\mathcal{C}_y^k = \mathcal{C}_y^j = \mathcal{C}_y^i$. By (a), the concepts of x are unchanged after $\mathcal{A}_h$ and since $k > j = i > h$, $\mathcal{C}_x^k = \mathcal{C}_x^j = \mathcal{C}_x^i = \mathcal{C}_x^h$. Now if $\mathcal{C}_x^k = \mathcal{C}_x^i$, $\mathcal{C}_y^k = \mathcal{C}_y^i$ and $\mathcal{C}_y^i \subseteq \mathcal{C}_x^i$ we conclude $\mathcal{C}_y^k \subseteq \mathcal{C}_x^k$ and y is blocked at k .

The “only if” case.

We prove the contrapositive. Take an arbitrary $k > i$, and assume y is not blocked in $\mathcal{A}_i$.

We show y is not blocked in $\mathcal{A}_k$. There are two subcases:

- i. y is a named individual. By Observation 1, y is not blocked in $\mathcal{A}_k$;
- ii. y is a parameter. Since y is not blocked each older individual x does not block y in $\mathcal{A}_i$ and $\mathcal{C}_y^i \not\subseteq \mathcal{C}_x^i$. There are two subcases:
 - 1) $k \leq j$. By Lemma 4.10 the rule instances applied in the sequence from steps i to k introduce concept assertions of the form $C(y)$. We conclude $\mathcal{C}_y^k \supseteq \mathcal{C}_y^i$. By (a), the concepts of older individuals are unchanged by later sequences, for each x older than y , $\mathcal{C}_x^k = \mathcal{C}_x^i$. We conclude that $\mathcal{C}_y^i - \mathcal{C}_x^i \neq \emptyset$ and hence $\mathcal{C}_y^k - \mathcal{C}_x^k \neq \emptyset$, i.e. $\mathcal{C}_y^k \not\subseteq \mathcal{C}_x^k$ and y is not blocked in $\mathcal{A}_k$;
 - 2) $k > j$. Following the argument in 1) y is not blocked up to and including $\mathcal{A}_j$ and by (a), the concepts of y and the concepts of individuals older than y remain unchanged after $\mathcal{A}_j$. We conclude y is not blocked in $\mathcal{A}_k$.

□

The following observations summarise the behaviour under the static blocking strategy:

Observation 2. *A parameter introduced by an instance of the $\rightarrow\exists\forall$ rule at $\mathcal{A}_i$ is either: blocked at $\mathcal{A}_i$ whence it remains blocked in $\mathcal{A}_k$, $k > i$; or it is not blocked whence it remains unblocked in $\mathcal{A}_k$, $k > i$.*

Observation 3. *The language of $p\text{-}\mathcal{ALC}$ restricts quantified concepts to simple concepts. By Observation 2, if a parameter p is blocked then it must have been blocked on introduction (and remains blocked) and the concepts of p are simple. It cannot be blocked if it was not blocked on introduction. We conclude that if a parameter p is blocked then the concepts of p are simple.*

Observation 4. *Any parameter p that is blocked is blocked by at least one unblocked individual. There may be other individuals that block p which are blocked. However, the oldest individual that blocks p must be unblocked (only an older individual can block).*

Observation 5. *The precondition of each rule restricts its application to unblocked individuals. Hence, if a rule is applied to an individual x during the development of a branch then x was unblocked in the branch when the rule was applied. By Observation 1, named individuals are*

never blocked and by Observation 2, parameters are either blocked on introduction and no rules are applied to them, or they remain unblocked. Hence, we conclude that if an expansion rule is applied to an individual x in the development of the branch then x is not blocked in the completed branch.

4.3.2 Termination

The proof that Algorithm 2 terminates for any finite input knowledge base relies on Lemma 4.14 which captures that a finite $p\text{-}\mathcal{ALC}$ knowledge base has a finite number of sub-formulae.

For the purpose of our analysis the notion of the set of analytic sub-formulae of $\mathcal{K}$ is introduced. It defines the set of formulae that can be introduced by Algorithm 2. Recall that $\mathcal{F}(\mathcal{K})$ denotes the subformula of $\mathcal{K}$ (Definition 3.2) and $\mathcal{Q}(\mathcal{K})$ denotes the quantified concepts in $\mathcal{K}$ (Definition 4.7).

Definition 4.13 (Analytic sub-formulae of a $p\text{-}\mathcal{ALC}$ knowledge base). *Let $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base. The analytic sub-formulae of $\mathcal{K}$, denoted $\mathcal{F}_A(\mathcal{K})$, are $\mathcal{F}(\mathcal{K}) \cup \mathcal{Q}(\mathcal{K})$.*

Lemma 4.14 (Every $\mathcal{K}$ has finite analytic sub-formulae). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a finite $p\text{-}\mathcal{ALC}$ knowledge base and $\mathcal{F}_A(\mathcal{K})$ denote the analytic sub-formulae of $\mathcal{K}$ as given by Definition 4.13. Then $\mathcal{F}_A(\mathcal{K})$ is finite.*

Proof. By assumption $\mathcal{K}$ is finite and therefore the number of axioms in $\mathcal{A}, \mathcal{T}, \mathcal{A}_d$ and $\mathcal{T}_d$ is finite and each axiom is of finite size. The definition of $\mathcal{F}_A(\mathcal{K})$ given in Definition 4.13 introduces:

- at most one finite concept for each assertion in $\mathcal{A}$;
- at most one finite concept for each defeasible assertion in $\mathcal{A}_d$;
- at most one finite concept for each TBox axiom in $\mathcal{T}$;
- at most one finite concept for each defeasible TBox axiom in $\mathcal{T}_d$;
- at most two finite concepts for each $\mathcal{Q}(\mathcal{K})$;

- the (finite) analytic sub-formulae of all such finite concepts.

Hence, $\mathcal{F}_A(\mathcal{K})$ is finite. □

Clearly, all concepts in a branch generated by Algorithm 2 belong to $\mathcal{F}_A(\mathcal{K})$.

Proposition 4.15 (Algorithm 2 is terminating). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a finite $p\text{-}\mathcal{ALC}$ knowledge base. Then every branch b generated by Algorithm 2 applied to $\mathcal{K}$ either closes or is completed after a finite number of rule applications.*

Proof. The proof is by contradiction. Let $\mathcal{F}_A(\mathcal{K})$ denote the set of analytic sub-formulae of $\mathcal{K}$. By assumption, $\mathcal{K}$ is finite and by Lemma 4.14, $\mathcal{F}_A(\mathcal{K})$ is finite. Let us assume for contradiction that there exists a branch b that includes an infinite sequence of rule applications. Such a sequence can be formed by either (i) performing an infinite number of rule applications to a (finite) set of individuals, or (ii) introducing infinitely many individuals to the sequence. Taking each case:

- (i) By assumption, the sequence is infinite and the number of parameters introduced (and therefore the total number of individuals in the sequence) is finite. Hence, at some point in the development of the sequence the $\rightarrow_{\exists\forall}$ -rule is no longer applicable. The precondition of each rule guarantees that a specific instance of that rule will be applied to an individual at most once. By the finiteness of $\mathcal{F}_A(\mathcal{K})$ and of the number of individuals in the sequence, the number of unique assertions is also finite. Hence, the sequence terminates, a contradiction with the assumption of an infinite sequence.
- (ii) By assumption, infinitely many individuals are introduced into the sequence. There are infinitely many unblocked parameters introduced. By Definition 4.4, an individual y , introduced at step k of a sequence, is unblocked if there is no older individual x such that $\mathcal{C}_y^k \subseteq \mathcal{C}_x^k$. All concepts introduced belong to $\mathcal{F}_A$. Hence, $\mathcal{C}_y^k \subseteq \mathcal{F}_A(\mathcal{K})$ and $\mathcal{C}_x^k \subseteq \mathcal{F}_A(\mathcal{K})$. There are at most $2^{|\mathcal{F}_A(\mathcal{K})|}$ unique sets S where $S \subseteq \mathcal{F}_A(\mathcal{K})$. Now by the finiteness of $\mathcal{F}_A(\mathcal{K})$, eventually any possible parameter introduced is blocked and the sequence terminates, a contradiction with the assumption of an infinite sequence.

□

4.3.3 All branches close

Recall from Definition 3.4 condition (i) that a preferred interpretation of a $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$. Hence, a knowledge base with an inconsistent $\langle \mathcal{A}, \mathcal{T} \rangle$ (is not credible) has no preferred interpretations. We show that Algorithm 2 returns $\perp$ for such knowledge bases.

Proposition 4.16 (Algorithm 2 returns $\perp$ iff $\langle \mathcal{A}, \mathcal{T} \rangle$ is inconsistent). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a finite $p\text{-}\mathcal{ALC}$ knowledge base and $\mathcal{B}$ be the set of open branches generated by Algorithm 2. Then $\langle \mathcal{A}, \mathcal{T} \rangle$ is inconsistent if, and only if, $\mathcal{B} = \emptyset$.*

Proof. First suppose that $\mathcal{B} = \emptyset$. In particular, one of the tableau branches developed by Algorithm 2 will be the one in which $\mathcal{A}_o = \mathcal{A}_d$ and every application of the rule $\rightarrow_{\mathcal{T}_d}$ leaves $\mathcal{A}_e$ unchanged. By assumption the result is $\perp$. Since this is the tableau for $\langle \mathcal{A}, \mathcal{T} \rangle$ it shows $\langle \mathcal{A}, \mathcal{T} \rangle$ is inconsistent by the property of completeness for $\mathcal{ALC}$ tableaux. Next, suppose that $\langle \mathcal{A}, \mathcal{T} \rangle$ is inconsistent. In each branch developed by Algorithm 2, the $\mathcal{ALC}$ rules $\rightarrow_{\sqcap}, \rightarrow_{\sqcup}, \rightarrow_{\forall}, \rightarrow_{\exists\forall}$ and $\rightarrow_{\mathcal{T}}$ are applied to the axioms of $\mathcal{A}$ and to the axioms that result from such expansions. By the soundness of $\mathcal{ALC}$ tableaux, any tableau that applies such rules will close. Algorithm 2 returns $\perp$ for such cases. Hence, $\mathcal{B} = \emptyset$. □

4.3.4 Saturation

We define and prove seven properties of the open completed branches returned by Algorithm 2. These properties are used to capture the notion of a *saturated* branch. Informally, a saturated branch: (1) includes the original ABox; (2) includes a subset of the defeasible ABox (without weights) and records those not included in the omitted set; (3) has applied the $\rightarrow_{\mathcal{T}}$ -rule to every individual, (4) has applied the $\rightarrow_{\mathcal{T}_d}$ -rule to every individual and records those that did not introduce axiom instances in the omitted set; (5) has applied every valid $\rightarrow_{\sqcap}$ and $\rightarrow_{\sqcup}$ -rule

expansion; (6) has applied every valid $\rightarrow_Q$ -rule expansion; and (7) has applied every valid $\rightarrow_V$ and $\rightarrow_{\exists V}$ -rule expansion. These properties will be used later to show that the Algorithm operates correctly.

We begin by making two observations:

Observation 6. *Every open branch returned by Algorithm 2 is completed.*

Observation 7. *No steps of Algorithm 2 remove elements from a branch.*

Definitions 4.17-4.21 are used to characterise the properties of an open saturated branch that do not relate to blocked individuals. In contrast, Definition 4.22 captures the properties of axioms in the branch relating to blocked individuals. This distinction is important because no rules are applied to blocked individuals and hence, in a completed branch, they do not satisfy Definitions 4.17-4.21.

Definition 4.17 ($\mathcal{A}_d$ omissions). *Let $\mathcal{A}$ be an ABox in which no individual is blocked, $\mathcal{A}_d$ be a defeasible ABox and $\mathcal{O}$ be a set of axiom instances. $\mathcal{O}$ is said to record the omissions of $\mathcal{A}_d$ from $\mathcal{A}$ iff*

(i) *for each axiom $C(x)^{[w]} \in \mathcal{A}_d$ then either $C(x) \in \mathcal{A}$ or $\langle C(x), x \rangle \in \mathcal{O}$; and*

(ii) *for each axiom $R(x, y)^{[w]} \in \mathcal{A}_d$ then either $R(x, y) \in \mathcal{A}$ or $\langle R(x, y), x \rangle \in \mathcal{O}$.*

Definition 4.18 ($\mathcal{T}$ -saturated). *Let $\mathcal{A}$ be an ABox in which no individuals are blocked and $\mathcal{T}$ be a TBox. $\mathcal{A}$ is called $\mathcal{T}$ -saturated iff for every individual x in $\mathcal{A}$ and each axiom $C \sqsubseteq D \in \mathcal{T}$ then $(\neg C \sqcup D)(x) \in \mathcal{A}$.*

Definition 4.19 ($\mathcal{T}_d$ -saturation omissions). *Let $\mathcal{A}$ be an ABox in which no individuals are blocked, $\mathcal{T}_d$ be a defeasible TBox and $\mathcal{O}$ be a set of axiom instances. $\mathcal{O}$ is said to record the omissions of $\mathcal{T}_d$ -saturation from $\mathcal{A}$ iff for every individual x in $\mathcal{A}$ and each axiom $C \sqsubseteq D^{[w]} \in \mathcal{T}_d$ then either $(\neg C \sqcup D)(x) \in \mathcal{A}$ or $\langle C \sqsubseteq D^{[w]}, x \rangle \in \mathcal{O}$.*

Definition 4.20 (Boolean downward saturated). *Let $\mathcal{A}$ be an ABox in which no individuals are blocked. $\mathcal{A}$ is called boolean downward saturated iff $\mathcal{A}$ satisfies the following properties for each individual x in $\mathcal{A}$:*

(i) If $(C \sqcap D)(x) \in \mathcal{A}$ then $C(x) \in \mathcal{A}$ and $D(x) \in \mathcal{A}$

(ii) If $(C \sqcup D)(x) \in \mathcal{A}$ then $C(x) \in \mathcal{A}$ or $D(x) \in \mathcal{A}$

Definition 4.21 (QC-split). Let $\mathcal{A}$ be an ABox in which no individuals are blocked and $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base. $\mathcal{A}$ is QC-split w.r.t. $\mathcal{K}$ iff for each individual x in $\mathcal{A}$ and each concept $C \in \mathcal{Q}(\mathcal{K})$ then $C(x) \in \mathcal{A}$ or $\neg C(x) \in \mathcal{A}$.

Definition 4.22 (Quantifier downward saturated). Let $\mathcal{A}$ be an ABox. $\mathcal{A}$ is quantifier downward saturated iff $\mathcal{A}$ satisfies the following properties for each unblocked individual x in $\mathcal{A}$:

(i) If $(\exists R.C)(x) \in \mathcal{A}$ then there exists some y s.t. $R(x, y) \in \mathcal{A}$ and $C(y) \in \mathcal{A}$

(ii) If $(\forall R.C)(x) \in \mathcal{A}$, $R(x, y) \in \mathcal{A}$ then $C(y) \in \mathcal{A}$

Proposition 4.23 (Open branches are saturated). Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible $p\text{-}\mathcal{ALC}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$. Let $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ be an open completed branch for $\mathcal{K}$ and $\mathcal{A}'_b$ denote the axioms in $\mathcal{A}_b$ not involving blocked parameters. Then b satisfies the following properties:

1. $\mathcal{A} \subseteq \mathcal{A}'_b$;
2. $\mathcal{O}_b$ records the omissions of $\mathcal{A}_d$ from $\mathcal{A}'_b$;
3. $\mathcal{A}'_b$ is $\mathcal{T}$ -saturated;
4. $\mathcal{O}_b$ records the omissions of $\mathcal{T}_d$ -saturation from $\mathcal{A}'_b$;
5. $\mathcal{A}'_b$ is boolean downward saturated;
6. $\mathcal{A}'_b$ is QC-split w.r.t. $\mathcal{K}$;
7. $\mathcal{A}_b$ is quantifier downward saturated.

An open branch b is said to be saturated iff it satisfies properties 1-7.

Proof. Properties 1-6 relate to $\mathcal{A}'_b$, the subset of $\mathcal{A}_b$ that involves only unblocked individuals, whereas Property 7 relates to $\mathcal{A}_b$, the whole ABox including blocked individuals. No individual in $\mathcal{A}'_b$ is blocked. Recall that (Observation 5) under the static blocking strategy, if a rule is applied to an unblocked individual then the individual remains unblocked. This property is used throughout the proof to justify that an axiom introduced by a rule application is in $\mathcal{A}'_b$.

1. The axioms in $\mathcal{A}$ are assigned to the branch in step 3. Hence, $\mathcal{A} \subseteq \mathcal{A}_b$. All individuals occurring in the axioms of $\mathcal{A}$ are named, and by Observation 1, are not blocked. We conclude that $\mathcal{A} \subseteq \mathcal{A}'_b$.
2. Suppose $\mathcal{A}_d$ includes an axiom of the form $C(x)^{[w]}$. For any (unblocked) individual x in $\mathcal{A}'_b$ there are two cases:
 - (i) $C(x)^{[w]}$ was not chosen as part of $\mathcal{A}_o$ in step 1. $C(x)$ was added to $\mathcal{A}_b$ in step 3 and since $x \in N_I$ by Observation 1 x is not blocked in the completed branch, $C(x) \in \mathcal{A}'_b$.
 - (ii) $C(x)^{[w]}$ was chosen as part of $\mathcal{A}_o$ in step 1. $\langle C(x)^{[w]}, x \rangle$ was assigned to $\mathcal{O}_{init}$ (in step 2) and therefore $\langle C(x)^{[w]}, x \rangle \in \mathcal{O}_b$ (step 12);

The argument for defeasible axioms of the form $R(x, y)^{[w]}$ is similar.

3. Suppose $\mathcal{T}$ includes an axiom of the form $C \sqsubseteq D$. For any (unblocked) individual x in $\mathcal{A}'_b$ there are two cases
 - (i) the $\rightarrow_{\mathcal{T}}$ rule precondition was unsatisfied because $(\neg C \sqcup D)(x)$ was already in $\mathcal{A}_b$;
 - (ii) the $\rightarrow_{\mathcal{T}}$ rule precondition was initially satisfied, the rule was applied and $(\neg C \sqcup D)(x)$ was added to $\mathcal{A}_b$.

By Observation 5, x is unblocked when the $\rightarrow_{\mathcal{T}}$ -rule is applied and remains unblocked. Hence, $(\neg C \sqcup D)(x)$ is also in $\mathcal{A}'_b$.

4. Suppose $\mathcal{T}_d$ includes an axiom of the form $C \sqsubseteq D^{[w]}$. For any (unblocked) individual x in $\mathcal{A}'_b$ there are two cases:

- (i) the $\rightarrow_{\tau_d}$ rule precondition was unsatisfied because $(\neg C \sqcup D)(x)$ was already in $\mathcal{A}_b$ and by Observation 5, $(\neg C \sqcup D)(x)$ is also in $\mathcal{A}'_b$.
- (ii) the $\rightarrow_{\tau_d}$ rule precondition was initially satisfied, the rule was applied, and there are two subcases:
 - a) the rule added $(\neg C \sqcup D)(x)$ to $\mathcal{A}_b$ and by Observation 5, $(\neg C \sqcup D)(x)$ is also in $\mathcal{A}'_b$.
 - b) the rule added $\langle (C \sqsubseteq D)^{[w]}, x \rangle$ to $\mathcal{O}_b$.

5. There are subcases for each of the boolean downward saturated properties:

- (i) Suppose $(C \sqcap D)(x) \in \mathcal{A}'_b$ and therefore x is not blocked. There are two subcases:
 - a) the $\rightarrow_{\sqcap}$ rule precondition was unsatisfied because both $C(x)$ and $D(x)$ were already present in $\mathcal{A}_b$;
 - b) the $\rightarrow_{\sqcap}$ rule precondition was initially satisfied, the rule was applied and $C(x)$ and $D(x)$ were added to $\mathcal{A}_b$.

By Observation 5, $C(x)$ and $D(x)$ are also in $\mathcal{A}'_b$.

- (ii) Suppose $(C \sqcup D)(x) \in \mathcal{A}'_b$ and therefore x is not blocked. There are two subcases:
 - a) the $\rightarrow_{\sqcup}$ rule precondition was unsatisfied because either $C(x)$ or $D(x)$ were already present in $\mathcal{A}_b$;
 - b) the $\rightarrow_{\sqcup}$ rule precondition was initially satisfied, the rule was applied and either $C(x)$ or $D(x)$ was added to $\mathcal{A}_b$.

By Observation 5, $C(x)$ or $D(x)$ is also in $\mathcal{A}'_b$.

6. Suppose an axiom of $\mathcal{K}$ includes a quantified concept $C \in \mathcal{Q}(\mathcal{K})$. For any (unblocked) individual x in $\mathcal{A}'_b$ there are two subcases:

- a) the $\rightarrow_Q$ rule precondition was unsatisfied because either $C(x)$ or $\neg C(x)$ is already present in $\mathcal{A}_b$;
- b) the $\rightarrow_Q$ rule precondition was initially satisfied, the rule was applied and either $C(x)$ or $\neg C(x)$ was added to $\mathcal{A}_b$.

By Observation 5, $C(x)$ and $\neg C(x)$ are also in $\mathcal{A}_b$.

7. There are subcases for each of the quantifier downward saturated properties:

- (i) Suppose $(\exists R, C)(x) \in \mathcal{A}_b$. $(\exists R, C)$ is not simple, by Observation 3 x is not a blocked parameter and by Observation 1 only parameters become blocked. We conclude x is not blocked and therefore application of the $\rightarrow_{\exists\forall}$ rule must be considered. There are two subcases:
 - a) the $\rightarrow_{\exists\forall}$ rule precondition was unsatisfied because there already existed some y s.t. $R(x, y) \in \mathcal{A}_b$ and $C(y) \in \mathcal{A}_b$;
 - b) the $\rightarrow_{\exists\forall}$ rule precondition was initially satisfied, the rule was applied and introduced $R(x, p)$ and $C(p)$ to $\mathcal{A}_b$ for some parameter p fresh in the branch.
- (ii) Suppose $(\forall R.C)(x) \in \mathcal{A}_b$. $(\forall R.C)$ is not simple and, following a similar argument to (i), x is not blocked. There are two sub-cases:
 - a) $y \in N_I$ and therefore application of the $\rightarrow_{\forall}$ rule must be considered. There are two further subcases:
 - I. the $\rightarrow_{\forall}$ rule precondition was unsatisfied because there already existed some named individual y s.t. $R(x, y) \in \mathcal{A}_b$, $C(y) \in \mathcal{A}_b$ and $\mathcal{A}_b$;
 - II. the $\rightarrow_{\forall}$ rule precondition was not satisfied, the $\rightarrow_{\forall}$ rule was applied to y where $R(x, y) \in \mathcal{A}_b$ and introduced $C(y)$.
 - b) $y \notin N_I$, it was a parameter introduced by the application of a $\rightarrow_{\exists\forall}$ rule instance. The rule application introduced $R(x, y)$, $D(y)$ for some existentially quantified concept $\exists R.D$. In addition, the rule introduces $C(y)$ for each $(\forall R.C)(x)$ and the branch satisfies Definition 4.20 condition (ii) because by Proposition 4.12 condition (a), the concepts of x remain unchanged after the $\rightarrow_{\exists\forall}$ rule instance is applied.

We conclude b satisfies properties 1-7. □

In the next section we prove the correctness of operation of Algorithm 2 and show that given a

credible knowledge base $\mathcal{K}$, the minimal branches obtained from the Algorithm correspond to preferred interpretations of $\mathcal{K}$.

4.4 Soundness

We have already shown that open branches are returned for a knowledge base $\mathcal{K}$ iff $\mathcal{K}$ is credible. Our notion of the *soundness* of Algorithm 2 states that for a credible $\mathcal{K}$, we can construct an interpretation of $\mathcal{K}$, based on an m -distant open branch, that satisfies all the axioms recorded in the expanded ABox, and hence $\langle \mathcal{A}, \mathcal{T} \rangle$, with a distance no greater than m . No interpretation of a credible n -inconsistent $\mathcal{K}$ that satisfies the non-defeasible axioms has a distance smaller than n . Hence, as a corollary of soundness, the distance of a branch is no less than n . An outline of the proof of soundness is shown in Figure 4.13.

A Herbrand interpretation based on an open branch from Algorithm 2 is obtained using the construction given in Definition 4.24.

Definition 4.24 (Open branch interpretation). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible knowledge base with signature $\langle N_I, N_C, N_R \rangle$, $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ be an open completed branch for $\mathcal{K}$, $P(b)$ denote the set of unblocked parameters introduced in the branch and $\mathcal{A}'_b$ be the set of axioms in $\mathcal{A}_b$ not involving blocked individuals. Let Δ_b be the Herbrand domain based on the signature of $\mathcal{K}$ in which N_I is augmented by the unblocked parameters $P(b)$. The Herbrand interpretation $\mathcal{I}_b$ of $\mathcal{K}$ based on b is defined as follows. For every $x, y \in \Delta_b$, $A \in N_C$ and $R \in N_R$: $A(x)$ is true iff $A(x) \in \mathcal{A}'_b$; $R(x, y)$ is true iff either $R(x, y) \in \mathcal{A}'_b$, or $R(x, z) \in \mathcal{A}_b$ and z is blocked by y .*

The proof of soundness relies on Lemma 4.25 which captures that the axioms of an open branch are satisfied in an interpretation based on an open branch.

Lemma 4.25 (Axioms are satisfied in an open branch interpretation). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible knowledge base with signature $\langle N_I, N_C, N_R \rangle$, $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ be an open completed branch for $\mathcal{K}$ and $\mathcal{I}_b$ be a Herbrand interpretation based on b . Then $\mathcal{I}_b$ satisfies every axiom in $\mathcal{A}_b$.*

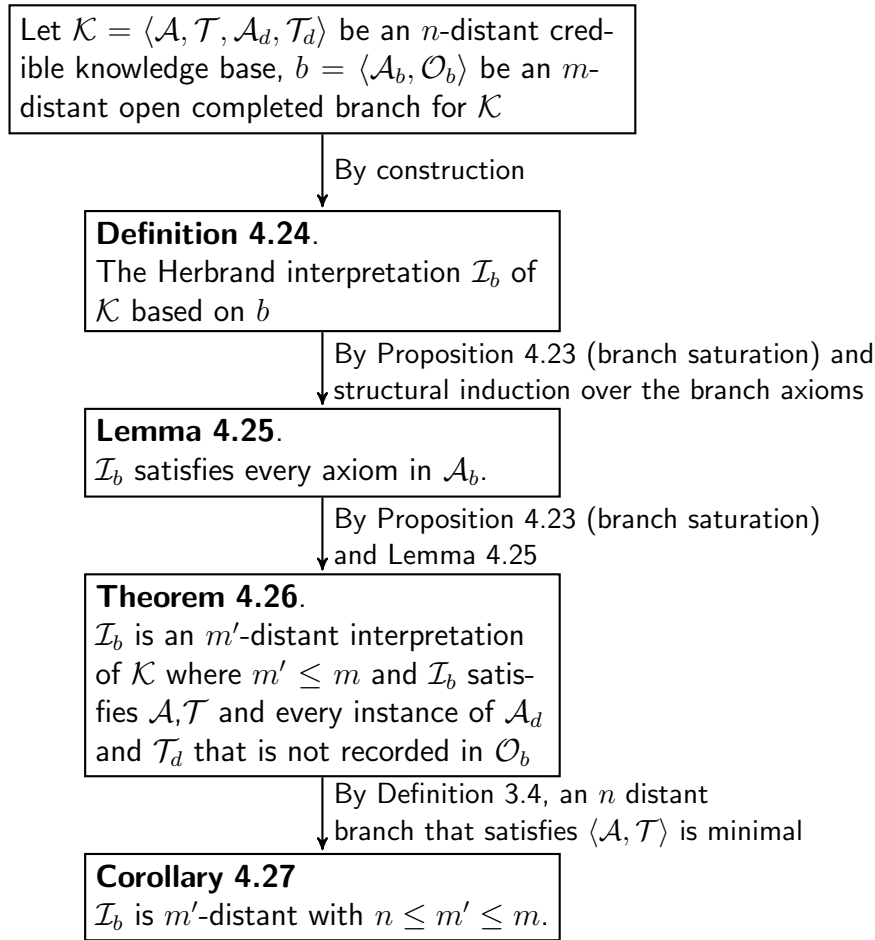


Figure 4.13: The proof outline for soundness of Algorithm 2. We begin by constructing an interpretation based on an open completed branch. By the saturation property of the branch (Proposition 4.23) this interpretation satisfies the axioms recorded in the ABox of the open branch (Lemma 4.25). Next, we use the branch properties to show soundness (Theorem 4.26) that such an interpretation: (i) is a model of $\langle \mathcal{A}, \mathcal{T} \rangle$, (ii) satisfies each of the defeasible axioms in $\mathcal{A}_d$ and the defeasible axiom instances of $\mathcal{T}_d$ not recorded in the omitted set of the branch. and (iii) records every omitted axiom instance in the omitted set. Since the construction satisfies the ABox axioms and records each of the omitted instances but does not require that the omitted instances are false in the interpretation, we conclude that the distance of the interpretation is less than or equal to the distance of the branch. Corollary 4.27 follows directly from soundness and Definition 3.4 because no interpretation that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ can be less than n distant.

Proof. $\mathcal{I}_b$ is a Herbrand interpretation with a domain Δ_b , the named individuals N_I augmented by the set of unblocked parameters in b . First, we deal with axioms that do not involve blocked parameters. Let $\mathcal{A}'_b$ denote the axioms in $\mathcal{A}_b$ that do not involve blocked parameters. There are two types of axiom in $\mathcal{A}'_b$, concept assertions of the form $C(x)$ where x is not blocked, and role assertions of the form $R(x, y)$ where x and y are not blocked. We show, by structural induction that every concept assertion $C(x)$ and role assertion $R(x, y)$ in $\mathcal{A}'_b$ is satisfied in $\mathcal{I}_b$:

- Base Case. Concept assertion $A(x)$ where $A \in N_C$. By Definition 4.24, $A(x)$ is made true in $\mathcal{I}_b$ because $A \in N_C$ and $A(x) \in \mathcal{A}'_b$. Concept assertion $\neg C(x)$ where $C \in N_C$. By Definition 4.24 $C(x)$ is made false in $\mathcal{I}_b$; thus $\mathcal{I}_b$ satisfies $\neg C(x)$. Role assertion of the form $R(x, y)$. From Definition 4.24 $R(x, y) \in \mathcal{A}'_b$ is made true in $\mathcal{I}_b$.
- Inductive step. Assume as inductive hypothesis that $C(x), D(x) \in \mathcal{A}'_b$. The proof then goes by cases:
 - $(C \sqcap D)(x)$. By Proposition 4.23(5), b is also boolean downward saturated. Hence, $\mathcal{A}'_b$ includes both $C(x)$ and $D(x)$. By the induction hypothesis, $\mathcal{I}_b$ satisfies $C(x)$ and $D(x)$. Thus $\mathcal{I}_b$ satisfies $(C \sqcap D)(x)$.
 - $(C \sqcup D)(x)$. Similar to the case of $(C \sqcap D)(x)$.
 - $(\exists R.C)(x)$. x is unblocked. By Proposition 4.23(7), b is also quantifier downward saturated. Therefore, there exists some y such that $R(x, y) \in \mathcal{A}_b$ and $C(y) \in \mathcal{A}_b$. There are two subcases:
 - (i) y is not blocked. Since x and y are unblocked, $R(x, y) \in \mathcal{A}'_b$ and $C(y) \in \mathcal{A}'_b$. By the base case and the induction hypothesis, $\mathcal{I}_b$ satisfies $R(x, y)$ and $C(y)$. Hence, $\mathcal{I}_b$ satisfies $(\exists R.C)(x)$.
 - (ii) y is blocked. By Definition 4.4 and Observation 4, there exists at least one unblocked individual w older than y s.t. $C(w) \in \mathcal{A}'_b$. By Definition 4.24 $R(x, w)$ is made true in $\mathcal{I}_b$ because $R(x, y) \in \mathcal{A}_b$ and $w \in \Delta_b$ blocks y . By the induction hypothesis, $\mathcal{I}_b$ satisfies $C(w)$. Thus, $\mathcal{I}_b$ satisfies $(\exists R.C)(x)$ as witnessed by w .
 - $(\forall R.C)(x)$. Let y be an arbitrary name such that $R(x, y) \in \mathcal{A}_b$. x is unblocked. By Proposition 4.23(7) b is also quantifier downward saturated. Hence $C(y) \in \mathcal{A}_b$. There are two subcases:
 - (i) y is not blocked. Since x and y are not blocked, $R(x, y) \in \mathcal{A}'_b$ and $C(y) \in \mathcal{A}'_b$. By the base case and the induction hypothesis $\mathcal{I}_b$ satisfies $R(x, y)$ and $C(x)$.
 - (ii) y is blocked. By Definition 4.4 and Observation 4, there exists at least one unblocked individual w older than y s.t. $C(w) \in \mathcal{A}'_b$. By Definition 4.24 $R(x, w)$

is made true in $\mathcal{I}_b$ because $R(x, y) \in \mathcal{A}_b$ and $w \in \Delta_b$ blocks y . By the induction hypothesis, $\mathcal{I}_b$ satisfies $C(w)$.

We conclude $\mathcal{I}_b$ satisfies $(\forall R.C)(x)$.

Finally, we deal with axioms involving blocked individuals. By Observation 1, a blocked individual is a parameter. As seen in sub-case (ii) of the $(\exists R.C)(x)$ case and sub-case (ii) of the $(\forall R.C)(x)$ case, the presence of these axioms is necessary to ensure that $\mathcal{I}_b$ satisfies the axioms in $\mathcal{A}'_b$. Definition 4.24 uses the blocked individuals to guide the construction of $\mathcal{I}_b$ using only unblocked individuals. Informally, a blocked parameter can be viewed as an “alias” for an unblocked individual that blocks it. The axioms involving blocked parameters are satisfied by $\mathcal{I}_b$ when each blocked parameter appearing in the axiom is replaced by an unblocked individual that blocks the parameter.

We conclude the interpretation based on the branch satisfies every axiom in b . $\square$

Theorem 4.26 (Soundness of Algorithm 2). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible knowledge base, $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ be an m -distant open completed branch for $\mathcal{K}$ and $\mathcal{I}_b$ be the Herbrand interpretation based on b . Then $\mathcal{I}_b$ is an m' -distant interpretation of $\mathcal{K}$ where $m' \leq m$ and $\mathcal{I}_b$ satisfies $\mathcal{A}, \mathcal{T}$ and every instance of $\mathcal{A}_d$ and $\mathcal{T}_d$ that is not recorded in $\mathcal{O}_b$.*

Proof. Let $\mathcal{A}'_b$ denote the axioms in $\mathcal{A}_b$ that do not involve blocked parameters. Proposition 4.23(1) ensures that $\mathcal{A} \subseteq \mathcal{A}'_b$. By Lemma 4.25, $\mathcal{I}_b$ satisfies the axioms in $\mathcal{A}_b$, hence $\mathcal{I}_b$ satisfies the axioms in $\mathcal{A}$. By Proposition 4.23(3), $\mathcal{A}'_b$ is $\mathcal{T}$ -saturated and again by Lemma 4.25, $\mathcal{I}_b$ satisfies every axiom of the form $(\neg C \sqcup D)(x)$ in $\mathcal{A}_b$. Hence, $\mathcal{I}_b$ satisfies the axioms in $\mathcal{T}$. By Proposition 4.23(2), $\mathcal{O}_b$ records the omissions of $\mathcal{A}_d$ in $\mathcal{A}'_b$ and those axioms not omitted are included in $\mathcal{A}'_b$. By Lemma 4.25, $\mathcal{I}_b$ satisfies the axioms in $\mathcal{A}_b$, hence $\mathcal{I}_b$ satisfies the axioms in $\mathcal{A}_d$ not recorded in $\mathcal{O}_b$. By Proposition 4.23(4), $\mathcal{O}_b$ records the omissions of $\mathcal{T}_d$ -saturation of $\mathcal{A}'_b$ and those axiom instances not omitted are included in $\mathcal{A}'_b$. By Lemma 4.25, $\mathcal{I}_b$ satisfies every axiom of the form $(\neg C \sqcup D)(x)$. Hence, $\mathcal{I}_b$ satisfies the axiom instance of $\mathcal{T}_d$ not recorded in $\mathcal{O}_b$. The omitted axioms recorded in $\mathcal{O}_b$ may or may not be falsified in $\mathcal{I}_b$, hence $\mathcal{I}_b$ is at most m -distant and $m' \leq m$. $\square$

Example 4.9 (continued). Recall the tableau generated for $\mathcal{K}_{inc}$ shown in Figure 4.10. Table 4.5 shows each of the five open branches and the characteristics of the interpretation based on those branches. Notice that the 3-distant branch results in a 2-distant interpretation. The interpretation based on the branch does not falsify $(\neg C \sqcup D)(b)$, and so it was unnecessary to include $\langle C \sqsubseteq D^{[1]}, b \rangle$ in the omitted set.

$\mathcal{A}_b$	$\mathcal{O}_b$	m	$C^{\mathcal{I}_b}$	$D^{\mathcal{I}_b}$	m'
$\{C(a), \neg D(a), C(b), (\neg C \sqcup D)(b), D(b)\}$	$\{\langle C \sqsubseteq D^{[1]}, a \rangle\}$	1	$\{a, b\}$	$\{b\}$	1
$\{C(a), \neg D(a), C(b), D(b)\}$	$\{\langle C \sqsubseteq D^{[1]}, a \rangle, \langle C \sqsubseteq D^{[1]}, b \rangle\}$	2	$\{a, b\}$	$\emptyset$	2
$\{C(a), \neg D(a), (\neg C \sqcup D)(b), \neg C(b)\}$	$\{\langle C(b)^{[1]}, b \rangle, \langle C \sqsubseteq D^{[1]}, a \rangle\}$	2	$\{a\}$	$\emptyset$	2
$\{C(a), \neg D(a), (\neg C \sqcup D)(b), D(b)\}$	$\{\langle C(b)^{[1]}, b \rangle, \langle C \sqsubseteq D^{[1]}, a \rangle\}$	2	$\{a\}$	$\{b\}$	2
$\{C(a), \neg D(a), C(b)\}$	$\{\langle C(b)^{[1]}, b \rangle, \langle C \sqsubseteq D^{[1]}, a \rangle, \langle C \sqsubseteq D^{[1]}, b \rangle\}$	3	$\{a\}$	$\emptyset$	2

Table 4.5: The five open branches developed for $\mathcal{K}_{inc}$, where $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$ denotes the branch, m is the branch distance and $\mathcal{I}_b$ is the m' -distant interpretation based on the branch, with $m' \leq m$.

Corollary 4.27 (Distances are greater than n). Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be an n -inconsistent credible knowledge base and b be an m -distant open completed branch for $\mathcal{K}$. Let $\mathcal{I}_b$ be the Herbrand interpretation based on b . Then $\mathcal{I}_b$ is m' -distant with $n \leq m' \leq m$.

Proof. By Theorem 4.26 $\mathcal{I}_b$ satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ and $\mathcal{I}_b$ is m' -distant with $m' \leq m$. By assumption, $\mathcal{K}$ is n -inconsistent and by Definition 3.4, an n -distant interpretation that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ is minimal. Hence $m' \geq n$. Now, $n \leq m'$ and $m' \leq m$. Hence, $n \leq m' \leq m$. $\square$

4.5 Completeness

The notion of the *completeness* of Algorithm 2 captures that for every m' -distant interpretation of an n -inconsistent knowledge base which satisfies the non-defeasible axioms, there is at least one branch generated by the algorithm that is at most m' -distant. Completeness of the algorithm is expressed by Theorem 4.30. The proof takes three steps and an outline of the proof is shown in Figure 4.14.

Lemma 4.28 (Branch based on an interpretation). Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible knowledge base and $\mathcal{I}$ be an interpretation of $\mathcal{K}$ satisfying $\langle \mathcal{A}, \mathcal{T} \rangle$. Then there exists a finite open

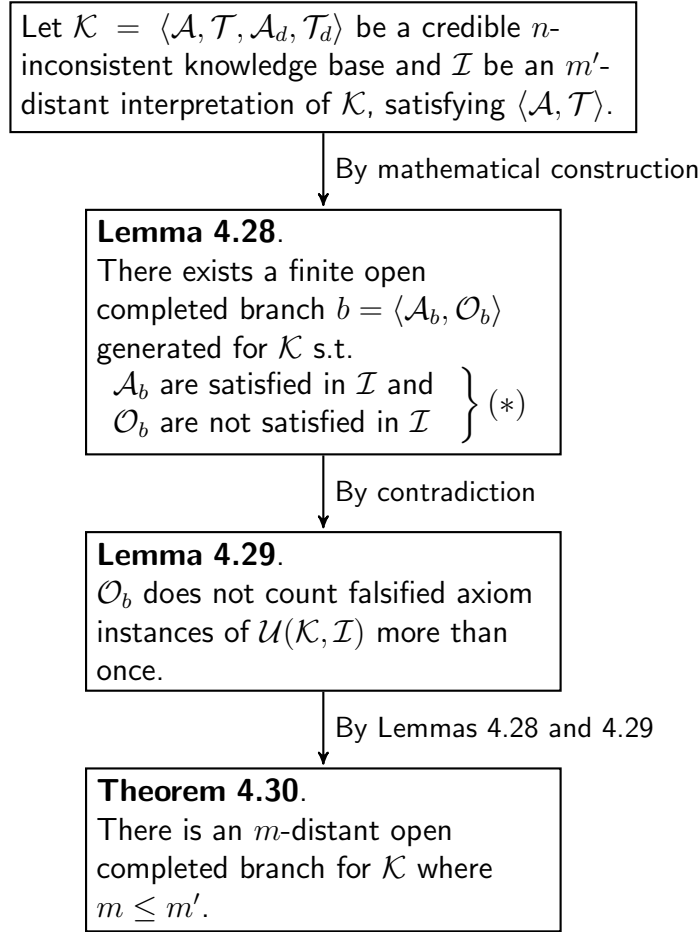


Figure 4.14: The proof outline for completeness of Algorithm 2, Theorem 4.30. First, we show that an arbitrary m' -distant interpretation that satisfies the non-defeasible axioms can be used to guide the algorithm such that it returns an open branch in which each axiom is satisfied by the guiding interpretation and each axiom instance recorded in the omitted set is made false in the guiding interpretation (Lemma 4.28). Now, since we only record axiom instances as omissions when they are false in the interpretation we would like to be able to conclude that the distance of the branch is at most m' -distant. However, this only follows if we have not counted the axiom instances falsified by the interpretation more than once. Our second step shows that no branch records axiom instances made false in the interpretation more than once (Lemma 4.29). The third step then concludes the proof of the Theorem.

completed branch $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ obtained from Algorithm 2 for $\mathcal{K}$, and an associated total function, $\pi : \text{Names}_b \rightarrow \Delta^{\mathcal{I}}$ where Names_b is the set of individual names in b , such that the following properties hold (See $(*)$ in Figure 4.14):

$$\text{if } x \in b \text{ is an unblocked individual then } \pi(x) \in \Delta^{\mathcal{I}} \quad (4.1)$$

$$\text{if } C(x) \in \mathcal{A}_b, \text{ and } x \text{ is not blocked then } \pi(x) \in C^{\mathcal{I}} \quad (4.2)$$

$$\text{if } R(x, y) \in \mathcal{A}_b, \text{ and } x \text{ and } y \text{ are not blocked then } \langle \pi(x), \pi(y) \rangle \in R^{\mathcal{I}} \quad (4.3)$$

$$\text{if } \langle C(x)^{[w]}, x \rangle \in \mathcal{O}_b, \text{ and } x \text{ is not blocked then } \pi(x) \notin C^{\mathcal{I}} \quad (4.4)$$

$$\text{if } \langle R(x, y)^{[w]}, x \rangle \in \mathcal{O}_b, \text{ and } x \text{ and } y \text{ are not blocked then } \langle \pi(x), \pi(y) \rangle \notin R^{\mathcal{I}} \quad (4.5)$$

$$\text{if } \langle C \sqsubseteq D^{[w]}, x \rangle \in \mathcal{O}_b, \text{ and } x \text{ is not blocked then } \pi(x) \notin (\neg C \sqcup D)^{\mathcal{I}} \quad (4.6)$$

$$b \text{ is open.} \quad (4.7)$$

Proof. We show by mathematical induction that Algorithm 2 can be used to construct an open branch $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ with a mapping π such that (b, π) satisfies properties 4.1-4.7 by using an m -distant interpretation $\mathcal{I}$.

In what follows (b_i, π_i) , $i > 0$ denotes the i th step in the construction of a branch b and associated mapping π where $b_i = \langle \mathcal{A}_i, \mathcal{O}_i \rangle$ denotes elements from the sequence ABoxes $\mathcal{A}_0 \subseteq \dots \subseteq \mathcal{A}_i$ and $\mathcal{O}_0 \subseteq \dots \subseteq \mathcal{O}_i$ denotes the corresponding sequence of omissions in the branch developed by Algorithm 2 and guided by $\mathcal{I}$ as follows

- Base case. (b_0, π_0) where $b_0 = \langle \mathcal{A}_0, \mathcal{O}_0 \rangle$. Initialise π_0 such that $\pi_0(a) = a^{\mathcal{I}}$ where a are the names in N_I . Choose the omitted set of defeasible ABox axioms $\mathcal{A}_o$ (in line 1) to include the axioms: $C(x)^{[w]} \in \mathcal{A}_d$ where $C(x)$ is falsified by $\mathcal{I}$; and $R(x, y)^{[w]} \in \mathcal{A}_d$ where $R(x, y)$ is falsified by $\mathcal{I}$. $\mathcal{O}_0$ is assigned the omitted axiom instances at line 2. For each omitted axiom instance $o \in \mathcal{O}_0$ there are two cases. If o is of the form:

- $\{\langle C(x)^{[w]}, x \rangle \mid C(x)^{[w]} \in \mathcal{A}_o\}$. $x \in N_I$ and by the initialisation of π_0 , $\pi_0(x) = x^{\mathcal{I}}$. By Observation 1, x is not blocked, and by the assignment of $\mathcal{A}_o$, $\pi_0(x) \notin C^{\mathcal{I}}$. Hence o satisfies 4.4.
- $\{\langle R(x, y)^{[w]}, x \rangle \mid R(x, y)^{[w]} \in \mathcal{A}_o\}$. $x, y \in N_I$ and by the initialisation of π_0 , $\pi_0(x) = x^{\mathcal{I}}$ and $\pi_0(y) = y^{\mathcal{I}}$. By Observation 1, x, y are not blocked, and by the assignment $\mathcal{A}_o$, $\langle \pi_0(x), \pi_0(y) \rangle \notin R^{\mathcal{I}}$. Hence o satisfies 4.5.

$\mathcal{A}_0$ are the axioms assigned to $\mathcal{A}_b$ at line 3. There are two sets of axioms:

- The non-defeasible axioms $\mathcal{A}$. For each axiom, $Z \in \mathcal{A}$, there are two sub-cases. If Z is of the form:

- * $C(x)$. $x \in N_I$ and by the initialisation of π_0 , $\pi_0(x) = x^{\mathcal{I}}$. By Observation 1, x is not blocked. By assumption $\mathcal{I}$ is a model of $\langle \mathcal{A}, \mathcal{T} \rangle$ and $\pi_0(x) \in C^{\mathcal{I}}$. Hence Z satisfies 4.2.
- * $R(x, y)$. $x, y \in N_I$ and by the initialisation of π_0 , $\pi_0(x) = x^{\mathcal{I}}$ and $\pi_0(y) = y^{\mathcal{I}}$. By Observation 1, x, y are not blocked. By assumption $\mathcal{I}$ is a model of $\langle \mathcal{A}, \mathcal{T} \rangle$ and $\langle \pi_0(x), \pi_0(y) \rangle \in R^{\mathcal{I}}$. Hence, Z satisfies 4.3.
- $(\mathcal{A}_d - \mathcal{A}_o)^{-W}$. For each axiom, $Z \in (\mathcal{A}_d - \mathcal{A}_o)^{-W}$, there are two sub-cases. If Z is of the form:

- * $C(x)$. $x \in N_I$ and by the initialisation of π_0 , $\pi_0(x) = x^{\mathcal{I}}$. By Observation 1, x is not blocked, and by the assignment of $\mathcal{A}_o$, $\pi_0(x) \in C^{\mathcal{I}}$. Hence Z satisfies 4.2.
- * $R(x, y)$. $x, y \in N_I$ and by the initialisation of π_0 , $\pi_0(x) = x^{\mathcal{I}}$ and $\pi_0(y) = y^{\mathcal{I}}$. By Observation 1, x, y are not blocked and by the assignment $\mathcal{A}_o$, $\langle \pi_0(x), \pi_0(y) \rangle \in R^{\mathcal{I}}$. Hence, Z satisfies 4.3.

By the construction, π_0 maps all named individuals to $\Delta^{\mathcal{I}}$ at initialisation. Only named individuals appear in b_0 . Hence (b_0, π_0) satisfies 4.1. The axioms recorded in $\mathcal{A}_0$ satisfy 4.2 and 4.3. The omissions recorded in $\mathcal{O}_0$ satisfies 4.4, 4.5 and (vacuously) 4.6. By 4.2 and 4.3, each of the axioms in $\mathcal{A}_0$ is true in $\mathcal{I}$, hence $\mathcal{A}_0$ is clash free and b_0 is open. Hence (b_0, π_0) satisfies Properties 4.1-4.7.

- The inductive case. Assume as inductive hypothesis (b_{k-1}, π_{k-1}) satisfies Properties 4.1-4.7, $\pi_k = \pi_{k-1}$ and the domain of π_k is unchanged. We show by cases a construction such that at each subsequent step k , (b_k, π_k) satisfies Properties 4.1-4.7. Observe that in all rule applications except $\exists R.C(x)$, the domain of π_k is unchanged and for $\exists R.C(x)$, the domain of π_k is changed.

- Case $(C \sqcap D)(x) \in \mathcal{A}_{k-1}$, x is not blocked and either $C(x) \notin \mathcal{A}_{k-1}$ or $D(x) \notin \mathcal{A}_{k-1}$. By the inductive hypothesis, $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.2. Hence $\pi_{k-1}(x) \in (C \sqcap D)^{\mathcal{I}}$. $\pi_k = \pi_{k-1}$. Hence, $\pi_k(x) \in (C \sqcap D)^{\mathcal{I}}$. Apply the $\rightarrow_{\sqcap}$ -rule such that $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{C(x), D(x)\}$. Hence, $\pi_k(x) \in C^{\mathcal{I}}$ and $\pi_k(x) \in D^{\mathcal{I}}$. By Proposition 4.12 property

- (b) x is not blocked in $\mathcal{A}_k$. We conclude (b_k, π_k) satisfies 4.2. By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.7 and b_{k-1} is open. Hence b_k is open.
- Case $(C \sqcup D)(x) \in \mathcal{A}_{k-1}$, x is not blocked, $C(x) \notin \mathcal{A}_{k-1}$ and $D(x) \notin \mathcal{A}_{k-1}$. By the inductive hypothesis, $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.2. Hence $\pi_{k-1}(x) \in (C \sqcup D)^{\mathcal{I}}$. $\pi_k = \pi_{k-1}$. Hence, $\pi_k(x) \in (C \sqcup D)^{\mathcal{I}}$ and either $\pi_k(x) \in C^{\mathcal{I}}$ or $\pi_k(x) \in D^{\mathcal{I}}$. Apply $\rightarrow_{\sqcup}$ rule such that: if $\pi_k(x) \in C^{\mathcal{I}}$ then $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{C(x)\}$ otherwise $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{D(x)\}$. By Proposition 4.12 property (b) x is not blocked in $\mathcal{A}_k$. We conclude (b_k, π_k) satisfies 4.2. By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.7 and b_{k-1} is open. Hence b_k is open.
 - Case $(\forall R.C)(x) \in \mathcal{A}_{k-1}$, $R(x, y) \in \mathcal{A}_{k-1}$, x is not blocked and $C(y) \notin \mathcal{A}_{k-1}$. By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.2. Hence, $\pi_{k-1}(x) \in (\forall R.C)^{\mathcal{I}}$ and $\langle \pi_{k-1}(x), \pi_{k-1}(y) \rangle \in R^{\mathcal{I}}$, therefore, $\pi_{k-1}(x) \in C^{\mathcal{I}}$. $\pi_k = \pi_{k-1}$. Hence $\pi_k(x) \in C^{\mathcal{I}}$. Apply the $\rightarrow_{\forall}$ -rule such that $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{C(x)\}$. By Proposition 4.12 property (b) x is not blocked in $\mathcal{A}_k$. We conclude (b_k, π_k) satisfies 4.2. By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.7 and b_{k-1} is open. Hence b_k is open.
 - Case $C \sqsubseteq D \in \mathcal{T}$, there exist an unblocked individual $x \in \mathcal{A}$ and $(\dot{\neg}C \sqcup D)(x) \notin \mathcal{A}_{k-1}$. By the inductive hypothesis, $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.1. Hence, $\pi_{k-1}(x) \in \Delta^{\mathcal{I}}$. By assumption, $\mathcal{I}$ is a model of $\langle \mathcal{A}, \mathcal{T} \rangle$, therefore $\pi_{k-1}(x) \in (\dot{\neg}C \sqcup D)^{\mathcal{I}}$. Let $\pi_k = \pi_{k-1}$. Hence, $\pi_k(x) \in (\dot{\neg}C \sqcup D)^{\mathcal{I}}$. Apply the $\rightarrow_{\mathcal{T}}$ rule such that $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{(\dot{\neg}C \sqcup D)(x)\}$. By Proposition 4.12 property (b) x is not blocked in $\mathcal{A}_k$. We conclude (b_k, π_k) satisfies 4.2. By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.7 and b_{k-1} is open. Hence b_k is open.
 - Case $C \sqsubseteq D^{[w]} \in \mathcal{T}_d$, there exist an unblocked individual $x \in \mathcal{A}$, $(\dot{\neg}C \sqcup D)(x) \notin \mathcal{A}_{k-1}$ and $\langle C \sqsubseteq D^{[w]}, x \rangle \notin \mathcal{O}_{k-1}$. By the inductive hypothesis, $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.1. Hence, $\pi_{k-1}(x) \in \Delta^{\mathcal{I}}$. $\pi_k = \pi_{k-1}$. Hence, $\pi_k(x) \in \Delta^{\mathcal{I}}$. There are two sub cases:
 - * $\pi_k(x) \in (\dot{\neg}C \sqcup D)^{\mathcal{I}}$. Apply the $\rightarrow_{\mathcal{T}_d}$ rule such that $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{(\dot{\neg}C \sqcup D)(x)\}$. By Proposition 4.12 property (b) x is not blocked in $\mathcal{A}_k$. We conclude (b_k, π_k) satisfies 4.2.
 - * $\pi_k(x) \notin (\dot{\neg}C \sqcup D)^{\mathcal{I}}$. Apply the $\rightarrow_{\mathcal{T}_d}$ rule such that $\mathcal{O}_k = \mathcal{O}_{k-1} \cup \{\langle C \sqsubseteq D^{[w]}, x \rangle\}$.

By assumption x is not blocked. Hence b_k satisfies 4.6.

By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.7 and b_{k-1} is open. Hence b_k is open.

- Case $C \in \mathcal{Q}(\mathcal{K})$, there exists an unblocked individual $x \in \mathcal{A}$, $C(x) \notin \mathcal{A}_{k-1}$ and $\neg C(x) \notin \mathcal{A}_{k-1}$. By the inductive hypothesis, $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.1. Hence, $\pi_{k-1}(x) \in \Delta^{\mathcal{I}}$. $\pi_k = \pi_{k-1}$. Hence, $\pi_k(x) \in \Delta^{\mathcal{I}}$. Apply the $\rightarrow_Q$ rule such that if $\pi_k(x) \in C^{\mathcal{I}}$ then $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{C(x)\}$ otherwise $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{\neg C(x)\}$. By Proposition 4.12 property (b) x is not blocked in $\mathcal{A}_k$. We conclude (b_k, π_k) satisfies 4.2. By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.7 and b_{k-1} is open. Hence b_k is open.
- Case $(\exists R.C)(x) \in \mathcal{A}_{k-1}$, x is not blocked, there does not exist y such that $R(x, y) \in \mathcal{A}_{k-1}$, $C(y) \in \mathcal{A}_{k-1}$ and no other non- $\rightarrow_{\exists\forall}$ rules apply. By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.2. Hence, $\pi_{k-1}(x) \in (\exists R.C)^{\mathcal{I}}$ and there exists some domain element $d \in \Delta^{\mathcal{I}}$ such that: (i) $\langle \pi_{k-1}(x), d \rangle \in R^{\mathcal{I}}$, $d \in C^{\mathcal{I}}$; and (ii) for each axiom $(\forall R.D)(x) \in \mathcal{A}_{k-1}$, $\pi_{k-1}(x) \in (\forall R.D)^{\mathcal{I}}$ and $d \in D^{\mathcal{I}}$. Apply the $\rightarrow_{\exists\forall}$ rule such that $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{R(x, p)\} \cup \{C(p) | (\exists R.C)(x) \in \mathcal{A}_{k-1}\} \cup \{D(p) | (\forall R.D)(x) \in \mathcal{A}_{k-1}\}$ where p is the name of the fresh parameter introduced. For each name ν in the domain of π_{k-1} , $\pi_k(\nu) = \pi_{k-1}(\nu)$ and for p , $\pi_k(p) = d$. By Proposition 4.12 property (b) x is not blocked in $\mathcal{A}_k$. There are two sub-cases:

- * p is not blocked. Hence, from (i) and (ii) above, (b_k, π_k) satisfies 4.2 and 4.3.
- * p is blocked. The preconditions of 4.2 and 4.3 exclude axioms with blocked individuals. Hence, (b_k, π_k) satisfies 4.2 and 4.3.

By the inductive hypothesis $(\mathcal{A}_{k-1}, \pi_{k-1})$ satisfies 4.7 and b_{k-1} is open. Hence b_k is open.

From Proposition 4.15, Algorithm 2 terminates, so let k be the final step and let (b, π) be the branch b_k and the map π_k at the final step. The induction shows that $\forall i : 0 \leq i \leq k$ Properties 4.1-4.7 hold for (b_i, π_i) . Hence, the branch b and map π satisfy Properties 4.1-4.7. $\square$

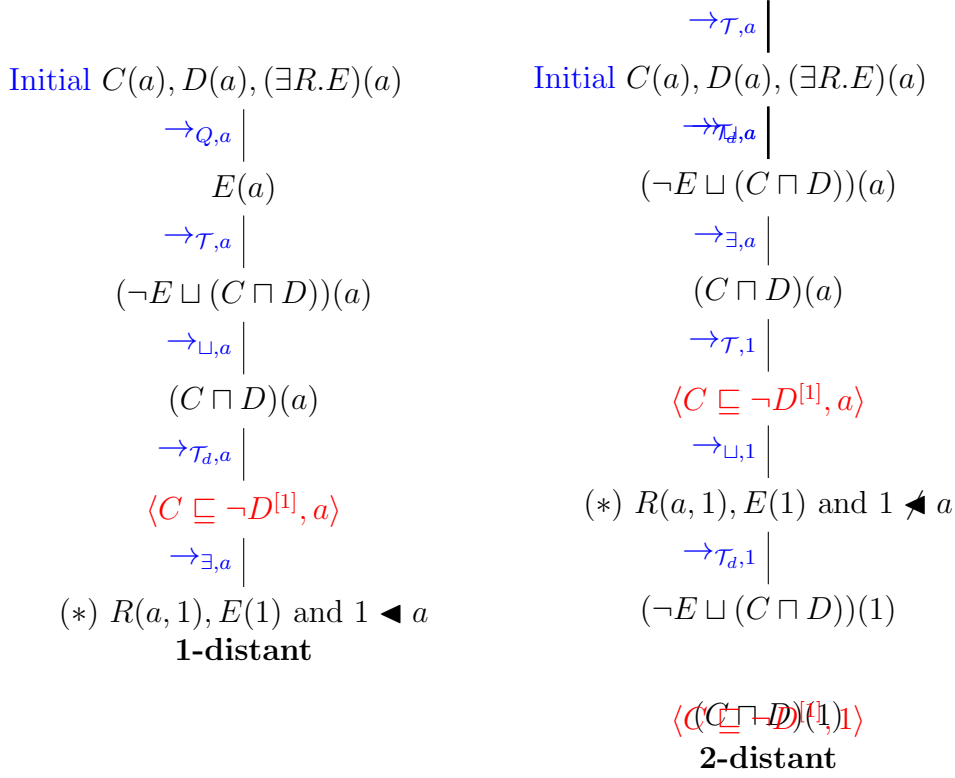
(b) Algorithm 2 without $\rightarrow_Q$

Figure 4.15: Branch (a) is developed for $\mathcal{K}_{1inc} = \langle \{C(a), D(a), (\exists R.E)(a)\}, \{E \sqsubseteq C \sqcap D\}, \emptyset, \{C \sqsubseteq \neg D^{[1]}\} \rangle$ by Algorithm 2, as guided by $\mathcal{I}_9$ using the construction of Lemma 4.28, where $C^{\mathcal{I}_9} = D^{\mathcal{I}_9} = E^{\mathcal{I}_9} = \{a\}$ and $R^{\mathcal{I}_9} = \{(a, a)\}$. Branch (b) is developed if the $\rightarrow_Q$ -rule is not applied.

Lemma 4.28 demonstrates a construction for an open completed branch b based on an interpretation $\mathcal{I}$ in which all the axioms recorded are satisfied by $\mathcal{I}$ and all the omissions are falsified in $\mathcal{I}$. Before proving that unsatisfied axiom instances in $\mathcal{I}$ are not counted more than once, we illustrate the construction of a branch from an interpretation of the knowledge base in Example 4.6. The example highlights the importance of the $\rightarrow_Q$ -rule in preventing double-counting by forcing blocking conditions.

Example 4.6 (continued). *Branch (a) in Figure 4.15 shows how the 1-distant Herbrand interpretation $\mathcal{I}_9$ from Figure 4.12 can be used to guide Algorithm 2 for $\mathcal{K}_{1inc}$.*

There are no defeasible ABox axioms, hence the branch is initialised with the non-defeasible ABox $\{C(a), D(a), (\exists R.E)(a)\}$ and π is initialised such that the domain of $\pi = \{a\}$ and $\pi(a) = a$. The $\rightarrow_Q$, $\rightarrow_{\mathcal{T}_d}$ and $\rightarrow_{\mathcal{T}}$ rules apply to a . These may be applied in any order. We will choose to apply $\rightarrow_Q$ first. $E(a)$ is true in $\mathcal{I}_9$, so we must choose to add $E(a)$ to the branch. Suppose

next, we choose to apply the $\rightarrow_{\mathcal{T}}$ -rule. $(\neg E \sqcup (C \sqcap D))(a)$ is added to the branch. Next, suppose we choose the $\rightarrow_{\sqcup}$ rule. $\neg E(a)$ is false in $\mathcal{I}_9$ and we must choose to add $(C \sqcap D)(a)$. Now, only the $\rightarrow_{\mathcal{T}_d}$ rule applies. $(\neg C \sqcup \neg D)(a)$ is false in $\mathcal{I}_9$ and we omit $\langle C \sqsubseteq \neg D^{[1]}, a \rangle$. Finally, the $\rightarrow_{\exists}$ -rule adds $R(a, 1)$ and $E(1)$ to the branch and maps $\pi(1) = a$. 1 is blocked by a at $(*)$ and the Algorithm terminates with a 1-distant open completed branch.

However, notice that without the application of the $\rightarrow_Q$ -rule, branch (b) in Figure 4.15 is developed. $E(a) \notin \mathcal{C}_a$ and 1 is not blocked at $(*)$, the $\rightarrow_{\mathcal{T}_d}$ is applied to 1 leading to the omission of $\langle C \sqsubseteq \neg D^{[1]}, 1 \rangle$ and increasing the distance to 2. In this case, since $\pi(1) = a$, the branch would record the unsatisfied instance of $C \sqsubseteq \neg D^{[1]}$ for a in $\mathcal{I}_9$ twice.

Lemma 4.29 (Inconsistencies are counted only once). *Let $\mathcal{I}$ be an m' -distant interpretation of a credible knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ with signature $\langle N_I, N_C, N_R \rangle$, where $\mathcal{I}$ satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$. Let $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ be an open branch and π a total mapping obtained from Algorithm 2 for $\mathcal{K}$ using the construction in Lemma 4.28. Then for each inconsistent axiom instance $\langle Z, u \rangle \in \mathcal{U}(\mathcal{K}, \mathcal{I})$ there are no two individual names $p, q \in b$ where $p \neq q$ such that $\langle Z, p \rangle, \langle Z, q \rangle \in \mathcal{O}_b$ and $\pi(p) = \pi(q) = u$.*

Proof. The proof is by contradiction. Suppose for contradiction, that for some inconsistent axiom instance $\langle Z, u \rangle \in \mathcal{U}(\mathcal{K}, \mathcal{I})$ there exist individual names $p, q \in b$ where $p \neq q$ such that $\langle Z, p \rangle, \langle Z, q \rangle \in \mathcal{O}_b$ and $\pi(p) = \pi(q) = u$. There are two subcases:

1. $p, q \in N_I$. By assumption b was obtained using the construction of Lemma 4.28. Hence, $\pi(p) = p^{\mathcal{I}}$ and $\pi(q) = q^{\mathcal{I}}$. By assumption $\mathcal{I}$ is an interpretation of $\mathcal{K}$ and by the UNA, $p^{\mathcal{I}} \neq q^{\mathcal{I}}$. Hence, $\pi(p) \neq \pi(q)$, a contradiction.
2. w.l.o.g. q is a parameter (as opposed to p). There are two sub-cases:
 - (i) p is not blocked and p is either a named individual or a parameter. Let $\mathcal{C}_q \subseteq \mathcal{Q}(\mathcal{K})$ denote the concepts of q , asserted by the $\rightarrow_{\exists\forall}$ -rule on the introduction of q . By Proposition 4.23 b satisfies property 6. Hence p is QC-split and let $\mathcal{C}_p \subseteq \mathcal{Q}(\mathcal{K})$ denote the quantified concepts of p in $\mathcal{A}_b$. By assumption, $\pi(p) = \pi(q) = u$ therefore

$\mathcal{C}_q \subseteq \mathcal{C}_p$. Hence, by Definition 4.4, p blocks q . The $\rightarrow_{\mathcal{T}_d}$ -rule is the only rule that introduces omitted axiom instances for a parameter and the precondition requires the parameter to be unblocked. A contradiction with the assumption that there exists some $\langle Z, q \rangle \in \mathcal{O}_b$.

- (ii) p is blocked. By Observation 1, p is a parameter. By Observation 4, there exists some older unblocked individual p' that blocks p and case (i) applies for p' an unblocked individual.

□

Theorem 4.30 (Completeness of Algorithm 2). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible n -inconsistent knowledge base and $\mathcal{I}$ be an m' -distant interpretation of $\mathcal{K}$, where $\mathcal{I}$ satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$. Then there is an m -distant open completed branch for $\mathcal{K}$ where $m \leq m'$.*

Proof. By Lemma 4.28 there is an open completed branch $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ based on $\mathcal{I}$ that satisfies Properties 4.2-4.6. $\mathcal{O}_b$ includes the rule instances of $\mathcal{A}_d$ falsified by $\mathcal{I}$ and the subset of the $\mathcal{T}_d$ rule instances falsified by $\mathcal{I}$ that are used in b . Let m be the distance of b . By Lemma 4.29, b does not record inconsistencies of $\mathcal{I}$ in $\mathcal{O}_b$ more than once. Hence, we conclude $m \leq m'$. □

4.6 Correctness

Finally, the notion of correctness captures the property that the distance of the minimal branches, obtained by applying Algorithm 2 to a credible n -inconsistent $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$. An outline of the proof of correctness is shown in Figure 4.16.

Theorem 4.31 (Correctness of Algorithm 2). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a credible n -inconsistent knowledge base. Let $\mathcal{B}$ be the set of m -minimal branches generated by Algorithm 2 applied to $\mathcal{K}$. Then $\mathcal{B}$ is non empty, $m = n$ and for each branch $b \in \mathcal{B}$ the interpretation $\mathcal{I}_b$ constructed from b is a preferred interpretation of $\mathcal{K}$.*

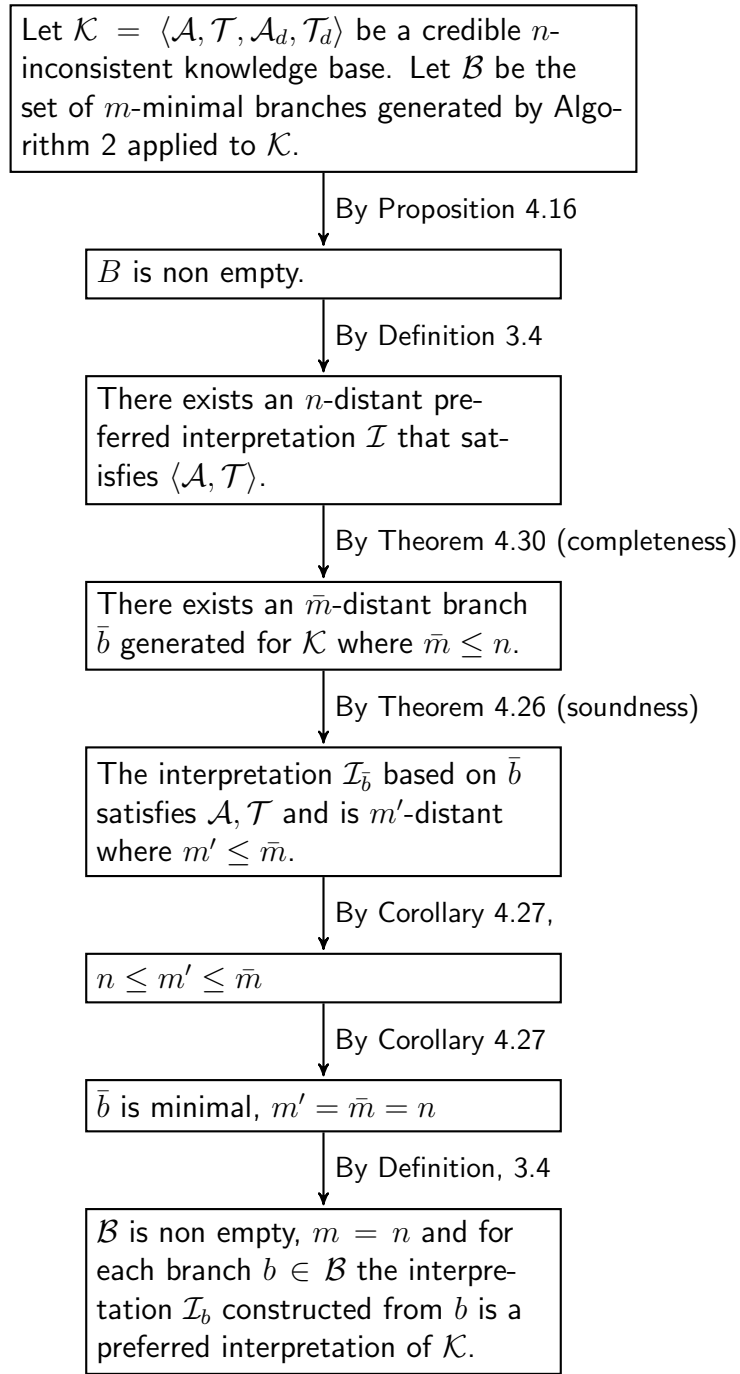


Figure 4.16: The proof outline for correctness of Algorithm 2, Theorem 4.31.

Proof. By assumption, $\mathcal{K}$ is credible and by Definition 3.4 condition (i), $\langle \mathcal{A}, \mathcal{T} \rangle$ is consistent. $\mathcal{K}$ is credible and therefore finite. Hence, by Proposition 4.16, the set of m -minimal branches $\mathcal{B}$ returned by applying Algorithm 2 to $\mathcal{K}$ is non empty. By assumption, $\mathcal{K}$ is n -inconsistent and by Definition 3.4, $\mathcal{K}$ has at least one n -distant preferred interpretation $\mathcal{I}$ that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$. Hence, by Theorem 4.30 (completeness) at least one $\bar{m}$ -distant branch $\bar{b}$ where $\bar{m} \leq n$ is

generated by Algorithm 2 when applied to $\mathcal{K}$. By Theorem 4.26 (soundness) the interpretation $\mathcal{I}_{\bar{b}}$ based on $\bar{b}$ satisfies $\mathcal{A}, \mathcal{T}$ and is m' -distant where $m' \leq \bar{m}$. By Corollary 4.27 $n \leq m' \leq \bar{m}$. Now, $n \leq m' \leq \bar{m}$ and $\bar{m} \leq n$ from which we obtain $m' = \bar{m} = n$. Finally, by Corollary 4.27 any open branch that satisfies $\langle \mathcal{A}, \mathcal{T} \rangle$ is at least n -distant. We conclude that $\bar{b}$ is in fact minimal, that $\bar{b} \in \mathcal{B}$ and that $m = n$. Hence, each $b \in \mathcal{B}$ is n -distant, the interpretation $\mathcal{I}_b$ based on b is n -distant and by Definition, 3.4 $\mathcal{I}_b$ is a preferred interpretation of $\mathcal{K}$. $\square$

4.7 Summary

We have presented a terminating, sound and complete algorithm for $p\text{-}\mathcal{ALC}$. Given the above results, to check for a credible knowledge base $\mathcal{K}$ whether $\mathcal{K} \models C(x)$, the idea is to exploit Theorem 3.13 and show the proof by refutation. First, we enumerate all the branches generated by applying Algorithm 2 to $\mathcal{K}$ to identify the distance n of the minimal branches. This can be done by using a branch-and-bound search (or similar). By Theorem 4.31 (Correctness) this obtains the n -inconsistency of $\mathcal{K}$. Then Algorithm 2 can be applied to $\mathcal{K}' = \langle \mathcal{A} \cup \neg C(x), \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ to search for an exactly n -distant branch. If such a branch is found, by Theorem 4.31 $\mathcal{K}'$ is n -inconsistent and we conclude that $\mathcal{K} \not\models C(x)$. Otherwise, either (i) all branches are closed and by Proposition 4.16 $\mathcal{K}'$ was inconsistent or (ii) the branches were m -distant for $m > n$ and, by Theorem 4.31, $\mathcal{K}'$ is m -inconsistent for some $m > n$. In either case, we can conclude that $\mathcal{K} \models C(x)$. In the next chapter we show how the computation of such branches can be implemented using Answer Set Programming (ASP), exploiting ASP optimisation features to identify preferred interpretations.

Chapter 5

Implementation

In this chapter we show how Algorithm 2 is implemented using Answer Set Programming. The choice of Answer Set Programming as a method for implementing the algorithm has been driven by the following considerations. Solvers are capable of tackling hard problems, even those belonging to **NP** and harder. ASP includes a notion of preference through the inclusion of weak constraints which allows the identification of optimal solutions. Furthermore, basing the implementation on a programming language with well defined semantics facilitates analysis and moreover, verification of correctness of the implementation. The recent development of incremental grounding and solving in ASP allows rules to be added to a program between iterations and this feature can be exploited to support the introduction of new constants that represent parameters during the execution of the $p\text{-}\mathcal{ALC}$ algorithm.

Before diving into the details of the implementation, we give an introduction to the answer set solver we have chosen for the implementation and include key properties of answer set programs that are used later to demonstrate the correctness of our implementation. Next, we present our encoding of a $p\text{-}\mathcal{ALC}$ knowledge base into an answer set program and show how this program can be combined with a fixed program such that when solved incrementally, the combined program has answer sets that represent tableau branches. Exploiting the optimisation features of ASP, we show that the optimal answer sets represent minimal branches obtained by the tableau and hence the implementation can be used to find the n -inconsistency of a knowledge

base and perform proofs by refutation. We conclude the chapter with a proof of correctness of the implementation by showing that our program terminates, and that it is sound and complete.

5.1 Background

Answer Set Programming is a form of logic programming. Our implementation uses *clingo* (version 4.5.3), from the Potassco Answer set solving collection [oP12]. The input language for *clingo* conforms to the *ASP-Core-2* [CFG⁺12] syntax and semantics. The *clingo* solver was selected for the implementation as no other ASP system¹ supported incremental grounding. For brevity, we present only the subset of the language used in the implementation.

5.1.1 Syntax and Semantics

We begin by introducing the syntax and semantics of disjunctive logic programs. The features that are specific to the *ASP-Core-2* specification and to the *clingo* solver are then described separately.

Syntax

Terms are *constants* (integers or strings starting with a lower case letter), *variables*, represented as strings starting with an upper case letter, and *functional terms* of the form $f(t_1, \dots, t_n)$ where f is a *functor* (a string starting with a lower case letter), and t_i , for $0 \leq i \leq n$, are terms. *Atoms* are of the form $p(t_1, \dots, t_n)$ where p is a *predicate* name (a string starting with a lower case letter), $n > 0$ is the *arity* of the predicate, and $t_1, \dots, t_n$ are terms. A *positive literal* is an atom b . A *negative literal* is a negated atom $\text{not } b$ where *not* denotes negation by failure. A *rule* r takes the form: $h_1 \mid \dots \mid h_m \leftarrow b_1, \dots, b_n$. where $m \geq 0$, $n \geq 0$, h_i are atoms, the symbol “ $\mid$ ” denotes a disjunction, $b_1, \dots, b_n$ are literals and the symbol “ $\leftarrow$ ” denotes a conjunction. $\text{head}(r) = \{h_1, \dots, h_m\}$ (and $\text{body}(r) = \{b_1, \dots, b_n\}$) is called the *head* (resp. *body*)

¹For example, *dlv* from dlvsystem.com.

of r . $body^+(r)$ ($body^-(r)$) are the set of atoms occurring in positive (resp. negative) literals in $body(r)$. A *fact* is a rule with an empty body (read as true) and a single variable free head atom. An *integrity constraint* is a rule with an empty head (read as false). A rule r is said to be *safe* if every variable that appears in r appears in at least one positive body literal in the body of r . A *program* P is a (possibly infinite) set of safe rules. A *ground* program (rule or atom) is a program (resp. rule or atom) without variables. A program is said to be *positive* if no rule in the program includes a negative literal. A program is said to be *disjunctive* if at least one rule in the program includes a disjunction and *non-disjunctive* otherwise.

Semantics

The semantics is based on the *Herbrand interpretations* of *ground* programs and is formalised next.

Definition 5.1 (Grounding). *Let P be a program. The Herbrand universe of P , denoted $\mathbb{U}_P$, consists of (ground) terms that can be constructed from the constants and functors appearing in P . The Herbrand base of P , denoted $\mathbb{H}_P$, is the set of all (ground) atoms that can be built by combining predicate names appearing in P with terms from $\mathbb{U}_P$ as arguments. A ground instance of a rule $r \in P$, is obtained by substituting each variable appearing in r with an element from $\mathbb{U}_P$. The grounding of P , denoted $gnd(P)$, is the set of ground instances of all the rules in P .*

Definition 5.2 (Model). *Let P be a program, $gnd(P)$ be the grounding of P . $I \subseteq \mathbb{H}_P$ is a Herbrand interpretation of $gnd(P)$ where an atom $a \in \mathbb{H}_P$ is true in I iff $a \in I$ and a negative literal $not\ a$ is true in I iff $a \notin I$. A rule $r \in gnd(P)$ is said to be satisfied by I iff some $h \in \{h_1, \dots, h_m\}$ is true w.r.t. I when $b_1, \dots, b_n$ are true w.r.t. I . I is a model of P iff every rule in $gnd(P)$ is satisfied by I . For integrity constraints, the head of the rule is empty. An integrity constraint is satisfied iff some $b_i \in b_1, \dots, b_n$ is false w.r.t. I .*

Definition 5.3 is based on the the GL-Reduct [GL91] which differs from [CFG⁺12] which uses the *FLP* semantics [FPL10]. The answer sets obtained under the two semantics coincide for the language fragment used in this implementation [FPL10].

Definition 5.3 (Answer Set). *Let P be a program. The reduct of P w.r.t. I , denoted P^I , is the set of rules obtained from $\text{gnd}(P)$ by deleting*

- (i) *each rule instance that has a literal not b in its body where $b \in I$, and*
- (ii) *the negative literals in the bodies of the remaining rules.*

I is an answer set of P if I is a model of the reduct P^I and no interpretation $I' \subset I$ is a model of P^I . I is said to be a $\subseteq$ -minimal model of the reduct P^I . The set of answer sets of a program P is denoted $\text{AS}(P)$. A program having no answer sets is said to be unsatisfiable.

Example 5.1. *Consider the ASP program:*

$$\Pi_{5.1} = \left\{ d. (1), \quad c \mid b \leftarrow d. (2), \quad b \leftarrow \text{not } f. (3), \quad e \leftarrow \text{not } b. (4) \right\}.$$

$\Pi_{5.1}$ has an answer set $I_1 = \{b, d\}$. We can check that I_1 is an answer set of $\Pi_{5.1}$ by considering the reduct of $\Pi_{5.1}$ w.r.t. I_1 :

$$\Pi_{5.1}^{I_1} = \left\{ d. (1), \quad c \mid b \leftarrow d. (2), \quad b. (3) \right\}.$$

Each rule in $\Pi_{5.1}^{I_1}$ is satisfied by I_1 and no proper subset of I_1 satisfies the rules of the reduct $\Pi_{5.1}^{I_1}$. For instance, removing d makes (1) unsatisfied and removing b makes both (2) and (3) unsatisfied. The requirement for answer sets to be subset minimal precludes other models of $\Pi_{5.1}^{I_1}$ from being answer sets. For instance, $\{b, c, d\}$ satisfies all rule instances, and is therefore a model of $\Pi_{5.1}^{I_1}$, but it is not a $\subseteq$ -minimal model of $\Pi_{5.1}^{I_1}$. Hence, it is not an answer set.

The reduct provides us with a method of checking that an interpretation is an answer set. However, finding and enumerating all answer sets is a much harder task. The search strategies implemented by contemporary solvers such clingo include adopting specific strategies for programs (or even parts of programs) that exhibit particular properties. In Section 5.1.4 we review properties that are relevant to our implementation and associated computational complexity results that are published in the literature.

The grounding of a program with variables can be very large and is infinite when the program includes functors. Informally, for many programs not every rule in the grounding of the program is *relevant* to the computation of answer sets. We say two programs are *equivalent* iff they have exactly the same answer sets. We restrict our attention to $\mathcal{FD}$ (denoting finite domain) programs given in [CCIL08]. For every $\mathcal{FD}$ program P , there exists an *intelligent grounding* of P , a finite ground program $fd(P) \subseteq gnd(P)$ where $fd(P)$ is equivalent to P , that is $AS(fd(P)) = AS(gnd(P)) = AS(P)$. The formalisation of the necessary syntactic conditions that define this class of programs is given in Section 5.1.4.

Modern answer set solvers use sophisticated algorithms to generate a finite ground program that is equivalent to the input program. Informally, the grounding algorithms establish truth values for a subset of the atoms in the Herbrand Base. For instance, a ground atom that appears as a fact in a program P is true in every answer set of P . Similarly, a ground atom that does not appear in the head of any rule in $gnd(P)$ is false in every answer set. The truth values are used to identify the relevant ground rule instances in $gnd(P)$. Given a ground rule instance $r_g \in gnd(P)$ and a ground atom a with an established truth value, r_g is not relevant if (i) $a \in body^+(r_g)$ and a is false or (ii) $a \in body^-(r_g)$ and a is true. In the remaining (relevant) rule instances of $gnd(P)$ the rule instances can be *simplified* by omitting literals with established truth values.

We use the notation $(rule : V_1/t_1, \dots, V_n/t_n)$ where $n > 1$ to denote the substitution of each variable V_x appearing in $(rule)$ by the term t_x where $1 \leq x \leq n$.

Example 5.2. Consider the ASP program:

$$\Pi_{5.2} = \left\{ \begin{array}{ll} drives(p, c1). & (1) \\ drives(q, c2) \mid drives(q, c3). & (2) \\ car(c2). & (3) \\ fast(c3). & (4) \\ car(Y) \mid bus(Y) \leftarrow drives(X, Y). & (5) \\ bus(Y) \leftarrow drives(X, Y), not\ fast(Y). & (6) \end{array} \right\}.$$

The rules of $\Pi_{5.2}$ are safe. In particular, the variable Y that is present in the heads of rules (5) and (6) also appears in $drives(X, Y)$, a positive body literal in these rules. The clingo solver generates the following (equivalent) ground program for $\Pi_{5.2}$:

$$ground(\Pi_{5.2}) = \left\{ \begin{array}{ll} drives(p, c1). & (1) \\ drives(q, c2) | drives(q, c3). & (2) \\ car(c2). & (3) \\ fast(c3). & (4) \\ bus(c1) | car(c1). [\leftarrow drives(p, c1).] & (5 : X/p, Y/c1) \\ bus(c2) | car(c2) \leftarrow drives(q, c2). & (5 : X/q, Y/c2) \\ bus(c3) | car(c3) \leftarrow drives(q, c3). & (5 : X/q, Y/c3) \\ bus(c1). [\leftarrow drives(p, c1), not fast(c1).] & (6 : X/p, Y/c1) \\ bus(c2) \leftarrow drives(q, c2). [not fast(c2).] & (6 : X/q, Y/c2) \end{array} \right\}.$$

The clingo grounder detects that rule instances of (5) and (6) where Y/p (or Y/q) are not relevant. For instance, consider the rule instance $r_g = (5 : X/p, Y/p)$. $drives(p, p) \in body^+(r_g)$ and there is no rule instance $r_i \in gnd(P)$ where $drives(p, p) \in head(r_i)$. Hence, $drives(p, p)$ is false in every answer set and r_g is not relevant. The clingo grounder simplifies the relevant rules of $ground(\Pi_{5.2})$ by omitting the body literals shown in square brackets, $[\]$. $drives(p, c1)$ is asserted as a fact in (1) and must be true in every answer set. Atoms $fast(c1)$ and $fast(c2)$ do not appear in the head of any relevant rule instance and are false in every answer set. $fast(c3)$ is asserted in (4) and must be true in every answer set. Hence, the instance $(6 : X/q, Y/c3)$ can never be satisfied, and the instance is not relevant. $\Pi_{5.2}$ has three answer sets:

$$\begin{aligned} I_1 &= \{drives(p, c1), bus(c1), drives(q, c2), car(c2), bus(c2), fast(c3)\} \\ I_2 &= \{drives(p, c1), bus(c1), car(c2), drives(q, c3), fast(c3), car(c3)\} \\ I_3 &= \{drives(p, c1), bus(c1), car(c2), drives(q, c3), fast(c3), bus(c3)\} \end{aligned}$$

Integrity constraints can be used to *prune out* answer sets. From Definition 5.2, an integrity constraint is satisfied iff at least one body literal is false w.r.t. to the interpretation.

Example 5.2 (continued). Adding the integrity constraint $\leftarrow \text{car}(X), \text{bus}(X). (7)$ to $\Pi_{5.2}$ has the effect of eliminating I_1 because the ground instance $\leftarrow \text{car}(c2), \text{bus}(c2). (7 : X/c2)$ is not satisfied in I_1 .

Our implementation exploits a number of language features beyond the basic syntax and semantics described thus far.

Weak constraints

The *ASP-Core-2* specification defines weak constraints that are used to distinguish *optimal* answer sets. A *weak constraint* is a labelled integrity constraint of the form:

$$\leftarrow b_1, \dots, b_n. [w, t_1, \dots, t_m]$$

where $n \geq 1$, $b_1, \dots, b_n$ are literals, w is an integer, $m \geq 1$, $t_1, \dots, t_m$ are terms and $[w, t_1, \dots, t_m]$ is called the *label*. The notion of safety is extended to weak constraints in that all variables must appear within a positive body literal. Ground instances of weak constraints are defined through variable substitutions by terms from the Herbrand Universe $\mathbb{U}_P$ in the same way as for rules.

Informally, a weak constraint is satisfied if the integrity constraint obtained by removing the label is satisfied. However, unlike integrity constraints, an answer set is not required to satisfy every weak constraint.

Definition 5.4 (Unsatisfied weak constraints). *Let I be an answer set of P . The unsatisfied weak rule instances of P w.r.t. I are:*

$$\text{weak}(P, I) = \{ \langle w, t_1, \dots, t_m \rangle \mid \leftarrow b_1, \dots, b_n. [w, t_1, \dots, t_m] \text{ in } \text{gnd}(P) \text{ where } b_1 \dots b_n \text{ are true in } I \}$$

The notion of an answer set's *optimality* captures the sum of the weights of the weak constraint instances that are not satisfied in the answer set.

Definition 5.5 (Optimality). *Let I be an answer set of P . The optimality of I is:*

$\sum_{w|\langle w, t_1, \dots, t_m \rangle \in \text{weak}(P, I)} w$, *the sum of w for each unsatisfied weak constraint ground instance. I is an optimal answer set of P if no other answer set of P has a smaller optimality than I .*

The label in a ground weak constraint serves as an identifier that groups together rule instances when computing the optimality. That is, any non-empty set of unsatisfied weak constraints having the same ground label $[w, t_1, \dots, t_m]$ introduce one tuple $\langle w, t_1, \dots, t_m \rangle$ to $\text{weak}(P, I)$.

Example 5.2 (continued). *Adding rules (7)..(9) to $\Pi_{5.2}$ where*

$$\leftarrow \text{drives}(q, Y), \text{car}(Y). \quad [1, q, \text{car}(Y)] \quad (7)$$

$$\leftarrow \text{drives}(q, Y), \text{bus}(Y). \quad [2, q, \text{bus}(Y)] \quad (8)$$

$$\leftarrow \text{drives}(q, Y), \text{fast}(Y). \quad [3, q, \text{fast}(Y)] \quad (9)$$

distinguishes I_2 and I_3 by optimality. In both I_2 and I_3 , $(9 : Y/c3)$ is unsatisfied, leading to a weight of 3. However, $(7 : Y/c3)$ is unsatisfied in I_2 adding a weight of 1 whereas $(8 : Y/c3)$ is unsatisfied in I_3 adding a weight of 2. We conclude that I_2 has an optimality of 4 and I_3 has an optimality of 5. Hence, I_2 is the (unique) optimal answer set of P .

Comparison predicates

The *ASP-Core-2* specification defines a set of built-in *comparison predicates* that are used to compare terms during grounding. A *comparison atom* is written in infix notation as $t_1 \text{ op } t_2$ where t_1 and t_2 are terms and op is an arity 2 *comparison predicate* from the set of operators $\{ ==, !=, <, >, <=, >= \}$.

Definition 5.6. *Let P be a program, $\text{gnd}(P)$ be the grounding of P and I be a Herbrand interpretation of $\text{gnd}(P)$. A (ground) comparison atom, $t_1 \text{ op } t_2$, is true w.r.t. I iff t_1 compared to t_2 by the operator $\text{math}(\text{op})$ evaluates to true; where $\text{math}(\text{op})$ denotes the corresponding element of $\{ =, \neq, <, >, \leq, \geq \}$ and the comparisons are evaluated arithmetically when both t_1 and t_2 are integers, and evaluated lexicographically otherwise.*

Example 5.3. Consider the ASP program:

$$\Pi_{5.3} = \left\{ \begin{array}{l} v(c1). \quad pas(c1, 4). \quad owns(p, c1). \\ v(c2). \quad pas(c2, 8). \quad owns(p, c2). \\ v(c3). \quad pas(c3, 20). \\ car(X) \leftarrow v(X), pas(c1, P), P \leq 4. \\ minibus(X) \leftarrow v(X), pas(c1, P), P == 8. \\ bus(X) \leftarrow v(X), pas(c1, P), P > 8. \\ drives(X, Y) = v(Y), owns(X, Y), X == Y. \end{array} \right\}.$$

The grounding of $\Pi_{5.3}$ obtained from *clingo* is:

$$\Pi_{5.3} = \left\{ \begin{array}{l} v(c1). \quad pas(c1, 4). \quad owns(p, c1). \\ v(c2). \quad pas(c2, 8). \quad owns(p, c2). \\ v(c3). \quad pas(c3, 20). \\ car(c1). [\leftarrow v(c1), pas(c1, 4), 4 \leq 4.] \\ minibus(c2). [\leftarrow v(c2), pas(c2, 8), 8 == 8.] \\ bus(X). [\leftarrow v(c3), pas(c3, 20), 20 > 8.] \\ drives(p, c1). [\leftarrow v(c1), owns(p, c1), c1 == c1.] \\ drives(X, Y). [\leftarrow v(c2), owns(q, c2), c2 == c1.] \end{array} \right\}.$$

During grounding, comparison atoms are evaluated to obtain a truth value and these are used to determine the relevant rule instances. For instance, for the rule instance:

$$minibus(c1) \leftarrow v(c1), pas(c1, 4), 4 == 8.$$

$4 == 8$ evaluates to false, indicating that the rule instance is not relevant.

For $\Pi_{5.3}$, the *clingo* grounder simplifies the relevant program (shown by) to a set of facts by removing literals that are established as true.

5.1.2 Reasoning tasks

The clingo solver offers access to a range of ASP reasoning tasks. Our implementation relies on computing the optimal answer sets, formalised next.

Definition 5.7 (Reasoning tasks). *Let P be a program and $AS(P)$ denote the set of answer sets of P . The core reasoning tasks are:*

- Find an answer set of P : *Given a program P find (non-deterministically) $A \in AS(P)$ or, where $AS(P) = \emptyset$, return **unsatisfiable**.*
- Find an optimal answer set of P : *Given a program P that includes weak constraints, find (non-deterministically) (A, o) where $A \in AS(P)$ is an optimal answer set of P with optimality o or, where $AS(P) = \emptyset$, return **unsatisfiable***
- Find a t -bounded optimal answer set of P : *Given a program P , that includes weak constraints, and t , a target upper bound on the optimality of answer sets returned, find (non-deterministically) (A, o) where $A \in AS(P)$ is an optimal answer set of P with optimality $o \leq t$ or, where $o > t$ or $AS(P) = \emptyset$, return **unsatisfiable**.*

The bounded search for optimal answer sets is useful in cases where the optimality is expected to be above or below a known threshold. A bounded search for optimal answer sets of P is often considerably faster than the corresponding unbounded search.

5.1.3 Features specific to clingo

Our implementation takes advantage of functionality that is specific to the clingo system and is not part of the ASP-core-2 specification.

Incremental grounding and solving

The clingo system includes support for *incremental* grounding and solving under scripted control². Informally, the idea is that we can ground and solve a *base* program and then add further ground rule instances to the base program that are dependent on the answer sets found. The additional rule instances are defined in a separate program and its rule instances are added to the ground program without re-grounding the base program. The combined program is then solved, and if required the process can be repeated to further extend the program.

An incremental program is encoded as a set of *sub-programs*. The start of each (sub-)program is indicated by `#program n(c1, ..., cm)` where n is the name of the program, $m \geq 0$ and $c_1 \dots c_m$ are *parameters*. Rules that appear before the first program directive are automatically assigned the program named *base* and each `#program` directive defines the end of any such previous program.

We use the notation P_n to refer to the (possibly non-ground) program named n and use $P_n(t_1, \dots, t_m)$ where $n \geq 0$ to denote the (possibly non-ground) *instance* of program n created by replacing each parameters c_i in a rule $r \in P_n$ by t_i where $0 \leq i \leq m$. Program instances are *cumulatively* ground under script control such that the ground rule instances obtained are added to any previously existing ground rule instances. The ground terms that are substituted for variables to obtain $gnd(P_n(t_1, \dots, t_m))$ are those from the combined Herbrand Universe of any previously grounded rules together with the rules of $P_n(t_1, \dots, t_m)$.

Example 5.4. Let $\Pi_{5.4}$ be an incremental program:

$$\Pi_{5.4} = \left\{ \begin{array}{ll} i(a). & (1) \qquad \qquad \#program \text{ add}(p) \quad (4) \\ i(b). & (2) \qquad \qquad i(p). \quad (5) \\ k(X, X) \leftarrow i(X). & (3) \qquad \qquad k(p, X) \leftarrow i(X) \quad (6) \end{array} \right\}$$

Rules (1-3) are assigned to the sub-program *base* and rules (4-6) to the program *add*. Consider the following sequence of incremental grounding and solving for $\Pi_{5.4}$.

²Our implementation uses the Lua scripting language, see <http://www.lua.org>.

1. *Ground the base program (1) – (3):*

$i(a). \quad i(b). \quad k(a, a) \quad k(b, b).$

Solving the resulting program yields 1 answer set $I_0 = \{i(a), i(b), k(a, a), k(b, b)\}$

2. *Suppose we now include knowledge about g by instantiating the program $\text{add}(g)$:*

$i(g). \quad k(g, X).$

The program $\text{add}(g)$ is ground over the Herbrand universe $\{a, b, g\}$:

$i(g). \quad k(p, g) \leftarrow i(g). \quad k(p, a) \leftarrow i(a). \quad k(p, b) \leftarrow i(b).$

and solving the resulting program yields 1 answer set:

$I_1 = I_0 \cup \{i(g), k(g, a), k(g, b), k(g, g)\}$. *The ground rules of the program base are unchanged and lead to the atoms in I_0 being present in I_1 .*

3. *The term p supplied to the grounder in $\text{add}(p)$ may be any valid ground term and can include terms already present in the program. For instance, instantiating the grounding of $\text{add}(f(a))$ (after step 2) and solving the resulting program yields 1 answer set:*

$I_2 = I_1 \cup \{i(f(a)), k(f(a), a), k(f(a), b), k(f(a), g), k(f(a), f(a))\}$.

The technique is of particular value where the number of constants required to obtain a solution to a problem can not be predicted. The answer sets representing solutions can be inspected at each iteration and new rules instantiated where required.

Custom functions

The clingo system permits functions of the form $@f(t_1, \dots, t_m)$ to appear within terms, where $m \geq 0$ and $t_1, \dots, t_m$ are terms. The prefix $@$ signifies that f is a *custom function* and is implemented in the scripting language. During grounding each custom function is replaced by the ground term obtained by evaluating the supplied script function. It is assumed that the Herbrand universe for grounding includes new terms introduced due to the evaluation of scripted functions and that the calls to such functions always return a term of finite size.

5.1.4 Properties of Answer Set Programs

In this section we review a number of existing techniques that are used in the analysis of answer set programs. The notation used in the definitions and theorems quoted have been adjusted slightly to follow the conventions used in this thesis.

Finitely Ground Programs

We review the necessary conditions that determine the class of Finitely Ground Programs ($\mathcal{FG}$ Programs) given in [CCIL08]. This property is used to show that a given program has an equivalent finite ground program. In the following, the notation: $p||n||$ is used to denote the n -th argument of an ASP predicate p with a fixed arity $k \geq 0$ and $\bar{t}$ denotes a tuple of terms.

Definition 1[CCIL08]. *The Argument Graph $\mathcal{G}^A(P)$ of a program P is a directed graph containing a node for each argument $p||i||$ of a predicate p of P ; there is an edge $(q||j||, p||i||)$ iff there is a rule $r \in P$ such that: (a) an atom $p(\bar{t})$ appears in the head of r ; (b) an atom $q(\bar{v})$ appears in $body^+(r)$; (c) the i -th argument of $p(\bar{t})$ and the j -th argument of $q(\bar{v})$ share the same variable. The set of arguments of P , denoted $ARGS(P)$, is the set of nodes in $\mathcal{G}^A(P)$.*

Given a program P , an argument $p||i||$ is said to be recursive with $q||j||$ if there exists a cycle in $\mathcal{G}^A(P)$ involving both $p||i||$ and $q||j||$.

Example 5.5. *Consider the program:*

$$\Pi_{5.5} = \{ q(f(a)). (1) \ p(X) \leftarrow q(f(X)). (2) \ q(X) \leftarrow p(X). (3) \ r(X, Y) \leftarrow p(X), q(Y). (4) \}$$

The argument graph of $\mathcal{G}^A(\Pi_{5.5})$ is shown in Figure 5.1.

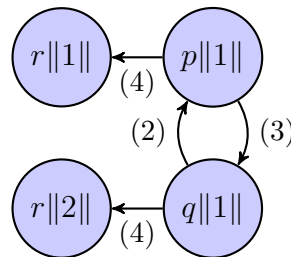


Figure 5.1: The argument graph $\mathcal{G}^A(\Pi_{5.5})$

Definition 10 [CCIL08]. *Given a program P , the set of finite-domain ($\mathcal{FD}$) arguments of P is the maximal (w.r.t. set inclusion) set $FD(P) \subseteq ARGS(P)$ such that $q||k|| \in FD(P)$ iff every rule $r \in P$ in which predicate q appears in the head of r satisfies the following condition. Let t be the term corresponding to argument $q||k||$ in the head of r . Then,*

1. *either t is variable-free, or*
2. *t is a subterm of (the term of) some finite-domain argument of a predicate that occurs within an atom in $body^+(r)$, or*
3. *every variable appearing in t also appears in (the term of) a finite-domain argument of a predicate that occurs within an atom in $body^+(r)$ which is not recursive with $q||k||$.*

If all arguments in $ARGS(P)$ are $\mathcal{FD}$ arguments, then P is said to be an $\mathcal{FD}$ program.

Informally, the idea is that we can identify relevant rule instances by establishing a finite set of possible values for each argument that appears within the ground program. The syntactic constraints eliminate programs that have infinitely many answer sets and those with infinite answer sets. (See [CCIL08] Corollary 2).

Example 5.6. *Consider $\Pi_{5.6a}$ (taken from Example 9 in [CCIL08]) and $\Pi_{5.6b}$:*

$$\Pi_{5.6a} = \{ q(f(0)). (1) \quad q(X) \leftarrow q(f(X)). (2) \} \quad \Pi_{5.6b} = \{ q(0). (3) \quad q(f(X)) \leftarrow q(X). (4) \}$$

We show that $\Pi_{5.6a}$ is an $\mathcal{FD}$ program by checking that each argument in $ARGS(\Pi_{5.6a})$ satisfies the conditions of Definition 10 [CCIL08]. $ARGS(\Pi_{5.6a}) = \{q||1||\}$. q appears in the head of (1) in which the term $f(0)$ is variable free and satisfies condition 1. q appears in the head of (2) in which the term X is a sub-term of $f(X)$ the term of the argument $f||1||$ in the positive body literal $q(f(X))$ and satisfies condition 2. We conclude that $\Pi_{5.6a}$ is an $\mathcal{FD}$ program. In contrast, for $\Pi_{5.6b}$ $ARGS(\Pi_{5.6b}) = \{q||1||\}$. q appears in the head of (3) in which the term $f(0)$ is variable free and satisfies condition 1. q appears in the head of (4) in which the term $f(X)$ is (i) not variable free; not (ii) a sub-term of X , the only term of an argument in a positive body

literal; and (iii) X is recursive. We conclude that $\Pi_{5.6b}$ is not an $\mathcal{FD}$ program. The answer set of $\Pi_{5.6b}$ is not finite, it contains atoms with the increasing terms $f(0), f(f(0)), \dots$

Head Cycle Free programs

Informally, *head-cycle free* programs [BED94] capture a class of disjunctive programs that can be transformed into non-disjunctive logic programs by “shifting” the head atoms into the body. They are of particular interest because computing answer sets for such programs has lower computational complexity than for programs that are not head-cycle free [LPF⁺06]. These complexity results will be summarised in Section 5.1.4. In the following, P is a ground program, $atoms(P)$ denotes the set of atoms appearing in P .

Definition 2 [LTW04]. *The (positive) dependency graph of a program P is given by $G_P = (atoms(P), E_P)$, where $E_P \subseteq atoms(P) \times atoms(P)$ is defined by the condition that $(p, q) \in E_P$ iff there exists some $r \in P$ such that $p \in body^+(r)$ and $q \in head(r)$.*

An atom $a \in atoms(P)$ is said to be *dependent on* atom $b \in atoms(P)$ if b is *reachable* from a in the directed graph G_P . A dependency graph G_P includes a *directed cycle* if there exists a set of atoms $\mathcal{C} = \{a_1, \dots, a_n\} \subseteq atoms(P)$ where $n > 1$ such that $(a_i, a_{i+1}) \in E_P$ for $1 \leq i \leq (n-1)$ and (a_{n-1}, a_0) . A cycle $\mathcal{C}$ is said to *go through* an atom a iff $a \in \mathcal{C}$.

A program is *head-cycle free* iff its dependency graph does not contain directed cycles that go through two atoms that belong to the head of the same rule [BED94].

Example 5.7. *Consider the following programs:*

$$\Pi_{5.7a} = \{ b|c. \} \quad \Pi_{5.7b} = \{ b|c. \quad b \leftarrow c. \quad c \leftarrow b \}$$

$G_{\Pi_{5.7a}} = (\{b, c\}, \{\})$ and $\Pi_{5.7a}$ is head-cycle free. $\Pi_{5.7a}$ has two answer sets $\{b\}$ and $\{c\}$ and is equivalent to $\{ b \leftarrow \text{not } c. \quad c \leftarrow \text{not } b. \}$. Whereas, $G_{\Pi_{5.7b}} = (\{b, c\}, \{(b, c), (c, b)\})$ and $\Pi_{5.7b}$ includes a directed cycle that goes through b and c appearing the head of the rule $b|c$. $\Pi_{5.7b}$ has

one answer set $\{b, c\}$ and is not equivalent to $\{ b \leftarrow \text{not } c. \quad c \leftarrow \text{not } b. \quad b \leftarrow c. \quad c \leftarrow b. \}$ which is unsatisfiable.

Locally Stratified Programs

We recall the conditions for local stratification [Prz88] of a disjunctive logic program. Locally stratified programs are of interest because answer sets of such programs can be constructed using algorithms that have lower computational complexity than those for unstratified programs. Informally, such algorithms operate on a locally stratified equivalent finite ground program and construct the answer sets bottom up. They start from the rules at the lowest stratum using a modified form of the T_p operator [Prz88]. At each stratum the truth values for literals in the previous stratum are established and used to simplify the remaining rules. The stratification ensures that all the negative literals are simplified out of the program. The disjunctions lead to the development of multiple answer sets and these are checked for minimality as the algorithm progresses. The designation *local* captures that the stratification is based on the relative positions of ground atoms in rule instances (i.e. within the head, positive body literals or negative body literals), rather than relative positions of predicates within rules as in stratified programs. In Definitions 5.8 and 5.9 that follow, the notion of *locally stratified* programs from Definition 4.1 [Prz88] is expressed in terms of the *definitions* of atoms adapted from Definition 2 [ABW88].

Definition 5.8 (Definitions of predicates and atoms). *Let r be a rule of a program P . Then a ground atom a_g is said to be defined in a rule instance $r_g \in \text{gnd}(P)$ iff $a_g \in \text{head}(r_g)$ and r_g is called a definition of a_g .*

Definition 5.9 (Locally stratified program). *A program P is locally stratified if and only if there is a partition $\mathcal{P}$ of $\text{gnd}(P)$, $\{P_1, \dots, P_s\}$ such that the following conditions hold for each ground rule instance $r_g \in \text{gnd}(P_k)$ where $k = 1, \dots, s$:*

- (i) *Every definition of $h_g \in \text{head}(r_g)$ is in P_k ;*
- (ii) *Every definition of $b_g \in \text{body}^+(r_g)$ is in $\bigcup_{j \leq k} P_j$;*

(iii) Every definition of $b_g \in \text{body}^-(r_g)$ is in $\bigcup_{j < k} P_j$.

Each element of the partition $\mathcal{P}$ is called a *stratum* of $\text{gnd}(P)$.

Notice that atoms that are not defined at all vacuously satisfy the conditions.

Example 5.8. Consider the following program:

$$\Pi_{5.8} = \left\{ p(f(0)). (1), \quad p(g(1)). (2), \quad p(h(X)) | q(X) \leftarrow p(f(X)), \text{ not } p(g(X)) (3) \right\}$$

The program has an infinite grounding. However, it is locally satisfied, as seen by partitioning the ground program $\text{gnd}(\Pi_{5.8})$ such that stratum 1 contains rule (2) and stratum 2 contains rule (1) and all instances of rule (3).

The Splitting set Theorem

Splitting sets [LT94] provide us with a powerful tool to analyse the answer sets of programs. Informally, a splitting set can be used to simplify the process to obtain the answer sets of a ground program. A set of atoms S splits a program P into a top program and a bottom program if (1) only the rules in P that define atoms in S appear in the bottom program and (2) these rules do not depend on atoms defined in the rules in the top program. We can now find the answer sets of the bottom program alone. Now, since the top program may be dependent on atoms of S , an answer set X of the bottom program is used to simplify the rules in the top program by assigning each body literal formed from an atom in S to the value assigned in X . Finally, the splitting set theorem captures that each answer set of P can be written as $X \cup Y$ where X is an answer set of the rules in the bottom program and Y is an answer set of the simplified rules of the top program. The following definitions are based on those given for ground disjunctive logic program in [LT94] Section 3.

Definition 5.10 (Splitting Set). Let P be a ground program and S be a set of ground atoms. S is a splitting set of P iff, for every rule $r \in P$, if $\text{head}(r) \cap S \neq \emptyset$ then $\text{atoms}(r) \subseteq S$. S

is also said to **split** P . The set of rules $r \in P$ such that $\text{atoms}(r) \subseteq S$ is the bottom of P relative to S , denoted by $\text{bot}_S(P)$. The set $\text{top}_S(P) = P - \text{bot}_S(P)$ is the top of P relative to S .

Definition 5.11 (Partial Evaluation). Let P be a ground program, and let S and X be sets of atoms. The partial evaluation of P w.r.t. S and X is $e_S(P, X)$, the set of rules obtained from P as follows: For each rule $r \in P$ where $\text{body}^+(r) \cap S$ is part of X and $\text{body}^-(r) \cap S$ is disjoint from X take the rule r' defined by $\text{head}(r') = \text{head}(r)$, $\text{body}^+(r') = \text{body}^+(r) \setminus S$ and $\text{body}^-(r') = \text{body}^-(r) \setminus S$.

Splitting Set Theorem [LT94] Let S be a splitting set of a ground program P . A set of ground atoms is an answer set for P iff it can be written in the form $X \cup Y$ where X is an answer set of $\text{bot}_S(P)$ and Y is an answer set of $e_S(\text{top}_S(P), X)$.

Example 5.9. Consider the following program:

$$\Pi_{5.8} = \left\{ \begin{array}{l} r.(1), \quad p|q \leftarrow r. (2), \quad s|t \leftarrow \text{not } q. (3), \quad u \leftarrow s, \text{not } p. (4) \end{array} \right\}$$

Let $S = \{p, q, r\}$. S splits $\Pi_{5.8}$ such that $\text{bot}_S(\Pi_{5.8})$ contains rules (1, 2) and $\text{top}_S(\Pi_{5.8})$ contains rules (3, 4). $\text{bot}_S(\Pi_{5.8})$ has two answer sets $I_{t1} = \{r, p\}$ and $I_{t2} = \{r, q\}$. Simplifying $\text{top}(\Pi_{5.8})$ we obtain:

$$\begin{aligned} e_S(\text{top}_S(\Pi_{5.8}, I_{t1})) &= \left\{ \begin{array}{l} s|t. \end{array} \right\} \text{ with answer sets } I_{b1} = \{s\} \text{ and } I_{b2} = \{t\} \\ e_S(\text{top}_S(\Pi_{5.8}, I_{t2})) &= \left\{ \begin{array}{l} u \leftarrow s. \end{array} \right\} \text{ with answer set } I_{b3} = \{\} \end{aligned}$$

Applying Theorem 5.1.4 for $\Pi_{5.8}$ we obtain three answer sets $I_{t1} \cup I_{b1} = \{r, p, s\}$, $I_{t1} \cup I_{b2} = \{r, p, t\}$ and $I_{t2} \cup I_{b3} = \{r, q\}$.

Complexity

We begin by introducing complexity classes that belong to **PH**, the polynomial hierarchy [AB09]. These classes can be defined using the notion of an **NP oracle**, a hypothetical entity that decides any problem in **NP** in deterministic polynomial time (**P**). The classes Δ_n^P (and Σ_n^P) where $n \geq 2$

Brave	$\{\}$	$\{w\}$	$\{not_s\}$	$\{not_s, w\}$	$\{not\}$	$\{not, w\}$
$\{\}$	$\mathbf{P}$	$\mathbf{P}$	Δ_2^P	Δ_2^P	Σ_2^P	Δ_2^P
$\{v_h\}$	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P
$\{v\}$	Σ_3^P	Δ_4^P	Σ_3^P	Δ_4^P	Σ_3^P	Δ_4^P

Table 5.1: The Complexity of Brave Reasoning for fragments of ASP as quoted in [LPF⁺06] Table 5. The fragments are given by the combinations of the following program characteristics: stratified negation (not_s), arbitrary negation (not), head-cycle free disjunction (v_h), arbitrary disjunction (v) and weak constraints (w).

are the class of problems that can be solved in deterministic polynomial (resp. non-deterministic polynomial) time with the use of $(n-1)$ $\mathbf{NP}$ oracles. The notation $C_1^{C_2}$ is used to denote the class of decision problems that are solvable in C_1 with the use of a C_2 oracle. Formally, $\Delta_0^P = \Sigma_0^P = \mathbf{P}$, and for $k \geq 1$, $\Delta_k^P = \mathbf{P}^{\Sigma_{k-1}^P}$ and $\Sigma_k^P = \mathbf{NP}^{\Sigma_{k-1}^P}$. For instance, $\Delta_2^P = \mathbf{P}^{\mathbf{NP}}$ and $\Sigma_3^P = \mathbf{NP}^{\mathbf{NP}^{\mathbf{NP}}}$.

We focus on the *brave reasoning decision problem*: given a program P and a ground atom a decide if a is true in at least one (optimal) answer set of P . The complexity of the brave reasoning decision problem for P provides a method for characterising the related task of finding an (optimal) answer of P [LPF⁺06]. Informally, the problem of finding an (optimal) answer set of P can be translated into the a brave reasoning task by deciding if a is true in at least one (optimal) answer set of $P \cup \{a.\}$ where a is an atom not present in P . The complexity is sensitive to the structure of a program. Table 5.1 shows the complexity of brave reasoning for non-ground programs with *bounded-arity* [EFFW07], where there is an upper bound on the arities of predicates³.

The classes within the polynomial hierarchy are known to be in $\mathbf{PSPACE}$ [AB09], for $k > 1$, $\Sigma_k^P \subseteq \Delta_{k+1}^P \subseteq \mathbf{PSPACE}$. Note that $\Delta_1^P = \mathbf{P}$ and $\Sigma_1^P = \mathbf{NP}$.

5.2 Encoding the $p\text{-}\mathcal{ALC}$ tableau algorithm in ASP

We begin with an informal overview of our approach. A knowledge base $\mathcal{K}$ is represented as a set of ASP facts and these are combined with a fixed set of non-ground ASP rules to form

³These results are broadly one complexity level higher than the corresponding complexities for ground programs (c.f. [LPF⁺06] Table 1)

the program $ASP(\mathcal{K})$. The fixed rules are designed to faithfully reproduce the actions of the tableau rules of Algorithm 2. When $ASP(\mathcal{K})$ is ground and solved incrementally, it leads to a representation of a tableau branch within each answer set for which the optimality of the answer set corresponds to the distance of the branch, or leads to no answer sets for knowledge bases for which all branches of the tableau close. Recall from Chapter 4 that the query $\mathcal{K} \models C(x)$? can be answered by refutation in two steps, first finding the n -inconsistency of $\mathcal{K}$ and then checking for the existence of an n -optimal interpretation of $\mathcal{K} \cup \neg C(x)$. Our implementation of Algorithm 2 in ASP allows us to follow the same approach. We perform an unbounded search for an optimal answer set of $ASP(\mathcal{K})$ to obtain the optimality o of $ASP(\mathcal{K})$. This indicates that $\mathcal{K}$ is o -inconsistent. Then we consider $ASP(\mathcal{K}')$, where $\mathcal{K}'$ is given by $\mathcal{K}$ augmented with $\neg C(x)$. In this second step we perform a bounded search for an o -optimal answer set. If such an answer set is not found then $\mathcal{K}'$ is either unsatisfiable or m -optimal for $m > o$, and in both cases we can conclude $\mathcal{K} \models C(x)$.

5.2.1 The representation of a $p\text{-}\mathcal{ALC}$ knowledge base

In this section we define τ , an operator that is used to translate a given $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$ into a set of ASP facts denoted $\mathcal{K}^\tau$. $p\text{-}\mathcal{ALC}$ concepts are represented as ground ASP terms using unary or binary function symbols to denote the constructors and constant symbols to concept and role names. We show later in Section 5.2.2 how this representation of concepts as ASP terms allows us to write non-ground rules that instantiate all the necessary tableau rule expansions by exploiting the grounder.

We will assume that each name N in the signature $sig(\mathcal{K})$ can be translated into an ASP constant N^τ by mapping the first letter to its lower case⁴. Concepts are translated to ground ASP terms using the unary or binary function symbols *neg*, *and*, *or*, *oSome* and *oAll* to denote the constructors $\neg, \sqcap, \sqcup, \exists, \forall$ (respectively). An ordering is imposed on the concepts prior to translation to ASP terms that ensures a unique representation. For example, concepts $D \sqcup (C \sqcap B \sqcap \neg A)$ and $(\neg A \sqcap B \sqcap C) \sqcup D$ are re-ordered as $D \sqcup (B \sqcap C \sqcap \neg A)$ and represented as

⁴This facilitates visual inspection of the answer sets generated. However, the implementation could be modified to add a fixed prefix to all names.

or($d, \text{and}(b, \text{and}(c, \text{neg}(a)))$) in ASP. Similarly, $\exists R.(\forall S.C \sqcup D \sqcup \exists R.\neg E)$ is ordered as $\exists R.(D \sqcup \exists R.\neg E \sqcup \forall S.C)$ and is represented as $\text{oSome}(r, \text{or}(d, \text{or}(\text{osome}(r, \text{neg}(e)), \text{oAll}(S, C))))$.

The reordering makes use of a total ordering over concepts denoted $<_o$ as defined below.

Definition 5.12 (Constructor and arguments of a concept). *Let C be a concept. $\text{cnstr}(C)$ and $\text{cargs}(C)$ denote (respectively) the constructor in C , and the set of concept arguments in C where*

$$\text{cnstr}(C) = \begin{cases} \alpha & \text{if } C \text{ is a concept name} \\ \neg & \text{if } C = \neg C_1 \\ \sqcap & \text{if } C = C_1 \sqcap \dots \sqcap C_n \\ \sqcup & \text{if } C = C_2 \sqcup \dots \sqcup C_n \\ \exists & \text{if } C = \exists R.C_1 \\ \forall & \text{if } C = \forall R.C_1 \end{cases} \quad \text{cargs}(C) = \begin{cases} \{C_1, \dots, C_n\} & \text{if } C = C_1 \sqcap \dots \sqcap C_n \\ \{C_2, \dots, C_n\} & \text{if } C = C_2 \sqcup \dots \sqcup C_n \end{cases}$$

$\text{carg}(C)$ denotes the concept C_1 if $C = \neg C_1$, $C = \exists R.C_1$ or $C = \forall R.C_1$.

$\text{role}(C)$ denotes the role R if $C = \exists R.C_1$ or $C = \forall R.C_1$.

Definition 5.13 (Concept ordering). *Let C_1 and C_2 be concepts. The relation $<_o$ is defined as $C_1 <_o C_2$ iff $\text{cnstr}(C_1) <_1 \text{cnstr}(C_2)$, or if $\text{cnstr}(C_1) = \text{cnstr}(C_2)$ and case*

$$\begin{cases} \text{cnstr}(C_1) = \alpha & C_1 <_{\text{lex}} C_2 \\ \text{cnstr}(C_1) = \neg & \text{carg}(C_1) <_o \text{carg}(C_2) \\ \text{cnstr}(C_1) = \forall(\exists) & \text{role}(C_1) <_{\text{lex}} \text{role}(C_2), \\ & \text{or } \text{role}(C_1) = \text{role}(C_2) \wedge \text{carg}(C_1) <_o \text{carg}(C_2) \\ \text{cnstr}(C_1) = \sqcap(\sqcup) & \text{cargs}(C_1) \text{ are ordered according to } <_o \end{cases}$$

The relation $<_1$ is given by: $\alpha <_1 \neg <_1 \sqcup <_1 \sqcap <_1 \exists <_1 \forall$ and $<_{\text{lex}}$ denotes a lexicographic ordering.

It is easy to show that $<_0$ is irreflexive, antisymmetric, transitive and total, i.e. it is a total order relation.

Definition 5.14 (*p-ALC concepts in ASP*). Let C be a concept. C^τ denotes the ASP term obtained by (i) ordering the arguments of the n -ary constructor in C by $<_o$ to form C^o , (ii) translating C^o inductively, according to Table 5.2 and (iii) mapping the first letter of each concept name and role name to lower case.

<i>p-ALC</i> ordered concept C^o	ASP term C^τ
$\top$	<i>thing</i>
$\perp$	<i>neg(thing)</i>
(C)	C^τ
$\neg C$	<i>neg</i> (C^τ)
$C_1 \sqcap \dots \sqcap C_n$	<i>and</i> ($C_1^\tau, \text{and}(C_2^\tau, \text{and}(\dots, C_n^\tau))$)
$C_1 \sqcup \dots \sqcup C_n$	<i>or</i> ($C_1^\tau, \text{or}(C_2^\tau, \text{or}(\dots, C_n^\tau))$)
$\exists R.C$	<i>oSome</i> (R^τ, C^τ)
$\forall R.C$	<i>oAll</i> (R^τ, C^τ)

Table 5.2: The mappings used to translate an ordered concept C^o to an ASP term C^τ

The axioms in the knowledge base are encoded as facts of $ax/2$, in which the first argument is a translation of the axiom that excludes any associated weight and the second argument records the associated weight > 0 . Recall from Chapter 3, that the weights that are assigned to defeasible axioms are positive, where the larger the weight the more you want to avoid falsifying the axiom and that the non-defeasible axioms have no associated weight. To maintain a fixed arity for ax , the non-defeasible axioms are labelled with a weight of 0. A weight $w = 0$ thus identifies a non-defeasible axiom (see rule (a1) of the encoding in Section 5.2) and a weight $w > 0$ identifies a defeasible axiom (see rules (a3,a4)).

Facts of the atoms $i/1$, $c/1$ and $r/1$ capture the names in the signature of the knowledge base.

Definition 5.15 (*p-ALC knowledge base in ASP*). Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a knowledge base with signature $\langle N_I, N_C, N_R \rangle$. The encoding in ASP of $\mathcal{K}$, $sig(\mathcal{K})$ and $\mathcal{Q}(\mathcal{K})$ are defined as

follows:

$$\begin{aligned}
K^\tau &= \{ax(Z^\tau, 0). \mid Z \in \mathcal{A} \cup \mathcal{T}\} \cup \{ax(Z^\tau, w). \mid Z^{[w]} \in \mathcal{A}_d \cup \mathcal{T}_d\} \\
sig(\mathcal{K})^\tau &= \{i(x^\tau). \mid x \in N_I\} \cup \{c(C^\tau). \mid C \in N_C\} \cup \{r(R^\tau). \mid R \in N_R\} \\
\mathcal{Q}(\mathcal{K})^\tau &= \{qc(C^\tau). \mid C \in \mathcal{Q}(\mathcal{K})\}
\end{aligned}$$

where the translations by τ are given in Table 5.3.

$p\text{-}\mathcal{ALC}$ axiom Z	ASP term Z^τ
$C(x)$	$ca(C^\tau, x^\tau)$
$R(x, y)$	$ra(R^\tau, x^\tau, y^\tau)$
$C \sqsubseteq D$	$sc(C^\tau, D^\tau)$

Table 5.3: The mappings used to translate a $p\text{-}\mathcal{ALC}$ axiom Z to an ASP term Z^τ

Example 5.10. Consider the knowledge base from Example 3.5.

$$\mathcal{K}_{\overline{pat}} = \left\langle \left\{ \begin{array}{l} G(a), \neg S(a), (\forall R. \neg S)(a), \\ S(b), (\exists R. G)(b), (\forall R. S)(b) \end{array} \right\}, \{H \sqsubseteq \neg S\}, \{R(a, b)^{[1]}\}, \{G \sqsubseteq H^{[1]}\} \right\rangle$$

The knowledge base and its signature are encoded by the ASP facts

$$\begin{aligned}
\mathcal{K}_{\overline{pat}}^\tau &= \left\{ \begin{array}{lll} ax(ca(g, a), 0). & ax(ca(neg(s), a), 0). & ax(ca(oAll(r, neg(s)), a), 0) \\ ax(ca(s, b), 0). & ax(ca(oSome(r, g), b), 0). & ax(ca(oAll(r, s), b), 0). \\ ax(sc(h, neg(s)), 0). & ax(ra(r, a, b), 1). & ax(sc(g, h), 1). \end{array} \right\} \\
sig(\mathcal{K}_{\overline{pat}})^\tau &= \left\{ i(a). \quad i(b). \quad c(g). \quad c(h). \quad c(s) \quad r(r). \right\} \\
\mathcal{Q}(\mathcal{K}_{\overline{pat}})^\tau &= \left\{ qc(s). \quad qc(g). \right\}
\end{aligned}$$

5.2.2 Encoding the expansion rules

Recall that a branch is a pair $\langle \mathcal{A}_b, \mathcal{O}_b \rangle$ where $\mathcal{A}_b$ is a set of ABox axioms and $\mathcal{O}_b$ is the set of omitted axiom instances. The axioms in $\mathcal{A}_b$ are represented in ASP by the predicates $isa/2$ and $hasa/3$ and the omitted axiom instances in $\mathcal{O}_b$ by the predicate $u/2$. Each $C(x) \in \mathcal{A}_b$ is represented in ASP as $isa(x^\tau, C^\tau)$, each $R(x, y) \in \mathcal{A}_b$ is represented as $hasa(x^\tau, R^\tau, y^\tau)$ and

each $\langle Z^{[w]}, x \rangle \in \mathcal{O}_b$ is represented as $u(Z^\tau, x^\tau, w)$. The representation of a knowledge base and its signature in ASP includes a constant x^τ for each named individual x in the signature. However, the $\rightarrow_{\exists\forall}$ -rule of the tableau also introduces a set of parameters and these must be added to the program. The *need/2* predicate is used to indicate where a parameter is needed to be introduced as a witness. The idea is that where the $\rightarrow_{\exists\forall}$ -rule would introduce a fresh p for $(\exists R.C)(x) \in \mathcal{A}_b$, $need(x^\tau, oSome(R^\tau, C^\tau))$ appears in the answer set of the program. New rules are added to the program incrementally that ensure that when a fresh $p \in P$ is needed in the branch a corresponding constant p^τ is introduced in the ASP program.

The full ASP encoding of a $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$ is given by $ASP(\mathcal{K}) = \mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau \cup P_b \cup P_{cum}$ where P_b and P_{cum} are defined below. $P_{base} = \mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau \cup P_b$ denotes the *base* program and is instantiated once. P_{cum} is a *cumulative* program and instantiated once for each time a parameter is needed under control of a Lua script.

The rules in P_b

The program P_b implements the first sub-subsequence of a branch and involves only the named individuals from the signature of $\mathcal{K}$.

$$isa(X, C) \leftarrow ax(ca(C, X), 0). \quad (a1)$$

$$hasa(X, R, Y) \leftarrow ax(ra(R, X, Y), 0). \quad (a2)$$

$$isa(X, C) \mid u(ca(C, X), X, W) \leftarrow ax(ca(C, X), W), W > 0. \quad (a3)$$

$$:\sim u(ca(C, X), X, W). \quad [W, ca, C, X] \quad (a4)$$

$$hasa(X, R, Y) \mid u(ra(R, X, Y), X, W) \leftarrow ax(ra(R, X, Y), W), W > 0. \quad (a5)$$

$$:\sim u(ra(R, X, Y), X, W). \quad [W, ra, R, X, Y] \quad (a6)$$

Rules (a1,a2) correspond to initialisation of $\mathcal{A}_b$ with the axioms of $\mathcal{A}$. Rules (a3-a6) correspond to the omission of a subset of the axioms of $\mathcal{A}_d$ recorded in $\mathcal{O}_b$ and the remainder being added to $\mathcal{A}_b$. The supported inference of ground atoms representing non-defeasible ABox axioms is unconditional, whereas that of atoms representing defeasible ABox axiom instances is subject to

choice using disjunction (see rule (a3)). The corresponding weight, the “penalty” for omitting an instance, is captured by the use of weak constraints (see rule (a4)). For example, given a non-defeasible ABox $(C \sqcap D)(a)$ in $\mathcal{K}$, answer sets of $ASP(\mathcal{K})$ will include $ax(ca(and(c, d), a), 0)$. The atom $isa(and(c, d), a)$ will be included in every answer set by (a1). Similarly, if the defeasible ABox axiom $R(a, b)^{[2]}$ is in $\mathcal{K}$, the ASP program $\mathcal{K}^\tau$ will include $ax(ra(a, b), 2)$. By rule (a5), either $hasa(a, r, b)$ or $u(ra(r, a, b), a, 2)$ will be in each answer set. The weak constraint (a6) increases the total weight of the answer set by 2, meaning the answer set is less optimal when $u(ra(r, a, b), a, 2)$ is added to it.

$$isa(X, C) \leftarrow isa(X, and(C, D)). \quad (e1)$$

$$isa(X, D) \leftarrow isa(X, and(C, D)). \quad (e2)$$

$$isa(X, C) \mid isa(X, D) \leftarrow isa(X, or(C, D)). \quad (e3)$$

$$isa(Y, C) \leftarrow isa(X, oAll(R, C)), hasa(X, R, Y). \quad (e4)$$

$$isa(X, or(@neg(C), D)) \leftarrow ax(sc(C, D), 0), i(X). \quad (e5)$$

$$isa(X, or(@neg(C), D)) \mid u(sc(C, D), X, W) \leftarrow ax(sc(C, D), W), i(X), W > 0. \quad (e6)$$

$$:\sim u(sc(C, D), X, W). \quad [W, sc, C, D, X] \quad (e7)$$

Rules (e1-e7) capture the expansion rules. (e1, e2) implement $\rightarrow_\sqcap$, (e3) implements $\rightarrow_\sqcup$, (e4) implements $\rightarrow_\forall$, (e5) implements $\rightarrow_\tau$ and (e6, e7) implements $\rightarrow_{\tau_d}$. The symbol $@neg$ is a custom function and ensures that terms representing negated concepts are expressed correctly in negation normal form; if X is an ASP term representing concept C , $@neg(X) = (\neg C)^\tau$. For instance, during grounding: the term $@neg(a)$ is evaluated to the term $neg(a)$; $@neg(neg(a))$ to the term a ; and $neg(and(a, b))$ to the term $or(neg(a), neg(b))$.

$$isa(X, thing) \leftarrow i(X). \quad (e8)$$

$$isa(X, C) \mid isa(X, @neg(C)) \leftarrow qc(C), i(X). \quad (e9)$$

$$\leftarrow isa(X, neg(thing)). \quad (e10)$$

$$\leftarrow isa(X, C), isa(X, neg(C)), c(C). \quad (e11)$$

$$hw(X, oSome(R, C)) \leftarrow isa(X, oSome(R, C)), hasa(X, R, Y), isa(Y, C). \quad (e12)$$

$$need(X, oSome(R, C)) \leftarrow isa(X, oSome(R, C)), not hw(X, oSome(R, C)). \quad (e13)$$

$$used(X) \leftarrow i(X). \quad (e14)$$

Rule (e8) captures the property that every named individual has to belong to the “top” concept and rule (e9) implements the $\rightarrow_Q$ -rule for each of the named individuals. Rules (e10 and e11) guarantee that answer sets include only consistent expansions. Rules (e12 and e13) capture the $\rightarrow_{\exists\forall}$ tableau rule, where the atom $hw(X, oSome(R, C))$ means that “ X has a witness to the concept $\exists R.C$ ”. Where no such witness exists, $need/2$ labels that a parameter must be introduced. However, since we do not know a priori how many parameters are required, new parameters and their associated rules are introduced as needed, as explained next. The $used/1$ predicate in (e14) maintains a record of the individual names that are represented in a branch. All the named individuals are represented in every branch. Algorithm 3 shows the iterative grounding and solving process used to search for optimal answer sets.

Algorithm 3: The ASP grounding and solving algorithm

Input: $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$, a $p\text{-}\mathcal{ALC}$ knowledge base**Input:** t , a target optimality**Output:** **unsatisfiable** or an optimal answer set with associated optimality

```

1  $\mathcal{N} := \emptyset$ ;
2  $\mathcal{N}_S := \emptyset$  ;
3  $f := 1$ ;
4  $P_g := \text{Ground}(K^\tau \cup \text{sig}(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau \cup P_b)$  ;
5 while  $\text{Solve}(P_g, t)$  is not unsatisfiable do
6   Assert  $(S, o) := \text{Solve}(P_g, t)$ ;
7    $\mathcal{N}_S := \{ \text{need}(X, o\text{Some}(R, C)) \mid \text{need}(X, o\text{Some}(R, C)) \in S \}$  ;
8   if  $\mathcal{N} \cup \mathcal{N}_S = \mathcal{N}$  then return  $(S, o)$  ;
9   for  $\text{need}(X, o\text{Some}(R, C)) \in (\mathcal{N}_S \setminus \mathcal{N})$  do
10      $P_g := P_g \cup \text{Ground}(P_{cum}(f, X, o\text{Some}(R, C)))$ ;
11      $f := f + 1$ ;
12   end
13 end
14 return unsatisfiable ;

```

$\text{Ground}(P)$ takes as an input P a non-ground program and returns a finite (intelligent) instantiation of $\text{gnd}(P)$. We prove in Section 5.3 that there will always be a finite ground instantiation at each step. $\text{Solve}(P_g, t)$ takes as input a finite ground program P_g and an optimality bound t . If t is non-negative then Solve initiates a t -bounded search for an optimal answer set. Solve returns an answer set and its associated optimality or **unsatisfiable** if P_g is unsatisfiable or the optimal answer sets of P_g have an optimality $> t$. If t is negative then Solve initiates an unbounded search for an optimal answer set. Solve returns an answer set and its associated optimality, or **unsatisfiable** where P_g is unsatisfiable. $\mathcal{N}_S$ is used to record the $\text{need}/2$ atoms found in an answer set. $\mathcal{N}$ records the cumulative set of $\text{need}/2$ atoms found and f is an integer used to create a unique fresh parameter identifier(PID) for each of the parameters introduced. P_g stores the incrementally ground program. We use the notation P_g^i , S^i and o^i to denote the

values of P_g , S and o after i iterations of the main loop (lines 5-13) have been completed.

Initially, the program $\mathcal{K}^\tau \cup \text{sig}(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau \cup P_b$ is grounded (line 3) obtaining P_g^0 and solved (at line 5) obtaining some answer set (S^0, o^0) or **unsatisfiable**. If the program is unsatisfiable the algorithm terminates returning **unsatisfiable**. Otherwise, the *need/2* atoms are retrieved from the answer set (line 7). If no new *need/2* atoms are found (line 8) the answer set and its optimality are returned. If *need/2* atoms are found, each $\text{need}(X, o\text{Some}(R, C))$ atom instance in S^0 indicates that a parameter is needed to serve as a witness to the individual X for the concept represented as $o\text{Some}(R, C)$. There may be more than one new need atom found and the inner for loop (lines 9-12) iterates over $|N_S \setminus \mathcal{N}|$ new need atoms cumulatively extending the program P_g^0 with instances of the program $\text{Ground}(P_{cum}(f, X, o\text{Some}(R, C)))$ (line 10). Each time f is incremented (line 11) to ensure that a unique PID is created for each parameter. P_g^1 denotes the ground program at completion of this inner loop (line 12). The resultant program is solved again, either returning (S^1, o^1) or **unsatisfiable**. Subsequent iterations are similarly carried out and terminate either when no further parameters are needed or the solver returns **unsatisfiable**. The final answer set generated is called an *optimal answer set* of the program $ASP(\mathcal{K})$.

The rules in P_{cum}

The program P_{cum} implements the rules required to introduce and expand concepts for a parameter. It begins with the *#program* directive which instructs the grounder to postpone grounding the subsequent rules until requested under script control. The $\text{need}(X, o\text{Some}(R, C))$ atom instances and associated unique PIDs are used to assign the three arguments in P_{cum} . Each triple of arguments $_p$ (a PID), $_i$ (the witnessed individual name X), and $_c$ (a term $o\text{Some}(R, C)$ representing the quantified concept), leads to a set of ground instantiation of P_{cum} .

$$\text{\#program cum}(_p, _i, _c) \tag{c1}$$

$$\text{used}(_p) \leftarrow \text{need}(_i, _c). \tag{c2}$$

$$\text{cea}(_p, C, _i) \leftarrow \text{used}(_p), o\text{Some}(R, C) = _c. \tag{c3}$$

$$cea(_p, C, _i) \leftarrow used(_p), isa(_i, oAll(R, C)). \quad (c4)$$

$$dnb(Y, _p) \leftarrow used(_p), used(Y), Y! = _p, cea(_p, C, _i), not isa(Y, C). \quad (c5)$$

$$b(_p) \leftarrow used(_p), used(Y), Y! = _p, not dnb(Y, _p). \quad (c6)$$

$$hasa(_i, R, _p) \leftarrow used(_p), oSome(R, C) = _c. \quad (c7)$$

$$isa(_p, C) \leftarrow used(_p), oSome(R, C) = _c. \quad (c8)$$

$$isa(_p, C) \leftarrow used(_p), isa(_i, oAll(R, C)). \quad (c9)$$

$$isa(_p, thing) \leftarrow used(_p). \quad (c10)$$

Rules (c2-c10) capture the $\rightarrow_{\exists V}$ rule with respect to fresh parameters. Parameters serving as a witness are labelled as used (c2). Each concept C introduced for a parameter $_p$ by the application of the $\rightarrow_{\exists V}$ -rule is recorded in instances of predicate $cea/3$ (c3,c4). Rules (c5-c6) keep track of the blocking mechanism. Atom $dnb(Y, _p)$ states that “ Y does not block $_p$ ”, and atom $b(_p)$ that “ $_p$ is blocked”. Since the grounding calls to P_{cum} are sequential, each used Y was introduced within an earlier grounding step and represents an *older* individual within an expansion. Mutual blocking is prevented by enforcing $Y! = _p$. The parameter is made a witness (c7-c9) and parameters that are used are added to the top concept (c10). Rules (c11-c23) expand used, unblocked parameters and follow similar patterns to rules for named individuals (e1-e13):

$$isa(_p, C) \leftarrow isa(_p, and(C, D)), used(_p), not b(_p). \quad (c11)$$

$$isa(_p, D) \leftarrow isa(_p, and(C, D)), used(_p), not b(_p). \quad (c12)$$

$$isa(_p, C) \mid isa(_p, D) \leftarrow isa(_p, or(C, D)), used(_p), not b(_p). \quad (c13)$$

$$isa(_p, or(@neg(C), D)) \leftarrow ax(sc(C, D), 0), used(_p), not b(_p). \quad (c15)$$

$$isa(_p, or(@neg(C), D)) \mid u(sc(C, D), _p, W) \leftarrow$$

$$ax(sc(C, D), W), W > 0, used(_p), not b(_p). \quad (c16)$$

$$:\sim u(sc(C, D), _p, W), used(_p), not b(_p). \quad [W, sc, C, D, _p] \quad (c17)$$

$$isa(_p, C) \mid isa(_p, @neg(C)) \leftarrow qc(C), used(_p), not b(_p) \quad (c19)$$

$$\leftarrow isa(_p, neg(thing)). \quad (c20)$$

$$\leftarrow isa(-p, C), isa(-p, neg(C)), c(C). \quad (c21)$$

$$need(-p, oSome(R, C)) \leftarrow isa(-p, oSome(R, C)), used(-p), not b(-p). \quad (c23)$$

The omission of labels (c14, c18 and c22) facilitates the comparison of the rule form with rules (e1-e13). Equivalents of (e4, e8 and e12) are not needed for parameters because the $\rightarrow_{\forall}$ rule is subsumed by (c9) as part of the $\rightarrow_{\exists\forall}$ rule; parameters are assigned to $\top$ by (c10); and labelled as used by (c2).

We next show that the implementation correctly implements Algorithm 2.

5.3 Properties of the translation

For the purposes of analysis, we will denote by Π^i the program obtained from $ASP(\mathcal{K})$ with i instances of the cumulative program P_{cum} where $i \geq 0$. Due to the process of incremental grounding of Π^i , $gnd(\Pi^i)$ consists of $gnd(P_{base})$ and $gnd(P_{cum_j})$, $1 \leq j < i$ where each $gnd(P_{cum_j})$ denotes the grounding over the Herbrand Universe of $P_{base} \cup P_{cum_m}$ for $1 \leq m \leq j$. For the avoidance of doubt, Π^i is not ground and is distinct from P_g^i used for a ground program instantiation after i iterations of Algorithm 3 in Section 5.2.

$ASP(\mathcal{K})$ belongs to the class of Finitely Ground programs

We show that given a knowledge base $\mathcal{K}$ the program $ASP(\mathcal{K})$ belongs to the class of Finitely Ground Programs ($\mathcal{FG}$ Programs).

Lemma 5.16 ($ASP(\mathcal{K})$ belongs to $\mathcal{FG}$). *Let $\mathcal{K}$ be an $p\text{-}\mathcal{ALC}$ knowledge base and Π^i be the ASP representation of $\mathcal{K}$ including i cumulative instantiations of P_{cum} . Then there exists a finite ground program $\Pi_g^i \subseteq gnd(\Pi^i)$ such that $AS(\Pi_g^i) = AS(\Pi^i)$ and $AS(\Pi_g^i)$ is a set of finitely many finite answer sets.*

Proof. The proof follows directly by considering the class of Finitely Ground Programs ($\mathcal{FG}$ Programs) defined in [CCIL08]. We show that Π^i is in the class of Finite Domain programs

r	Term (t) : argument (a) in head(r)	atom(s) of body⁺(r)	C
(a1)	$X : isa 1 , C : isa 2 $	$ax(ca(X, C), 0)$	2,2
(a2)	$X : hasa 1 , R : hasa 2 , Y : hasa 3 $	$ax(ra(R, X, Y), 0)$	2,2,2
(a3)	$X : isa 1 , C : isa 2 ,$ $ca(C, X) : u 1 , X : u 2 , W : u 3 $	$ax(ca(C, X), W), 0)$	2,2 2,2,2
(a5)	$X : hasa 1 , R : hasa 2 , Y : hasa 3 ,$ $ra(R, X, Y) : u 1 , X : u 2 , W : u 3 $	$ax(ra(R, X, Y), W), 0)$	2,2,2 2,2,2
(e1)	$X : isa 1 , C : isa 2 $	$isa(X, and(C, D))$	2,2
(e2)	$X : isa 1 , D : isa 2 $	$isa(X, and(C, D))$	2,2
(e3)	$X : isa 1 , C : isa 2 , D : isa 2 $	$isa(X, or(C, D))$	2,2,2
(e5)	$X : isa 1 , or(@neg(C), D)) : isa 2 $	$i(X), ax(sc(C, D), 0)$	2,3
(e6)	$X : isa 1 , or(@neg(C), D)) : isa 2 ,$ $sc(C, D) : u 1 , X : u 2 , W : u 3 $	$i(X), ax(sc(C, D), W)$	2,3 2,2,2
(e8)	$X : isa 1 , thing : isa 2 $	$i(X)$	2,1
(e9)	$X : isa 1 , C : isa 2 , @neg(C) : isa 2 $	$i(X), qc(C)$	2,2,2
(e12)	$X : hwi 1 , oSome(R, C) : hwi 2 $	$isa(X, oSome(R, C))$	2,2
(e13)	$X : need 1 , oSome(R, C) : need 2 $	$isa(X, oSome(R, C))$	2,2
(e14)	$X : used 1 $	$i(X)$	2

Table 5.4: For each rule $r \in P_b$, each term (t) of an argument (a) occurring in head(r) satisfies at least one condition C.

r	Term (t) : argument (a) in head(r)	atom(s) of body⁺(r)	C
(c2)	$\neg p : used 1 $		1
(c3)	$\neg p : cea 1 , C : cea 2 , \neg i : cea 3 $	$oSome(R, C) : \neg c$	1,2,1
(c4)	$\neg p : cea 1 , C : cea 2 , \neg i : cea 3 $	$isa(\neg i, oAll(R, C))$	1,2,1
(c5)	$Y : dnb 1 , \neg p : dnb 2 $	$used(Y)$	2,1
(c6)	$\neg p : b 1 $		1
(c7)	$\neg p : hasa 1 , R : hasa 2 , \neg i : hasa 3 $	$oSome(R, C) : \neg c$	1,2,1
(c8)	$\neg p : isa 1 , C : isa 2 $	$oSome(R, C) : \neg c$	1,2
(c9)	$\neg p : isa 1 , C : isa 2 $	$isa(\neg i, oAll(R, C))$	1,2
(c10)	$\neg p : isa 1 , thing : isa 2 $		1,1
(c11)	$\neg p : isa 1 , C : isa 2 $	$isa(\neg p, and(C, D))$	1,2
(c12)	$\neg p : isa 1 , D : isa 2 $	$isa(\neg p, and(C, D))$	1,2
(c13)	$\neg p : isa 1 , C : isa 2 , D : isa 2 $	$isa(\neg p, or(C, D))$	1,2,2
(c15)	$\neg p : isa 1 , or(@neg(C), D)) : isa 2 $	$ax(sc(C, D), 0)$	1,3
(c16)	$\neg p : isa 1 , or(@neg(C), D)) : isa 2 ,$ $u 1 : sc(C, D), \neg p : u 2 , W : u 3 $	$ax(sc(C, D), W)$	1,3 2,1,2
(c19)	$\neg p : isa 1 , C : isa 2 , @neg(C) : isa 2 $	$qc(C)$	1,1,2
(c23)	$\neg p : need 1 , thing : need 2 $		1,2

Table 5.5: For each rule $r \in P_{cum}$, each term (t) of an argument (a) occurring in head(r) satisfies at least one condition C.

($\mathcal{FD}$) by showing that for every argument in $ARGS(\Pi^i)$ that each term of the argument that occurs in the head of a rule satisfies at least one of the three syntactic conditions of Definition 10 in [CCIL08]. First consider the argument $ax||1||$. The predicate $ax/1$ only occurs in the heads of rules in the facts in $\mathcal{K}^\tau$. The term corresponding to $ax||1||$ is ground (it represents an axiom) and therefore satisfies condition 1. For arguments, $c||1||$, $r||1||$ and $i||1||$, the facts in $sig(\mathcal{K})^\tau$ include atoms of the predicates $c/1$, $r/1$ and $i/1$ with a ground term representing, respectively, a concept name, a role name or an individual name. Each term satisfies condition 1. For argument $qc[1]$, each fact in $\mathcal{F}(\mathcal{K})^\tau$ has a single a ground term representing a concept name and therefore satisfies condition 1. Tables 5.4 and 5.5 show for the rules of programs P_b and P_{cum} (resp.) the arguments in $ARGS(\Pi^i)$ that occur in the head of the rule, the corresponding term, the relevant positive body literal and the condition satisfied for each term. Note that in rules (e5-e6, c15-c16) the variables C and D occur within the term of the argument $isa||2||$ as $or(@neg(C), D)$. C and D also occur as terms of the argument $ax||1||$ within a positive body atom. All definitions of $ax/1$ appear within the facts $\mathcal{K}^\tau$, hence $ax/1$ is not recursive in $isa||2||$ and each argument $isa||2|| = or(@neg(C), D)$ satisfies condition 3.

We therefore conclude by Theorem 7 in [CCIL08] that Π^i is also in the class of Finite Grounding ($\mathcal{FG}$) programs. From its corollary, Corollary 2, Π^i has finitely many finite answer sets and $AS(\Pi^i)$ is computable. Finally, since Π^i is an $\mathcal{FG}$ program, Theorem 1 in [CCIL08] can be applied to conclude the existence of a finite ground program (an intelligent instantiation) $\Pi_g^i \subseteq gnd(\Pi^i)$ having the same answer sets as the program Π^i . $\square$

ASP($\mathcal{K}$) excluding all constraints is locally stratified

The set of constraints in Π^i will be denoted Con^i .

Lemma 5.17 (*ASP($\mathcal{K}$) is locally stratified*). *Let $\mathcal{K}$ be a p- $\mathcal{ALC}$ knowledge base and Π^i be the ASP representation of $\mathcal{K}$ including i cumulative instantiations of P_{cum} . Then $\Pi^i - Con^i$ is locally stratified.*

Proof. The program $\Pi^i - Con^i$ will be locally stratified (Definition 5.9) if its ground rule

instances can be partitioned into a finite set of strata $P_1, \dots, P_s$ such that every ground rule instance r_g is assigned to stratum k where: (i) every definition of each atom in $head(r_g)$ is in stratum k , (ii) every definition of each atom in $body^+(r_g)$ is in stratum $j \leq k$ and (iii) every definition of each atom in $body^-(r_g)$ is in stratum $j < k$.

The ground rule instances can be assigned to strata as follows. The facts from $sig(\mathcal{K}^\tau)$ and $\mathcal{Q}(\mathcal{K})^\tau$ involving $ax/2$, i/a , $c/1$, $r/1$ and $qc/1$ are in stratum 0. Ground instances of the rules (a1-a3), (a5), (e1-e6), (e8-e9), (e12) and (e14) from P_b are assigned to stratum 1; and ground instances of (e13), which includes the only negative literal in P_b (*not* $hw(X, oSome(R, C))$), are assigned to stratum 2. Ground instances in each program P_{cum_j} are assigned as follows. Stratum $3j$ consists of groundings of (c2-c5), stratum $3j + 1$ has the groundings of (c6) and stratum $3j + 2$ has the ground instances of (c7-c16), (c19) and (c23).

Note that every atom defined in a rule instance of P_{cum_j} includes the term $_p$, where $_p$ denotes the unique constant representing the fresh parameter introduced at this step. Each term $_p$ is unique and therefore not part of the Herbrand universe of any earlier grounding step. We conclude that all atoms defined by rule instances of P_{cum_j} are defined in strata $3j$, $3j + 1$ and $3j + 2$, and therefore that the rule instances in these strata satisfy condition (i). The terms $_i$ and $_c$ in atoms are constant symbols representing an individual being witnessed and an existentially quantified concept (respectively). $_i$ is either a named individual or an existing (older) parameter. Hence, every body literal that includes the term $_i$ is defined in a stratum $l < 3j$. In rule (c6) the positive body atom $used(Y)$, is paired with atom $Y! = _p$ and therefore each $used(Y)$ instance is either defined in a stratum $l < 3j$ or not defined at all. We conclude that the rule instances in strata $3j$, $3j + 1$ and $3j + 2$ satisfy condition (ii). Finally, in the body of rule (c5) the negative literal *not isa*(Y, C) occurs with positive literals $used(Y)$ and $Y! = _p$. By the same argument, instances are defined in a stratum $l < 3j$. We conclude rule instance in strata $3j$, $3j + 1$ and $3j + 2$ satisfy condition (iii). $\square$

ASP($\mathcal{K}$) is head cycle free

In the following, we say a term t is *larger* than term u if u occurs within t .

Lemma 5.18 ($ASP(\mathcal{K})$ is HCF). *Let $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base and Π^i be the ASP representation of $\mathcal{K}$ including i cumulative instantiations of P_{cum} . Then Π^i is head cycle free.*

Proof. The program Π^i will be head cycle free iff the dependency graph of $gnd(\Pi^i)$ does not contain directed cycles that go through two literals that belong to the head of the same rule [BED94].

The proof is by contradiction. Suppose for contradiction there is a ground rule instance $r_g \in gnd(\Pi^i)$ such that there is a directed cycle in the dependency graph of Π^i passing through $h_1 \in head(r_g)$ and $h_2 \in head(r_g)$ where $h_1 \neq h_2$.

The proof goes by cases by considering the disjunctive rules $r \in \Pi^i$:

- The defeasible ABox rules (a3, a5) in P_b . In each such rule $u(T, X, W) \in head(r)$ and $ax(T, W) \in body^+(r)$ where variable X appears within the term T . $ax/2$ atoms only appear in the heads of rules as facts of K^τ . Each instance of $u(T, X, W)$ is dependent on an instance of $ax(T, W)$ and no instances of $ax/2$ are dependent on any other rule instances. Hence, these rule instances are head cycle free.
- The $\rightarrow_Q$ -rule (e9) in P_b . $isa(X, C) \in head(r)$, $isa(X, @neg(C)) \in head(r)$, $i(X) \in body^+(r)$ and $qc(C) \in body^+(r)$. $i/1$ atoms only appear in the heads of rules as facts of $sig(\mathcal{K})^\tau$. $qc/1$ atoms only appear in the heads of rules as facts of $\mathcal{F}(\mathcal{K})^\tau$. Hence, these rule instances are head cycle free.
- The defeasible TBox rule (e6) in P_b . $u(T, X, W) \in head(r)$, $ax(T, W) \in body^+(r)$ and $i(X) \in body^+(r)$. $ax/2$ atoms only appear in the heads of rules as facts of K^τ . $i/1$ atoms only appear in the heads of rules as facts of $sig(\mathcal{K})^\tau$. Hence, instances of (e6) are head cycle free.
- For the $\rightarrow_\square$ -rule implementation (e3). $isa(X, C) \in head(r)$ and $isa(X, D) \in head(r)$. We consider the sub-cases of rules that define $isa/2$ atoms:

- (a1, a3, e5, e6, e8 and e9). The dependencies are on facts in $\mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{F}(\mathcal{K})^\tau$.

- (e3). $isa(X, C)$ and $isa(X, D)$ in the head are dependent on $isa(X, or(C, D))$ in the body, an atom with the larger term $or(C, D)$.
- (e1 and e2). $isa(X, C)$ and $isa(X, D)$ in the head are dependent on $isa(X, and(C, D))$ in the body, an atom with the larger term $and(C, D)$.
- (e4). $isa(X, C)$ in the head is dependent on $isa(X, oAll(R, C))$ in the body, an atom with the larger term $oAll(R, C)$.

The term in each $isa/2$ body atom is always larger than the corresponding term in the head, hence prohibiting the development of a cycle.

- The $\rightarrow_Q$ -rule (c19) for parameter $_p$ in P_{cum} . $isa(_p, C) \in head(r)$, $isa(_p, @neg(C)) \in head(r)$, $qc(C) \in body^+(r)$ and $used(_p) \in body^+(r)$. $used(_p)$ is dependent on $need(_i, _c)$ (c2) which in turn is dependent on $isa/2$ atoms for some older individual $_i$. Such atoms are not dependent on any atoms in the P_{cum} instantiation of $_p$ because the grounding of rules for individual $_i$ are established in an iteration before $_p$ is introduced. $qc/1$ atoms only appear in the heads of rules as facts of $\mathcal{F}(\mathcal{K})^\tau$. Hence, the term in each $isa/2$ body atom is always larger than the corresponding term in the head, hence prohibiting the development of a cycle.
- The defeasible TBox rule (c16) for parameter $_p$ in P_{cum} . $u(T, _p, W) \in head(r)$, $ax(T, W) \in body^+(r)$ and $used(_p) \in body^+(r)$. $ax/2$ atoms only appear in the heads of rules as facts of K^τ . $used(_p)$ is dependent on $need(_i, _c)$ (c2) which in turn is dependent on $isa/2$ atoms for some older individual $_i$. Hence instances of (c16) are head cycle free.
- For the $\rightarrow_\sqcup$ -rule implementation (c13) for parameter $_p$ in P_{cum} . $isa(_p, C) \in head(r)$ and $isa(_p, D) \in head(r)$. We consider the sub-cases of rules that define $isa/2$ atoms
 - (c7, c8). The dependencies are on $used(_p)$.
 - (c10, c16 and c19). The dependencies are on $used(_p)$ and facts in $\mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{F}(\mathcal{K})^\tau$.

- (c13). $isa(_p, C)$ and $isa(_p, D)$ in the head are dependent on $used(_p)$ and $isa(_p, or(C, D))$, an atom with the larger term $or(C, D)$.
- (c11 and c12). $isa(_p, C)$ and $isa(_p, D)$ in the head are dependent on $used(_p)$ and $isa(_p, and(C, D))$, an atom with the larger term $and(C, D)$.
- (c4). $isa(_p, C)$ in the head is dependent on $used(_p)$ and $isa(_p, oAll(R, C))$, an atom with the larger term $oAll(R, C)$.

Each rule instance is dependent on $used(_p)$, in turn dependent on $need(_i, _c)$ and hence $isa/2$ atoms for some older individual $_i$. The term in each $isa/2$ body atom is always larger than the corresponding term in the head, hence prohibiting the development of a cycle. Hence, these rule instances are cycle free.

□

There exists an iterative construction for each answer set of $ASP(\mathcal{K})$

We show that an answer set of Π^i can be obtained by computing the answer sets of a sequence of positive ground programs that contribute to the final answer set. Our approach generalises the iterative construction used for locally stratified disjunctive programs [Prz88] to disjunctive programs that are ground incrementally and may include integrity and weight constraints.

Lemma 5.19 (Construction for $AS(ASP(\mathcal{K}))$). *Let $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base and Π^i be the ASP representation of $\mathcal{K}$ including i cumulative instantiations of P_{cum} . Let $\mathbb{H}_j$ denote the Herbrand base of rule instances of Π^i from strata 0 to j , $\mathbb{S}_j$ denote the set $S_0 \cup \dots \cup S_j$ and G_j denote the ground rules in $gnd(\Pi^i)$ from stratum j . Let S be a set of atoms. Then S is an answer set of Π^i iff S satisfies $Cons^i$ and $S = \bigcup_{j=0}^{3i+2} S_j$ where S_0 is the unique answer set of G_0 and for $1 \leq j \leq 3i+2$, S_j is an answer set of the positive program $e_{\mathbb{H}_{j-1}}(G_j, \mathbb{S}_{j-1})$.*

Proof. Let $\pi_0 = gnd(\Pi^i)$. By Lemma 5.17 the rules excluding the constraints in π_0 are locally stratified. We first show that we can repeatedly split and simplify the program starting from π_0 . Clearly, $\mathbb{H}_0$ splits π_0 into $bot_{\mathbb{H}_0}(\pi_0)$, the rule instances from stratum 0, and $top_{\mathbb{H}_0}(\pi_0)$, all

remaining rule instances including the constraints (the constraints have no head atom and therefore appear in the top). Let S_0 be an answer set of $bot_{\mathbb{H}_0}(\pi_0)$ and for $1 \leq j \leq 3i + 2$ we split π_{j-1} using $\mathbb{H}_{j-1}$ such that:

- the bottom is $bot_{\mathbb{H}_{j-1}}(\pi_{j-1})$ and the top is $top_{\mathbb{H}_{j-1}}(\pi_{j-1})$;
- and define $\pi_j = e_{\mathbb{H}_{j-1}}(top_{\mathbb{H}_{j-1}}(\pi_{j-1}), S_{j-1})$, the simplified top, and S_j is an answer set of $bot_{\mathbb{H}_j}(\pi_j)$.

By the stratification, the rules in the bottom $bot_{\mathbb{H}_j}(\pi_j)$ are the simplified rules from stratum j . They are not dependent on atoms in higher strata and all negative literals are defined in strata below and are simplified out. $bot_{\mathbb{H}_j}(\pi_j)$ is a positive program. Applying the Splitting Set theorem S is an answer set π_0 iff $S = S_0 \cup (S_1 \cup (S_2 \cup \dots (S_{3i+2}) \dots))$ and S satisfies the simplified constraints from $Cons^i$. Let G_j denote the rule instances at stratum j and G_{3j+3} denote the rule instances from the constraints in $Cons^i$. The instances used to compute S_j are:

- The facts G_0 where $j = 0$ and;
- The rule instances of $bot_{\mathbb{H}_j}(e_{\mathbb{H}_{j-1}}(top_{\mathbb{H}_{j-1}}(\dots e_{\mathbb{H}_0}(top_{\mathbb{H}_0}(\Pi^i), S_0) \dots), S_{j-1}))$ where $1 \leq j \leq 3i + 2$. From Definitions 5.11 and 5.10 the rules at each stratum j can be simplified in a single step using $e_{\mathbb{H}_{j-1}}(G_j, S_{j-1})$ where $1 \leq j \leq 3i + 2$
- At the final split on $\mathbb{H}_{3j+2}$, S must satisfy the simplified constraints $e_{\mathbb{H}_{3i+2}}(G_{3i+3}, S)$.

□

The iterative construction of an answer set of Π^i using the splitting set theorem is illustrated in Figure 5.2.

The representation of a branch in an answer set

To show the correspondence between the answer sets obtained by Algorithm 3 and the branches developed by Algorithm 2 we first need to formalise how a branch will be represented in ASP.

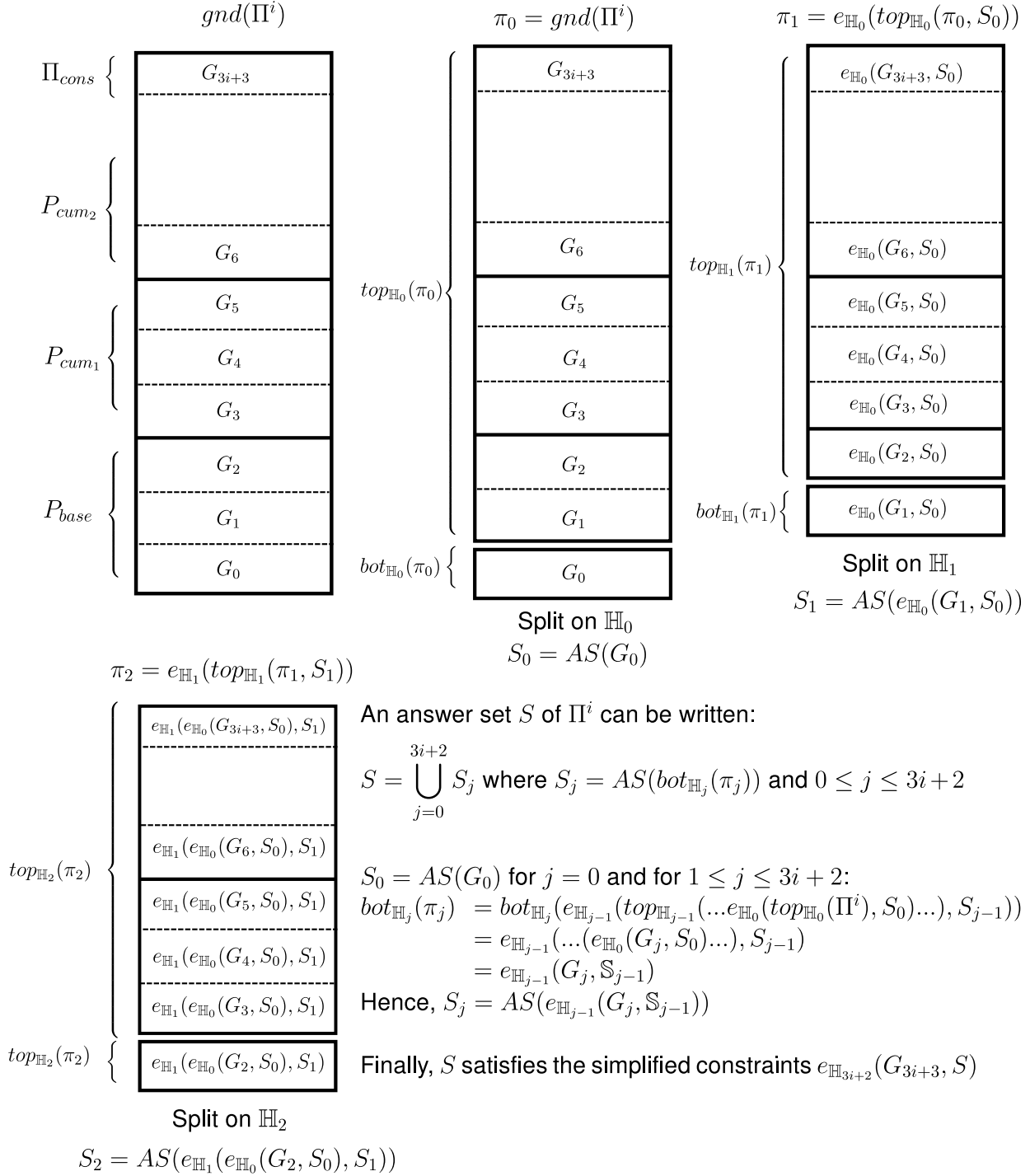


Figure 5.2: An illustration of the iterative construction of an answer set of Π^i through repeated applications of the Splitting Set Theorem.

Given a branch b generated for the knowledge base $\mathcal{K}$ Definition 5.20 formalises b^τ , the set of ASP atoms that represent b . The atoms in b^τ capture the branch axioms and the omitted set, together with auxiliary information about the individuals used in the branch. This includes identifying which individuals need witnesses, the parameters that are used as a witnesses and

blocking information about each individual.

Definition 5.20 (Branch in ASP). *Let $\mathcal{K}$ be a knowledge base with signature $\langle N_I, N_C, N_R \rangle$ and $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ be a branch returned by Algorithm 2 for $\mathcal{K}$ using a set of fresh parameters $P(b)$. Let*

$$\mathcal{A}_b^\tau = \{ isa(x^\tau, C^\tau) \mid C(x) \in \mathcal{A}_b \text{ or } (C = \top \text{ and } (x \in N_I \cup P(b))) \} \cup$$

$$\{ hasa(x^\tau, C^\tau, y^\tau) \mid R(x, y) \in \mathcal{A}_b \}$$

$$\mathcal{O}_b^\tau = \{ u(Z^\tau, x^\tau, w) \mid \langle Z^{[w]}, x \rangle \in \mathcal{O}_b \}$$

$$\mathcal{N}_{used}^\tau = \{ used(x^\tau) \mid x \in N_I \}$$

$$\mathcal{N}_{hw}^\tau = \{ hw(x^\tau, oSome(R^\tau, C^\tau)) \mid (\exists R.C)(x) \in \mathcal{A}_b \text{ and } x \in N_I \text{ and}$$

$$\text{there exists } y \in N_I \text{ s.t. } R(x, y) \in \mathcal{A}_b \text{ and } C(y) \in \mathcal{A}_b \}$$

$$\mathcal{N}_{need}^\tau = \{ need(x^\tau, oSome(R^\tau, C^\tau)) \mid (\exists R.C)(x) \in \mathcal{A}_b \text{ and } x \in N_I \text{ and}$$

$$\text{there exists no } y \in N_I \text{ s.t. } R(x, y) \in \mathcal{A}_b \text{ and } C(y) \in \mathcal{A}_b \}$$

$$\mathcal{W}_{used}^\tau = \{ used(x^\tau) \mid x \in P(b) \}$$

$$\mathcal{W}_{cea}^\tau = \{ cea(x^\tau, C^\tau) \mid x \in P(b) \text{ and } R(i, x) \in \mathcal{A}_b \text{ and } ((\exists R.C)(i) \in \mathcal{A}_b \text{ or } (\forall R.C)(i) \in \mathcal{A}_b) \}$$

$$\mathcal{W}_{dnb}^\tau = \{ dnb(y^\tau, p^\tau) \mid p \in P(b), y \in N_I \cup P(b), y \text{ is older than } p \text{ and}$$

$$\text{there exists a concept } C \text{ where } C(p) \in \mathcal{A}_b \text{ and } C(y) \notin \mathcal{A}_b \}$$

$$\mathcal{W}_b^\tau = \{ b(p^\tau) \mid p \in P(b) \text{ and } p \text{ is blocked} \}$$

$$\mathcal{W}_{need}^\tau = \{ need(x^\tau, oSome(R^\tau, C^\tau)) \mid (\exists R.C)(x) \in \mathcal{A}_b \text{ and } x^\tau \in P(B) \}$$

Then $b^\tau = \mathcal{K}^\tau \cup Sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau \cup \mathcal{A}_b^\tau \cup \mathcal{O}_b^\tau \cup \mathcal{N}_{used}^\tau \cup \mathcal{N}_{hw}^\tau \cup \mathcal{N}_{need}^\tau \cup \mathcal{W}_{used}^\tau \cup \mathcal{W}_{cea}^\tau \cup \mathcal{W}_{dnb}^\tau \cup \mathcal{W}_b^\tau \cup \mathcal{W}_{need}^\tau$ is the representation of b in ASP.

The elements of $\mathcal{A}_b^\tau$ and $\mathcal{O}_b^\tau$ capture the representations of the axioms and the omissions in the branch. The elements of $\mathcal{N}_{used}^\tau$ capture the set of named individuals in the branch. The elements of $\mathcal{N}_{hw}^\tau$ capture the set of named individuals in the branch that are required to satisfy an existential quantified concept where a named individual serves as the witness in the branch. The elements of $\mathcal{N}_{need}^\tau$ capture the set of named individuals in the branch that are required to

satisfy an existential quantified concept where a parameter serves as the witness in the branch. The elements of $\mathcal{W}_{used}^\tau$ capture the parameters in the branch. The elements of $\mathcal{W}_{cea}^\tau$ capture the set of concepts that are witnessed by a parameter. The elements of $\mathcal{W}_{dnb}^\tau$ capture for each parameter in the branch the (older) individuals that do not block it. The elements of $\mathcal{W}_b^\tau$ capture the parameters that are blocked in the branch. The elements of $\mathcal{W}_{need}^\tau$ capture the set of parameters in the branch that are required to satisfy an existential quantified concept where a parameter serves as the witness in the branch.

Definition 5.21 ($\subseteq$ -minimality). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be a p - $\mathcal{ALC}$ knowledge base and b be an n -minimal open branch returned by Algorithm 2 for $\mathcal{K}$. $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ is said to be an n - $\subseteq$ -minimal branch if there is no other branch $\bar{b} = \langle \mathcal{A}_{\bar{b}}, \mathcal{O}_{\bar{b}} \rangle$ s.t. $\mathcal{A}_{\bar{b}} \subset \mathcal{A}_b$.*

Observation 8. *Given a branch b developed for $\mathcal{K}$ with signature $\langle N_I, N_C, N_R \rangle$, the $isa/2$ atoms in b^τ are of the form $isa(x^\tau, C^\tau)$ and the $hasa/2$ atoms in b^τ are of the form $hasa(x^\tau, R^\tau, y^\tau)$, where $x, y \in N_I \cup P(b)$, $R \in N_R$ and $C \in \mathcal{F}_A(\mathcal{K})$.*

Observation 9. *The iterative grounding and solving operation of Algorithm 3 leads to the instantiation of each P_{cum} as $need/2$ atoms are detected within answer sets. This leads to the possibility of parameters that are introduced but are not “used” in the final answer set obtained. Such parameters contribute no atoms to the final answer set and the order of instantiation of parameters is not important to the proof of completeness in Theorem 4.30.*

The correspondence between minimal branches and optimal answer sets is expressed by Theorems 5.22 and 5.24 and is given next.

Theorem 5.22 (Correspondence forward). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be an n -inconsistent p - $\mathcal{ALC}$ knowledge base, $\mathcal{B}$ be the set of $\subseteq$ - m -distant branches obtained from Algorithm 2 applied to $\mathcal{K}$. Then for each m -distant branch $b \in \mathcal{B}$ there exists an i such that there is a program Π^i , based on the ASP representation of $\mathcal{K}$, such that b^τ is an answer set of Π^i with optimality m .*

Proof. Recall that Π^i denotes the program obtained from $ASP(\mathcal{K})$ with i instances of the cumulative program P_{cum} .

We choose i to be the number of subsequences in $b-1$ where $Subs_\sigma$ denotes subsequence σ of b , where $0 \leq \sigma \leq i$. Let $\mathbb{A}_\sigma$ and $\mathbb{O}_\sigma$ denote the assertions and the omissions (respectively) added to the branch in $Subs_\sigma$. Let $p_\sigma, \sigma > 0$ be the parameter introduced as a witness in subsequence $Subs_\sigma$.

We can partition b^τ as follows:

$$\begin{aligned}
S_0 &= \mathcal{K}^\tau \cup Sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau \\
S_1 &= \mathbb{A}_0^\tau \cup \mathbb{O}_0^\tau \cup \mathcal{N}_{used}^\tau \cup \mathcal{N}_{hw}^\tau \\
S_2 &= \mathcal{N}_{need}^\tau \\
S_{3\sigma} &= \{used(p_\sigma^\tau) | used(p_\sigma^\tau) \in \mathcal{W}_{used}^\tau\} \cup \{cea(p_\sigma^\tau, C^\tau) | cea(p_\sigma^\tau, C^\tau) \in \mathcal{W}_{cea}^\tau\} \cup \\
&\quad \{dnb(x^\tau, p_\sigma^\tau) | dnb(x^\tau, p_\sigma^\tau) \in \mathcal{W}_{dnb}^\tau\} \\
S_{3\sigma+1} &= \{b(p_\sigma^\tau) | b(p_\sigma^\tau) \in \mathcal{W}_b^\tau\} \\
S_{3\sigma+2} &= \mathbb{A}_\sigma^\tau \cup \mathbb{O}_\sigma^\tau \cup \{need(p_\sigma^\tau, oSome(R^\tau, C^\tau)) | need(p_\sigma^\tau, oSome(R^\tau, C^\tau)) \in \mathcal{W}_{need}^\tau\}
\end{aligned}$$

Let $\mathbb{H}_j$ denote the Herbrand base of rule instances of Π^i from strata 0 to j , where $0 \leq j \leq 3i+2$, $\mathbb{S}_j$ denote the set $S_0 \cup \dots \cup S_j$ and $\mathcal{A}'_b$ denote the axioms of $\mathcal{A}_b$ not involving blocked parameters. Let P_0 be the facts in stratum 0 of Π^i and let $P_j = e_{\mathbb{H}_{j-1}}(G_j, \mathbb{S}_{j-1})$ where G_j are the ground rule instances of Π^i from stratum j . By Lemma 5.19 b^τ is an answer set of Π^i iff b^τ satisfies $Cons^i$ and $b^\tau = \bigcup_{j=0}^{3\sigma+2} S_j$ where S_j is an answer set of P_j .

We show that for each, $j, 0 \leq j \leq 3\sigma+2$, S_0 is a model of P_j . The proof goes by cases:

- P_0 is the set of the facts $\mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau$ in stratum 0 of Π^i . By the partition, S_0 contains $\mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau$. S_0 is the unique answer of P_0 .
- $P_1 = e_{\mathbb{H}_0}(G_1, S_0)$ where G_1 denotes the ground rule instances from stratum 1. We show first, that S_1 is a model of P_1 . Assume for contradiction, that S_1 is not a model of P_1 . If S_1 is not a model of P_1 then there exists some rule instance $r_g \in P_1$ not satisfied by S_1 . We show this leads to a contradiction for every type of r_g :

- (a1) By Lemma 5.17, instances of $ax/1$ are defined in stratum 0. By Definition 5.15, $ax(ca(C^\tau, x^\tau), 0) \in S_0 \leftrightarrow C(x) \in \mathcal{A}$. Hence r_g , derived from (a1) are facts

of the form $isa(X^\tau, C^\tau)$ where $ax(ca(C^\tau, X^\tau), 0) \in S_0$. If r_g is not satisfied by S_1 then $isa(X^\tau, C^\tau) \notin S_1$. By Proposition 4.23(1), $C(x) \in \mathcal{A}'_b$. By Lemma 4.10 and Observation 1, $C(x) \in \mathbb{A}_0$. By the partition, $C(x) \in S_1$ a contradiction.

- (a2) is similar to (a1).
- (a3). By Lemma 5.17, instances of $ax/1$ are defined in stratum 0. By Definition 5.15, for all $w > 0$, $ax(ca(C^\tau, x^\tau), w) \in S_0 \leftrightarrow C(x)^{[w]} \in \mathcal{A}_d$. Hence r_g , derived from (a3) are rules of the form $isa(X^\tau, C^\tau) | u(ca(C^\tau, x^\tau), x^\tau, w)$. where $C(x)^{[w]} \in \mathcal{A}_d$. If r_g is not satisfied by S_1 then $isa(X^\tau, C^\tau) \notin S_1$ and $u(ca(C^\tau, x^\tau), x^\tau, w) \notin S_1$. By Proposition 4.23(2), $C(x)^{[w]} \in \mathcal{A}'_b$ or $\langle C(x)^{[w]}, x \rangle \in \mathcal{O}_b$. By Lemma 4.10 and Observation 1, $C(x)^{[w]} \in \mathbb{A}_0$ or $\langle C(x)^{[w]}, x \rangle \in \mathbb{O}_0$. By the partition, $isa(X^\tau, C^\tau) \in S_1$ or $u(ca(C^\tau, x^\tau), x^\tau, w) \in S_1$, a contradiction.
- (a5) is similar to (a3).
- (e1). By Observation 8, if r_g is not satisfied by S_1 then $isa(x^\tau, C^\tau) \notin S_1$ and $isa(x^\tau, and(C^\tau, D^\tau)) \in S_1$. The partition assigns $\mathbb{A}_0^\tau$ to S_1 . By Definition 5.20, $(C \sqcap D)(x) \in \mathbb{A}_0$. By Proposition 4.23(5) $\mathcal{A}'_b$ is boolean downward saturated and $C(x) \in \mathcal{A}'_b$. By Lemma 4.10 and Observation 1, $C(x) \in \mathbb{A}_0$. By the partition, $isa(x^\tau, C^\tau) \in S_1$, a contradiction.
- (e2) and (e3) are similar to (e1).
- (e4). By Observation 8, if r_g is not satisfied by S_1 then $isa(y^\tau, C^\tau) \notin S_1$ where $isa(x^\tau, oAll(R^\tau, C^\tau)) \in S_1$ and $hasa(x^\tau, R^\tau, y^\tau) \in S_1$. The partition assigns $\mathbb{A}_0^\tau$ to S_1 . By Definition 5.20, $(\forall R.C)(x) \in \mathbb{A}_0$ and $R(x, y) \in \mathbb{A}_0$. By Proposition 4.23(7), $\mathcal{A}'_b$ is quantifier saturated and $C(y) \in \mathcal{A}'_b$. By Lemma 4.10 and Observation 1, $C(y) \in \mathbb{A}_0$. By the partition, $isa(y^\tau, C^\tau) \in S_1$, a contradiction.
- (e5). By Lemma 5.17, instances of $ax/1$ and $i/1$ are defined in stratum 0. By Definition 5.15, $ax(sc(C^\tau, D^\tau), 0) \in S_0 \leftrightarrow C \sqsubseteq D \in \mathcal{T}$ and $i(x^\tau) \in S_0 \leftrightarrow x \in N_I$. Hence r_g , derived from (e5), are facts of the form $isa(x^\tau, or((\dot{\neg}C)^\tau, D^\tau))$ where $ax(sc(C^\tau, D^\tau), 0) \in S_0$. If r_g is not satisfied by S_1 then $isa(x^\tau, or((\dot{\neg}C)^\tau, D^\tau)) \notin S_1$. By Proposition 4.23(3) $\mathcal{A}'_b$ is $\mathcal{T}$ -saturated and $(\dot{\neg}C \sqcup D)(x) \in \mathcal{A}'_b$. By Lemma 4.10

- and Observation 1, $(\neg C \sqcup D)(x) \in \mathbb{A}_0$. By the partition, $isa(x^\tau, or((\neg C)^\tau, D^\tau)) \in S_1$, a contradiction.
- (e6). By Lemma 5.17, instances of $ax/1$ and $i/1$ are defined in stratum 0. By Definition 5.15, for $w > 0$, $ax(sc(C^\tau, D^\tau), w) \in S_0 \leftrightarrow C \sqsubseteq D \in \mathcal{T}_d$ and $i(x^\tau) \in S_0 \leftrightarrow x \in N_I$. Hence r_g , derived from (e6), are rules of the form $isa(x^\tau, or((\neg C)^\tau, D^\tau)) | u(sc(C^\tau, D^\tau), x^\tau, w)$. where $ax(sc(C^\tau, D^\tau), w) \in S_0$. If r_g is not satisfied by S_1 then $isa(x^\tau, or((\neg C)^\tau, D^\tau)) \notin S_1$ and $u(sc(C^\tau, D^\tau), x^\tau, w) \notin S_1$. By Proposition 4.23(4) $\mathcal{A}'_b$ is $\mathcal{T}_d$ -saturated and $(\neg C \sqcup D)(x) \in \mathcal{A}'_b$ or $\langle C \sqsubseteq D, x \rangle \in \mathcal{O}_b$. By Lemma 4.10 and Observation 1, $(\neg C \sqcup D)(x) \in \mathbb{A}_0$ or $\langle C \sqsubseteq D, x \rangle \in \mathbb{O}_0$. By the partition, $isa(x^\tau, or((\neg C)^\tau, D^\tau)) \in S_1$ or $u(sc(C^\tau, D^\tau), x^\tau, w) \in S_1$, a contradiction.
 - (e8). By Lemma 5.17, instances of $i/1$ are defined in stratum 0. By Definition 5.15, $i(x^\tau) \in S_0 \leftrightarrow x \in N_I$. Hence r_g , derived from (e8), are facts of the form $isa(x^\tau, thing)$. If r_g is not satisfied by S_1 then $isa(x^\tau, thing) \notin S_1$. By Definition 5.15, if $x \in N_I$ then $isa(x^\tau, thing) \in \mathcal{A}_b$. By Lemma 4.10 and Observation 1, $isa(x^\tau, thing) \in \mathbb{A}_0$. By the partition, $isa(x^\tau, thing) \in S_1$, a contradiction.
 - (e9). By Lemma 5.17, instances of $i/1$ and $qc/1$ are defined in stratum 0. By Definition 5.15, $i(x^\tau) \in S_0 \leftrightarrow x \in N_I$ and $qc(C^\tau) \in S_0 \leftrightarrow C \in \mathcal{F}(\mathcal{K})$. Hence r_g , derived from (e9), are rules of the form $isa(x^\tau, C^\tau) | isa(x^\tau, (\neg C)^\tau)$. If r_g is not satisfied by S_1 then $isa(x^\tau, C^\tau) \notin S_1$ and $isa(x^\tau, (\neg C)^\tau) \notin S_1$. By Proposition 4.23(6) $\mathcal{A}'_b$ is QC-split w.r.t. $\mathcal{K}$ and $C(x) \in \mathcal{A}'_b$ or $\neg C(x) \in \mathcal{A}'_b$. By Lemma 4.10 and Observation 1, $C(x) \in \mathbb{A}_0$ or $\neg C(x) \in \mathbb{A}_0$. By the partition, $isa(x^\tau, C^\tau) \in S_1$ or $isa(x^\tau, neg(C^\tau)) \in S_1$, a contradiction.
 - (e12). By Observation 8, if r_g is not satisfied by S_1 then $hw(x^\tau, oSome(R^\tau, C^\tau)) \notin S_1$, $isa(y^\tau, oSome(R^\tau, C^\tau)) \in S_1$, $hasa(x^\tau, R^\tau, y^\tau) \in S_1$, and $isa(y^\tau, C^\tau) \in S_1$. The partition assigns $\mathbb{A}_0^\tau$ to S_1 . By Definition 5.20, $(\exists R.C)(x) \in \mathbb{A}_0$, $R(x, y) \in \mathbb{A}_0$ and $C(y) \in \mathbb{A}_0$. By Definition 5.20, $hw(x^\tau, R^\tau, y^\tau) \in \mathcal{N}_{hw}$. By the partition, $hw(x^\tau, R^\tau, y^\tau) \in S_1$, a contradiction.
 - (e14). Similar to (e8).

We conclude S_1 is a model of P_1 .

The simplified program P_1 is a positive disjunctive program. Hence, to show that S_1 is an answer set of P_1 it is sufficient to show that S_1 is also a $\subseteq$ minimal model of P_1 . Assume for contradiction that there is an atom a in S_1 such that $S'_1 = S_1 \setminus \{a\}$ is a model of P_1 . The proof goes by cases on the atoms a assigned to S_1 by the partition:

- $a \in \mathbb{A}_0^\tau$. By assumption, a represents an ABox axiom introduced to b in subsequence 0 by a step in Algorithm 2. We consider each step that introduces ABox axioms in subsequence 0.
 - * $isa(x^\tau, C^\tau)$ where $C(x) \in \mathcal{A}$ (branch initialisation). By Lemma 5.17, instances of $ax/1$ are defined in stratum 0. By Definition 5.15, $ax(ca(C^\tau, x^\tau), 0) \in S_0 \leftrightarrow C(x) \in \mathcal{A}$. Hence, the simplified rule instance of (a1) in P_1 , $isa(X^\tau, C^\tau)$, is not satisfied by S'_1 .
 - * $hasa(x^\tau, R^\tau, y^\tau)$ where $R(x, y) \in \mathcal{A}$ (branch initialisation). Similar to $isa(x^\tau, C^\tau) \in \mathcal{A}$ where $C(x) \in \mathcal{A}$ (branch initialisation).
 - * $isa(x^\tau, C^\tau)$ where $C(x)^{[w]} \in \mathcal{A}_d$ is not omitted (branch initialisation). By Proposition 4.23(2) $\mathcal{O}_b$ records the omissions of $\mathcal{A}_d$ from $\mathcal{A}_b$. Hence $u(ca(C^\tau, x^\tau), x^\tau, w) \notin \mathcal{O}_b$. By the partition, $u(ca(C^\tau, x^\tau), x^\tau, w) \notin S'_1$. By Lemma 5.17, instances of $ax/1$ are defined in stratum 0. By Definition 5.15, for all $w > 0$, $ax(ca(C^\tau, x^\tau), w) \in S_0 \leftrightarrow C(x)^{[w]} \in \mathcal{A}_d$. Hence, the simplified rule instance of (a3) in P_1 , $isa(X^\tau, C^\tau) | u(ca(C^\tau, x^\tau), x^\tau, w)$, is not satisfied by S'_1 .
 - * $hasa(x^\tau, R^\tau, y^\tau)$ where $R(x, y)^{[w]} \in \mathcal{A}_d$ is not omitted (branch initialisation). Similar to $isa(x^\tau, C^\tau)$ where $C(x)^{[w]} \in \mathcal{A}_d$ is not omitted (branch initialisation).
 - * $isa(x^\tau, C^\tau)$ introduced by the $\rightarrow_{\sqcap}$ -rule. By Proposition 4.23(5) $\mathcal{A}'_b$ is boolean downward saturated. There are two subcases:
 - $(C \sqcap D) \in \mathcal{A}'_b$. By Lemma 4.10 and Observation 1, $(C \sqcap D) \in \mathbb{A}_0$. By the partition, $and(C^\tau, D^\tau) \in S'_1$. Hence, the simplified rule instance of (e1) in P_1 , $isa(x^\tau, C^\tau) \leftarrow isa(x^\tau, and(C^\tau, D^\tau))$ is not satisfied for S'_1 .
 - $(D \sqcap C) \in \mathcal{A}'_b$. By Lemma 4.10 and Observation 1, $(D \sqcap C) \in \mathbb{A}_0$. By the

- partition, $and(D^\tau, C^\tau) \in S'_1$. Hence, the simplified rule instance of (e2) in P_1 , $isa(x^\tau, C^\tau) \leftarrow isa(x^\tau, and(D^\tau, C^\tau))$ is not satisfied for S'_1 .
- * For $isa(x^\tau, C^\tau)$ where $C(x)$ was introduced by the $\rightarrow_\sqcup$, $\rightarrow_\tau$, and $\rightarrow_Q$ -rules follow a similar patterns to the $\rightarrow_\sqcap$ rule.
 - * $isa(x^\tau, or((\dot{\neg}C)^\tau, D^\tau))$ where $(\dot{\neg}C \sqcup D)(x)$ was introduced for $C \sqsubseteq D \in \mathcal{T}$ by the $\rightarrow_{\mathcal{T}_d}$ -rule. By Proposition 4.23(4) $\mathcal{A}'_b$ is $\mathcal{T}_d$ -saturated. Hence $\langle C \sqsubseteq D, x \rangle \notin \mathcal{O}_b$. By the partition $u(sc(C^\tau, D^\tau), x^\tau, w) \notin S'_1$. By Lemma 5.17, instances of $ax/1$ and $i/1$ are defined in stratum 0. By Definition 5.15, for $w > 0$, $ax(sc(C^\tau, D^\tau), w) \in S_0 \leftrightarrow C \sqsubseteq D \in \mathcal{T}_d$ and $i(x^\tau) \in S_0 \leftrightarrow x \in N_I$. Hence, the simplified rule instance of (e6) in P_1 , $isa(x^\tau, or((\dot{\neg}C)^\tau, D^\tau)) | u(sc(C^\tau, D^\tau), x^\tau, w)$ is not satisfied for S'_1 .
 - * $isa(x^\tau, thing)$, required by Definition 5.20 for all $x \in N_I$. By Lemma 5.17, instances of $i/1$ are defined in stratum 0. By Definition 5.15, $i(x^\tau) \in S_0 \leftrightarrow x \in N_I$. Hence, the simplified rule instance of (e8) in P_1 , $isa(x^\tau, thing)$ is not satisfied by S'_1 .
- $a \in \mathbb{O}_0^\tau$. We consider all cases that introduce a to $\mathbb{O}_0$:
- * $u(ca(C^\tau, x^\tau), x^\tau, w)$ where $C(x)^{[w]} \in \mathcal{A}_d$ is omitted at step 0. By Proposition 4.23(2) $\mathcal{O}_b$ records the omissions of $\mathcal{A}_d$ from $\mathcal{A}_b$. Hence $isa(x^\tau, C^\tau) \notin \mathcal{O}_b$. By the partition, $isa(x^\tau, C^\tau) \notin S'_1$. By Lemma 5.17, instances of $ax/1$ are defined in stratum 0. By Definition 5.15, for all $w > 0$, $ax(ca(C^\tau, x^\tau), w) \in S_0 \leftrightarrow C(x)^{[w]} \in \mathcal{A}_d$. Hence, the simplified rule instance of (a3) in P_1 , $isa(X^\tau, C^\tau) | u(ca(C^\tau, x^\tau), x^\tau, w)$ is not satisfied by S'_1 .
 - * $u(ra(R^\tau, x^\tau, y^\tau), x^\tau, w)$ where $R(x, y)^{[w]} \in \mathcal{A}_d$ is omitted at step 0. Similar to $u(ca(C^\tau, x^\tau), x^\tau, w)$ where $C(x)^{[w]} \in \mathcal{A}_d$ is omitted at step 0.
 - * $u(sc(C^\tau, D^\tau), x^\tau, w)$ where $\langle C \sqsubseteq D, x \rangle$ was introduced for $C \sqsubseteq D \in \mathcal{T}$ by the $\rightarrow_{\mathcal{T}_d}$ -rule. By Proposition 4.23(4) $\mathcal{A}'_b$ is $\mathcal{T}_d$ -saturated. Hence $isa(x^\tau, or((\dot{\neg}C)^\tau, D^\tau)) \notin \mathcal{A}_b$. By the partition $isa(x^\tau, or((\dot{\neg}C)^\tau, D^\tau)) \notin S'_1$. By Lemma 5.17, instances of $ax/1$ and $i/1$ are defined in stratum 0. By Definition 5.15, for $w > 0$, $ax(sc(C^\tau, D^\tau), w) \in S_0 \leftrightarrow C \sqsubseteq D \in \mathcal{T}_d$ and $i(x^\tau) \in S_0 \leftrightarrow x \in N_I$. Hence, the

simplified rule instance of (e6) in P_1 , $isa(x^\tau, or((\neg C)^\tau, D^\tau)) | u(sc(C^\tau, D^\tau), x^\tau, w)$, is not satisfied for S'_1 .

- $used(x^\tau) \in \mathcal{N}_{used}^\tau$ required by Definition 5.20 for all $x \in N_I$. Similar to $isa(x^\tau, thing)$.
- $hw(x^\tau, oSome(R^\tau, y^\tau)) \in \mathcal{N}_{hw}^\tau$. By Definition 5.20, $(\exists R.C)(x) \in \mathbb{A}_0$, $R(x, y) \in \mathbb{A}_0$ and $C(y) \in \mathbb{A}_0$. By the partition, $isa(y^\tau, oSome(R^\tau, C^\tau)) \in S_1$, $hasa(x^\tau, R^\tau, y^\tau) \in S_1$ and $isa(y^\tau, C^\tau) \in S_1$. Hence, the rule instance of (e12) in P_1 , $hw(x^\tau, oSome(R^\tau, C^\tau)) \leftarrow isa(y^\tau, oSome(R^\tau, C^\tau)), hasa(x^\tau, R^\tau, y^\tau), isa(y^\tau, C^\tau)$, is not satisfied for S'_1 .

We conclude that S_1 is an answer set of P_1 .

- $P_2 = e_{\mathbb{H}_1}(G_2, S_0 \cup S_1)$ where G_2 denotes the ground rule instances of (e13) the only rule in stratum 2. Assume for contradiction, that S_2 is not a model of P_2 . If S_2 is not a model of P_2 then there exists some rule instance $r_g \in P_2$ not satisfied by S_2 . By Observation 8, r_g , derived from (e13), is a fact of the form $need(x^\tau, oSome(R^\tau, C^\tau))$ where $isa(x^\tau, oSome(R^\tau, C^\tau)) \in S_1$ and $hw(x^\tau, oSome(R^\tau, C^\tau)) \notin S_1$. If r_g is not satisfied by S_1 then $need(x^\tau, oSome(R^\tau, C^\tau)) \notin S_2$. The partition assigns $\mathbb{A}_0^\tau$ to S_1 . Hence, $(\exists R.C)(x) \in \mathbb{A}_0$. The partition assigns $\mathcal{N}_{hw}^\tau$ to S_1 . By Definition 5.20 there is no named witness $y \in \mathbb{A}_0$ such that $R(x, y), C(y) \in \mathbb{A}_0$. By Definition 5.20, $need(x^\tau, oSome(R^\tau, C^\tau)) \in S_2$ a contradiction. We conclude S_2 is a model of P_2 . Consider $S'_2 = S_2 \setminus need(x^\tau, oSome(R^\tau, C^\tau))$ where $need(x^\tau, oSome(R^\tau, C^\tau)) \in \mathcal{N}_{need}$. We note that P_2 is comprised entirely of simplified facts. S'_2 falsifies the corresponding simplified fact of (e13) in P_2 . We conclude that S_2 is a $\subseteq$ minimal model of S_2 and hence an answer set of P_2 .
- $P_{3\sigma} = e_{\mathbb{H}_{3\sigma-1}}(G_{3\sigma}, \mathbb{S}_{3\sigma-1})$ where $G_{3\sigma}$ denotes the ground rule instances from stratum 3σ . By assumption, the rules of $G_{3\sigma}$ were instantiated by $P_{cum}(p_\sigma^\tau, i^\tau, oSome(R^\tau, C^\tau))$, where $i \in N_I \cup P(b)$, $R \in N_R$ and C is a concept. Hence, there exists an earlier subsequence, $Subs_\rho$, $\rho < \sigma$ where $(\exists R.C)(i) \in \mathbb{A}_\rho$. Assume for contradiction that $S_{3\sigma}$ is not a model of $P_{3\sigma}$. If $S_{3\sigma}$ is not a model of $P_{3\sigma}$ then there exists some rule instance $r_g \in P_{3\sigma}$ not satisfied by $S_{3\sigma}$. We show this leads to a contradiction for every type of r_g :

- (c2). By assumption p_σ is introduced because there is no witness y for $(\exists R.C)(i) \in \mathbb{A}_\rho$ such that $R(x, y), C(y) \in \mathbb{A}_\rho$. There are two sub cases:
 - * i is a named individual. By Definition 5.20, $need(i^\tau, oSome(R^\tau, C^\tau)) \in \mathcal{N}_{need}^\tau$. By the partition $need(i^\tau, oSome(R^\tau, C^\tau)) \in \mathbb{S}_{3\sigma-1}$. If r_g is not satisfied by $S_{3\sigma}$ then $used(p_\sigma^\tau) \notin S_{3\sigma}$. By Definition 5.20 $used(p_\sigma^\tau) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma^\tau) \in S_{3\sigma}$, a contradiction.
 - * i is an older parameter. By Definition 5.20, $need(i^\tau, oSome(R^\tau, C^\tau)) \in \mathcal{W}_{need}^\tau$. By the partition $need(i^\tau, oSome(R^\tau, C^\tau)) \in \mathbb{S}_{3\sigma-1}$. If r_g is not satisfied by $S_{3\sigma}$ then $used(p_\sigma^\tau) \notin S_{3\sigma}$. By Definition 5.20 $used(p_\sigma^\tau) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma^\tau) \in S_{3\sigma}$, a contradiction.
- (c3). If r_g is not satisfied by $S_{3\sigma}$ then $cea(p_\sigma^\tau, C^\tau, i^\tau) \notin S_{3\sigma}$ where $used(p_\sigma) \in S_{3\sigma}$. By Definition 5.20 $used(p_\sigma^\tau) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma^\tau) \in S_{3\sigma}$. By assumption $(\exists R.C)(i) \in \mathbb{A}_\rho$. By Definition 5.20 $cea(p_\sigma^\tau, C^\tau, i^\tau) \in \mathcal{W}_{cea}$. By the partition $cea(p_\sigma^\tau, C^\tau, i^\tau) \in S_{3\sigma}$, a contradiction.
- (c4). If r_g is not satisfied by $S_{3\sigma}$ then $cea(p_\sigma^\tau, D^\tau, i^\tau) \notin S_{3\sigma}$ where $used(p_\sigma) \in S_{3\sigma}$, $isa(i^\tau, oAll(R^\tau, D^\tau)) \in \mathbb{S}_{3\sigma-1}$. By Definition 5.20 $used(p_\sigma^\tau) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma^\tau) \in S_{3\sigma}$. By Definition 5.20, $(\forall R.D)(i) \in \mathbb{A}_\rho$. By Definition 5.20 $cea(p_\sigma^\tau, D^\tau, i^\tau) \in \mathcal{W}_{cea}$. By the partition $cea(p_\sigma^\tau, D^\tau, i^\tau) \in S_{3\sigma}$, a contradiction.
- (c5). If r_g is not satisfied by $S_{3\sigma}$ then $dnb(y^\tau, p_\sigma^\tau) \notin S_{3\sigma}$ where $used(p_\sigma) \in S_{3\sigma}$, $used(y^\tau) \in \mathbb{S}_{3\sigma-1}$, $isa(y^\tau, D^\tau) \notin \mathbb{S}_{3\sigma-1}$ and $cea(p_\sigma^\tau, D^\tau, i^\tau) \in S_{3\sigma}$. By Definition 5.20 $used(p_\sigma^\tau) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma^\tau) \in S_{3\sigma}$. By Definition 5.20 there exists an individual y that is older than p_σ where $D(y) \notin \mathcal{A}_b$. By Definition 5.20 $cea(p_\sigma^\tau, D^\tau, i^\tau) \in \mathcal{W}_{cea}$. There are two subcases
 - * $(\exists R.D)(i^\tau) \in \mathbb{A}_\rho$. By Proposition 4.23(7), $\mathcal{A}_b$ is quantifier downward saturated and $D(p_\sigma) \in \mathcal{A}_b$. By Definition 4.4, y does not block p_σ . By Definition 5.20 $dnb(y^\tau, p_\sigma^\tau) \in \mathcal{W}_{dnb}$. By the partition, $dnb(y^\tau, p_\sigma^\tau) \in S_{3\sigma}$, a contradiction.
 - * $(\forall R.D)(i^\tau) \in \mathbb{A}_\rho$. By Proposition 4.23(7), $\mathcal{A}_b$ is quantifier downward saturated and $D(p_\sigma) \in \mathcal{A}_b$. By Definition 4.4, y does not block p_σ . By Definition 5.20, $dnb(y^\tau, p_\sigma^\tau) \in \mathcal{W}_{dnb}$. By the partition, $dnb(y^\tau, p_\sigma^\tau) \in S_{3\sigma}$, a contradiction.

We conclude $S_{3\sigma}$ is a model of $P_{3\sigma}$. We note that $P_{3\sigma}$ is comprised entirely of simplified facts. The argument for $\subseteq$ -minimality is similar to P_2 . We conclude $S_{3\sigma}$ is an answer set of $P_{3\sigma}$.

- $P_{3\sigma+1} = e_{\mathbb{H}_{3\sigma}}(G_{3\sigma+1}, \mathbb{S}_{3\sigma})$ where $G_{3\sigma+1}$ denotes the ground rule instances of (c6), the only rule in stratum $3\sigma + 1$. Assume for contradiction that $S_{3\sigma+1}$ is not a model of $P_{3\sigma+1}$. If $S_{3\sigma+1}$ is not a model of $P_{3\sigma+1}$ then there exists some rule instance $r_g \in P_{3\sigma+1}$ not satisfied by $S_{3\sigma+1}$. If r_g is not satisfied by $S_{3\sigma+1}$ then $b(p_\sigma^\tau) \notin S_{3\sigma+1}$ where $used(p_\sigma) \in \mathbb{S}_{3\sigma}$, $used(y^\tau) \in \mathbb{S}_{3\sigma}$ and $dnb(y^\tau, p_\sigma^\tau) \notin \mathbb{S}_{3\sigma}$. By Definition 5.20, $used(p_\sigma) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma) \in \mathbb{S}_{3\sigma}$. By assumption, y is older than p_σ . Hence, by Definition 5.20 and the partition, $used(p_\sigma) \in \mathbb{S}_{3\sigma}$. By the partition, $dnb(y^\tau, p_\sigma^\tau) \notin \mathcal{W}_{dnb}^\tau$. By Definition 5.20, y blocks p_σ . By Definition 5.20, $b(p_\sigma^\tau) \in \mathcal{W}_b^\tau$. By the partition, $b(p_\sigma^\tau) \in S_{3\sigma+1}$, a contradiction. We conclude $S_{3\sigma+1}$ is a model of $P_{3\sigma+1}$. We note that $P_{3\sigma}$ is comprised entirely of simplified facts. The argument for $\subseteq$ -minimality is similar to P_2 . We conclude $S_{3\sigma+1}$ is an answer set of $P_{3\sigma+1}$.

- $P_{3\sigma+2} = e_{\mathbb{H}_{3\sigma+1}}(G_{3\sigma+2}, \mathbb{S}_{3\sigma+1})$ where $G_{3\sigma+2}$ denotes the ground rule instances of the rules in stratum $3\sigma + 2$. By assumption, $G_{3\sigma+2}$ was instantiated by $P_{cum}(p_\sigma^\tau, i^\tau, oSome(R^\tau, C^\tau))$, where $i \in N_I \cup P(b)$, $R \in N_R$ and C is a concept. Hence, there exists an earlier subsequence $Subs_\rho$, $\rho < \sigma$ where $(\exists R.C)(i) \in \mathbb{A}_\rho$. Assume for contradiction, that $S_{3\sigma+2}$ is not a model of $P_{3\sigma+2}$. If $S_{3\sigma+2}$ is not a model of $P_{3\sigma+2}$ then there exists some rule instance $r_g \in P_{3\sigma+2}$ not satisfied by $S_{3\sigma+1}$. We show this leads to a contradiction for every type of r_g :

- (c7). By Definition 5.20, $used(p_\sigma) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma) \in \mathbb{S}_{3\sigma+1}$. If r_g is not satisfied by $S_{3\sigma+2}$ then $hasa(i^\tau, R^\tau, p_\sigma^\tau) \notin S_{3\sigma+2}$. By Proposition 4.23(7), $\mathcal{A}_b$ is quantifier downward saturated and $R(i, p_\sigma) \in \mathcal{A}_b$. By the partition $hasa(i^\tau, R^\tau, p_\sigma^\tau) \in S_{3\sigma+2}$, a contradiction.
- (c8). Similar to (c7).
- (c9). By Definition 5.20, $used(p_\sigma) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma) \in \mathbb{S}_{3\sigma+1}$. If r_g is not satisfied by $S_{3\sigma+2}$ then $isa(p_\sigma^\tau, D^\tau) \notin S_{3\sigma+2}$, and $isa(i^\tau, oAll(R^\tau, D^\tau)) \in$

- $\mathbb{S}_{3\sigma+1}$. By the partition, $(\forall R.D)(i^\tau) \in \mathbb{A}_\rho$. By Proposition 4.23(7), $\mathcal{A}_b$ is quantifier downward saturated and $D(p_\sigma) \in \mathcal{A}_b$. By the partition $isa(p_\sigma^\tau, D^\tau) \in S_{3\sigma+2}$, a contradiction.
- (c10). By Definition 5.20, $used(p_\sigma) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma) \in \mathbb{S}_{3\sigma+1}$. If r_g is not satisfied by $S_{3\sigma+2}$ then $isa(p_\sigma^\tau, thing) \notin S_{3\sigma+2}$. By Definition 5.20 $isa(p_\sigma^\tau, thing) \in \mathcal{A}_b$. By the partition $isa(p_\sigma^\tau, thing) \in S_{3\sigma+2}$, a contradiction.
 - (c11). By Definition 5.20, $used(p_\sigma) \in \mathcal{W}_{used}^\tau$. By the partition, $used(p_\sigma) \in \mathbb{S}_{3\sigma+1}$. By Observation 8, if r_g is not satisfied by $S_{3\sigma+2}$ then $isa(p_\sigma^\tau, C^\tau) \notin S_{3\sigma+2}$, $isa(p_\sigma^\tau, and(C^\tau, D^\tau)) \in S_{3\sigma+2}$ and $b(p_\sigma^\tau) \notin \mathbb{S}_{3\sigma+1}$. By Definition 5.20, $b(p_\sigma^\tau) \notin \mathcal{W}_b$. Hence p_σ is not blocked. The partition assigns $\mathbb{A}_\sigma^\tau$ to $S_{3\sigma+2}$. By Definition 5.20, $(C \sqcap D)(p_\sigma) \in \mathbb{A}_{3\sigma+2}$. By Proposition 4.23(5) $\mathcal{A}'_b$ is boolean downward saturated and $C(p_\sigma) \in \mathcal{A}'_b$. By the partition, $isa(p_\sigma^\tau, C^\tau) \in S_{3\sigma+2}$, a contradiction.
 - Rules (c12, c13, c15, c16, c19 and c23) are similar to ((e2, e3, e5, e6, e9 and e13), taking into account blocking by following the pattern in (c11).

We conclude $S_{3\sigma+2}$ is a model of $P_{3\sigma+2}$. The argument for $\subseteq$ -minimality follows similar patterns as for the rules for individuals in P_1 . We conclude that $S_{3\sigma+2}$ is an answer set of $P_{3\sigma+2}$.

Finally, we show b^τ is a model of $Cons^i$. Assume for contradiction that b^τ is not a model of $Cons^i$. If b^τ is not a model of $Cons^i$ then there exists some rule instance $r_g \in Cons^i$ of an integrity constraint not satisfied by $S_{3\sigma}$. We show this leads to a contradiction for every type of r_g :

- (e10). By Observation 8, if r_g is not satisfied by b^τ then $isa(x^\tau, neg(thing)) \in b^\tau$. By assumption b is open. Hence $isa(x^\tau, neg(thing)) \notin b^\tau$, a contradiction.
- (e11). By Lemma 5.17, instances of $c/1$ are defined in stratum 0. By Definition 5.15, $c(C^\tau) \in sig(\mathcal{K})^\tau \leftrightarrow C \in N_C$. If r_g is not satisfied by b^τ then $isa(x^\tau, C^\tau) \in b^\tau$ and $isa(x^\tau, neg(C)) \in b^\tau$. By assumption b is open. Hence, there is no C such that $C(x) \in b$ and $\neg C(x) \in b$, a contradiction.

- (c20). Similar to (e10).
- (c21). Similar to (e11).

A model that fails to satisfy a weak constraint adds the associated weight to the optimality. We show that the weight of each element in $\mathbb{O}_b$ is recorded in o the optimality of the branch by considering weak constraints of type of r_g :

- (a4). By Observation 8, if r_g is not satisfied then $u(ca(C^\tau, x^\tau), w) \in b^\tau$. By Definition 5.20, $\mathbb{O}_0^\tau \in b^\tau$. Hence the optimality o includes w for each $\langle C(x)^{[w]}, x \rangle \in \mathbb{O}_0$.
- (a6) and (e7) are similar to (a4).

We conclude that b^τ satisfies $Cons^i$ and $o = m$. Hence, b^τ is an answer set of Π^i with an optimality of m . $\square$

To show that the optimality of an optimal answer set obtained by Algorithm 3 correctly reflects the n -distance of the knowledge base, we first show, in Lemma 5.23, that an o -optimal answer set can be used to guide Algorithm 2 to an open completed branch with a distance of at most o .

Lemma 5.23 (Open branch based on answer set). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be an n -inconsistent p -ALC knowledge base, Π^i be a program obtained from $ASP(\mathcal{K})$ using Algorithm 3 and S be an optimal answer set of Π^i with optimality o . Then there exists an m -distant branch b obtained from Algorithm 2 applied to $\mathcal{K}$ such that $S \subseteq b^\tau$ and $m \leq o$.*

Proof. We show that given an answer set S of Π^i , S can be used to guide Algorithm 2 to a completed open tableau branch $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ from $\mathcal{K}$. By Observation 9, the program Π^i may include parameters that are not “used” in the answer set. A parameter p_σ introduced by $P_{cum}(p_\sigma^\tau, i^\tau, (\exists R.C)^\tau)$ is not used if atom $need(x^\tau, ((\exists R.C)(x))) \notin S$. The atom $need(x^\tau, ((\exists R.C)(x)))$ appears in the body of the rule instance of (c2) and does not appear in the head of any other rule instance in Π^i . We conclude $used(p) \notin S$. The atom $used(p)$ occurs

in the body of all rule instances of (c3-c23). Hence, no atoms relating to p are introduced to S . Let $\Pi^k \subseteq \Pi^i$ denote the rule instances that exclude the unused instantiations of P_{cum} where $k \leq i$.

Let $AO_{j,\sigma}^\tau = \mathcal{A}_b^\tau \cup \mathcal{O}_b^\tau \cup \{b(p_\rho) | b(p_\rho) \in W_b, \rho < \sigma\}$ be the atoms that represent A_b , O_b and the blocking conditions at step j in subsequence σ of branch b . Let b_σ denote the branch at the end of subsequence σ . The proof is by well-founded induction using $<_{lex}$, the lexicographic ordering of pairs (j, σ) , where j is a step in a subsequence Sub_σ , of b .

Let (j, σ) be arbitrary step j in subsequence σ . We prove that $AO_{j,\sigma}^\tau \subseteq S$ and where no more rules apply in subsequence σ , $b_\sigma^\tau \subseteq S$. Assume as induction hypothesis that for all $(j', \sigma') <_{lex} (j, \sigma)$ it is the case that $AO_{j',\sigma'}^\tau \subseteq S$ and where no more rules apply in subsequence σ , $b_{\sigma'}^\tau \subseteq S$.

The proof goes by cases:

- $j = 0, \sigma = 0$. Initialisation of the branch. By Definition 5.15, $\mathcal{K}^\tau \subseteq \Pi^k$. Hence, $\mathcal{K}^\tau \subseteq S$. By rule (a1), for each $C(x) \in \mathcal{A}$, $(C(x))^\tau \in S$. By rule (a2), for each $R(x, y) \in \mathcal{A}$, $(R(x, y)^{[w]})^\tau \in S$. By rule (a3), for each $C(x)^{[w]} \in \mathcal{A}_d$, either $(C(x)^{[w]})^\tau \in S$ or $\langle (C(x)^{[w]}), x \rangle^\tau \in S$. By rule (a5), for each $R(x, y)^{[w]} \in \mathcal{A}_d$, either $(R(x, y)^{[w]})^\tau \in S$ or $\langle (R(x, y)^{[w]}), x \rangle^\tau \in S$. At line 1 of Algorithm 2, choose the smallest $\mathcal{A}_o$ such that $(\mathcal{A}_d \setminus \mathcal{A}_o)^\tau \subseteq S$. At line 2 of Algorithm 2, assign $\mathcal{O}_b = \{C(x)^{[w]} | C(x)^{[w]} \in \mathcal{A}_o\} \cup \{R(x, y)^{[w]} | R(x, y)^{[w]} \in \mathcal{A}_o\}$. At line 3 of Algorithm 2, assign $\mathcal{A}_b = \mathcal{A} \cup \mathcal{A}_o$. We conclude $AO_{0,0}^\tau \subseteq S$.
- $j > 0, \sigma = 0$. The proof goes by cases for each applicable rule instance:
 - $\rightarrow_{\sqcap}$ -rule. The preconditions are: $(C \sqcap D)(x) \in \mathcal{A}_b$ where $x \in N_I$ is not blocked, $C(x) \notin \mathcal{A}_b$ and $D(x) \notin \mathcal{A}_b$. By Observation 1, x is not blocked. By the induction hypothesis, $((C \sqcap D)(x))^\tau \in S$. By rule (e1), $(C(x))^\tau \in S$. By rule (e2), $(D(x))^\tau \in S$. In line 6 of Algorithm 2, assign $\mathcal{A}_b$ to the valid expansion $\mathcal{A}_b \cup \{C(x), D(x)\}$. Hence, $AO_{j,0}^\tau \subseteq S$.

- $\rightarrow_{\sqcup}$ -rule. The preconditions are: $(C \sqcup D)(x) \in \mathcal{A}_b$ where $x \in N_I$ is not blocked, $C(x) \notin \mathcal{A}_b$ and $D(x) \notin \mathcal{A}_b$. By Observation 1, x is not blocked. By the induction hypothesis, $((C \sqcup D)(x))^\tau \in S$. The valid expansions of $\mathcal{A}_b$ are $\mathcal{A}_{e1} = \{C(x)\}$ and $\mathcal{A}_{e2} = \{D(x)\}$. By rule (e3), $(C(x))^\tau \in S$ or $(D(x))^\tau \in S$. In line 6 of Algorithm 2, choose for $\mathcal{A}_b$, the expansion $\mathcal{A}_b \cup \mathcal{A}_{e1}$ if $(C(x))^\tau \in S$, or $\mathcal{A}_b \cup \mathcal{A}_{e2}$ if $(D(x))^\tau \in S$. Hence, $AO_{j,0}^\tau \subseteq S$.
- $\rightarrow_{\forall}$ -rule. The preconditions are: $(\forall R.C)(x) \in \mathcal{A}_b$ where $x \in N_I$ is not blocked, $R(x, y) \in \mathcal{A}_b$ and $C(y) \notin \mathcal{A}_b$. By Observation 1, x is not blocked. By the induction hypothesis, $((\forall R.C)(x))^\tau \in S$ and $R(x, y)^\tau \in S$. By rule (e4), $(C(y))^\tau \in S$. In line 6 of Algorithm 2, assign $\mathcal{A}_b$ to the valid expansion $\mathcal{A}_b \cup \{C(y)\}$. Hence, $AO_{j,0}^\tau \subseteq S$.
- $\rightarrow_{\mathcal{T}}$ -rule. The preconditions are: $C \sqsubseteq D \in \mathcal{T}$ where $x \in N_I$ is not blocked and $(\dot{\neg}C \sqcup D)(x) \notin \mathcal{A}_b$. By Observation 1, x is not blocked. By Definition 5.15, $\mathcal{K}^\tau \subseteq \Pi^k$ and $\text{sig}(\mathcal{K})^\tau \subseteq \Pi^k$. Hence, $(C \sqsubseteq D)^\tau \in S$ and $i(x^\tau) \in S$. By rule (e6), $((\dot{\neg}C \sqcup D)(x))^\tau \in S$. In line 6 of Algorithm 2, assign $\mathcal{A}_b$ to the valid expansion $\mathcal{A}_b \cup ((\dot{\neg}C \sqcup D)(x))^\tau$. Hence, $AO_{j,0}^\tau \subseteq S$.
- $\rightarrow_{\mathcal{T}_d}$ -rule. The preconditions are: $C \sqsubseteq D^{[w]} \in \mathcal{T}_d, w > 0$ where $x \in N_I$ is not blocked, $(\dot{\neg}C \sqcup D)(x) \notin \mathcal{A}_b$ and $\langle C \sqsubseteq D^{[w]}, x \rangle \notin \mathcal{O}_b$. By Observation 1, x is not blocked. By Definition 5.15, $\mathcal{K}^\tau \subseteq \Pi^k$ and $\text{sig}(\mathcal{K})^\tau \subseteq \Pi^k$. Hence, $(C \sqsubseteq D^{[w]})^\tau \in S$ and $i(x^\tau) \in S$. By rule (e6), $((\dot{\neg}C \sqcup D)(x))^\tau \in S$ or $(\langle C \sqsubseteq D^{[w]}, x \rangle)^\tau \in S$. In line 6 of Algorithm 2, choose for $\mathcal{A}_b$, the valid expansion $\mathcal{A}_b \cup ((\dot{\neg}C \sqcup D)(x))^\tau$ if $((\dot{\neg}C \sqcup D)(x))^\tau \in S$ otherwise choose $\mathcal{A}_b$ and add $\langle C \sqsubseteq D^{[w]}, x \rangle$ to $\mathcal{O}_b$ in line 7. Hence, $AO_{j,0}^\tau \subseteq S$.
- $\rightarrow_Q$ -rule. The preconditions are: $x \in N_I$ is not blocked, for some $C \in \mathcal{Q}(\mathcal{K})$, $C(x) \notin \mathcal{A}_b$ and $\neg C(x) \notin \mathcal{A}_b$. By Observation 1, x is not blocked. By Definition 5.15, $\text{sig}(\mathcal{K})^\tau \in \Pi^k$ and $\mathcal{Q}(\mathcal{K}) \in \Pi^k$. Hence, $i(x^\tau)$ and $qc(C^\tau) \in S$. By rule (e9), $(C(x))^\tau \in S$ or $(\neg C(x))^\tau \in S$. In line 6 of Algorithm 2, choose for $\mathcal{A}_b$, the valid expansion $\mathcal{A}_b \cup \{(C(x))^\tau\}$ if $(C(x))^\tau \in S$ otherwise choose $\mathcal{A}_b \cup \{(\neg C(x))^\tau\}$. Hence, $AO_{j,0}^\tau \subseteq S$.
- No further rules apply in subsequence 0. By Definition 5.15, $\mathcal{K}^\tau \cup \text{sig}(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K}) \subseteq \Pi^k$. Hence, $\mathcal{K}^\tau \cup \text{sig}(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K}) \subseteq S$. By rule (e12), $\text{used}(x^\tau) \in S$. By the induction

hypothesis $AO_{j',0}^\tau \subseteq S$, $j' < j$. For each $(\exists R.C)(x) \in \mathcal{A}_b$ there are two subcases:

- * There is some y such that $R(x, y) \in \mathcal{A}_b$ and $C(y) \in \mathcal{A}_b$. By the induction hypothesis, $((\exists R.C)(x))^\tau \in S$, $(R(x, z))^\tau \in S$ and $(C(z))^\tau \in S$. By rule (e12), $hw(x^\tau, (\exists R.C)^\tau) \in S$. By rule (e13), $need(x^\tau, (\exists R.C)^\tau) \notin S$.
- * There is no y such that $R(x, y) \in \mathcal{A}_b$ and $C(y) \in \mathcal{A}_b$. By the induction hypothesis, $((\exists R.C)(x))^\tau \in S$, and for every $z \in N_I$, if $(R(x, z))^\tau \in S$ then $(C(z))^\tau \in S$. By rule (e12), $hw(x^\tau, (\exists R.C)^\tau) \notin S$. By rule (e13), $need(x^\tau, (\exists R.C)^\tau) \in S$.

We conclude $\mathcal{N}_{used}^\tau$, $\mathcal{N}_{hw}^\tau$ and $\mathcal{N}_{need}^\tau$ are correctly represented in S . Hence, $b_0^\tau \subseteq S$.

- $j = 0, \sigma > 0$. The $\rightarrow_{\exists\forall}$ -rule. The preconditions are: $(\exists R.C)(x) \in \mathcal{A}_b$, $x \in N_I \cup P(b)$ is not blocked, there is no y such that $R(x, y) \in \mathcal{A}_b$ and $C(y) \in \mathcal{A}_b$. There may be more than one such applicable rule. By the inductive hypothesis, $((\exists R.C)(x))^\tau \in S$ and $need(x^\tau, ((\exists R.C)(x))^\tau) \in S$. By assumption, a parameter, p was introduced by $P_{cum}(p^\tau, (\exists R.C)^\tau, x^\tau)$. Select the rule instance to apply by identifying the $need/2$ atom that corresponds to the introduction of the lowest integer parameter p not yet introduced to the branch. By rule (c2), $used(p^\tau) \in S$. By rule (c7), $(R(x, p))^\tau \in S$. By rule (c8), $(C(p))^\tau \in S$. By the inductive hypothesis, for each $(\forall R.D)(x) \in \mathcal{A}_b$, $((\forall R.D)(x))^\tau \in S$ and by rule (c9), $(D(p))^\tau \in S$. By rule (c10), $isa(p^\tau, thing) \in S$. In line 6 of Algorithm 2, assign $\mathcal{A}_b$ to the valid expansion $\mathcal{A}_b \cup \{C(p), R(x, p)\} \cup \{D(p) | (\forall R.D)(p) \in \mathcal{A}_b\}$. By rule (c3), $cea(p^\tau, C^\tau) \in S$. By rule (c4), $cea(p^\tau, D^\tau) \in S$. By the inductive hypothesis, for each individual y , older than p , $used(y^\tau) \in S$ and $C(y) \in \mathcal{A}_b \rightarrow (C(y))^\tau \in S$. By rule (c5), for each individual y that does not block p , $dnb(y^\tau, p^\tau) \in S$. There are two sub-cases:

- p is blocked. There exists some y such that $used(y^\tau) \in S$, $dnb(y^\tau, p^\tau) \notin S$. By rule (c6), $b(p^\tau) \in S$.
- p is not blocked. For each y such that $used(y^\tau) \in S$, $dnb(y^\tau, p^\tau) \in S$. By rule (c6), $b(p^\tau) \notin S$.

We conclude $\mathcal{W}_b^\tau$ for p is correctly represented in S and $AO_{0,\sigma}^\tau \subseteq S$.

- $j > 0, \sigma > 0$. The proof goes by cases for each applicable rule instance:
 - $\rightarrow_{\sqcap}$ -rule. The preconditions are: $(C \sqcap D)(p) \in \mathcal{A}_b$ where $p \in P(b)$ is not blocked, $C(p) \notin \mathcal{A}_b$ and $D(p) \notin \mathcal{A}_b$. By rule (c2), $used(p^\tau) \in S$. By the inductive hypothesis, $((C \sqcap D)(p))^\tau \in S$ and $b(p^\tau) \notin S$. By rule (c11), $(C(p))^\tau \in S$. By rule (c12), $(D(p))^\tau \in S$. In line 6 of Algorithm 2, assign $\mathcal{A}_b$ to the valid expansion $\mathcal{A}_b \cup \{C(p), D(p)\}$. Hence, $AO_{j,\sigma}^\tau \subseteq S$.
 - $\rightarrow_{\sqcup}$ -rule. Is similar to the $\rightarrow_{\sqcup}$ -rule for a named individual, using ASP rule (c13) and the blocking condition of the $\rightarrow_{\sqcap}$ -rule for a parameter.
 - $\rightarrow_{\mathcal{T}}$ -rule. Is similar to the $\rightarrow_{\mathcal{T}}$ -rule for a named individual, using ASP rule (c15) and the blocking condition of the $\rightarrow_{\sqcap}$ -rule for a parameter.
 - $\rightarrow_{\mathcal{T}_d}$ -rule. Is similar to the $\rightarrow_{\mathcal{T}_d}$ -rule for a named individual, using ASP rule (c17) and the blocking condition of the $\rightarrow_{\sqcap}$ -rule for a parameter.
 - $\rightarrow_Q$ -rule. Is similar to the $\rightarrow_Q$ -rule for a named individual, using ASP rule (c19) and the blocking condition of the $\rightarrow_{\sqcap}$ -rule for a parameter.
 - No rules apply in subsequence σ . By the inductive hypothesis $b_{\sigma'}^\tau \subseteq S$, $\sigma' < \sigma$ and $AO_{j',\sigma}^\tau \subseteq S$, $j' < j$. By rule (c2), $\mathcal{W}_{used}^\tau$ for p is correctly represented in S . By rules (c3 and c4), $\mathcal{W}_{cea}^\tau$ for p is correctly represented in S . By rule (c5), $\mathcal{W}_{dnb}^\tau$ for p is correctly represented in S . For each $(\exists R.C)(p) \in \mathcal{A}_b$, by the inductive hypothesis, $((\exists R.C)(p))^\tau \in S$ and by rule (c23), $need(p^\tau, (\exists R.C)^\tau) \in S$. We conclude $\mathcal{W}_{need}^\tau$ for p is correctly represented in S . Hence, $b_\sigma^\tau \subseteq S$.

By Lemma 4.15, Algorithm 2 terminates. Suppose it terminates with $b = b_l$ after l parameters are introduced. Then clearly $l \leq k$. The m -distance of b is given by the sum of the weights of the axioms recorded in $\mathcal{O}_b$ which were added under the guidance of the $u/3$ atoms in S . We conclude that $m \leq o$. By the inductive proof, $b_l^\tau \subseteq S$. Hence, $b^\tau \subseteq S$.

□

Theorem 5.24 (Correspondence back). *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ be an n -inconsistent p -ALC knowledge base, Π^i be a program obtained from $ASP(\mathcal{K})$ using Algorithm 3 and S be an optimal*

answer set of Π^i with optimality o . Then $o = n$.

Proof. Given S , an o optimal answer set of a program Π^i obtained from Algorithm 3 for an n -inconsistent $\mathcal{K}$ we prove that the optimality of o is n by showing that for S to be reported as optimal answer set, o must be equal to n .

First, we show the existence of an answer set S_n from some Π^i obtained from Algorithm 3 that has an optimality of n . By assumption, $\mathcal{K}$ is n -distant and by Theorem 4.31, there is an n -distant branch returned by Algorithm 2. By Theorem 5.22, a $\subseteq$ - m -distant (any m) branch b is shown to be an answer set of some Π^i . Therefore, in particular, we can put $m = n$ (since we know there is such a branch) to give subset minimal n -distant branch b_n which has an answer set S_n of some Π^i with optimality n .

We show that no answer set obtained from some Π^i obtained from Algorithm 3 can have an optimality less than n . Assume for contradiction that S_m is obtained from some Π^i obtained from Algorithm 3 where $m < n$. S_m is a $\subseteq$ - m -optimal answer set of $ASP(\mathcal{K})$ and, by Lemma 5.23, can be used to guide Algorithm 2 to an open complete branch $b_{m'}$ that is m' -distant, where $m' \leq m$. Hence, $b_{m'}$ is an open branch with distance $m' < n$. By Corollary 4.27, no open branch obtained from Algorithm 2 can have a distance less than n , a contradiction.

We conclude that any answer set obtained by Algorithm 3 that is reported as optimal has an optimality equal to n . By assumption, S is an optimal answer set obtained from Algorithm 3 having a distance of o . We conclude that for S to be reported as optimal, then $o = n$. $\square$

Proposition 5.25 (Termination). *Let $\mathcal{K}$ be a p - $\mathcal{ALC}$ knowledge base. Then Algorithm 3 applied to $ASP(\mathcal{K})$ terminates.*

Proof. The proof is by contradiction. Recall that Π^i denotes $ASP(\mathcal{K})$ formed from $P_{base} = \mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K}) \cup P_b$ and i instances of P_{cum} . Suppose for a contradiction that Algorithm 3 does not terminate when applied to $ASP(\mathcal{K})$. There are three circumstances to consider.

1. The termination of $\mathbf{Ground}(\Pi^i)$. By Lemma 5.16, Π^i belongs to the class of $\mathcal{FG}$ programs.

Hence, Π^i has a finite grounding and each call to $\mathbf{Ground}(\Pi^i)$ terminates.

2. The termination of $\text{Solve}(\Pi^i)$. By Lemma 5.16, Π^i belongs to the class of $\mathcal{FG}$ programs. Hence, Π^i has a finite number of answer sets and each call to $\text{Solve}(\Pi^i)$ terminates.
3. The finiteness of the sequence of P_{cum} instantiations introduced by Algorithm 3. Each P_{cum} is instantiated from a *need/2* atom found in an answer set of Π^k , where $k < i$. The terms X^τ and $oSome(R^\tau, C^\tau)$ in each *need/2* atom denote an witness for individual X concept belonging to concept $\exists R.C$ in an open (but not yet completed) branch. By Lemma 4.15 Algorithm 2 is terminating and we conclude a finite number of such *need/2* atoms are introduced by Algorithm 3.

□

5.4 Complexity

We can obtain the complexity of the program Π^i at each iteration by considering the structure of the program.

Proposition 5.26. *Let $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base and Π^i be the ASP representation of $\mathcal{K}$ including i cumulative instantiations of P_{cum} . The task of finding an optimal answer set of Π^i is characterised by the complexity class Δ_3^P .*

Proof. The task can be characterised by considering the complexity of the brave reasoning task. We show the complexity of the brave reasoning task for Π^i by considering the characteristics of the program. Π^i is constructed from the facts $\mathcal{K}^\tau \cup sig(\mathcal{K})^\tau \cup \mathcal{Q}(\mathcal{K})^\tau$ with P_b and i instances of P_{cum} . Π^i is therefore a disjunctive logic program that includes weight constraints and with no predicates with an arity that exceeds 4. From Lemma 5.16 Π^i is finitely ground and we can obtain a finite grounding. From Lemma 5.18 Π^i is head cycle free and from Lemma 5.17 is locally stratified. From the results in Table 5.1 we obtain the complexity of the brave reasoning task for Π^i is in Δ_3^P . □

This complexity result provides insight into the complexity of finding an answer set at each iteration. Notice however, that the complexity of the whole task is much greater. We know from Table 2.5 that the complexity of the satisfiability tableau for $\mathcal{ALC}$ is in EXP. This is reflected in our implementation by the need for exponentially many iterations.

For the second part of proofs by refutations, where the n -inconsistency is known, the complexity at each iteration is lowered.

Proposition 5.27. *Let $\mathcal{K}$ be a $p\text{-}\mathcal{ALC}$ knowledge base and Π^i be the ASP representation of $\mathcal{K}$ including i cumulative instantiations of P_{cum} and $n \geq 0$ be an integer. The task of finding an n -bounded optimal answer set of Π^i is characterised by the complexity class Σ_2^P .*

Proof outline. From Lemma 5.16 Π^i is finitely ground and we can obtain a finite grounding. From Lemma 5.18 Π^i is head cycle free. We follow the technique used in [BLR00] Lemma 15. We can decide if there is an n -bounded optimal answer set of Π^i as follows: (1) Guess and check if a (polynomial-sized) interpretation I is an answer set of Π^i . (2) Check if the sum of all weak constraints not satisfied by I is $\leq n$. Now, Π^i is head cycle free (v_h) and can be translated into a normal (*not*) program⁵. By [EFFW07] Lemma 2 case 3, (1) can be checked in coNP and (2) can be checked in P. We conclude the problem can be solved with one NP oracle call and is therefore in Σ_2^P . $\square$

5.5 Summary

We have presented a sound, complete and terminating implementation of Algorithm 2 as an incremental clingo answer set program. The implementation can be used to obtain preferred consequences of a $p\text{-}\mathcal{ALC}$ knowledge base using proofs by refutation performed in two steps. The implementation as presented has been developed to mirror the exact operation of the tableau algorithm and encoded to aid readability. Further optimisation of both the tableau and the ASP are possible, a number of such optimisations are discussed in Chapter 7.

⁵It is now not locally stratified.

Chapter 6

Evaluation

This chapter presents an evaluation of our ASP implementation. The aims of this evaluation are twofold: (i) to demonstrate empirically that the ASP program correctly implements the tableau algorithm presented in Chapter 4 and (ii) to provide insight into performance by measuring computational time with respect to the number of inconsistencies in a knowledge base. Performance was evaluated using synthetic inconsistent ontologies generated by *OWLGen*, a tool that we developed for this purpose. We begin by introducing the methodology behind the design of the experiments and their analysis which are used in the evaluation.

6.1 Introduction and Methodology

To demonstrate that Algorithm 2 is correctly implemented in ASP we have applied three techniques: automated unit testing, manual test case verification and cross-checking results obtained during the performance testing experiments. The unit tests were used to verify that each rule of the tableau algorithm has been correctly implemented in ASP. The manual test cases are a suite of hand-created knowledge bases. For consistent knowledge bases, inferences were checked by comparing the answer sets generated (or unsatisfiability) against inferences using the Hermit reasoner in Protégé [NSD⁺01]. For inconsistent knowledge bases, inferences were checked against a *p-ALC* tableau created by hand. The performance evaluation experiments,

described in Section 6.5, were obtained using synthetically generated inconsistent knowledge bases. Due to the paucity of suitable inconsistent knowledge bases, we created OWLGena tool for creating inconsistent knowledge bases under parametric control (see Section 6.4). Our approach enables three methods of cross-checking:

1. In each experiment, inferences obtained from a consistent knowledge base are compared against those predicted by Hermit.
2. Inferences derived from n -inconsistent knowledge bases using a single threaded ASP solver are compared with the same inference tasks performed by a multi-threaded ASP solver. In the latter case, the solver in each thread is instantiated with different configuration parameters allowing multiple solutions to be explored in parallel. For each knowledge base we check that the optimality obtained from single and multi-threaded solvers are equal.
3. Finally, the inferences obtained from a uniform knowledge base are compared with the results of the same knowledge base but with non-uniform weights. We expect that adding weights generally increases rather than diminishes the possible consequences that are entailed. The experimental results confirm that more consequences are inferred by a corresponding non-uniform knowledge base.

The design of the performance evaluation experiments and the synthetic knowledge bases has been driven by considering the variables that we anticipated would impact the operation of the tableau. We saw in Chapter 2 that the number of, and also the form of, the axioms present in an $\mathcal{ALC}$ knowledge base determines the complexity of reasoning. The influence of form is dominated by the *and*-branching and the *or*-branching of the tableau. Since Algorithm 2 is based on the $\mathcal{ALC}$ satisfiability tableau we anticipated that it is also sensitive to these variables. Each defeasible axiom present in the knowledge base leads to branching in the tableau. Reasoning would therefore be sensitive to the number of defeasible axioms. The n -inconsistency of a p - $\mathcal{ALC}$ knowledge base captures an underlying measure of its inconsistency and therefore reasoning would be sensitive to this variable. Finally, the weights in each of the

defeasible axioms influence the selection of the preferred branches. The impact on performance of adding non uniform weights to a uniform knowledge base is evaluated.

A key objective of our experimental design was to characterise the influence of each variable whilst fixing (or at least minimising changes to) the other variables. However, the interdependence between the variables makes this non-trivial. To accommodate inconsistency within a knowledge base requires that some of the axioms are made defeasible. Clearly, as the number of inconsistencies increases, it may be necessary to increase the number of defeasible axioms. To mitigate this, we first characterise the impact of making axioms defeasible in the absence of inconsistencies and then characterise the influence of adding inconsistencies under the assumption that only those axioms that are involved in at least one inconsistency are made defeasible. We argue that this represents a reasonable choice since making more than such axioms inconsistent does not change the preferred consequences. Ideally, to evaluate a technique that aims to perform reasoning in the presence of inconsistencies, one would investigate the results obtained from using existing inconsistent knowledge bases.

6.2 Verification of correctness

The unit test framework is implemented in Python. Each unit test case is defined by an *ALC* knowledge base expressed in OWL Functional Style syntax and an associated *conformance specification* for the answer sets that are generated. Informally, the conformance specification defines the ASP ground atoms that are expected in an answer set, where “|” is used to specify alternative atoms or sets of atoms (using “&”). To ensure comprehensive testing, the clingo solver is instructed to generate all possible answer sets for each unit test knowledge base. This differs from the non-deterministic search for an answer set used by the main implementation that was described in Chapter 5.

Definition 6.1. A conjunction is a “&” delimited list of at least one ground ASP atom. A disjunction is a “|” delimited list of at least one conjunct. A conformance specification is a “;” delimited list of disjuncts or *UNSAT*. Let $\mathcal{S}$ be an answer set and $\mathcal{X}$ a conformance specification.

$\mathcal{S}$ conforms to $\mathcal{X}$ if for each disjunct $D \in \mathcal{X}$ there exists a conjunct $C \in D$ such that $C \subseteq \mathcal{S}$. The conformance specification *UNSAT* is used to indicate that no answer sets are expected.

Example 6.1. Test `t0055` is designed to test the implementation of the $\rightarrow\exists\forall$ -rule for a named individual. The OWL ontology `t0055_Named_ExistsForall.owl` (see Appendix A) represents the knowledge base $K_{t0055} = \langle \{(\exists R.C)(i), (\forall R.D)(i)\}, \emptyset, \emptyset, \emptyset \rangle$. Let $b = \langle \mathcal{A}_b, \mathcal{O}_b \rangle$ denote a branch developed by Algorithm 2 for K_{t0055} . By Proposition 4.23, b satisfies properties 1-7. By property 1 $\mathcal{A} \subseteq \mathcal{A}_b$ and hence $\{(\exists R.C)(i), (\forall R.C)(i)\} \subseteq \mathcal{A}_b$. By property 7, b is quantifier downward saturated and there is some individual x such that $\{R(i, x), C(x), D(x)\} \subseteq \mathcal{A}_b$. Since there is no named individual that could serve as a suitable witness a parameter must be introduced. The conformance specification for test `t0055` is:

```
isa(i,oSome(r,c));isa(i,oAll(r,d));hasa(i,r,1);isa(1,c);isa(1,d);isa(1,thing);
need(i,oSome(r,c));cea(1,c,i);cea(1,d,i);used(1);
dnb(i,1)|isa(i,c)&isa(i,d)&b(1)
```

The first line contains the ASP representations of the axioms that are expected in $\mathcal{A}_b$. The second line verifies that the atoms in $\mathcal{N}_{need}^\tau, \mathcal{W}_{cea}^\tau$ and $\mathcal{W}_{used}^\tau$ have been correctly introduced. Finally, the third line captures that there are two possible cases:

- $dnb(i, 1)$, indicating that individual i does not block parameter 1; or
- $isa(i, c), isa(i, d)$ and $b(1)$ indicates 1 is blocked by i .

The unit test cases are split into 3 sets: tests `t0001` to `t0015` are used to check that the axioms represented in OWL Functional style syntax are correctly encoded as facts in ASP; tests `t0050` to `t0058` verify that the meta-level representation of named individuals in a branch satisfy the saturation properties; and tests `t0100` to `t0108` verify that the meta-level representation of parameters in a branch satisfy the saturation properties.

A set of small knowledge bases were created and used to verify inferences by comparing them with the inferences obtained using the Hermit reasoner accessed through Protégé.

Example 6.2. The knowledge base `software.owl` (see Appendix B) has a signature with 26 individuals, 13 concepts and 4 roles. The ABox is 30 concept assertions and 22 role assertions.

The *TBox* has 4 equivalences, 8 concept inclusions, 2 disjoint classes, 4 role restrictions and 4 domain restrictions. Eliminating non-simple concepts from the knowledge base increased the number of equivalences to 7.

Finding the n -inconsistency (in this case $n = 0$) and performing refutations was fast (less than a second on the machine specified for performance evaluation in the next section) and allowed results to be compared readily with *Protégé*.

Some knowledge bases have been specifically developed to study the incremental grounding behaviour.

Example 6.3. The ontology `expbranch.owl` (see Appendix A) is designed to illustrate the worst case complexity of reasoning in $\mathcal{ALC}$. The concept structure is taken from [BCM⁺07].

$$\mathcal{K}_{exp} = \left\langle \left\{ \begin{array}{l} (\exists R_1. \forall R_2. \forall R_3. C_{11} \sqcap \\ \exists R_1. \forall R_2. \forall R_3. C_{12} \sqcap \\ \forall R_1. (\exists R_2. \forall R_3. C_{21} \sqcap \\ \exists R_2. \forall R_3. C_{22} \sqcap \\ \forall R_2. (\exists R_3. C_{31} \sqcap \\ \exists R_3. C_{32})))(a) \end{array} \right\}, \emptyset, \emptyset, \emptyset \right\rangle$$

Applying the classical $\mathcal{ALC}$ Algorithm to $\mathcal{K}_{exp}$ results in tableau with a single branch using $2^1 + 2^2 + 2^3 = 14$ parameters, 14 role assertions and $1 + 3 + (2^1 \times 4) + (2^2 \times 4) + (2^3 \times 3) = 52$ concept assertions. $\mathcal{K}_{expN}$ is the equivalent non-defeasible p - $\mathcal{ALC}$ knowledge base in which each non-simple concept C in $\mathcal{K}_{exp}$ is replaced by a new unique name E_n (a different name for each different non-simple concept) and equivalence axiom $E_n \equiv C$ is added to the knowledge base:

$$\mathcal{K}_{expN} = \left\langle \left\{ \begin{array}{l} (\exists R_1. E_1 \sqcap \\ \exists R_1. E_3 \sqcap \\ \forall R_1. E_5)(a) \end{array} \right\}, \left\{ \begin{array}{ll} E_1 \equiv \forall R_2. E_2, & E_5 \equiv \exists R_2. E_6 \sqcap \exists R_2. E_7 \sqcap \forall R_2. E_8, \\ E_2 \equiv \forall R_3. C_{21}, & E_6 \equiv \forall R_3. C_{11}, \\ E_3 \equiv \forall R_2. E_4, & E_7 \equiv \forall R_3. C_{22}, \\ E_4 \equiv \forall R_3. C_{12}, & E_8 \equiv \exists R_3. C_{32} \sqcup \exists R_3. C_{31} \end{array} \right\}, \emptyset, \emptyset \right\rangle$$

To find the n -inconsistency (in this case $n = 0$), the single threaded solver took 4 grounding

iterations and introduced 60 parameters, of which 6 were used in the model. When all the assertions are made defeasible in $\mathcal{K}_{\text{exp}N}$, finding the n -inconsistency required 19 grounding iterations and introduced 197 parameters of which 53 were used in the model. Using the multi-threaded reasoner yielded widely varying results. Over 10 repeated runs, for the non-defeasible knowledge base between 6 and 22 iterations were required and for the defeasible knowledge base, between 17 and 34 iterations.

Example 6.3 illustrates that making axioms defeasible can lead to more parameters being introduced, and hence more iterations are required. This is due to the increased branching of the tableau and the resulting impact on performance is explored in 6.5.1, Experiment 1. Notice that the number of parameters used in the final model is not significant. However, the presence of large numbers of unused parameters highlights that the implementation introduces more parameters than are needed during the non-deterministic search for a model. This is exacerbated when using a multi-threaded solver. Hence, we expect that introducing a heuristic to control the process could significantly improve performance. Strategies for incorporating heuristics are discussed in Section 8.1.1. The variation in the multi-threaded solver results observed in Example 6.3 and the results obtained from the multi-threaded performance evaluation in Section 6.5.1 are examined in more detail in the discussion in Section 6.6.

In summary, the implementation was verified by the unit tests and then by a series of hand checked entailments. Additional verification was achieved by extensive cross-checking during the performance experiments and is described in the next section.

6.3 Knowledge bases for performance evaluation

As discussed in [MBP13], it is desirable, where possible, to evaluate systems against knowledge bases that reflect those in use. Unfortunately, sourcing suitable knowledge bases for the performance evaluation proved difficult on two accounts. Firstly, the DL language fragment $\mathcal{ALC}$ is rarely used in practical applications. As seen in Chapter 2, $\mathcal{ALC}$ has a simple syntax, and the techniques developed for $\mathcal{ALC}$ can often be adapted for features found in more expressive log-

ics. However, most knowledge bases in use are designed to exploit the tractability afforded by the less expressive languages (such as $\mathcal{EL}$ and $DL-Lite_{core}$) or use features beyond $\mathcal{ALC}$. Secondly, the current generation of DL reasoners are designed to work with consistent knowledge bases and inconsistencies are repaired before use in any realistic production environment. With this in mind, we undertook an extensive search for two types of knowledge bases: inconsistent knowledge bases used by other teams working in related fields and for any $\mathcal{ALC}$ knowledge bases (consistent or otherwise) available in the existing corpora of knowledge bases.

6.3.1 Inconsistent knowledge bases

Our survey of inconsistency handling in Description logics from Section 2.4 highlights that there are relatively few systems that have been empirically evaluated and that target the logic $\mathcal{ALC}$. Table 6.1 summarises this work noting the languages used in the empirical evaluation and identifying candidate knowledge bases that could be suitable for our own evaluation.

Source	Target language	Evaluated language	Candidate Knowledge Base
[KPK06]	$SROIQ$	$DL-Lite_{core}$	None
[QJPD11]	$\mathcal{ALC}$	$\mathcal{ALC}$	MiniTambis and Proton 100
[CMMV13b]	$\mathcal{ALC}$	$\mathcal{ALC}$	Synthetic
[MMH13]	$SROIQ$	$SROIQ$	amino-acid
[ZXLV14]	$SHOIN(\mathcal{D})$	various	bird
[BFPS15]	$SROIQ(D)$	$\mathcal{EL}^{++}$	None

Table 6.1: A summary of research on inconsistent and non-classical knowledge bases indicating the target language, the evaluated and possible candidate knowledge bases for use in evaluation.

In [CMMV13b], the authors resorted to synthetic generation of the knowledge bases in the evaluation due to the lack of suitable knowledge bases and shared their dataset with us. However, the dataset contained only *incoherent* knowledge bases each having an empty ABox.

The knowledge bases MiniTambis and Proton 100 were obtained and examined. However, they are *incoherent* and contain no ABox. The bird knowledge base was rejected because it includes only 5 concepts, no roles and 2 individuals. The amino-acid knowledge base was identified as a potential candidate. It is both consistent and coherent; it has a signature with

Ontology	S/H	$ N_I $	$ N_C $	$ N_R $	CI	EQ	DC
premissis202	S	10	50	1	0	47	0
sudoku	S	81	91	0	0	81	846
CMMI	H	213	309	4	304	97	1030
unit-ontology	H	217	510	2	650	67	0
grid-prime	H	6	117	11	149	36	1
amino-acid	H	20	46	5	238	12	199

Table 6.2: $\mathcal{ALC}$ knowledge base characteristics indicating if they are synthetic (S) or hand crafted (H), the number of names in the signature $\langle N_I, N_C, N_R \rangle$ and the numbers of concept inclusions (CI), Equivalences (EQ) and Disjoint classes (DC).

46 named concepts, 5 roles and 20 individuals; and has a TBox with 238 concept inclusions, 12 concept equivalences and 199 disjoint concepts.

6.3.2 Corpora of knowledge bases

In [MBP13], the authors describe a mechanism to extract a corpus of OWL DL ontologies based on specific search criteria by sampling existing corpora. The approach has been implemented for the Manchester OWL Repository [Mat] and allows searches of three of the largest ontology repositories: Oxford Ontology Library(OOL), the BioPortal and MOWLCorp. Applying the criteria: 1) is an $\mathcal{ALC}$ knowledge base, 2) has more than 10 ABox axioms and 3) has more than 10 TBox axioms yielded 3 ontologies from the Oxford Ontology Library(OOL), 1 from BioPortal and 27 from MOWLCorp.

On closer inspection only 15 of these ontologies are unique; the remainder are versions or derivatives of the core 15. Of these: the 3 from OOL and 1 from the BioPortal included no individuals and 6 ontologies had only a basic TBox with inclusions of the form $A \sqsubseteq B$ where all A and B were named concepts (they had been classified as $\mathcal{ALC}$ because they included one or more disjoint concepts). Finally, of the remainder, there were 2 synthetic ontologies and 3 hand-crafted ontologies. The first synthetic ontology **sudoku** was a highly specialised representation of a Sudoku board and the second **premissis202** is a unit test ontology with a TBox comprised entirely of non-primitive definitions (equivalences) and no hierarchical structure or roles.

Table 6.2 summarises the characteristics of all the potentially useful $\mathcal{ALC}$ knowledge bases that

were identified. Appendix B provides a brief overview the hand crafted knowledge bases in Table 6.2. Initial testing demonstrated that our system did not return models for the larger knowledge bases **CMMI** and **unit-ontology** within two hours (before defeasible axioms or inconsistencies had been added). These knowledge bases were deemed unsuitable for the evaluation. Three techniques for sourcing additional knowledge bases were discussed with the repository team at Manchester:

1. Creating knowledge bases from entirely random but syntactically conformant $\mathcal{ALC}$ ABox and TBox axioms;
2. Taking more expressive knowledge bases and cutting out all the axioms that involved constructs not expressible in $\mathcal{ALC}$;
3. Constructing new knowledge bases from *fragments* of existing $\mathcal{ALC}$ knowledge bases.

The first technique, creating entirely random knowledge bases, does not achieve this because most knowledge bases include some structure (for instance, the TBox may be definitorial and/or hierarchical) and many reasoning systems are optimised to take such properties into account. The second technique raised some questions over its legitimacy because the available constructs may impact the structure of the knowledge base during the authoring process, hence what remains may not be representative of $\mathcal{ALC}$ knowledge bases. The third approach is challenging to implement, since the presence of internal structures make establishing the criteria for what constitutes a reasonable *fragment* is non-trivial. We decided on a compromise, to synthesise randomised knowledge bases that are built to reflect the internal structures observed in knowledge bases from the corpora.

6.4 Synthetic knowledge bases

The synthetic knowledge bases are generated using the Java OWL-API [HB11]. The API provides a programmatic interface to create and modify a knowledge base and allows a description logic reasoner to be instantiated to perform reasoning on the developing knowledge base.

We exploit the OWL-API explanation library [HPS08] which utilises the reasoner to compute justifications for a specified entailment.

6.4.1 Generating a consistent knowledge base

Our aim is to generate knowledge bases that broadly reflect those within the corpora. We begin with an informal overview of structures observed in knowledge bases and introduce some notation to refer to these structures.

As discussed in Chapter 2, terminologies are hierarchical structures. The TBox can be viewed as a tree structure with the $\top$ concept forming the root of the tree. The first level includes a set of “foundational” concepts that we call *primary concepts*. Primary concepts are usually made disjoint as they reflect fundamentally different notions within the domain of interest. Below each primary concept, sits a branching hierarchy of more specialised concepts that we call the *sub-concepts* of the primary concept. We refer to a primary concept together with its sub-concepts as a *sub-domain*. For instance, a knowledge base might include three sub-domains, rooted in the primary concepts *Animal*, *Vegetable* and *Mineral*. For brevity, we may refer to a sub-domain by the name of its primary concept.

A recurring structure found in many knowledge bases are “distinguishing properties” that we call *features*. Features are represented in three parts, (1) a concept (the *feature concept*) that captures the notion being represented, (2) a set of pairwise disjoint sub-concepts (the *sub-features*) that represent distinguishing characteristics of the property, (3) a specialised role (a *feature role*) that associates individuals with the characteristics and is restricted for use with this property. Named individuals are rarely asserted to a feature concept or its sub-concepts. The existence of such (unnamed) individuals is inferred by the existentially quantified role restriction. For instance, the property of electrical charge can be captured as (1) the feature concept *Charge*, (2) a disjoint union of the sub-features *Positive* and *Negative* expressed by $Charge \equiv Positive \sqcup Negative$ and $Positive \sqsubseteq \neg Negative$, together with (3) the feature role *hasCharge* range restricted to *Charge* by $\top \sqsubseteq \forall hasCharge.Charge$. Now the properties of

an anion can be captured using $Anion \sqsubseteq \exists hasCharge.Positive$. For brevity, we may refer to feature by the name of the feature concept.

The roles that represent relationships between individuals in the sub-domains are usually domain and role restricted. For instance, $hasDaughter$ might be domain restricted to $Human$ and range restricted to $Woman$. We define a *relation* as a role together with a range and a domain restriction and again refer to a relation using the name of the role. We refer to primitive definitions for sub-domain concepts as an *embellishment* of the concept. They capture key characteristics associated with the concept. For instance $Anion \sqsubseteq hasCharge.Positive$ is an embellishment for the concept $Anion$. Finally, we call non-primitive definitions that are used to capture specialised concepts *equivalances*. These often represent the “outputs” of a knowledge base. For instance, $Interesting \equiv \forall hasThis.That \sqcap \exists partOf.Theother$.

Example 6.4. *Following the patterns we have described, the primary concepts could be given as $Human$ and $Vehicle$ with the $Vehicle$ sub-domain including the sub-concepts BMW and $Reliant$. A feature representing the notion of colour is captured by the concept $Colour$, a disjoint union of the sub-features Red , $Yellow$ and $Blue$; together with a role $hasColour$. Similarly the notion of speed is captured by the concept $Speed$, a disjoint union of sub-features $Fast$ and $Slow$; with the role $hasSpeed$. Now the concept “red BMW” can be expressed as $BMW \sqcap \exists hasColour.Red$ and we can add embellishments such as $BMW \sqsubseteq hasSpeed.Fast$ and $Reliant \sqsubseteq hasSpeed.Slow$. The driving relation is expressed by the role $Drives$, the domain restriction $\exists Drives.\top \sqsubseteq Human$ and the range restriction $\top \sqsubseteq \forall Drives.Manufacturer$.*

Each synthetic $\mathcal{ALC}$ knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ is generated using our java tool OWLGen under the control of non-negative integer parameters **s**, **f**, **r**, **m**, **e**, **b**, **d**, **c**, **i** and **a**. The parameters are used as follows:

1. **s** determines the number of *sub-domains* generated, where the parameters **b** and **d** control the maximum branch factor and maximum branch depth for the subtrees;
2. **f** determines the number of *features* generated, where the parameter **b** determines the maximum number of sub-features;

3. **r** determines the number of *relations* generated;
4. **m** determines the number of *embellishments* added, where the parameter **c** governs the complexity of concepts generated¹;
5. **e** determines the number of *equivalences* generated, where the parameter **c** governs the complexity of concepts generated;
6. **i** determines the number of individuals created;
7. **a** determines the size of the ABox generated, a set of **a** (concept or role) assertions, where the parameter **c** governs the complexity of the concepts generated.

In addition to the generation of knowledge bases, OWLGen provides a range of features that can be used to extract information from knowledge bases represented in OWL syntax.

An overview of the structure of a synthetic TBox is illustrated in Figure 6.1. The method of construction for each part of the knowledge base is given next.

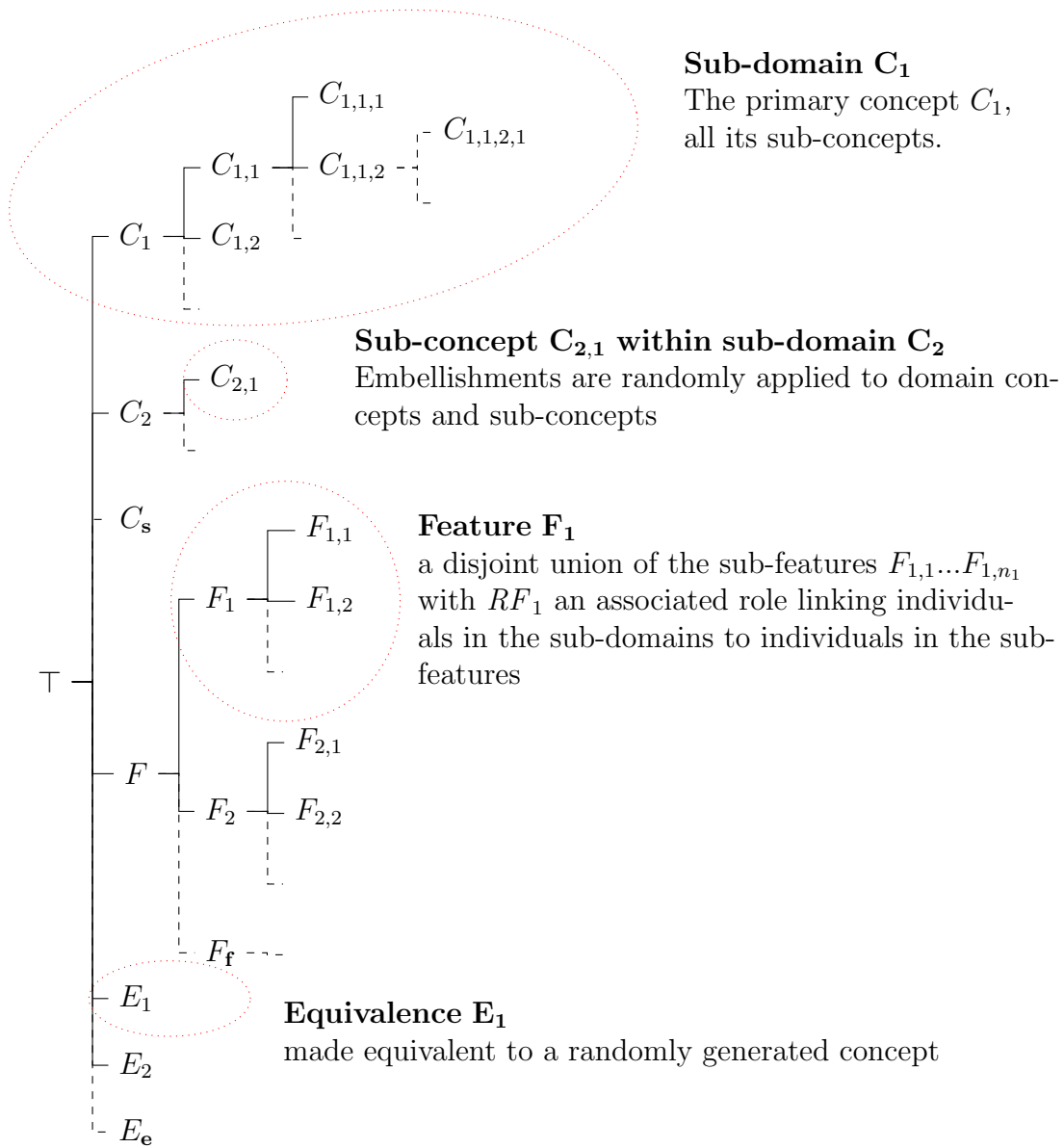
The sub-domains are constructed as follows:

- (i) **s** *primary concepts* named $C_1 \dots C_s$ are added to the signature;
- (ii) Each primary concept C_x , where $1 \leq x \leq s$, is made pairwise disjoint from the other primary concepts by adding the axioms

$$C_a \sqsubseteq \neg C_b \text{ where } a, b \leq s \text{ and } a \neq b$$

- (iii) Each primary concept is randomly sub-divided into a hierarchy of *sub-concepts* using a specified maximum branching depth (parameter **d**) and maximum branching factor (parameter **b**). The sub-concept names are added to the signature and take the form $C_{e_1, e_2, \dots, e_m}$ which uniquely identifies a sub-concept at a depth m .

¹**c** determines the maximum number of concepts that are generated within disjunctions and conjunctions.



The structure generated is controlled by a set of parameters:

- s** determines the number of sub-domains created;
- f** determines the number of features created;
- r** determines the number of relations created;
- m** determines the number of embellishments added
- e** determines the number of equivalences added
- b** determines the maximum branch factor of sub-domains and features;
- d** determines the maximum branch depth of sub-domains;
- c** determines the complexity of concepts generated.

Figure 6.1: The hierarchical structure of a TBox in a synthetic knowledge base.

- (iv) Each sub-concept $C_{e_1, \dots, e_{x-1}, e_x}$ where $2 \leq x \leq m$ is placed in the hierarchy by the axiom

$$C_{e_1, \dots, e_{x-1}, e_x} \sqsubseteq C_{e_1, \dots, e_{x-1}}$$

The features are constructed as follows:

- (i) The concept F and $\mathbf{f}$ *feature concepts* named $F_1 \dots F_{\mathbf{f}}$ are added to the signature, together with associated roles: $RF_1 \dots RF_{\mathbf{f}}$.
- (ii) Each feature F_x , where $x \leq \mathbf{f}$, is randomly sub-divided into a set of n_x *sub-features* $F_{x,y}$ where $2 \leq n_x \leq \mathbf{b}$ and $y \leq n_x$. These sub-features are added to the signature.
- (iii) Each feature F_x is made into a sub-concept of concept F and equivalent to the *disjoint union* of its sub-features by adding:

$$F_x \sqsubseteq F$$

$$F_x \equiv F_{x,1} \sqcup \dots \sqcup F_{x,n_x}$$

$$F_{x,a} \sqsubseteq \neg F_{x,b} \text{ where } a, b \leq n_x \text{ and } a \neq b$$

- (iv) The range of the associated role RF_x is restricted to the feature F_x by adding:

$$\top \sqsubseteq \forall RF_x.F_x$$

The relations are constructed as follows:

- (i) $\mathbf{r}$ roles named $R_1 \dots R_r$ are added to the signature.
- (ii) For each role R_x where $1 \leq x \leq \mathbf{r}$, the role is restricted to a domain D_x and a range G_x , concepts chosen randomly from the set of all sub-domains. The domain and range restrictions are expressed by the axioms:

$$\exists R_x.\top \sqsubseteq D_x$$

$$\top \sqsubseteq \forall R_x.G_x$$

The function **Concept** shown in Algorithm 4 is used to generate randomised complex concepts based on the signature. A priori knowledge of the TBox structure enables generation of candidate concepts that are likely to be satisfiable w.r.t. the TBox. However, the satisfiability of each generated candidate concept is checked against the evolving TBox using the active reasoner. If the generated concept is unsatisfiable then the candidate concept is rejected and an alternative concept is generated.

Algorithm 4: Concept

Input: $\langle \mathcal{S}, \mathcal{R}, \mathcal{F}, \mathcal{R}_F \rangle$
Input: S a sub-domain concept or *null*
Output: A random concept

```

1  $\mathcal{Z} := \{C, \neg C, C \sqcap D, C \sqcup D, \exists R.F, \forall R.F\}$  ;
2 if  $\text{Dom}(\mathcal{R}, S) \neq \emptyset$  then
3   |  $\mathcal{Z} := \mathcal{Z} \cup \{\exists R.S, \forall R.S\}$ ;
4 end
5 switch Choose( $\mathcal{Z}$ ) do
6   | case  $C$  do
7     | return Choose( $\mathcal{S}$ )
8   | case  $\neg C$  do
9     | return  $\neg$ Choose( $\mathcal{S}$ )
10  | case  $\exists R.F$  do
11    |  $R := \text{Choose}(\mathcal{R}_F)$  ;
12    | return  $\exists R.\text{Choose}(\text{Rng}(\mathcal{F}, R))$ 
13  | case  $\forall R.F$  do
14    |  $R := \text{Choose}(\mathcal{R}_F)$  ;
15    | return  $\forall R.\text{Choose}(\text{Rng}(\mathcal{F}, R))$ 
16  | case  $\exists R.S$  do
17    |  $R := \text{Choose}(\text{Dom}(\mathcal{R}, S))$  ;
18    | return  $\exists R.\text{Choose}(\text{Rng}(\mathcal{S}, R))$ 
19  | case  $\forall R.S$  do
20    |  $R := \text{Choose}(\text{Dom}(\mathcal{R}, S))$  ;
21    | return  $\forall R.\text{Choose}(\text{Rng}(\mathcal{S}, R))$ 
22 end
```

Concept takes as inputs: a tuple of sets $\langle \mathcal{S}, \mathcal{R}, \mathcal{F}, \mathcal{R}_F \rangle$ where $\mathcal{S}$ is the set of all sub-domain concepts, $\mathcal{R}$ is the set of all relations, $\mathcal{F}$ is the set of all features and $\mathcal{R}_F$ is the set of all feature roles; and a concept name S , where $S \in \mathcal{S}$ or is *null*. The function **Choose**($\mathcal{C}$) randomly selects exactly one element from a given set $\mathcal{C}$. The functions **Dom** and **Rng** are used to restrict the

generator to concepts that satisfy the domain and range constraints of the specified role. The function $\text{Dom}(\mathcal{R}, S)$ returns $\mathcal{R}$ if S is null, otherwise the set of roles for which S belongs to the same sub-domain as concept C_D , that appears within the domain constraint $\exists R.\top \sqsubseteq C_D \in \mathcal{T}$. When S is given, there is no guarantee that any relations exist where S satisfies the associated domain constraint. In such cases, role restrictions involving relations are not returned. The function $\text{Rng}(\mathcal{C}, R)$ returns the set of concepts in $(\mathcal{C} \cup \{\neg C \mid C \in \mathcal{C}\})$ that are not forbidden by C_R in $\top \sqsubseteq R.C_R$, the range constraint associated with R .

The embellishments are constructed as follows:

- (i) Each concept to be embellished, $M_n \in \mathcal{S}$, is chosen at random where $1 \leq n \leq \mathbf{m}$ and $\mathcal{S}$ denotes the concepts in the sub-domain hierarchies. For each M_n , a number x_n is chosen at random such that $1 \leq x_n \leq \mathbf{c}$.
- (ii) Concepts $S_1, \dots, S_{x_n}$ are generated using $\text{Concept}(\langle \mathcal{S}, \mathcal{R}, \mathcal{F}, \mathcal{R}_F \rangle, M_n)$.
- (iii) The embellishment Z_n is the axiom:

$$M_n \sqsubseteq S_1 \sqcap \dots \sqcap S_{x_n}$$

- (iv) M_n is added to $\mathcal{T}$ if M_n is satisfiable w.r.t. $\mathcal{T}$, otherwise an alternative is generated.

Examples of embellishments: $C_{1,2,1} \sqsubseteq C_3$ or $C_{4,2,1} \sqsubseteq C_3 \sqcap (\exists R_2.C_5) \sqcap \neg C_{1,2}$.

The equivalences are constructed as follows:

- (i) $\mathbf{e}$ concepts named $E_1 \dots E_{\mathbf{e}}$ are added to the signature.
- (ii) For each concept E_n , $1 \leq n \leq \mathbf{e}$, a randomised disjunction of the form $D_1 \sqcup \dots \sqcup D_x$ is generated such that $1 \leq x \leq c$ and each D_y where $1 \leq y \leq x$ is a randomised conjunction of the form $C_1 \sqcap \dots \sqcap C_{z_y}$ such that $1 \leq z_y \leq c$ and each concept C_w where $1 \leq w \leq z_y$ is a concept generated using $\text{Concept}(\langle \mathcal{S}, \mathcal{R}, \mathcal{F}, \mathcal{R}_F \rangle, \emptyset)$.

(iii) The equivalence Z_n is the axiom:

$$E_n \equiv D_1 \sqcup \dots \sqcup D_x$$

(iv) Z_n is added to $\mathcal{T}$ if E_n is satisfiable w.r.t. $\mathcal{T}$, otherwise an alternative is generated.

Examples of equivalences: $E_1 \equiv \exists RF_1.F_{1,2}$ or $E_4 \equiv (\exists R_3.C_{2,1}) \sqcup (\neg C_{3,2} \sqcap C_{4,2,1})$.

The ABox is constructed from **a** assertions constructed as follows:

- (i) Randomly select an individual $i \in N_I$ and generate Z an axiom of one of four types chosen at random:
 - (a) $S(a)$ where $S \in \mathcal{S}$ is randomly chosen;
 - (b) $\neg S(a)$ where $S \in \mathcal{S}$ is randomly chosen;
 - (c) $C(a)$ where C is generated using $\text{Concept}(\langle \mathcal{S}, \mathcal{R}, \mathcal{F}, \mathcal{R}_F \rangle, \emptyset)$;
 - (d) $R(i, j)$ where $j \in N_I$ is randomly chosen.
- (ii) Z is added to the ABox $\mathcal{A}$ if $\mathcal{K} \cup \{Z\}$ is consistent, otherwise an alternative is generated.

6.4.2 Generating an inconsistent knowledge base

Comparing the reasoning performance of a consistent knowledge base with the performance of the knowledge base with inconsistencies is challenging due to interdependencies between multiple variables. We addressed the dependency on the numbers of defeasible axioms by making each axiom involved in at least one inconsistency defeasible. We also require a method that does not materially change the number, the size or the complexity of the axioms in the knowledge base as inconsistencies are introduced.

Simply adding additional axioms to make the knowledge base inconsistent does not meet this goal if the number of axioms added is significant w.r.t. the numbers of axioms in the knowledge base. Similarly, adopting a simple one-in, one-out strategy may not meet the goal if the axioms

are not of comparable size and complexity. Our mitigation for this is to identify candidate axiom substitutions by randomly changing one name in a randomly selected ABox axiom to a different and randomly selected name from the signature having the same type. Algorithm 5 shows our initial design for introducing t inconsistencies into an $\mathcal{ALC}$ knowledge base. The output is a uniform $p\text{-}\mathcal{ALC}$ knowledge base in which each axiom involved in at least one inconsistency is made defeasible. Informally, given a knowledge base $\mathcal{K}$, we try to introduce n inconsistencies by randomly selecting and then modifying an ABox axiom. We then check to see if the modified axiom appears within all possible justifications for the inconsistency of $\mathcal{K}$. If it does, we keep the modified axiom and then repeat the process until we have made n changes.

Algorithm 5: MakeInconsistent

Input: $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \emptyset, \emptyset \rangle$, a consistent input knowledge base.
Input: t the target number of inconsistencies to be introduced.
Output: $\langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ an inconsistent $p\text{-}\mathcal{ALC}$ knowledge base.

```

1  $\mathcal{A}_d ::= \emptyset;$ 
2  $\mathcal{T}_d ::= \emptyset;$ 
3  $\mathcal{S} ::= \mathcal{A};$ 
4  $m ::= \text{MAX\_TRIES};$ 
5  $c ::= 0;$ 
6 while  $c < t$  and  $\mathcal{S} \neq \emptyset$  and  $m > 0$  do
7    $Z ::= \text{Choose}(\mathcal{S});$ 
8    $e ::= \text{Choose}(\text{sig}(Z));$ 
9    $\bar{Z} ::= \text{Replace}(Z, e, \text{Alt}(e, \text{sig}(\mathcal{K})));$ 
10   $\mathcal{A} ::= (\mathcal{A} \setminus \{Z\}) \cup \{\bar{Z}\};$ 
11   $\mathcal{E} ::= \text{Explain}(\langle \mathcal{A}, \mathcal{T} \rangle, \top \sqsubseteq \perp, \text{MAX\_EX});$ 
12  if  $\bar{Z} \in \mathcal{E}$  then
13     $\mathcal{S} ::= \mathcal{S} \setminus \{Z\};$ 
14     $c ::= c + 1;$ 
15     $m ::= 0;$ 
16  else
17     $\mathcal{A} ::= (\mathcal{A} \setminus \{\bar{Z}\}) \cup \{Z\};$ 
18     $m ::= m - 1;$ 
19  endif
20 end
21  $\mathcal{E} ::= \text{Explain}(\langle \mathcal{A}, \mathcal{T} \rangle, \top \sqsubseteq \perp);$ 
22 for  $Z \in \mathcal{E}$  do
23   if  $Z \in \mathcal{A}$  then
24      $\mathcal{A} = \mathcal{A} \setminus \{Z\};$ 
25      $\mathcal{A}_d = \mathcal{A}_d \cup \{Z^{[1]}\};$ 
26   else if  $Z \in \mathcal{T}$  then
27      $\mathcal{T} = \mathcal{T} \setminus \{Z\};$ 
28      $\mathcal{T}_d = \mathcal{T}_d \cup \{Z^{[1]}\};$ 
29   end
30 return  $\langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ 

```

Algorithm 6: MakeInconsistent (close approximation)

Input: $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$, a consistent $\mathcal{ALC}$ knowledge base.
Input: t the target number of inconsistencies to be introduced.
Output: $\langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ an inconsistent $p\text{-}\mathcal{ALC}$ knowledge base.

```

1  $\mathcal{A}_d ::= \emptyset;$ 
2  $\mathcal{T}_d ::= \emptyset;$ 
3  $\mathcal{E} ::= \emptyset;$ 
4  $m ::= \text{MAX\_TRIES};$ 
5  $c ::= 0;$ 
6 while  $c < t$  and  $\mathcal{A} \neq \emptyset$  and  $m > 0$  do
7    $Z ::= \text{Choose}(\mathcal{A});$ 
8    $e ::= \text{Choose}(\text{sig}(Z));$ 
9    $\bar{Z} ::= \text{Replace}(Z, e, \text{Alt}(e, \text{sig}(\mathcal{K})));$ 
10   $\mathcal{A} ::= (\mathcal{A} \setminus \{Z\}) \cup \{\bar{Z}\};$ 
11  if  $\text{Inconsistent}(\langle \mathcal{A}, \mathcal{T} \rangle)$  then
12     $\mathcal{E} = \text{Explain}(\langle \mathcal{A}, \mathcal{T} \rangle, \top \sqsubseteq \perp, \text{MAX\_EX});$ 
13    for  $Z \in \mathcal{E}$  do
14      if  $Z \in \mathcal{A}$  then
15         $\mathcal{A}_d = \mathcal{A}_d \cup Z^{[1]};$ 
16      else if  $Z \in \mathcal{T}$  then
17         $\mathcal{T}_d = \mathcal{T}_d \cup Z^{[1]};$ 
18      end
19     $\mathcal{A} ::= \mathcal{A} \setminus \{\bar{Z}\};$ 
20     $c ::= c + 1;$ 
21     $m ::= 0;$ 
22  else
23     $\mathcal{A} ::= (\mathcal{A} \setminus \{\bar{Z}\}) \cup \{Z\};$ 
24     $m ::= m - 1;$ 
25  endif
26 end
27 return  $\langle \mathcal{A} \setminus \mathcal{A}_d^{-W}, \mathcal{T} \setminus \mathcal{T}_d^{-W}, \mathcal{A}_d, \mathcal{T}_d \rangle$ 

```

The function $\text{Replace}(Z, a, b)$ is used to replace all occurrences of name a in an axiom Z with name b . The function $\text{Alt}(\text{SIG}, a)$ randomly selects an alternative name b for name a from a supplied signature $\text{SIG} = \langle N_I, N_C, N_R \rangle$ such that $a \neq b$, $a \in N_I \rightarrow b \in N_I$, $a \in N_C \rightarrow b \in N_C$ and $a \in N_R \rightarrow b \in N_R$. The constant MAX_TRIES is used to ensure that the algorithm terminates where no inconsistencies can be introduced by this method. For instance, if the knowledge base includes no ABox. For brevity, the error handling has been omitted from the algorithm. The

variable $\mathcal{S}$ records the set of ABox axioms that are available to swap out. $\mathcal{S}$ is initialised with the ABox $\mathcal{A}$ and the algorithm terminates if this becomes empty. The function $\text{Explain}(\mathcal{K}, Z, \text{max})$ is used to obtain the set of axioms involved in the entailment $\mathcal{K} \models Z$ where the bound max is imposed by the OWL Explanation API that used to implement this function. Explain computes the union of max justifications for $\mathcal{K} \approx Z$. The bound max is required because the search is not guaranteed to be complete and may not terminate. The justifications for axiom $\mathcal{K} \models \top \sqsubseteq \perp$ provides a (signature independent) mechanism to obtain all the axioms that contribute to the inconsistency of $\mathcal{K}$. Within the main loop of Algorithm 5 (lined 6-20) a swappable ABox axiom Z is chosen (line 7) and one name in its signature replaced to form $\bar{Z}$ (lines 8-9) and replaces Z in the $\mathcal{A}$ (line 10). The axioms involved in inconsistencies of the modified knowledge base $\langle \mathcal{A}, \mathcal{T} \rangle$ are obtained (line 11). If the modified axiom $\bar{Z}$ is involved in at least one inconsistency (line 12), the change is accepted, Z removed from those available for swap $\mathcal{S}$ (line 13) and the counters updated (lines 14-15). Otherwise, the change is rejected, the original axiom is restored (line 17) and the retry counter decremented (line 18). This continues until c swaps are completed. Finally, all the axioms involved in the consistency (line 21) of $\mathcal{K}$ are made defeasible (lines 23-28).

Two problems were encountered with Algorithm 5. Firstly, many of the guessed changes did not increase the distance of the knowledge base due to overlap between inconsistencies. Secondly, the explanations interface used by the Explain function takes as an input the number of justifications to generate. There is no simple way of predicting the number of justifications required to allow all axioms involved in inconsistency to be identified and identifying every inconsistent axiom during each iteration becomes unfeasible as the number of inconsistencies increased. Algorithm 6 seeks to address this by leaving out each of the modified axioms (see lines 10 and 19) which restores the satisfiability of the knowledge base while other candidate changes are generated. A new function $\text{Inconsistent}(\mathcal{K})$ is used (line 11) to check the satisfiability of the knowledge base using the active reasoner. The justifications for each inconsistency are computed (lines 12) and defeasible axioms recorded (lines 13-17) before removing the axiom from the ABox (line 19). This approach does not guarantee that every change leads to an increase in the n -inconsistency, but it does so more often than Algorithm 5. However, when using

this technique, some of the axioms that are involved in inconsistencies are not identified and hence are not made defeasible. These are due to inconsistencies arising from combinations of the axioms that were temporarily removed whilst computing the justifications. Our synthetic knowledge bases were generated using a $\text{MAX_EX} = 20$. For the purposes of understanding the scaling of the algorithm the technique serves as an approximation. An analysis of the variance between Algorithms 5 and 6 is provided in Experiment 2.

6.4.3 Generating queries

Recall, from Theorem 3.13, that to show that a query Q is a consequence of an n -inconsistent p - $\mathcal{ALC}$ knowledge base $\mathcal{K}$ requires checking if $\mathcal{K} \cup \{\neg Q\}$ has an n -distant preferred interpretation. We generate each candidate query Q to evaluate refutation proofs using a similar technique to introducing inconsistencies and rely on an active reasoner. If $\neg Q$ appears in at least one justification of the $\mathcal{ALC}$ knowledge base $\mathcal{K}^{-W} \cup \{\neg Q\} \models \top \sqsubseteq \perp$ then Q is *accepted* as a candidate for checking if $\mathcal{K} \approx Q$. Similarly, if candidate Q appears in no justifications of $\mathcal{K}^{-W} \cup \{\neg Q\} \models \top \sqsubseteq \perp$ is *accepted* as a candidate for checking if $\mathcal{K} \not\approx Q$. The function **Query** is used to generate the candidate queries and is formalised in Algorithm 7.

Algorithm 7: Query

Input: $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$, a credible p - $\mathcal{ALC}$ knowledge base.

Input: v the target outcome for the proof, either *True* or *False*

Output: Q a candidate query.

```

1  $m ::= \text{MAX\_TRIES};$ 
2 while  $m > 0$  do
3    $Q = \text{Concept}(\text{Spec}(\mathcal{K}), \emptyset);$ 
4   if  $\text{Inconsistent}(\langle \mathcal{A} \cup \mathcal{A}_d^{-W} \cup \neg Q, \mathcal{T} \cup \mathcal{T}_d^{-W} \rangle)$  then
5      $\mathcal{E} = \text{Explain}(\langle \mathcal{A} \cup \mathcal{A}_d^{-W} \cup \neg Q, \mathcal{T} \cup \mathcal{T}_d^{-W} \rangle, \top \sqsubseteq \perp, \text{MAX\_EX});$ 
6     if  $\neg Q \in \mathcal{E}$  and  $v$  is True then
7       return  $Q$ ;
8     else if  $\neg Q \notin \mathcal{E}$  and  $v$  is False then
9       return  $Q$ ;
10  else
11    if  $v$  is False then
12      return  $Q$ ;
13   $m ::= m - 1;$ 
14 end
```

The function **Spec** is used to obtain $\langle \mathcal{S}, \mathcal{R}, \mathcal{F}, \mathcal{R}_F \rangle$, the characteristics of the generated structure of the knowledge base (**Concept** uses $\langle \mathcal{S}, \mathcal{R}, \mathcal{F}, \mathcal{R}_F \rangle$ to guide the generation of concepts). The candidates for $\mathcal{K} \approx Q$ are not guaranteed to be preferred consequences of $\mathcal{K}$ because the presence of an axiom in *at least one* justification is not sufficient to guarantee that it is entailed. If we had selected axioms that appear in every explanation instead, then such axioms would always be entailed. However, this is too strong for our purpose when the distribution of non-defeasible and defeasible axioms with their associated weights are taken into account. We illustrate the operation of the algorithm through the following example.

Example 6.5. Let $\mathcal{K}_q = \langle \emptyset, \{C \sqsubseteq \neg E, D \sqsubseteq \neg E\}, \{D(x)^{[1]}, E(x)^{[1]}\}, \emptyset \rangle$ be a $p\text{-}\mathcal{ALC}$ knowledge base. Suppose we try to generate a query that is a preferred consequence of $\mathcal{K}_q$ using $\text{Query}(\mathcal{K}_q, \text{True})$ implemented by Algorithm 7. Let $Q = \neg C(x)$ be the axiom chosen by Algorithm 7 at line 3, for a query that will be proved for $\mathcal{K}_q$. Let $\mathcal{K}'_q$ denote the $\mathcal{ALC}$ knowledge base $\mathcal{K}_q^{-W} \cup \{\neg Q\} = \langle \{C(x), D(x), E(x)\}, \{C \sqsubseteq \neg E, D \sqsubseteq \neg E\} \rangle$. $\mathcal{K}'_q$ is inconsistent at line 4. There are two justifications of $\mathcal{K}'_q \models \top \sqsubseteq \perp$: $\mathcal{J}_1 = \{D(x), E(x), D \sqsubseteq \neg E(x)\}$ and $\mathcal{J}_2 = \{C(x), E(x), C \sqsubseteq \neg E(x)\}$. At line 5, $\mathcal{E} = \{C(x), D(x), E(x), C \sqsubseteq \neg E, D \sqsubseteq \neg E\}$. Now $\neg Q = C(x)$, $C(x) \in \mathcal{E}$ and v is True, hence Q is accepted as a candidate and returned at line 7. However, $\mathcal{K}_q$ is credible and 1-inconsistent (each preferred interpretation falsifies one of the defeasible axioms $D(x)^{[1]}$ or $E(x)^{[1]}$). In particular, the Herbrand interpretation $\{C(x), D(x), \neg E(x)\}$ of $\mathcal{K}_q$ is 1-distant. Hence $\mathcal{K}_q \not\approx \neg C(x)$.

In contrast, consider $\mathcal{K}''_q = \langle \emptyset, \{C \sqsubseteq \neg E, D \sqsubseteq \neg E\}, \{D(x)^{[1]}, E(x)^{[2]}\}, \emptyset \rangle$. $\mathcal{K}''_q$ is again 1-inconsistent but all interpretations that falsify $E(x)^{[2]}$ are 2-distant and $\mathcal{K}'' \approx \neg C(x)$.

The choice of **MAX_EX**, the upper bound for the number of explanations **Explain** searches for, may lead to some candidates for checking $\mathcal{K} \not\approx Q$ being accepted if $\neg Q$ appears in an explanation that was not found. In the experiments that follow, such cases present as failures to prove the generated query.

<i>KB</i>	$\mathcal{A}$	$\mathcal{T}$	N_I	N_C	N_R	CI	EQ	DC	s	f	r	m	e	b	d	c
$\mathcal{K}_{sma}$	20	107	20	75	6	86	8	4	3	3	3	20	5	4	4	2
$\mathcal{K}_{med}$	40	280	20	171	6	258	9	4	4	4	2	97	5	6	6	2

Table 6.3: Characteristics of the knowledge bases $\mathcal{K}_{sma}$ and $\mathcal{K}_{med}$. $|\mathcal{A}|$ and $|\mathcal{T}|$ are the number of axioms in the ABox and TBox; N_I , N_C and N_R are the numbers of individuals, concepts and roles in the signature; s, f, r, m and e are the input parameter values that were supplied to the synthetic knowledge base generator.

6.5 Experiments

All experiments were conducted on a workstation based on a 64bit Intel(R) Core(TM) i7-2600 CPU running at 3.40 GHz with 16GB of RAM and running the Debian-based Ubuntu linux version 14.04 (LTS), [<http://releases.ubuntu.com/14.04/>]. Version 4.5.3 of the *clingo* answer set solver was compiled to target the 64bit address model, support multi-threaded solving via Intel Thread Building Blocks (TBB) version 4.2 [Rei07] and support Lua scripting version 5.1.5 [<http://www.lua.org>]. Two synthetic knowledge bases (`sma.owl` and `med.owl`) were created for the purposes of the performance evaluations. The choice for suitable sizing for these knowledge bases was determined by gradually increasing the size and complexity of the generated candidate knowledge bases to find ones that the experiments could be run on the hardware using either the single threaded solver or the multi-threaded solver. The ontology `sma.owl` containing $\mathcal{K}_{sma}$ was used in both single and multi-threaded tests; `med.owl` containing $\mathcal{K}_{med}$ was used only in the multi-threaded tests. The characteristics of the knowledge bases are summarised in Table 6.3.

In the experiments, three tasks are evaluated for each $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}$: finding the n -inconsistency of the knowledge base, a proof by refutation for a query that is expected to be a consequence and a proof by refutation for a query that is not expected to be a consequence. To “find- n ”, the solver searches for one optimal answer set of $ASP(\mathcal{K})$ and records the returned optimality as n . For the refutation proofs, $\text{Query}(\mathcal{K}, \text{False})$ and $\text{Query}(\mathcal{K}, \text{True})$ are used to generate candidate queries. By Theorem 3.13 and the correspondence of Chapter 5, $\mathcal{K} \models Q$ iff there are no n -optimal models of $ASP(\mathcal{K} \cup \{\neg Q\})$. For each refutation the solver searches for an n -optimal model of $ASP(\mathcal{K} \cup \{\neg Q\})$ using the generated query and the n obtained from the first task. As noted in Section 6.4.3, the generated queries for proofs that are expected to

be consequences are not guaranteed. Discrepancies are reported in our experimental results as **proof-failed**. No absolute time limit was placed on the experiments. Instead, a **time-out** was recorded if the number of grounding iterations exceeded 40 iterations. This has the effect of curtailing branches in which very large numbers of parameters have been introduced. A justification for choosing this approach is provided in the discussion at the end of this section.

6.5.1 Experiment 1: Defeasible axioms

This experiment was designed to explore the impact on reasoning performance of making axioms defeasible within a consistent knowledge base. Informally, we create suitable candidate knowledge bases by taking an $\mathcal{ALC}$ knowledge base and making different percentages of the non-defeasible axioms defeasible by assigning them a weight of 1.

For a $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d, \mathcal{T}_d \rangle$ the percentage of defeasible axioms in $\mathcal{K}$ is given by $(|\mathcal{A}_d \cup \mathcal{T}_d|) / (|\mathcal{A} \cup \mathcal{A}_d \cup \mathcal{T} \cup \mathcal{T}_d|) \times 100$. With a slight abuse of notation, we use $\mathcal{A}^{[w]}$ and $\mathcal{T}^{[w]}$ to denote $\{Z^{[w]} | Z \in \mathcal{A}\}$ and $\{Z^{[w]} | Z \in \mathcal{T}\}$ respectively. Given an input $\mathcal{ALC}$ knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ and an integer percentage x , an x percent defeasible $p\text{-}\mathcal{ALC}$ knowledge base based on $\mathcal{K}$ is $\langle \mathcal{A} \setminus \mathcal{A}_x, \mathcal{T} \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}, \mathcal{T}_x^{[1]} \rangle$ where $\mathcal{A}_x \subseteq \mathcal{A}$, $\mathcal{T}_x \subseteq \mathcal{T}$ and $\text{abs}(x - (|\mathcal{A}_x|/|\mathcal{A}| \times 100))$ and $\text{abs}(x - (|\mathcal{T}_x|/|\mathcal{T}| \times 100))$ are minimized.

The average time taken over 10 runs for each of the three tasks was recorded for versions of the knowledge base $\mathcal{K}_{sma}$ containing between 0% and 100% of defeasible axioms at 10% intervals. In each run r at $x\%$ where $x > 0$, the three reasoning tasks were performed on a (different) x percent defeasible $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}_{x,r}$ created from $\mathcal{K}_{sma}$.

In the results obtained, no tasks were recorded as **time-out** or **proof-failed**. The results are shown in Figure 6.2. We do not expect any discrepancies in candidate queries that were generated for a consistent $\mathcal{ALC}$ knowledge base using queries generated by Algorithm 7. This property is captured by Proposition 6.5. We show that this property holds even with the limitation that $\text{Explain}(\mathcal{K}, Z)$ finds at least one (but not necessarily all) justification(s) for $\mathcal{K} \models Z$. We begin by showing that Algorithm 7 generates candidate queries from a defeasible knowl-

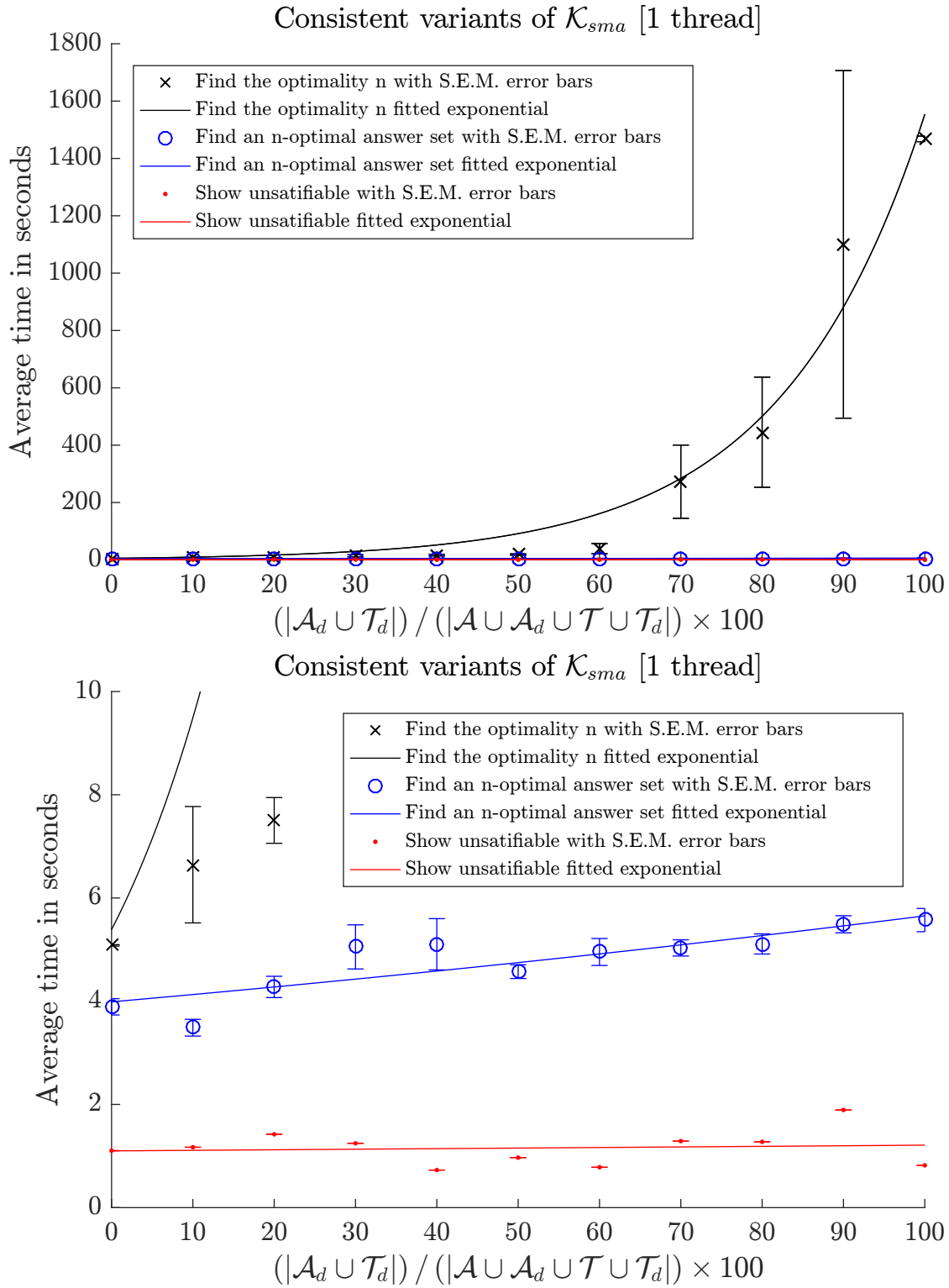


Figure 6.2: Upper graph: The times, averaged over 10 runs, taken to: find an n optimal answer set, disprove entailment by finding an n -optimal answer set and prove entailment by showing unsatisfiability for defeasible knowledge bases created from $\mathcal{K}_{sma}$ with varying percentages of defeasible axioms. Lower graph: exposes detail by restricting the y-axis to 10 seconds.

edge base $\mathcal{K}'$ that reflect the correct entailment by $\mathcal{K}$, the consistent $\mathcal{ALC}$ knowledge base from which $\mathcal{K}'$ was generated (Lemma 6.2). Next, we show that every defeasible knowledge base $\mathcal{K}'$ generated from $\mathcal{K}$ is 0-inconsistent, and that the consequences of $\mathcal{K}$ coincide with the preferred consequences of $\mathcal{K}'$ (Lemmas 6.3 and Lemma 6.4). Finally, since the query was correctly generated and the consequences coincide, the results obtained from checking entailment of a generated query will agree with the intended outcome for the query.

Lemma 6.2. *Let $\mathcal{K}' = \langle \mathcal{A}_k \setminus \mathcal{A}_x, \mathcal{T}_k \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}, \mathcal{T}_x^{[1]} \rangle$ be an $x\%$ defeasible knowledge base derived from a consistent knowledge base $\mathcal{K} = \langle \mathcal{A}_k, \mathcal{T}_k \rangle$ where $0 \leq x \leq 100$, $\mathcal{A}_x \subseteq \mathcal{A}$ and $\mathcal{T}_x \subseteq \mathcal{T}$. Let $\mathcal{Q}$ be query generated by calling $\text{Query}(\mathcal{K}', v)$ implementing Algorithm 7, where $v \in \{\text{True}, \text{False}\}$ and the Algorithm did not reach **MAX_TRIES**. Then, $\mathcal{K} \models \mathcal{Q}$ iff $v = \text{True}$.*

Proof. We show $\mathcal{K} \models \mathcal{Q}$ iff $v = \text{True}$ by considering the operation of the Algorithm 7.

- **The “if” case.** Suppose v is *True*. Let $\mathcal{Q}$ be the candidate query generated in line 3. In line 4, $\langle (\mathcal{A}_k \setminus \mathcal{A}_x) \cup (\mathcal{A}_x^{[1]})^{-W} \cup \neg \mathcal{Q}, (\mathcal{T}_k \setminus \mathcal{T}_x) \cup (\mathcal{T}_x^{[1]})^{-W} \rangle = \mathcal{K} \cup \{\neg \mathcal{Q}\}$. There are two subcases:
 1. $\mathcal{K} \cup \{\neg \mathcal{Q}\}$ is inconsistent. By assumption $\mathcal{K}$ is consistent, hence $\mathcal{K} \models \mathcal{Q}$ and $\neg \mathcal{Q}$ appears in every justification for $\mathcal{K} \cup \{\neg \mathcal{Q}\} \models \top \sqsubseteq \perp$. Hence, for any $\mathcal{E}$ formed from a non-empty subset of justifications for $\mathcal{K} \cup \{\neg \mathcal{Q}\} \models \top \sqsubseteq \perp$, $\neg \mathcal{Q} \in \mathcal{E}$ and $\mathcal{Q}$ is returned at line 7.
 2. $\mathcal{K} \cup \{\neg \mathcal{Q}\}$ is consistent and $\mathcal{Q}$ is rejected at line 11 and an alternative chosen.
- **The “only if” case.** We show the contrapositive. Suppose v is not true. By assumption, v not true implies v is *False*. Let $\mathcal{Q}$ be the candidate query generated in line 3. In line 4, $\langle (\mathcal{A}_k \setminus \mathcal{A}_x) \cup (\mathcal{A}_x^{[1]})^{-W} \cup \neg \mathcal{Q}, (\mathcal{T}_k \setminus \mathcal{T}_x) \cup (\mathcal{T}_x^{[1]})^{-W} \rangle = \mathcal{K} \cup \{\neg \mathcal{Q}\}$. There are two subcases:
 1. $\mathcal{K} \cup \{\neg \mathcal{Q}\}$ was inconsistent. By assumption, $\mathcal{K}$ is consistent, hence $\neg \mathcal{Q}$ appears in every justification for $\mathcal{K} \cup \{\neg \mathcal{Q}\} \models \top \sqsubseteq \perp$. Hence, for any $\mathcal{E}$ formed from a non-empty subset of justifications for $\mathcal{K} \cup \{\neg \mathcal{Q}\} \models \top \sqsubseteq \perp$, $\neg \mathcal{Q} \in \mathcal{E}$ and $\mathcal{Q}$ is rejected at line 8 and an alternative chosen.

2. $\mathcal{K} \cup \{\neg Q\}$ is consistent, $\mathcal{K} \not\models Q$ and Q is accepted and returned at line 12.

□

Lemma 6.3. *Let $\mathcal{K}' = \langle \mathcal{A}_k \setminus \mathcal{A}_x, \mathcal{T}_k \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}, \mathcal{T}_x^{[1]} \rangle$ be an $x\%$ defeasible knowledge base derived from a consistent knowledge base $\mathcal{K} = \langle \mathcal{A}_k, \mathcal{T}_k \rangle$ where $0 \leq x \leq 100$, $\mathcal{A}_x \subseteq \mathcal{A}$ and $\mathcal{T}_x \subseteq \mathcal{T}$. Then $\mathcal{K}'$ is 0-inconsistent.*

Proof. By assumption $\mathcal{K}$ is consistent and by Definition 2.10, there exists at least one model $\mathcal{I}$ of $\mathcal{K}$. $\mathcal{I}$ satisfies every axiom in $\mathcal{A}$ and $\mathcal{T}$. By assumption $\mathcal{K}'$ shares the same signature as $\mathcal{K}$, hence $\mathcal{I}$ is also an interpretation $\mathcal{K}'$. Clearly, $\mathcal{I}$ also satisfies $\mathcal{A} \setminus \mathcal{A}_x, \mathcal{T} \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}$ and $\mathcal{T}_x^{[1]}$. Hence $\mathcal{I}$ is a 0-distant preferred interpretation of $\mathcal{K}'$. We conclude $\mathcal{K}'$ is 0 distant. □

Lemma 6.4. *Let $\mathcal{K}' = \langle \mathcal{A}_k \setminus \mathcal{A}_x, \mathcal{T}_k \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}, \mathcal{T}_x^{[1]} \rangle$ be an $x\%$ defeasible knowledge base derived from a consistent knowledge base $\mathcal{K} = \langle \mathcal{A}_k, \mathcal{T}_k \rangle$ where $0 \leq x \leq 100$, $\mathcal{A}_x \subseteq \mathcal{A}$ and $\mathcal{T}_x \subseteq \mathcal{T}$. Then $\mathcal{K}' \approx Q$ iff $\mathcal{K} \models Q$.*

Proof. We show $\mathcal{K}' \not\models Q$ iff $\mathcal{K} \not\models Q$ by cases.

1. **The “if” case.** Suppose $\mathcal{K}' \not\models Q$. By Lemma 6.3 $\mathcal{K}'$ is 0-inconsistent. By Definition 3.4 there exists at least one 0-distant interpretation $\mathcal{I}$ that satisfies $\neg Q, \mathcal{A} \setminus \mathcal{A}_x, \mathcal{T} \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}$ and $\mathcal{T}_x^{[1]}$. Now, $\mathcal{I}$ satisfies $\mathcal{A}, \mathcal{T}$ and $\neg Q$, hence $\mathcal{K} \not\models Q$.
2. **The “only if” case.** Suppose $\mathcal{K} \not\models Q$. By Definition 2.10 there exists at least one interpretation $\mathcal{I}$ that satisfies $\mathcal{A}, \mathcal{T}$ and $\neg Q$. Now, $\mathcal{I}$ satisfies $\neg Q, \mathcal{A} \setminus \mathcal{A}_x, \mathcal{T} \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}$ and $\mathcal{T}_x^{[1]}$, hence $\mathcal{K}' \not\models Q$.

Now $\mathcal{K}' \approx Q$ iff $\mathcal{K} \models Q$ by the contrapositive. □

Proposition 6.5. *Let $\mathcal{K}' = \langle \mathcal{A}_k \setminus \mathcal{A}_x, \mathcal{T}_k \setminus \mathcal{T}_x, \mathcal{A}_x^{[1]}, \mathcal{T}_x^{[1]} \rangle$ be an $x\%$ defeasible knowledge base derived from a consistent knowledge base $\mathcal{K} = \langle \mathcal{A}_k, \mathcal{T}_k \rangle$ where $0 \leq x \leq 100$, $\mathcal{A}_x \subseteq \mathcal{A}$ and $\mathcal{T}_x \subseteq \mathcal{T}$. Let Q be query generated by calling $\text{Query}(\mathcal{K}', v)$ implementing Algorithm 7, where $v \in \{\text{True}, \text{False}\}$ and the Algorithm did not reach **MAX_TRIES**. Then $\mathcal{K}' \approx Q$ iff $v = \text{True}$.*

Proof. By Lemma 6.4, $\mathcal{K}' \approx Q$ iff $\mathcal{K} \models Q$. By Lemma 6.2, $\mathcal{K} \models Q$ iff $v = \text{True}$. Hence, $\mathcal{K}' \approx Q$ iff $v = \text{True}$. $\square$

The following observations summarise the performance results for the consistent knowledge base $\mathcal{K}_{sma}$:

- Increasing the percentage (and hence number of) defeasible axioms leads to a decrease in performance of all tasks;
- Finding the value of n is significantly harder than performing refutations once n is known;
- Proving a query is significantly faster than failing to prove a query.

Experiment 1 also contributes to our claim that Algorithm 2 is correctly implemented. The results of each of the 220 entailment tests is in agreement with those predicted by the classical reasoner.

6.5.2 Experiment 2: Inconsistency

This experiment examines how variation in the level of inconsistency in a knowledge base impacts reasoning performance. 80 inconsistent knowledge bases were created from $\mathcal{K}_{sma}$ using Algorithm 6. For each target inconsistency level t from 1 to 8, 10 knowledge bases were created using `MakeInconsistent( $\mathcal{K}_{sma}, t$ )`. The three tasks were performed for each knowledge base, recording the executions times, the distance n found, the percentage of defeasible axioms introduced and the results of the refutations.

In the results obtained, no tasks were recorded as `time-out`. 11 of the 90 candidate queries in tasks to show proof of entailment were recorded as `proof-failed`. No candidate queries for disproving entailment were recorded as `proof-failed`. The results for each of the three tasks were grouped by the n -inconsistency retrieved by “find n ” task. Figure 6.3 shows that the average percentage of the axioms involved in inconsistency for each group increases asymptotically towards 43% as n increases (Figure obtained from using MATLAB curve fit). The times taken

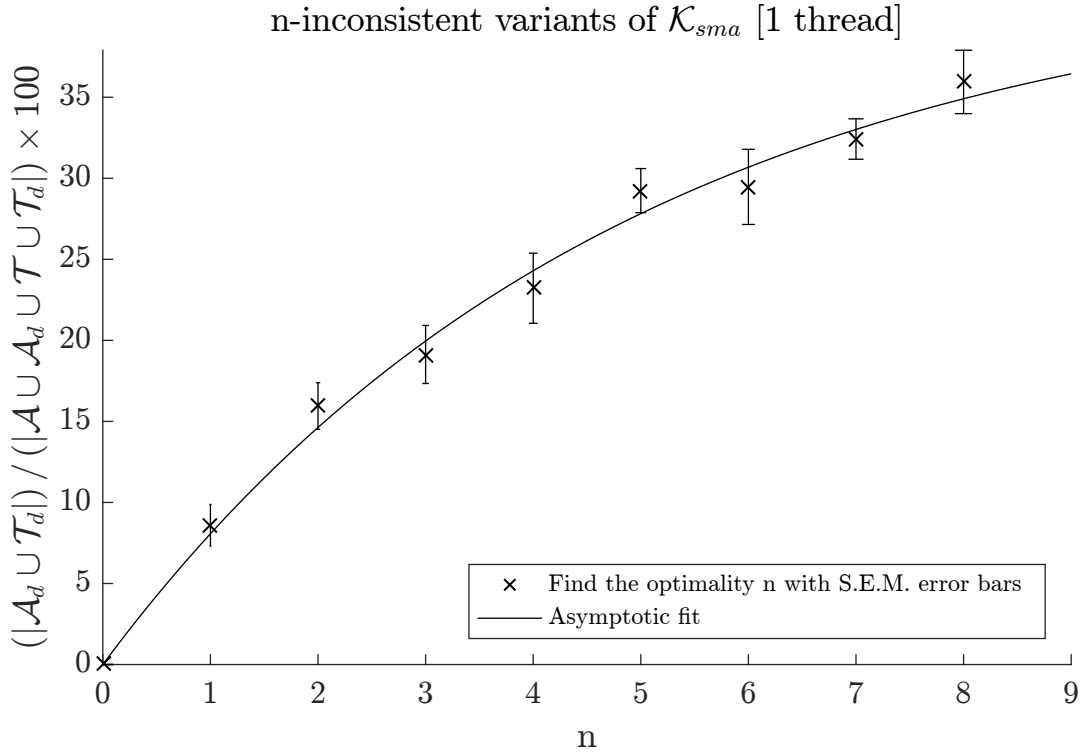


Figure 6.3: The average percentages of defeasibility at each n found in the n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$.

to perform a given reasoning task for different knowledge bases having the same n -inconsistency varies considerably. This is illustrated by Figure 6.4, in which the times taken to “find n ” are plotted as an X-Y scatter graph. As expected, the knowledge bases with higher n have higher percentages of inconsistency and therefore appear on the right hand side of the graph. Those having higher n -inconsistency are also harder to solve. Grouping the “find n ” results by n -inconsistency, we obtain average times and the average percentages of axioms introduced at each n as shown in Figure 6.5. Comparing Figure 6.5 to the corresponding (upper) plot for the consistent knowledge bases in 6.2 shows that the presence of inconsistency has a significant impact on the reasoning task. For instance, the relative increase at 30% of defeasible axioms is nearly 2 orders of magnitude. Furthermore, the exponential fit is of a higher order. Figure 6.6 compares the average times obtained for each of the three reasoning tasks. The three plots have a similar underlying shape. As before, refutations are significantly faster than finding n and proving a query is faster than disproving a query. The optimality reported for each answer set found for a “disprove query” task was cross-verified with the optimality recorded in the “find n ” task. There were no discrepancies.

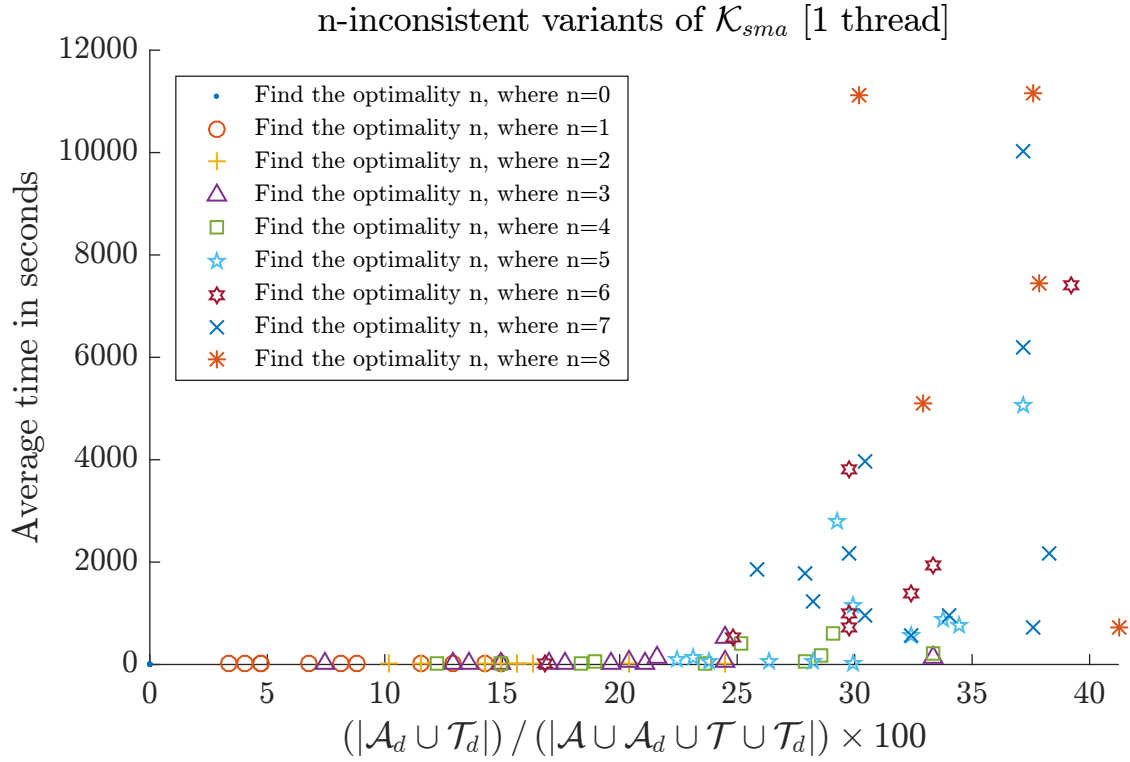


Figure 6.4: The times taken to find n for each of the n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$.

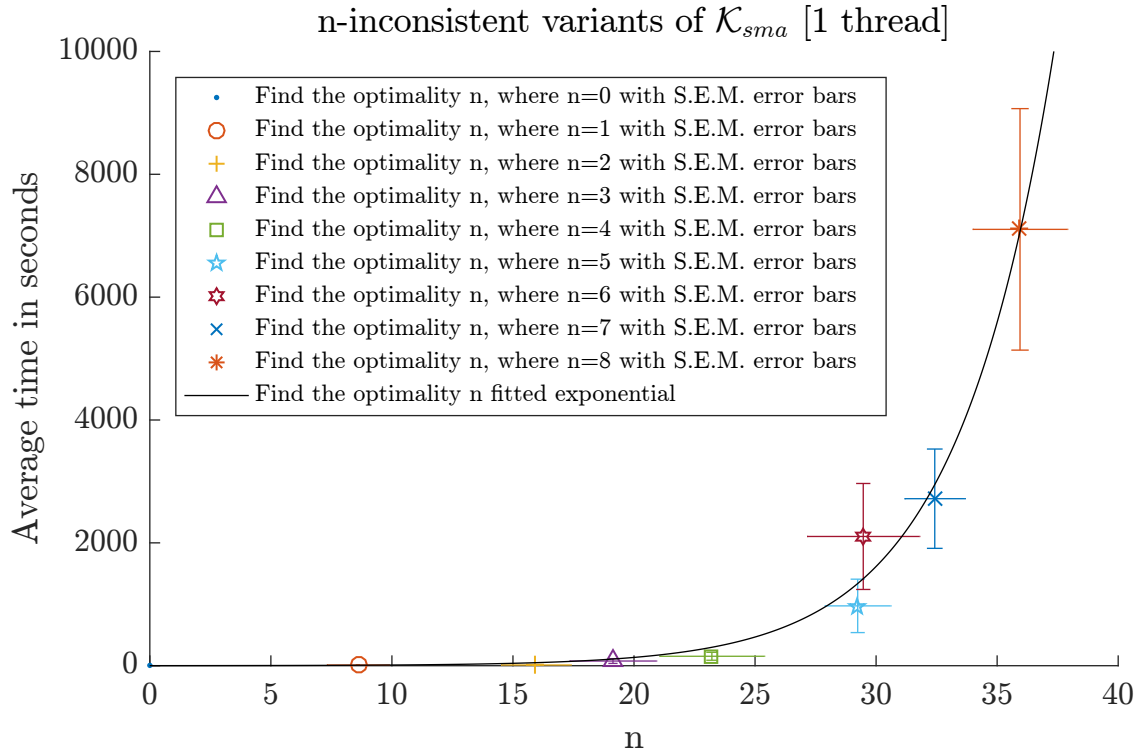


Figure 6.5: The average times taken to find n for n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$.

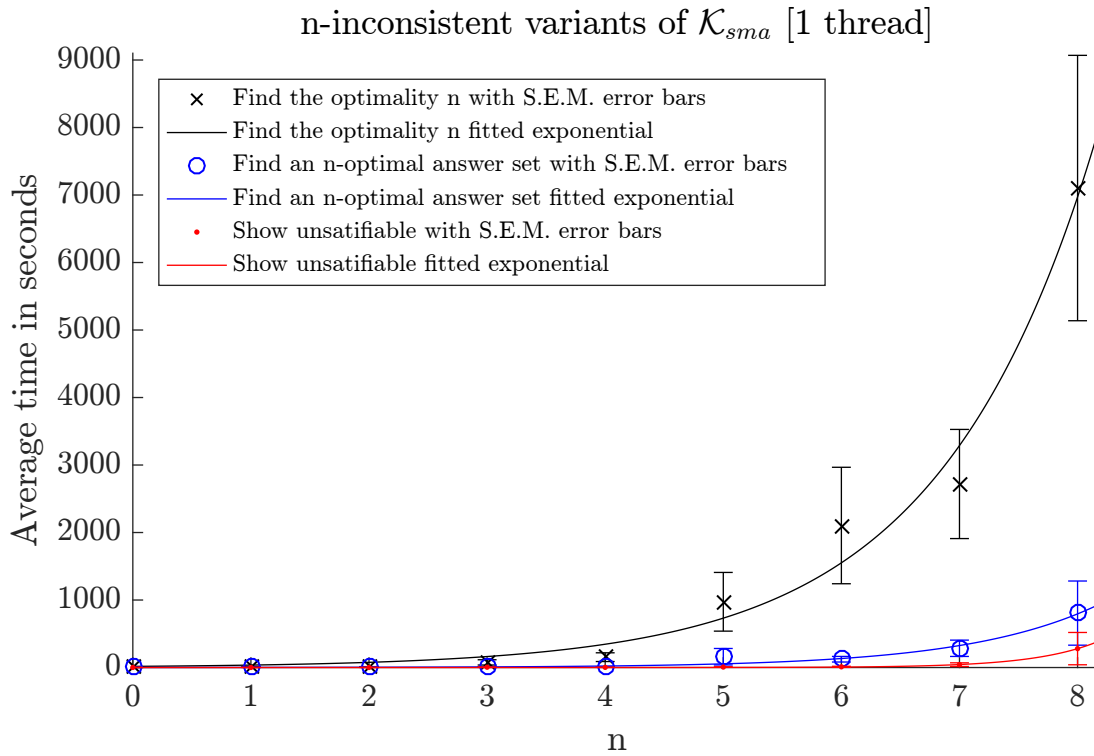


Figure 6.6: The average times taken to find n , prove queries and disprove queries for n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$.

We took the opportunity to compare the numbers of axioms that were identified as contributing to inconsistency (and therefore made defeasible) in each inconsistent knowledge base generated using Algorithm 6 with the results obtained for the same knowledge base obtained by recomputing these values using Algorithm 5. Figure 6.7 shows that the approximation is quite accurate, and that the differences do not exceed 6 axioms in any knowledge base.

6.5.3 Experiment 3: Multi-threaded solving

The knowledge bases generated in Experiment 2 were used to repeat the reasoning tasks of Experiment 2 with the solver configured to use 8 threads in “competitive mode”. No tasks were recorded as **time-out**. 11 of the 90 candidate queries in tasks to show proof of entailment were recorded as **proof-failed**. No candidate queries for disproving entailment were recorded as **proof-failed**. Figure 6.8 summarises the results. The results obtained in this test were cross-verified with results in Experiment 2. There were no discrepancies in the values of n or in the proofs of queries.

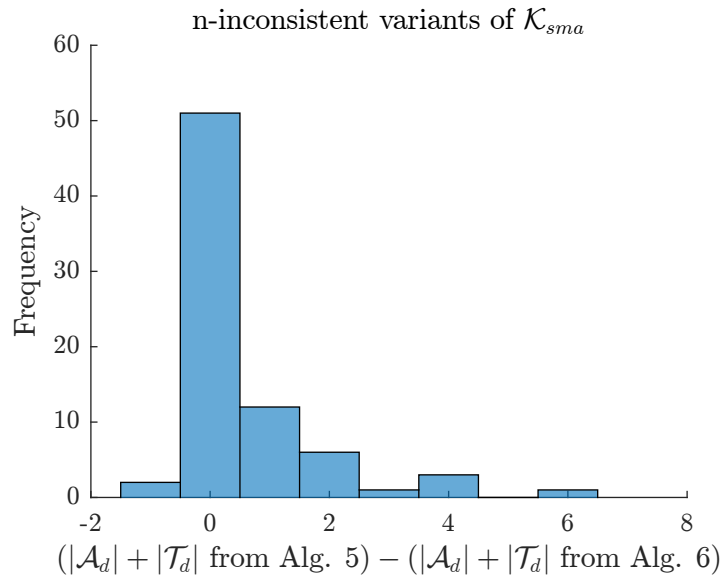


Figure 6.7: A histogram showing the difference between the total number of axioms made defeasible by Algorithm 5 and the total number of axioms made defeasible by Algorithm 6 for the set of inconsistent knowledge bases generated from $\mathcal{K}_{sma}$.

The reasoning tasks were repeated for the knowledge base $\mathcal{K}_{med}$ using the multi-threaded solver. 15 tasks were recorded as **time-out**. 10 of the 90 candidate queries in tasks to show proofs of entailment were recorded as **proof-failed**. 2 candidate queries for disproving entailment were recorded as **proof-failed**. The latter were investigated. The cause was traced to the limit chosen for the number of explanations to search for during the query generation. Both queries appeared in explanations of inconsistencies when the limit was increased from 20 to 40. The optimality reported for each answer set found for a “disprove query” task was cross-verified with the optimality recorded in the “find n ” task. There were no discrepancies. The results obtained are summarised in Figure 6.9. The results reflect similar patterns to those obtained for the smaller knowledge base $\mathcal{K}_{sma}$ using a single threaded solver shown in Figure 6.6. The *parallel efficiency* E was calculated for each “find n ” task for $\mathcal{K}_{sma}$ using $E = (T_1/T_N)/N$ where N threads is the number of threads used for the computation, T_1 and T_N are the lapsed times using 1 and N threads. These results are shown in Figure 6.10. We believe that the exponential nature of the efficiency increase with n is due to the method of implementation of the competitive multi-threaded solver. Each time a solution is found in a “winning” thread the remaining threads are killed. This allows many tableau branches to be explored simultaneously but does not require that every branch is completed.

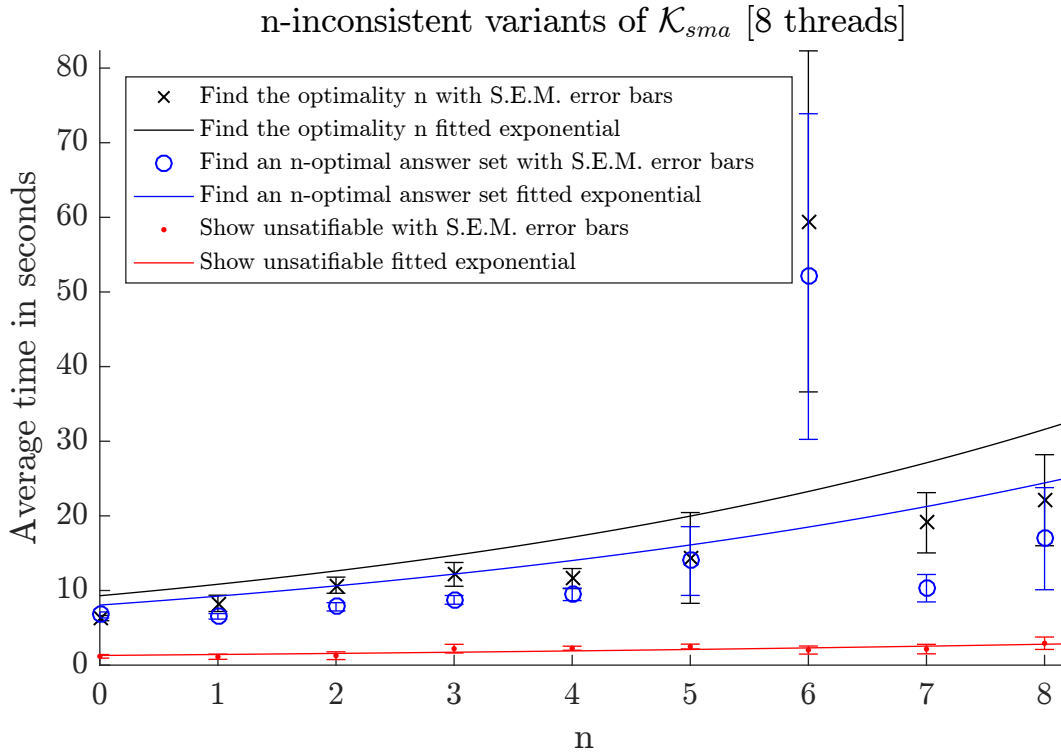


Figure 6.8: The average (lapsed) times taken to find n , prove queries and disprove queries for n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$ in Experiment 2 using a solver utilising up to 8 threads.

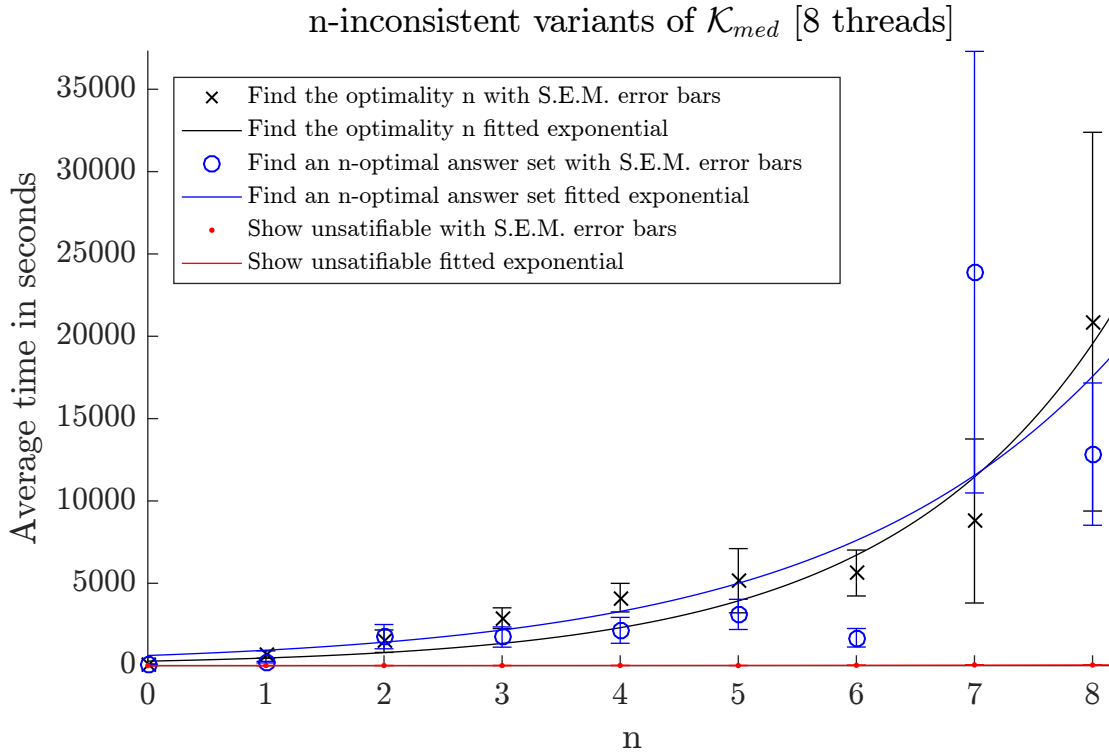


Figure 6.9: The average (lapsed) times taken to find n , prove queries and disprove queries for n -inconsistent knowledge bases generated from $\mathcal{K}_{med}$ in using a solver utilising up to 8 threads

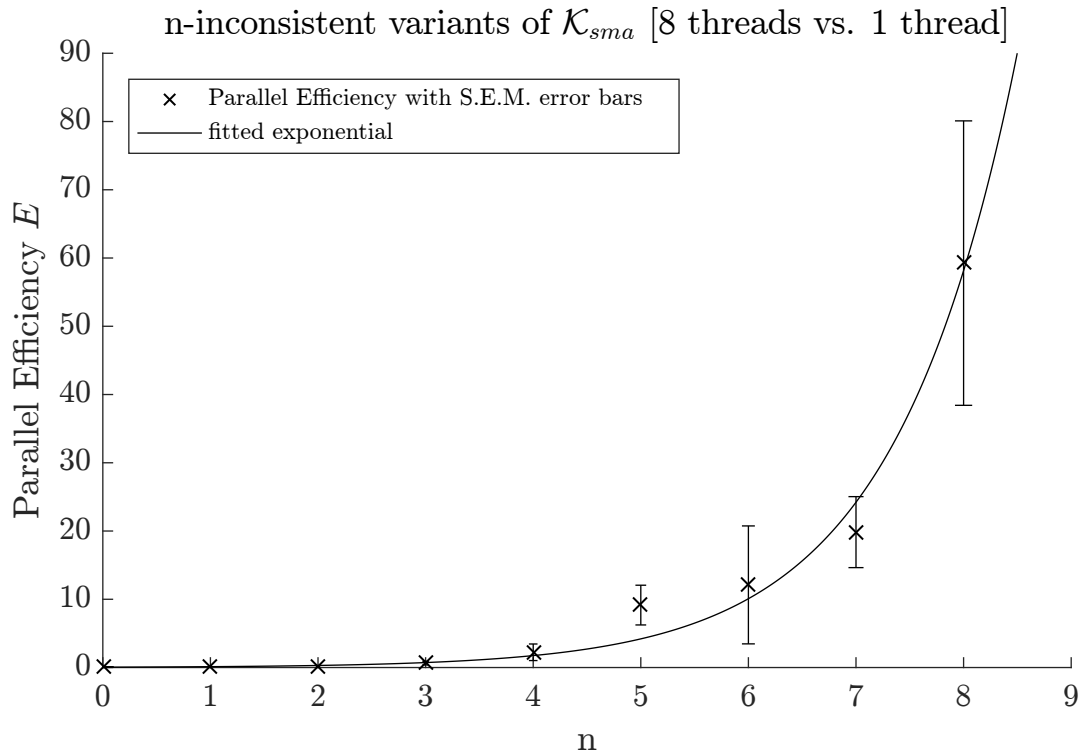


Figure 6.10: The average parallel efficiency for 8 threads vs. 1 thread when finding n for n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}$.

In conclusion, the use of a multi-threaded solver significantly decreases the lapsed solving time and hence allows larger knowledge bases to be tackled.

6.5.4 Experiment 4: Non-uniform weights

The weights assigned to the defeasible axioms are used to arbitrate between the inconsistent axioms. This experiment was designed to investigate the impact of choosing non-uniform weights on the performance of the reasoning tasks.

The results for reasoning with non-uniform knowledge bases were obtained as follows. The weight of each defeasible axiom in each of the 80 knowledge bases generated in Experiment 2 was re-assigned to randomly chosen values between 1 and 5. The reasoning tasks of Experiment 2 were repeated using the 80 non-uniform knowledge bases and a single threaded solver. No tasks were recorded as **time-out**. 8 of the 90 candidate queries in tasks to show proof of entailment were recorded as **proof-failed**. No candidate queries for disproving entailment

were recorded as **proof-failed**. The optimality reported for each answer set found for a “disprove query” task was cross-verified with the optimality recorded in the “find n ” task. There were no discrepancies. The 8 queries recorded as **proof-failed** were a proper subset of the 11 recorded in Experiment 2. We would expect that modifying the weights changes the entailed consequences as was illustrated in Example 3.5. The reduction in the number of candidates that fail to be proved where uniform weights are changed to random non-uniform weights cases lends support to the idea that assigning weights can be used to draw additional conclusions from an inconsistent knowledge base. Figure 6.11 compares the non-uniform result with the corresponding uniform result for the three task types.

We conclude that the selection of non-uniform weights for the defeasible axioms in knowledge bases generated from $\mathcal{K}_{sma}$ leads to increased reasoning performance. We believe that the reason for this is that there is an overall reduction of the numbers of branches that lead to an optimal branch w.r.t. the uniform case and this is reflected by the ASP implementation.

Finally, this experiment was repeated using the multi-threaded solver. No tasks were recorded as **time-out**. The same 8 of the 90 candidate queries in tasks to show proof of entailment were recorded as **proof-failed**. No candidate queries for disproving entailment were recorded as **proof-failed**. Each value of n (obtained for find n or disproving a query) was cross-verified with the corresponding result using the single threaded solver. There were no discrepancies and similar increases in reasoning performance observed.

6.6 Summary

The evidence obtained from the evaluation supports the claim that Algorithm 2 is correctly implemented in ASP. The performance testing shows that reasoning becomes intractable as the size and complexity knowledge bases increase. Increasing the numbers of defeasible axioms and the number of inconsistencies make reasoning significantly harder. In general, finding the n inconsistency of a knowledge base is harder than performing proofs by refutation and disproving a query harder than proving a query.

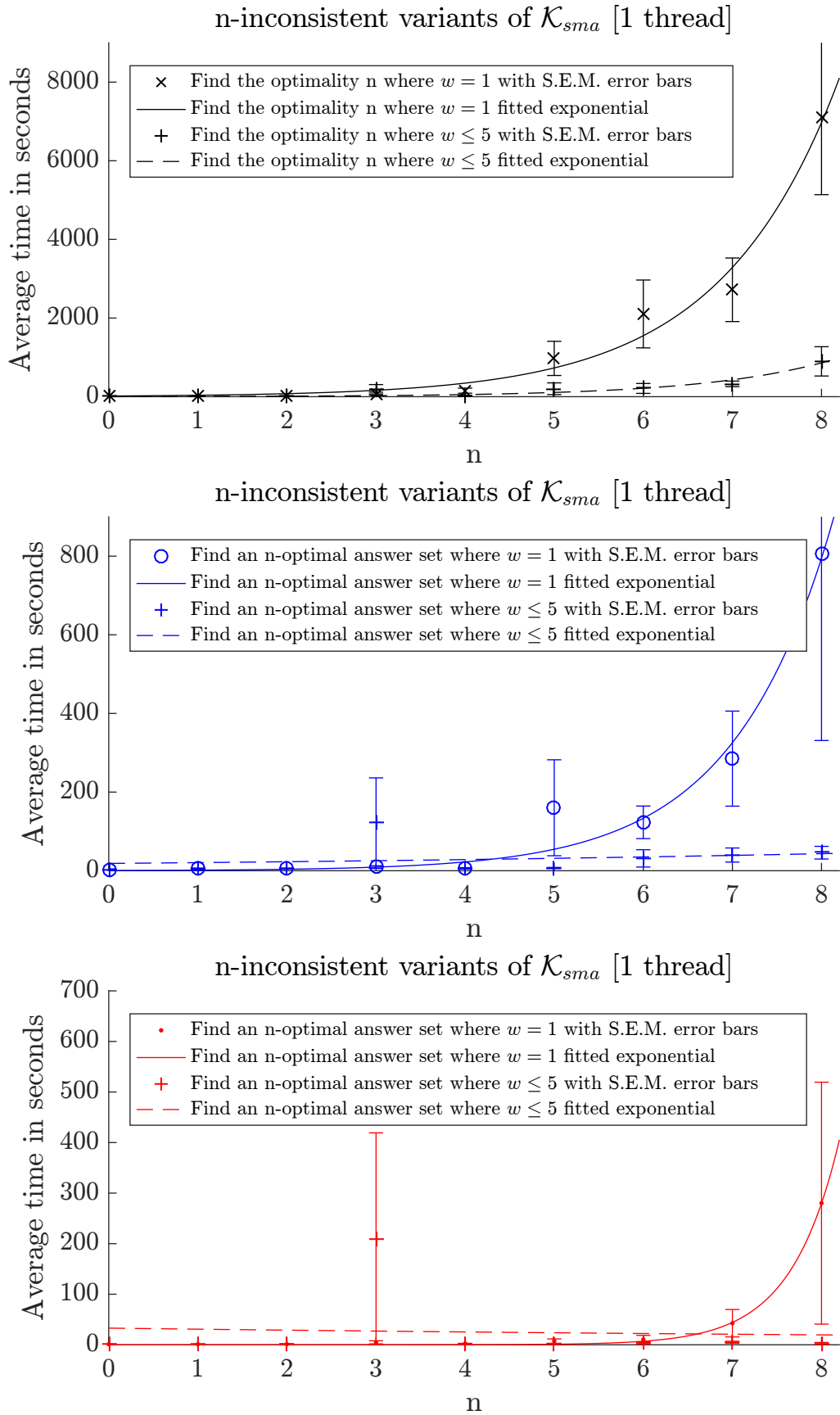


Figure 6.11: A comparison of the average times to find n , prove queries and disprove queries for uniform and non-uniform n -inconsistent knowledge bases generated from $\mathcal{K}_{sma}.1$

We observed throughout that the tasks which took the longest times were those in which many iterations were required to find an optimal answer set or to show unsatisfiability. In each iteration, additional parameters are introduced and hence the size of the grounding of the program increases. For solutions in which several branches close, there may be many parameters that have been introduced are not used in the final branch. However, each of these contribute to the overall size of the ground program. We believe it is this behaviour that contributes to the wide variation in the reasoning times observed using the multi-threaded solver. It is most noticeable in the multi-threaded solver because the use of competitive solvers allows many branches to be aborted and new branches explored before finding an optimal branch.

Our choice of ASP for our implementation prevents complete control over the search strategy. However, we have investigated a technique for steering the search towards solutions that introduce fewer parameters. An outline of this approach is presented in Chapter 8.

The underlying un-optimised $\mathcal{ALC}$ tableau is known to be intractable [HS99]. In particular, the $\rightarrow_{\mathcal{T}}$ rule is applied to each individual for every TBox axiom and leads to the introduction of a disjunction. One advantage of our $p\text{-}\mathcal{ALC}$ tableau being based on the $\mathcal{ALC}$ tableau is that existing optimisation techniques can be incorporated. Some preliminary experiments based on an implementation of the well known $\mathcal{T}$ optimisation rule [BCM⁺07] were promising and are presented in Chapter 8.

We plan to make OWLGen, our tool for generating synthetic knowledge bases, available to other research teams under an open-source license in the hope that it may prove useful to the community. The technique can readily be adapted for other Description Logics by adjusting the syntax of the concepts and axioms generated. We are also planning to extend the tool to allow inconsistency to be introduced into existing (non-synthetic) knowledge bases, as discussed in Chapter 8.

Chapter 7

Related Work

A survey of related approaches was given in Chapter 2. In this section we focus on work that is most closely related to $p\text{-}\mathcal{ALC}$. We also compare our encoding of Description Logics within ASP with other logical encodings of Description Logics within ASP found in the literature.

7.1 Repair Semantics

The $p\text{-}\mathcal{ALC}$ semantics is related to the *ABox Repair semantics* (*AR*-semantics) introduced in [LLR⁺10] for inconsistency-tolerant reasoning *DL-Lite*. The recently introduced *general repair semantics* [EL16] extend error-tolerance to both the ABox and the TBox by encoding *DL-Lite* within Datalog[±]. Our preliminary analysis suggests the approach in [EL16] has a closer relationship to $p\text{-}\mathcal{ALC}$ than that of the *AR*-semantics.

AR-Semantics

The *AR*-semantics allows inferences to be drawn from a consistent TBox together with an ABox that is not guaranteed to be consistent w.r.t. the TBox. Informally, a set of *repairs* is formed by considering maximal subsets of the ABox that are consistent with the TBox. The

consequences, based on these repairs, are defined as the entailments that are common to every model obtained from a consistent knowledge base formed from the TBox and one such repair.

We now recall the formal definition of the AR-semantics given in [LLR⁺10]. In the definitions that follow, the notation has been adjusted to reflect the notations used in this thesis, $Mod(\mathcal{K})$ denotes a model of $\mathcal{K}$ and $\models$ is used to denote classical entailment.

Definition 1 [LLR⁺10]. Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a knowledge base. An ABox Repair (AR) of $\mathcal{K}$ is a set $\mathcal{A}'$ of membership assertions such that:

1. $\mathcal{A}' \subseteq \mathcal{A}$,
2. $Mod(\langle \mathcal{A}', \mathcal{T} \rangle) \neq \emptyset$,
3. there exist no $\mathcal{A}''$ such that $\mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A}$ and $Mod(\langle \mathcal{A}'', \mathcal{T} \rangle) \neq \emptyset$.

The set of AR-repairs for $\mathcal{K}$ is denoted by $AR-Rep(\mathcal{K})$.

Informally, the repairs of a knowledge base are maximal subsets of the original ABox that are consistent with the TBox. ABox repair models are based on the above notion of repair and consistent entailment is the generalization of classical entailment to the ABox repair semantics.

Definition 2 [LLR⁺10]. Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a knowledge base. An interpretation $\mathcal{I}$ is an ABox repair model, or simply an AR-model, of $\mathcal{K}$ if there exists $\mathcal{A}' \in AR-Rep(\mathcal{K})$ such that $\mathcal{I} \models \langle \mathcal{A}', \mathcal{T} \rangle$. The set of ABox repair models of $\mathcal{K}$ is denoted by $AR-Mod(\mathcal{K})$.

Definition 3 [LLR⁺10]. Let $\mathcal{K}$ be a knowledge base, and let Z be an axiom. We say that Z is AR-consistently entailed, or simply AR-entailed, by $\mathcal{K}$, written $\mathcal{K} \models_{AR} Z$, if $\mathcal{I} \models Z$ for every $\mathcal{I} \in AR-Mod(\mathcal{K})$.

The $p\text{-}\mathcal{ALC}$ semantics is less cautious than the AR-semantics. To illustrate this, consider a knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ in $DL-Lite_{\mathcal{A}}$ that has an equivalent formulation in $\mathcal{ALC}$. $\mathcal{K}$ can be expressed as a uniform $p\text{-}\mathcal{ALC}$ knowledge base $\mathcal{K}' = \langle \emptyset, \mathcal{T}, \mathcal{A}^{[1]}, \emptyset \rangle$, where each defeasible axiom in $\mathcal{A}$ is assigned a weight of 1.

Example 7.1. Let $\mathcal{K}_{7.1} = \langle \{C(a), R(a, b), C(b)\}, \{C \sqsubseteq \neg \exists R\} \rangle$ and $\mathcal{K}_{7.1A} = \langle \{R(a, b), R(a, c), D(a)\}, \{\exists R \sqsubseteq \neg D\} \rangle$ be $DL\text{-}Lite_{\mathcal{A}}$ knowledge bases. $\mathcal{K}_{7.1}$ has two possible AR -repairs, the sets $\{R(a, b), C(b)\}$, and $\{C(a), C(b)\}$. For the repaired knowledge bases, $\langle \{R(a, b), C(b)\}, \{C \sqsubseteq \neg \exists R\} \rangle \models C(b)$ and $\langle \{C(a), C(b)\}, \{C \sqsubseteq \neg \exists R\} \rangle \models C(b)$. Hence, we conclude $\mathcal{K}_{7.1} \models_{AR} C(b)$. In $p\text{-}\mathcal{ALC}$, the 1-distant preferred interpretations of $\mathcal{K}_{7.1}$ can be partitioned into two sets: in the first, $R(a, b)^{[1]}$ is defeated, $C(a)^{[1]}$ and $C(b)^{[1]}$ are satisfied; in the second $C(a)^{[1]}$ is defeated and $C(b)^{[1]}$ is satisfied. Hence $\mathcal{K}'_{7.1} \approx C(b)$ and the semantics agree. In contrast, for $\mathcal{K}_{7.1A}$ there are two possible AR -repairs, the sets $\{R(a, b), R(a, c)\}$, and $\{D(a)\}$. Now, $\langle \{R(a, b), R(a, c)\}, \{\exists R \sqsubseteq \neg D\} \rangle \models \neg D(a)$ and $\langle \{D(a)\}, \{\exists R \sqsubseteq \neg D\} \rangle \models D(a)$. We conclude $\mathcal{K}_{7.1A} \not\models_{AR} D(a)$ and $\mathcal{K}_{7.1A} \not\models_{AR} \neg D(a)$. In contrast, $\mathcal{K}'_{7.1A} \approx \neg D(a)$ because an interpretation that corresponds to the first AR -repair defeats $D(a)^{[1]}$ and is 1-distant whereas for the second both $R(a, b)^{[1]}$ and $R(a, c)^{[1]}$ are defeated making the interpretation 2-distant.

Under the AR -semantics each minimal repair can remove any number of axioms whereas under the $p\text{-}\mathcal{ALC}$ semantics minimality is based on the sum of the defeated axiom weights. For uniform knowledge bases having no defeasible TBox and no non-defeasible ABox, all consequences under the AR -semantics are consequences under the $p\text{-}\mathcal{ALC}$ semantics, making the AR -semantics a *sound approximation* of the $p\text{-}\mathcal{ALC}$ semantics.

Theorem 7.1. Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a $DL\text{-}Lite_{\mathcal{A}}$ knowledge base with signature $\langle N_I, N_C, N_R \rangle$ and $\mathcal{K}' = \langle \emptyset, \mathcal{T}, \mathcal{A}^{[1]}, \emptyset \rangle$ be the corresponding $p\text{-}\mathcal{ALC}$ knowledge base. Let Z be an axiom of the form $C(x)$ where $x \in N_I$ and C is a concept expressible in both $DL\text{-}Lite_{\mathcal{A}}$ and $\mathcal{ALC}$. Then $\mathcal{K} \models_{AR} Z$ implies $\mathcal{K}' \approx Z$.

Proof. Assume by contradiction that $\mathcal{K}' \not\approx Z$. By assumption $\mathcal{T}$ is satisfiable and by construction every ABox axiom is defeasible. Hence, $\mathcal{K}'$ is credible and is n -inconsistent for some n . By Definition 3.4, there exists at least one n -distant preferred interpretation $\mathcal{I}$ of $\mathcal{K}'$ where $n \geq 0$ such that $\mathcal{I} \not\models Z$. Clearly, $\mathcal{I}$ is also a model of $\langle A_R, \mathcal{T} \rangle$ where A_R is obtained from $\mathcal{A}$ by removing the corresponding n axioms falsified in $\mathcal{I}$. By Definition 3.4 no interpretation of $\mathcal{K}'$ falsifies fewer defeasible axioms and we conclude $A_R \in AR\text{-}Rep(\mathcal{K})$. Now $\mathcal{I} \in AR\text{-}Mod(\mathcal{K})$, $\mathcal{I} \not\models Z$ and by Definition 3 [LLR⁺10] $\mathcal{K} \not\models_{AR} Z$. $\square$

The less cautious *CAR*-repair semantics are also considered in [LLR⁺10]. These rely on the notion of *consistent logical consequences* of a knowledge base.

Definition 7.2 (Page 8 [LLR⁺10]). *Given a knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$, we denote with $HB(\mathcal{K})$ the Herbrand Base of $\mathcal{K}$, i.e. the set of ABox assertions that can be built from the alphabet in $\mathcal{L}(\mathcal{K})$. Then we define the consistent logical consequences of $\mathcal{K}$ as the set $clc(\mathcal{K}) = \{\alpha \mid \alpha \in HB(\mathcal{K}) \text{ and there exists } S \subseteq \mathcal{A} \text{ such that } \text{Mod}(\langle S, \mathcal{T} \rangle) \neq \emptyset \text{ and } \langle S, \mathcal{T} \rangle \models \alpha\}$.*

Informally, the $clc(\mathcal{K})$ are the set of consequences that can be derived from any subset of the ABox axioms that are consistent with the TBox.

Definition 4 [LLR⁺10]. *Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a DL KB. A Closed ABox Repair (CAR) for $\mathcal{K}$ is a set $\mathcal{A}'$ of membership assertions such that:*

1. $\mathcal{A}' \subseteq clc(\mathcal{A})$,
2. $\text{Mod}(\langle \mathcal{A}', \mathcal{T} \rangle) \neq \emptyset$,
3. *there exist no $\mathcal{A}'' \subseteq clc(\mathcal{K})$ such that*
 - (a) $\text{Mod}(\langle \mathcal{A}'', \mathcal{T} \rangle) \neq \emptyset$,
 - (b) *and it is either $((\mathcal{A}'' \cap \mathcal{A}) \supset (\mathcal{A}' \cap \mathcal{A}))$ or $((\mathcal{A}'' \cap \mathcal{A}) = (\mathcal{A}' \cap \mathcal{A}))$ and $(\mathcal{A}'' \supset \mathcal{A}')$.*

The set of CAR-repairs for $\mathcal{K}$ is denoted by $CAR\text{-}Rep(\mathcal{K})$.

Informally, the CAR-repairs “maximally preserve” the original ABox with as many as possible of the consequences from $clc(\mathcal{K})$. The notions of $CAR\text{-}Mod(\mathcal{K})$ and $\models_{CAR}$ follow, respectively, Definition 2 [LLR⁺10] and Definition 3 [LLR⁺10]. Informally, the idea is to retain as many as possible of the consequences that can be derived by a repair and are consistent with the TBox.

Example 7.1 (continued.). *Returning to $\mathcal{K}_{7.1} = \langle \{C(a), R(a, b), C(b)\}, \{C \sqsubseteq \neg \exists R\} \rangle$. The consistent consequences are $clc(\mathcal{K}_{7.1}) = \{C(a), R(a, b), C(b)\}$ and $\mathcal{K}_{7.1}$ has two possible CAR-repairs, the sets $\{R(a, b), C(b)\}$, and $\{C(a), C(b)\}$. Again, for the repaired knowledge bases,*

$\langle \{R(a, b), C(b)\}, \{C \sqsubseteq \neg \exists R\} \rangle \models C(b)$ and $\langle \{R(a, b), C(b)\}, \{C \sqsubseteq \neg \exists R\} \rangle \models C(b)$. Hence, we conclude $\mathcal{K}_{7.1} \models_{CAR} C(b)$, agreeing with both AR and $p\text{-}\mathcal{ALC}$ semantics. Following a similar argument, $\mathcal{K}_{7.1A} \not\models_{CAR} \neg D(a)$, and in this instance, the CAR -semantics agree with the AR -semantics and disagree with the $p\text{-}\mathcal{ALC}$ semantics.

However, the idea of retaining as many consequences as possible used in CAR -semantics can lead to potentially un-intuitive results.

Example 7.2. Let $\mathcal{K}_{7.2} = \langle \{R(a, b), D(a)\}, \{\exists R \sqsubseteq \neg D, \exists R \sqsubseteq E\} \rangle$. The consistent consequences are $clc(\mathcal{K}_{7.2}) = \{R(a, b), D(a), \neg D(a), E(a)\}$ and there are two possible CAR -repairs, the sets $\{R(a, b), \neg D(a), E(a)\}$ and $\{D(a), E(a)\}$. Notice that the second set includes $E(a)$ since $\langle \{D(a), E(a)\}, \{\exists R \sqsubseteq \neg D, \exists R \sqsubseteq E\} \rangle$ is consistent, even though $E(a)$ was not given and cannot be derived from the surviving $ABox$ axiom $D(a)$ using the $TBox$. For the repaired knowledge bases, we obtain $\langle \{R(a, b), \neg D(a), E(a)\}, \{\exists R \sqsubseteq \neg D, \exists R \sqsubseteq E\} \rangle \models E(a)$ and $\langle \{D(a), E(a)\}, \{\exists R \sqsubseteq \neg D, \exists R \sqsubseteq E\} \rangle \models E(a)$. Hence, $\mathcal{K}_{7.2} \models_{CAR} E(a)$. In contrast, for the corresponding $p\text{-}\mathcal{ALC}$ knowledge base, $\mathcal{K}'_{7.2} \not\models E(a)$. There are two sets of 1-distant preferred interpretations. In one set: $D(a)^{[1]}$ is defeated, $R(a, b)^{[1]}$, $\exists R \sqsubseteq E$ are satisfied and $E(a)$ is true in the interpretation; in the other set: $R(a, b)^{[1]}$ is defeated, some preferred interpretations make $E(a)$ true and others make $E(a)$ false.

We conclude that the CAR -semantics are quite different from the $p\text{-}\mathcal{ALC}$ semantics.

Two further semantics, IAR - and $ICAR$ - were introduced in [LLR⁺10] as sound approximations of (respectively) the AR - and CAR -semantics with the aim of lowering the computational complexity of reasoning. It was shown in [LLR⁺10] that instance checking for the AR -semantics is in **coNP** and this is reduced to **P** for the IAR -semantics.

Definition 5. [LLR⁺10] Let $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ be a DL KB. An Intersection ABox Repair (IAR) for $\mathcal{K}$ is the set $\mathcal{A}'$ of membership assertions such that $\mathcal{A}' \models \bigcap_{\mathcal{A}_i \in AR\text{-Rep}(\mathcal{K})} \mathcal{A}_i$. The (singleton) set of IAR -repairs for $\mathcal{K}$ is denoted by $IAR\text{-Rep}(\mathcal{K})$.

Since IAR -semantics are a sound approximation of AR -semantics, we conclude that the IAR -semantics are also a sound approximation of $p\text{-}\mathcal{ALC}$ semantics under the conditions described

earlier. However, the *IAR* semantics are more cautious than *AR*.

Example 7.3. Let $\mathcal{K}_{7.3} = \langle \{C(a), D(a)\}, \{C \sqsubseteq \neg D, C \sqsubseteq E, D \sqsubseteq E, \} \rangle$. The two possible *AR*-repairs are the sets $\{C(a)\}$ and $\{D(a)\}$. Hence, the singleton *IAR* repair is $\{\}$. $\mathcal{K}_{7.3} \models_{AR} E(a)$ but $\mathcal{K}_{7.3} \not\models_{IAR} E(a)$.

The *ICR* semantics proposed in [Bie12] provide a closer approximation to *AR* by considering consequences based on the intersection of the closed consequence (as opposed to the union of repairs used in *ICAR*).

Implementations of the *IAR*-semantics have been evaluated in [RRGM12] and also in [LLR⁺15]. Both of these systems were evaluated against synthetic *DL-Lite* knowledge bases generated using the LUBM benchmark method [GPH05]. To our knowledge there is no implementation of the full *AR*-semantics to date.

Generalised Repair Semantics

In [EL16] *generalised repair* (GR) and *local generalised repair* (LGR) semantics are introduced. In the former TBox axioms are treated atomically, whereas the LGR semantics consider instances of TBox axioms in a manner very similar to *p-ALC*. Additionally, LG and LGR semantics divide the ABox and TBox into non-defeasible and defeasible axioms (referred to as *hard* and *soft* axioms respectively).

We focus on LGR. Informally, a *DL-Lite* knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ can be expressed in Datalog[±] as a pair (D, Σ) where D is a *database* that encodes the facts in $\mathcal{A}$ and Σ is a set of Datalog[±] *program rules* that encodes the axioms of $\mathcal{T}$. We omit the details of the exact encoding for brevity but note that each rule in Σ is non-ground and corresponds to a TBox axiom in $\mathcal{T}$. The axioms that are to be weakened in $\mathcal{K}$, and accommodate repairs, are obtained by expressing D as a *flexible data base* (D_h, D_s) where $D = D_h \cup D_s$ and Σ as a *flexible program* (Σ_h, Σ_s) where $\Sigma = \Sigma_h \cup \Sigma_s$. A *local general repair* now follows a similar pattern to an *AR*-repair: it is a pair $((h_h, h'_s), (\Sigma_h, \Sigma'_s))$ where $h'_s \subseteq \text{ground}(h_s)$ and $\Sigma'_s \subseteq \text{ground}(\Sigma_s)$ such that $((h_h, h'_s), (\Sigma_h, \Sigma'_s))$ is consistent. The consequences under the LGR semantics, written $\models_{LGR}$,

are then defined in terms of queries that hold for each repair. Again, the precise definitions of *ground*, *consistent* and *query* are omitted for brevity.

Just as the *AR*-semantics are more cautious than *p-ALC*, the LGR semantics are also more cautious than *p-ALC*, due to our semantics taking into account the weights of the defeated axioms. For knowledge bases that are expressible in both languages, we believe that the relationship between inferences obtained from the LGR semantics and those obtained from the *p-ALC* semantics is captured by Conjecture 7.3.

Conjecture 7.3. *Let $((D_{\mathcal{A}}, D_{\mathcal{A}_d}), (\Sigma_{\mathcal{T}}, \Sigma_{\mathcal{T}_d}))$ be an error-tolerant representation in $\text{Datalog}^{\pm}$ of the DL-Lite knowledge base $\mathcal{K} = \langle \mathcal{A}_L, \mathcal{T}_L \rangle$ with signature $\langle N_I, N_C, N_R \rangle$ where $\mathcal{A}_L = \mathcal{A} \cup \mathcal{A}_d$, $\mathcal{T}_L = \mathcal{T} \cup \mathcal{T}_d$ and $\Sigma_{\mathcal{T}}$ ($\Sigma_{\mathcal{T}_d}$) are the encodings of the axioms of $\mathcal{T}$ (resp. $\mathcal{T}_d$) as program rules. Let $\mathcal{K}' = \langle \mathcal{A}, \mathcal{T}, \mathcal{A}_d^{[1]}, \mathcal{T}_d^{[1]} \rangle$ be the defeasible representation of $\mathcal{K}$ as a uniform *p-ALC* knowledge base and Z be an axiom of the form $C(x)$ where $C \in N_C$ and $x \in N_I$. Then $\mathcal{K} \models_{\text{LGR}} Z$ implies $\mathcal{K}' \models Z$*

We are not aware of any implementation of the LGR semantics.

7.2 Encoding Description Logic in ASP

A range of different techniques for implementing Description Logic reasoning tasks by encoding a knowledge base as answer set programs have been explored in the literature.

At the simplest level, Description Logic Programs [GHVD03] capture the intersection of Description logics with logic programs. Unfortunately, the resulting language is not sufficiently expressive for most problems, for instance those expressed in *ALC*. Early attempts to encode more expressive Description Logics, for instance [AB02] and [Swi04], quickly identified the challenges associated with Skolemisation. Answer Set solvers require a fixed and finite Herbrand Universe to obtain a grounding whereas the Description Logic tableau algorithms introduce fresh witnesses for quantified roles. This was addressed in [Mot06] and [HMS07] by introducing a series of resolution steps to eliminate the quantifiers and obtain a translation to a disjunctive

program. However, we are not aware this work being extended to accommodate inconsistencies. The resulting programs are far removed from the original input. We anticipate that making the rules defeasible in such a way as to allow the identification of the axioms involved in inconsistencies would be non-trivial. To avoid the issues associated with encoding Description Logics within a logic program a number of *hybrid* logics have been introduced (see [KMH11] for a review). These techniques require that a knowledge base is split into two components, such that the first component exploits a classical Description Logic reasoner and the second component exploits a logic program reasoner such as an Answer Set solver. To our knowledge, our encoding is the first ASP encoding of an expressive description logic that exploits incremental grounding to overcome the issues relating to Skolemisation.

7.3 Generation of inconsistent knowledge bases

The LUBM benchmark system [GPH05] was used to synthesis inconsistent *DL-Lite* knowledge bases in [RRGM12] and also in [LLR⁺15]. For more expressive logics, OTAGen [OVD⁺08] can be used to generate consistent knowledge bases. However, OTAGen provides no mechanism to introduce inconsistency. The tool appears to generate knowledge bases using a method related to our own. The structure is hierarchical and based on the notion of connected *clusters* of concepts that are interconnected by roles. In [HPH15] inconsistencies (resulting in *incoherence*) were introduced into the TBoxes of existing $\mathcal{ALC}$ TBoxes from the Manchester OWL Repository¹ and a set of entirely synthetically generated TBoxes. This approach is not compatible with our own as we assume a coherent TBox. The synthetic TBoxes generated are hierarchical in structure and the ratios between the number of different types of constructors used can be adapted to reflect those observed in the corpora of real-world knowledge bases. Such an approach would be a way to improve our own technique to obtain knowledge bases that reflect those in the real world more closely.

¹<http://mowlrepo.cs.manchester.ac.uk/>

7.4 Summary

$p\text{-}\mathcal{ALC}$ is closely related to, but differs from, the recently introduced LGR repair semantics. Where inconsistency tolerance is not required for the TBOX, $p\text{-}\mathcal{ALC}$ is related to the AR and IAR -semantics but differs from the CAR - and $ICAR$ -semantics. Our encoding of $\mathcal{ALC}$ within ASP removes the need for a transformation to eliminate quantifiers as used in [Mot06] and [HMS07] by introducing new constants on an as-needed basis through incremental grounding.

Chapter 8

Conclusions and Future Work

This thesis has presented three contributions to the field of inconsistency tolerant reasoning. These are:

1. $p\text{-}\mathcal{ALC}$, a novel logic that allows meaningful inferences to be drawn from an inconsistent knowledge base. Integer weights are assigned to a subset of the axioms that identify them as defeasible, such that they can be falsified during inference incurring a penalty defined by the weight. An inconsistent knowledge base is made *credible* by ensuring the set of unlabelled non-defeasible axioms are consistent. We have shown that every credible knowledge base has an n -inconsistency that provides an underlying measurement for the level of inconsistency in the knowledge base and that the preferred consequences in $p\text{-}\mathcal{ALC}$ can be obtained in two steps using proof by refutation. $p\text{-}\mathcal{ALC}$ addresses a more expressive Description Logic than the ABox Repair (AR) semantics, $p\text{-}\mathcal{ALC}$ consequences are less cautious and $p\text{-}\mathcal{ALC}$ has the additional benefit of control over the arbitration of conflict resolution by varying the values of weights.
2. The TINference system, which computes the preferred consequences, constructing proofs by refutation based on a tableau algorithm, and which is implemented in ASP. We have shown that the tableau algorithm is terminating, sound and complete for all finite $p\text{-}\mathcal{ALC}$ knowledge bases and that the tableau algorithm has been correctly implemented in ASP.

This leads to a terminating, sound and complete system for computing preferred consequences. We have conducted an evaluation of the system demonstrating that the system is correctly implemented, and have gained insight into the scalability of the approach.

3. *OWLGen*, a tool that can be used to create synthetic inconsistent $\mathcal{ALC}$ knowledge bases that broadly reflect the structures found in the corpora of existing knowledge bases. The OWLGen tool allows parametric control over a wide range of variables to create knowledge bases of varying dimensions, complexity and level of inconsistency. The tool was used to generate knowledge bases for our evaluation and we will make it available for use by the wider community.

8.1 Future development of the existing implementation

In Chapter 3 we identified that further research is required to investigate how the selection of uniform vs. non-uniform weights and their distribution over TBox and ABox and to determine to what extent our approach could be refined or informed by findings from research on priority/preference orderings in defeasible reasoning and logic programming.

We have shown in Chapter 6 that our $p\text{-}\mathcal{ALC}$ system operates correctly and can be used for knowledge bases that include hundreds of axioms (for instance $\mathcal{K}_{med}$ includes 320 axioms). However, in the world of “big data” solutions are being sought for systems that scale to hundreds of thousands of axioms. In the following sections we set out future work which includes an examination of the underlying causes of intractability in the implementation and proposes techniques for improving performance.

8.1.1 Optimising the tableau

Algorithm 2 does not yet incorporate certain optimisations that are known to improve the tractability of reasoning for real-world knowledge bases.

Absorption

Absorption [HT00] is a technique for optimising the expansions due to the TBox employing *lazy unfolding* [BHS08], delaying the expansion of concepts until required. We recall the absorption approach given in section 4 of [HT00], following the notational conventions of [BHS08] and where N_C denotes the set of concept names in the signature of the knowledge base.

A TBox is said to be *primitive* [HT00] iff it consists entirely of axioms of the form $A \equiv C$ with $A \in N_C$, each $A \in N_C$ appears as at most one left-hand side of an axiom, and $\mathcal{T}$ is acyclic.

Definition 8.1 (Absorption [HT00]). *Given a TBox $\mathcal{T}$, absorption divides $\mathcal{T}$ into a triple of TBoxes $(\mathcal{T}_g, \mathcal{T}_{prim}, \mathcal{T}_{inc})$ such that*

- $\mathcal{T}_{prim}$ is primitive; and
- $\mathcal{T}_{inc}$ consists only of axioms of the form $A \sqsubseteq D$ where $A \in N_C$ and A is not defined in $\mathcal{T}_{prim}$.
- $\mathcal{T}_g = \mathcal{T} \setminus (\mathcal{T}_{prim} \cup \mathcal{T}_{inc})$

The tableau is then modified to exploit Theorem 3.1 [HT00] by applying the standard TBox rules of to the rules of $\mathcal{T}_g$ and the rules shown in Table 8.1.

Rules $\mathcal{R}$	Valid expansions of $\mathcal{A}$
$\equiv_1$ -rule	If $(A \equiv D) \in \mathcal{T}_{prim}$, $A \in N_C$ and $\exists$ an unblocked individual x in $\mathcal{A}$ and $A(x) \in \mathcal{A}$ and $D(x) \notin \mathcal{A}$ then $\mathcal{A}_e = \mathcal{A} \cup \{D(x)\}$
$\equiv_2$ -rule	If $(A \equiv D) \in \mathcal{T}_{prim}$, $A \in N_C$ and $\exists$ an unblocked individual x in $\mathcal{A}$ and $\neg A(x) \in \mathcal{A}$ and $\neg D(x) \notin \mathcal{A}$ then $\mathcal{A}_e = \mathcal{A} \cup \{\neg D(x)\}$
$\mathcal{T}_{lazy}$ -rule	If $(A \sqsubseteq D) \in \mathcal{T}_{inc}$, $A \in N_C$ and $\exists$ an unblocked individual x in $\mathcal{A}$ and $A(x) \in \mathcal{A}$ and $D(x) \notin \mathcal{A}$ then $\mathcal{A}_e = \mathcal{A} \cup \{D(x)\}$

Table 8.1: The tableau rules for lazy unfolding

The intuition behind absorption is as follows. An interpretation $\mathcal{I}$ based on an open branch assigns false to each concept $A \in N_C$ that is not asserted as $A(x)$ in the branch, which means

that x vacuously satisfies $A \sqsubseteq C$ in $\mathcal{I}$ when $x^{\mathcal{I}} \notin C^{\mathcal{I}}$. An additional rule called the $\mathcal{T}_{\text{lazy}}$ -rule is provided for when $A(x)$ is in the branch. Similarly, $A \equiv C$ can be written as $A \sqsubseteq C$ and $\neg A \sqsubseteq \neg C$, and satisfaction is guaranteed by the $\equiv_1$ and $\equiv_2$ -rules.

We have implemented this approach for our $p\text{-}\mathcal{ALC}$ tableau algorithm, although we have not yet proved the correctness of the operation for this enhancement. Preliminary results obtained are promising. Repeating the reasoning tasks in Experiment 3 for $\mathcal{K}_{\text{med}}$ using the optimised rules allowed all results to be computed in under 15 seconds. These results are summarised in Figure 8.1. The times taken to complete the hardest tasks using the unoptimised encoding

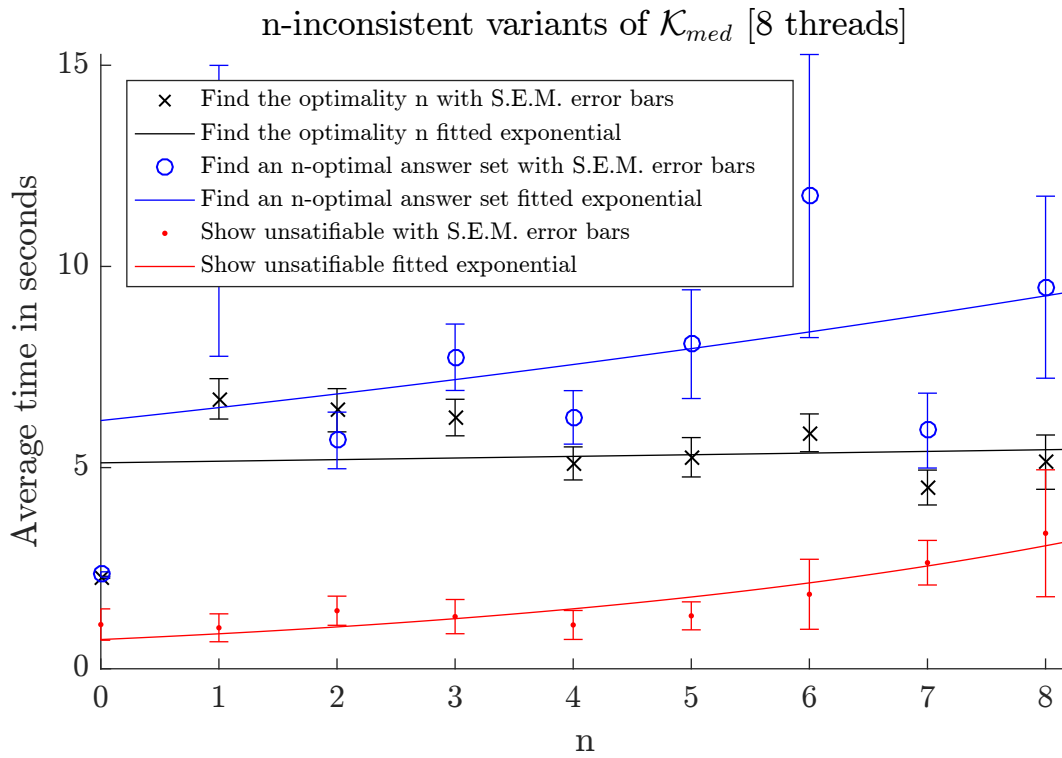


Figure 8.1: The average (lapsed) times taken to find n , prove queries and disprove queries for n -inconsistent knowledge bases generated from $\mathcal{K}_{\text{med}}$ in Experiment 2 using lazy unfolding rules and a solver utilising up to 8 threads.

(see Figure 6.9) were around 5 hours. A significant increase in performance is to be expected as the optimisation reduces the number of expansions where definitions are present and most knowledge bases (including our synthetically generated ones), due to their hierarchical nature, incorporate many definitions. The increase observed, up to 3 orders of magnitude, reflects the exponential relationship between reasoning time and the number of inconsistencies.

We note that the optimisation does not improve the underlying (worst-case) complexity of reasoning, but merely improves the performance for knowledge bases of a certain form. The results in Figure 8.1 also show that the times taken to find the optimality are only marginally greater than those for finding a model at a target optimality. It is not clear why this is the case and once the proof of correctness is established this will be require further investigation. We note, however, that the ASP rules we use to implement the additional tableau rules can introduce head-cycles that push up the complexity of the ASP program at each solving step.

Early clash detection

Algorithm 2 detects clashes between simple concepts, that is $A(x)$ with $\neg A(x)$ where A is a concept name. It is possible to detect certain clashes at an early stage, for example $C(x)$ with $\neg C(x)$ where C is not simple. Such optimisations may be possible but great care must be taken to ensure that any falsified axiom instances are still correctly recorded. The absorption technique goes some way towards addressing this and has the advantage of avoiding introducing the necessary machinery for detecting non-trivial clashes.

Heuristics to control the search

The tableau algorithms for Description Logics often employ heuristics to control *backtracking*, deciding where to pick-up when a branch closes, for instance *Dynamic Backtracking* [Gin93] in [TSL15] and *Dependency Backjumping* [Bak95] in [TH06]. Our implementation in ASP expresses the problem declaratively and therefore does not offer obvious methods of control. However, an implementation of Algorithm 2 that did not rely on ASP might lead to a more efficient implementation and such an approach is discussed in Section 8.3. There may be additional tableau optimisation techniques that are used in state-of-the-art reasoners that could be included, but the impact on the recording of falsified axioms must be carefully evaluated for each one.

The $\rightarrow_Q$ -rule is a somewhat “brute force” approach to ensure that the algorithm reaches

branches that represent preferred interpretations. Splitting on every quantified concept, especially where such concepts have definitions for more complex concepts, leads to exponential branching. It may be possible to refine the blocking strategy to recognise blocking in the presence of complex concepts in conjunction with the absorption technique. Further work is required to investigate this and explore other strategies to tackle the problem. However, if such strategies are complex, it may prove quite difficult to implement these in ASP.

Indistinguishable individuals

Any strategy that can reduce the number of individuals is likely to result in performance improvements since the inconsistencies introduced by every individual must be accounted for. In a large knowledge base it may be possible to identify, as a preprocessing step that analyses the facts, equivalence classes that group *indistinguishable* individuals, in the sense that the ABox enforces they belong to the same concepts and have the same kinds of role relationships. Such individuals then can be replaced when reasoning by a single *proxy*, a representative from the equivalence class. Adopting this strategy requires that the weights of falsified axiom instances for a proxy need to be multiplied by the size of the class they represent.

8.1.2 Optimising the ASP implementation

The current implementation was designed to reflect the tableau algorithm as closely as possible. One problem that emerges is that in the absence of any heuristics all the choices are arbitrary and in particular many fresh individuals can be introduced when it is not strictly necessary to do so. There is a significant overhead for the introduction of fresh individuals as the current answer set must be searched for *need*/2 atoms, then the necessary instances of P_{cum} must be grounded for the new parameters against the existing program and finally the solver reengaged. With increasing numbers of parameters the numbers of rules increases and grounding and solving becomes slower at each iteration. One approach for addressing this is to leverage the notion of *weighted priorities* in ASP.

Weak constraints in ASP may be written with an associated *priority* as:

$$\Leftarrow b_1, \dots, b_n.[w@p, t_1, \dots, t_m]$$

where $n, m \geq 1$, $b_1 \dots b_m$ are literals, $w \geq 0$, $p \geq 0$ and $t_1 \dots t_m$ are terms. w denotes the weight as before, p is an integer and denotes the *priority* of the constraint where higher values indicate a stronger priority. Informally, the idea is to define optimal answer sets as those that minimise the optimality (summed weights of falsified weak constraints) at the highest priority, then minimise the optimality at the next highest priority, all the way down to the lowest priority. The optimality of an answer set is then reported as a tuple of values, reflecting the optimality at each priority.

Applying priorities, we can express each weak constraint in P_b and P_{cum} at priority 2 and add weak constraints to minimise the *need/2* atoms at priority 1. The optimality reported at priority 2 of the answer sets obtained will reflect the distance of the preferred interpretations (the value obtained is unchanged). However where there are minimal branches requiring different numbers of parameters, only the answer sets that reflect use of the fewest numbers of parameters will be reported. The iterative search will favour the introduction of fewer parameters and we anticipate that this will increase performance without changing the correctness of the implementation.

8.1.3 Weighted priorities in $p\text{-}\mathcal{ALC}$

Extending the notion of prioritised weights to the language of $p\text{-}\mathcal{ALC}$ itself is very straight forward. Defeasible axioms are assigned to m priorities $p_1 \dots p_m$ where $p_x > p_{x-1} > 0$ for $1 \leq x \leq m$. Each defeasible axiom is an $\mathcal{ALC}$ axiom labelled by a weighted priority, $Z^{[w@p_x]}$ where $1 \leq x \leq m$ and the notion of the distance of an interpretation $\mathcal{I}$ of $\mathcal{K}$ is redefined as the tuple

$$d(\mathcal{U}(\mathcal{K}, \mathcal{I})) = \left(\sum_{\langle Z^{[w@p_m]}, u \rangle \in \mathcal{U}(\mathcal{K}, \mathcal{I})} w, \sum_{\langle Z^{[w@p_{(m-1)}]}, u \rangle \in \mathcal{U}(\mathcal{K}, \mathcal{I})} w, \dots, \sum_{\langle Z^{[w@p_1]}, u \rangle \in \mathcal{U}(\mathcal{K}, \mathcal{I})} w \right)$$

Preferred interpretations are obtained from Definition 3.4 by replacing the relation $<$ by the relation $\ll$ such that for interpretations $\mathcal{I}$ and $\mathcal{I}'$ of $\mathcal{K}$, $\mathcal{I} \ll \mathcal{I}'$ if $d(\mathcal{U}(\mathcal{K}, \mathcal{I})) \ll d(\mathcal{U}(\mathcal{K}, \mathcal{I}'))$ and

$(d_1, \dots, d_m) \ll (d'_1, \dots, d'_m)$ if there is some integer k such that $d_k \leq d'_k$ and for all $j < k$, $d_j = d'_j$. As before, the n -inconsistency of $\mathcal{K}$ is given by the distance of the preferred interpretations which is now a tuple. Our ASP implementation can incorporate this change by passing the associated priority with the weight to the weak constraint in P_b and P_{cum} rules.

The resultant logic allows very flexible control over how inconsistencies are accommodated through the choice of the priorities and weights.

8.2 Extending the approach for languages beyond $\mathcal{ALC}$

The reader will recall that in Chapter 1 we explained our choice of $\mathcal{ALC}$ for the underlying logic in our work. In particular we chose $\mathcal{ALC}$ because it includes, and permits unrestricted use of, the boolean, universal quantified role and existential quantified role constructors, a broad fragment of first order logic. As seen in Chapter 3, $\mathcal{ALC}$ (and $p\text{-}\mathcal{ALC}$) exhibit the finite model property. We consider how $p\text{-}\mathcal{ALC}$ might be extended to other Description Logics both with, and without, this property.

8.2.1 Languages with the finite-model property

Proof by refutation for queries of the form $R(a, b)$ is not yet accommodated in our system. The underlying reason for this is that the corresponding negated query $\neg R(a, b)$ does not belong to the language $\mathcal{ALC}$. For a consistent $\mathcal{ALC}$ knowledge base $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$, $\mathcal{K} \models R(a, b)$ iff $R(a, b) \in \mathcal{A}$. This is not the case in $p\text{-}\mathcal{ALC}$ since role assertions can be made defeasible and, since they can be defeated in preferred interpretations, their entailment is not guaranteed. This can be addressed by including support for role hierarchies and disjoint roles, which is straightforward. The resulting language includes non defeasible axioms of the form $R \sqsubseteq S$, $R \sqsubseteq \neg S$ and their defeasible counterparts $R \sqsubseteq S^{[w]}$, $R \sqsubseteq \neg S^{[w]}$. Recall from Section 2.2.2, negated role assertions of the form $\neg R(a, b)$ can be expressed by defining $R' \sqsubseteq \neg R$ and asserting $R'(a, b)$, thus thus allowing reasoning about roles membership by refutation.

$p\text{-}\mathcal{ALC}$ has assumed a unique names assumption; however, this could be relaxed introducing defeasible and non-defeasible variants of name agreement and disagreement axioms. The resulting logic could be useful when working with knowledge bases that include aliases for names.

For nominals, concrete domains, number restrictions and inverse roles (in combinations that retain the finite model property), we believe that the language, tableau and ASP implementations can be extended following the tableau techniques for these languages.

Our approach could be optimised for the languages in the *DL-Lite* family and therefore used to improve on the cautious reasoning using repair semantics observed in [LLR⁺10]. One way of doing this is to build the necessary language constructs into our tableau and the ASP implementation. However, recall from Chapter 2 that the *DL-Lite* languages are designed specifically to exploit efficient strategies for working with a large ABox. A more interesting approach would be to investigate if first-order rewriting could be adapted for use with our approach. Just as the IAR semantics approximate the AR semantics and the ICAR approximate the ICAR semantics we speculate there may be an equivalent $Ip\text{-}\mathcal{ALC}$ semantics that approximates the $p\text{-}\mathcal{ALC}$ semantics. Recent work in extending Ontology Based Data Access (OBDA) to more expressive logics in [BCS⁺16] hints that it may be possible to implement $p\text{-}\mathcal{ALC}$ in full, and perhaps even more expressive variants, using this technique.

8.2.2 Languages without the finite-model property

Our method for proof by refutation is reliant on Proposition 3.11, the proof of which hinges on the finite model property. In languages that lack the finite model property it is possible to construct a knowledge base that is only satisfiable in non-finite interpretations and each includes an infinite number of inconsistencies. For instance, if the preferential semantics of $p\text{-}\mathcal{ALC}$ is extended to $\mathcal{ALCFI}$ to form $p\text{-}\mathcal{ALCFI}$ by introducing inverse role relationships and functional role axioms, counting the inconsistent axiom instances leads to problems.

Example 8.1. Let $\mathcal{K}_{8.1} = \langle \{C(a)\}, \{C \sqsubseteq \exists R.C, C \sqsubseteq R^-.C, \text{func}(R), \text{func}(R^-)\}, \emptyset, \{C \sqsubseteq \perp^{[1]}\} \rangle$ be a $p\text{-}\mathcal{ALCFI}$ knowledge base. The requirement for both R and R^- to be functional

excludes models that include loops. An interpretation of $\mathcal{K}$ that satisfies the non-defeasible axioms requires that individual a belongs to concept C , has an infinite number of distinct R -successors belonging to concept C , and an infinite number of distinct R -predecessors belonging to C . Every one of these falsifies an instance of $C \sqsubseteq \perp^{[1]}$. We conclude that $\mathcal{K}_{8.1}$ does not have a finite m -inconsistency.

We speculate that it may be possible to address this by detecting and eliminating such cyclic structures in a TBox in a similar manner to finding and eliminating incoherence. Recall from Chapter 1 that the Protégé authoring tool [NSD⁺01] can be used to identify incoherence. This is achieved by checking the satisfiability of each concept. It may be possible to define a mechanism that checks the TBox for the necessary cyclic conditions that lead to an infinite sequence.

8.2.3 Synthetic knowledge bases

We plan to extend OWLGensuch that inconsistencies can be introduced into non-synthetic knowledge bases. The current iteration of the tool relies on the the structure within synthetic knowledge bases to guess changes that are “likely” to introduce inconsistencies. However, the approach can be adapted for general knowledge bases at the expense of some efficiency. We will also make our tool available to other research teams under an open-source license in the hope that it may prove useful to the community.

8.3 Alternative methods of implementation

Our current implementation has demonstrated the utility of $p\text{-}\mathcal{ALC}$ for reasoning in the presence of inconsistencies, although the current implementation does not scale beyond knowledge bases that include a few hundreds of axioms. We plan to revisit some of our key choices and explore alternative methods for implementation $p\text{-}\mathcal{ALC}$.

Without ASP

ASP has proved advantageous in that it permits a compact declarative representation of the knowledge base and the tableau rules. The grounder and solver then provide all of the necessary features for the implementation. However, as discussed in Section 8.1.1, heuristic control over the search is challenging to implement within ASP.

Creating a bespoke tableau implementation in language such as C++ and employing a branch-and-bound search would allow the flexibility to incorporate heuristics. Alternatively, the foundation of our tableau on existing algorithms for Description Logic would facilitate embedding our technique within a tableau based reasoner such as Hermit [GHM⁺14] or FACT++ [TH06]. Such an approach would benefit directly from the state-of-the-art developments in the reasoner implementation. However, pursuing such approaches requires a bespoke implementation for identifying the optimal branches. This could be implemented using a branch-and-bound search (or similar) whereas the necessary search machinery comes for free using ASP.

We recognise that it is possible to encode the problem for other state-of-the-art theorem proving and constraint solving systems. For instance, the problem might be encoded for AVATAR [Vor14] by expressing the axioms as a first order theory and defeasibility by applying constraints.

KLM axiomatisation

In Table 2.6 we identified various methods for handling planned exceptions in Description Logics exploiting Kraus, Lemman and Magidor’s approach for axiomatisation of non-classical logics [KLM90]. To our knowledge, this technique has only been applied for inconsistency tolerant reasoning in [QZ13]. Much of the work on KLM axiomatisation has focussed on inferences about the TBox, although inferences for the ABox have been considered in, $\mathcal{ALC}\text{-}\mathbf{T}$ [GGOP09], $\mathcal{ALC}\text{-}\mathbf{T}_{min}$ [GGOP13], $\mathcal{ALC}\text{-}\mathbf{T}_R$ [GGOP15] and Rational closure of the ABox [CMMV13a]. In the latter, inferences about individuals are drawn but typicality statements are not permitted within the ABox. In [GGOP15] the computational complexity of deciding if $C(a)$ belongs to the Rational closure of the ABox in $\mathcal{ALC}$ is shown to be EXP-complete, and therefore the same

as that for the underlying logic.

Given such favourable reasoning characteristics it would be interesting to investigate if the notion of weights (and priorities) used in $p\text{-}\mathcal{ALC}$ could be implemented as a KLM closure and hence lower the complexity of reasoning. During the early development of our work in this thesis, we explored using the cautious reasoning feature of clingo that allows the computation of the intersection of all answer sets of a program. The underlying idea is that cautious reasoning could be exploited to compute the preferred (or rational) consequences of a knowledge base using a suitable encoding of the KLM axiomatisation rules. Unfortunately, the solvers at that time did not permit incremental grounding while computing either brave or cautious answer sets and we eventually moved away from the approach. The current version of clingo now permits incremental reasoning whilst computing cautious and brave answer sets and this technique deserves further exploration.

8.4 Summary

We have demonstrated an implementation of $p\text{-}\mathcal{ALC}$ that can be used to perform inferences in the presence of inconsistencies. For future work we plan to focus on optimisation and enhancements to the current implementation. In the longer term, there are opportunities to consider implementations that replace ASP or, more radically, the tableau algorithm.

We believe that the need for systems that can draw inference from expressive inconsistent knowledge bases will continue to grow. As we build larger and more complex knowledge bases, that are interconnected through use of the web ontology languages, our reliance on consistent knowledge bases for reasoning will become less, and less, tenable. We hope that our contributions provide insight into solutions to this problem and to the broader challenges that lie ahead.

Appendix A

OWL 2 Functional Style Syntax

The knowledge bases discussed in Chapter 6 are stored in OWL 2 Functional Style Syntax (FSS) [BFH⁺09]. Table A.1 provides a (simplified) overview of how DL concepts and axioms are represented in OWL 2 FSS.

Concept	OWL 2 FSS Class expression	Axiom	OWL 2 FSS Axiom
$\neg C$	<code>ObjectComplementOf(C)</code>	$C \sqsubseteq D$	<code>SubClassOf($C D$)</code>
$C_1 \sqcap \dots \sqcap C_n$	<code>ObjectIntersectionOf($C_1 \dots C_n$)</code>	$C \equiv D$	<code>EquivalentClasses($C D$)</code>
$C_1 \sqcup \dots \sqcup C_n$	<code>ObjectUnionOf($C_1 \dots C_n$)</code>	$C \sqsubseteq \neg D$	<code>DisjointClasses($C D$)</code>
$\exists R.C$	<code>ObjectSomeValuesFrom($R C$)</code>	$\exists R.\top \sqsubseteq C$	<code>ObjectPropertyDomain($R C$)</code>
$\forall R.C$	<code>ObjectAllValuesFrom($R C$)</code>	$\top \sqsubseteq \forall R.C$	<code>ObjectPropertyRange($R C$)</code>
		$C(a)$	<code>ClassAssertion($C a$)</code>
		$R(a, b)$	<code>ObjectPropertyAssertion($R a b$)</code>

Table A.1: The representation of DL concepts and axioms in FSS

Each *entity* x in the signature of a knowledge base is introduced by a declaration statement of the form `Declare(x)`. An *annotation property* (a named comment type) called *priority* is declared using: `Declaration(AnnotationProperty(:priority))`. The annotation property is used to record the associated weight of each defeasible axioms and the statement `Annotation(:priority "0"^^xsd:integer)` is used to label that an axiom is non-defeasible.

The OWLGen tool generates synthetic knowledge bases in FSS and implements the translation by τ (as described in Chapter 5). OWLGen is used to translate OWL 2 FSS knowledge bases into ASP facts. The ontology prefix is omitted in the examples for brevity.

The knowledge base `t0055_Named_ExistsForall.owl`

The example test case $\mathcal{K}_{t0055}$ from Chapter 6.

```

Ontology(<http://www.doc.ic.ac.uk/~gbd10/owl/unit/t0055_Named_ExistsForall.owl>
Declaration(Class(:C))
Declaration(Class(:D))
Declaration(ObjectProperty(:R))
Declaration(NamedIndividual(:i))
Declaration(AnnotationProperty(:priority))
ClassAssertion(Annotation(:priority "0"^^xsd:integer) ObjectSomeValuesFrom(:R :C) :i)
ClassAssertion(Annotation(:priority "0"^^xsd:integer) ObjectAllValuesFrom(:R :D) :i)
)

```

The knowledge base `expbranch.owl`

```

Ontology(<http://www.doc.ic.ac.uk/~gbd10/owl/unit/expbranch>
Declaration(Class(:C11))
Declaration(Class(:C12))
Declaration(Class(:C21))
Declaration(Class(:C22))
Declaration(Class(:C31))
Declaration(Class(:C32))
Declaration(ObjectProperty(:R1))
Declaration(ObjectProperty(:R2))
Declaration(ObjectProperty(:R3))
Declaration(NamedIndividual(:a))
Declaration(AnnotationProperty(:priority))
ClassAssertion(Annotation(:priority "0"^^xsd:integer)
  ObjectIntersectionOf(
    ObjectSomeValuesFrom(:R1 ObjectAllValuesFrom(:R2 ObjectAllValuesFrom(:R3 :C11)))
    ObjectSomeValuesFrom(:R1 ObjectAllValuesFrom(:R2 ObjectAllValuesFrom(:R3 :C12)))
    ObjectAllValuesFrom(:R1 ObjectIntersectionOf(
      ObjectSomeValuesFrom(:R2 ObjectAllValuesFrom(:R3 :C21))
      ObjectSomeValuesFrom(:R2 ObjectAllValuesFrom(:R3 :C22))
      ObjectAllValuesFrom(:R2 ObjectIntersectionOf(
        ObjectSomeValuesFrom(:R3 :C31)
        ObjectSomeValuesFrom(:R3 :C32)))))) :a)
)

```

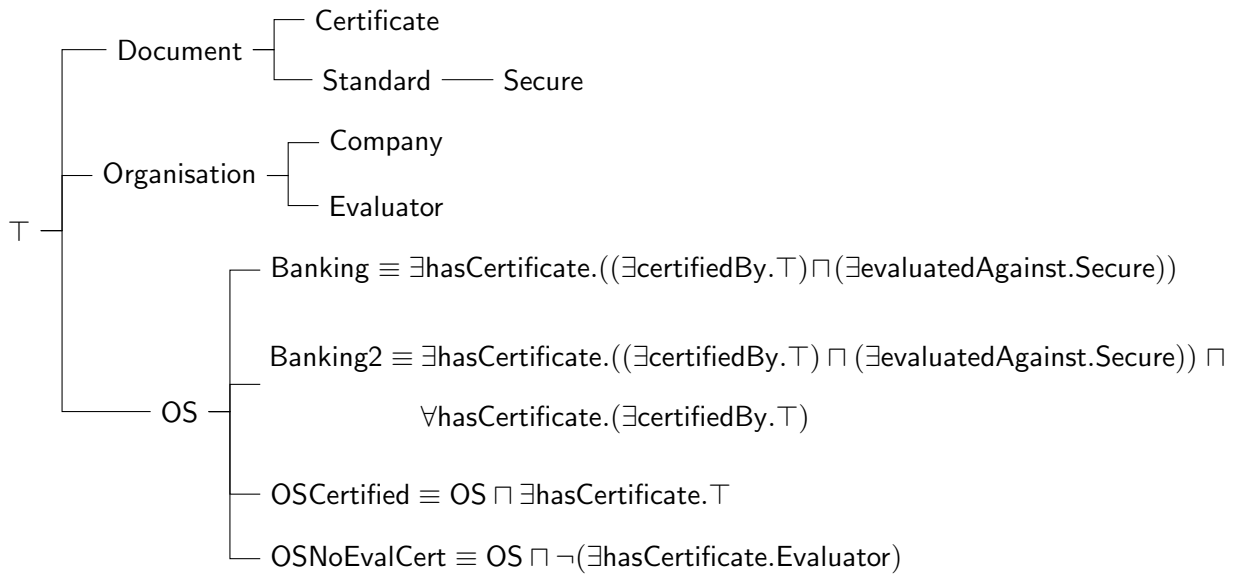
Appendix B

Knowledge bases found in the corpora

This Appendix includes information about the non-synthetic knowledge bases found in the corpora and described in Chapter 6. For each knowledge base we show its dimensions (as quoted in Table 6.3.2) and illustrate the hierarchical structure of the TBox. Examples of equivalences, features and assertional axioms are given to provide additional context.

software.owl

26 individuals, 13 concepts, 4 roles, 8 concept inclusions, 4 equivalences and 2 disjoint classes.



Document, Organisation and OS are mutually disjoint. Certificate and Standard are disjoint.

Example assertions:

Company(apple)

OS(c_aix7)

Evaluator(bsi)

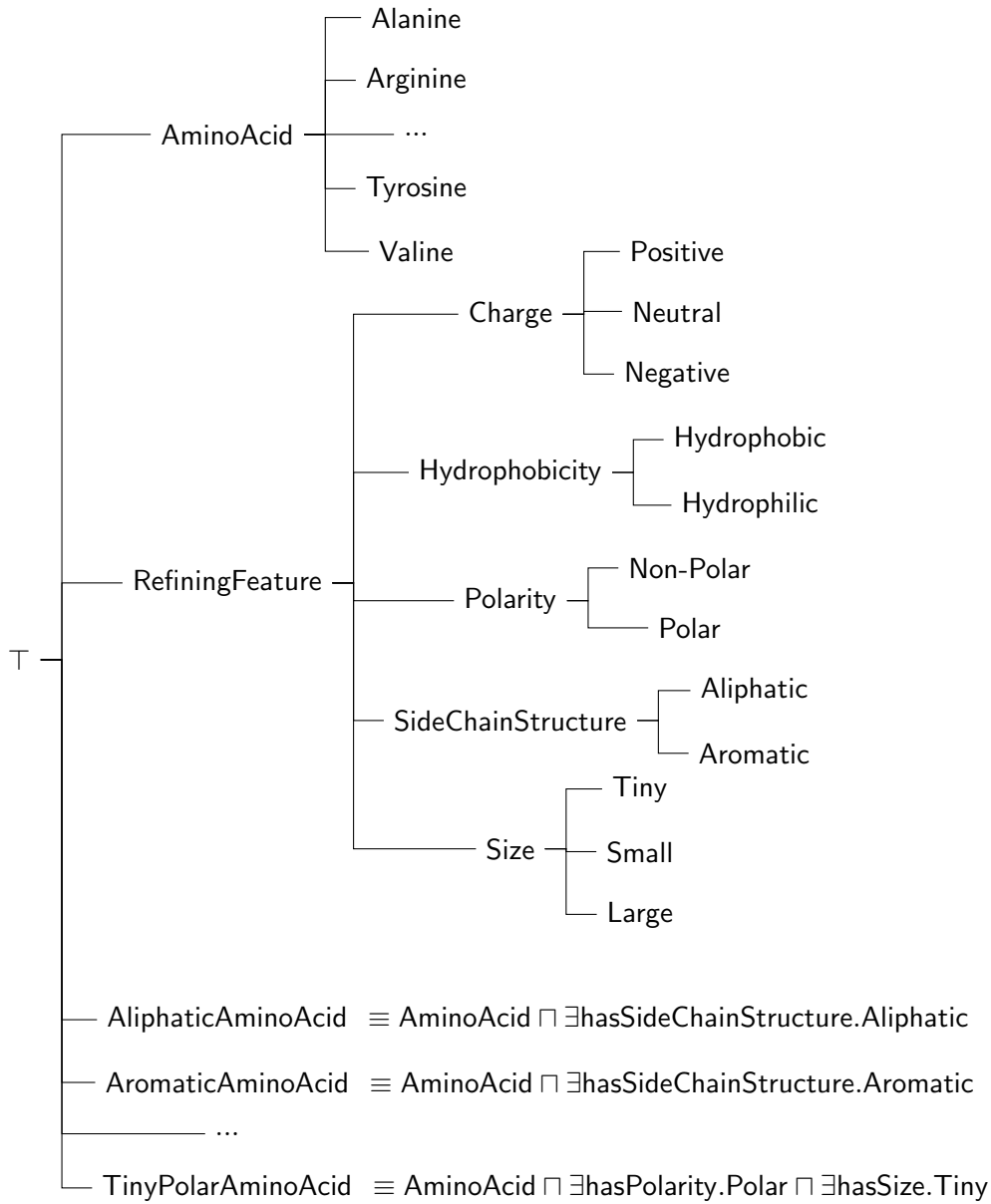
Standard(s_eal4plus)

certifiedBy(c_aix, bsi)

evaluatedAgainst(c_aix, s_eal4plus)

aminoacid.owl

20 individuals, 46 concepts, 5 roles, 238 concept inclusions, 12 equivalences and 199 disjoint classes.



Each amino acid is embellished. For example, for the concept Alanine:

Alanine $\sqsubseteq \forall$ hasCharge.Neutral	Alanine $\sqsubseteq \exists$ hasSideChainStructure.Aliphatic
Alanine $\sqsubseteq \exists$ hasCharge.Neutral	Alanine $\sqsubseteq \exists$ hasPolarity.Non-Polar
Alanine $\sqsubseteq \forall$ hasHydrophobicity.Hydrophobic	Alanine $\sqsubseteq \forall$ hasPolarity.Non-Polar
Alanine $\sqsubseteq \exists$ hasHydrophobicity.Hydrophobic	Alanine $\sqsubseteq \exists$ hasSize.Tiny
Alanine $\sqsubseteq \forall$ hasSideChainStructure.Aliphatic	Alanine $\sqsubseteq \forall$ hasSize.Tiny

20 individuals are instantiated, one to represent each acid:

Alanine(i_alanine)

Arganine(i_arginine)

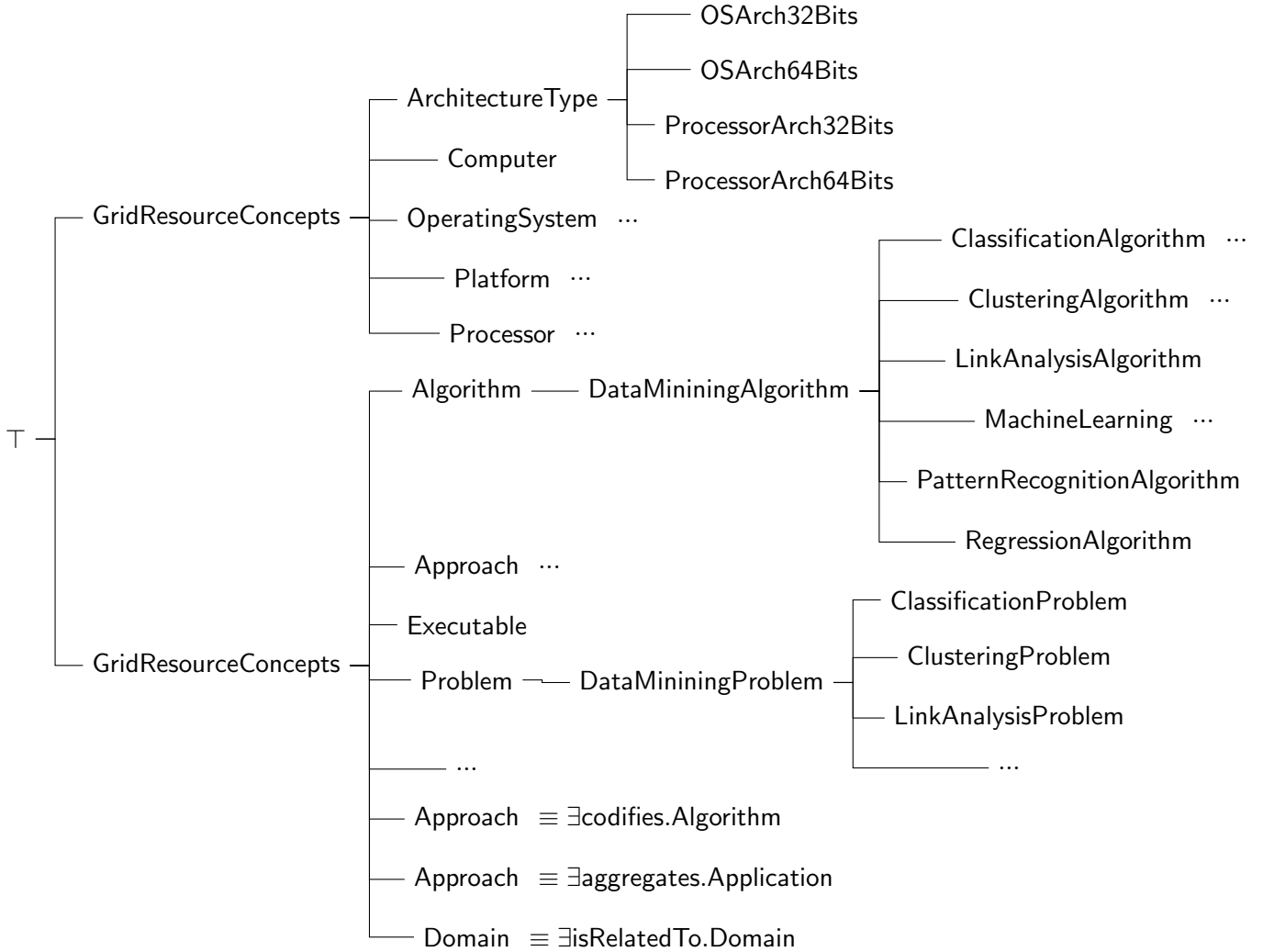
...

Tyrosine(i_tyrosine)

Valine(i_valine)

grid-prime.owl

6 individuals, 117 concepts, 11 roles, 149 concept inclusions, 36 equivalences and 1 disjoint class.



Example embellishments:

NaiveBayesAlgorithm $\sqsubseteq$ LearningAlgorithm

NaiveBayesAlgorithm $\sqsubseteq$ $\exists$ usedFor.ClassificationApproach

SVMAlgorithm $\sqsubseteq$ LearningAlgorithm

SVMAlgorithm $\sqsubseteq$ $\exists$ usedFor.ClassificationApproach

SVMAlgorithm $\sqsubseteq$ $\exists$ usedFor.RegressionApproach

Example assertions:

SVMAlgorithm(ClusteringApproach_Inst)

ClusteringProblem(ClusteringProblem_Inst)

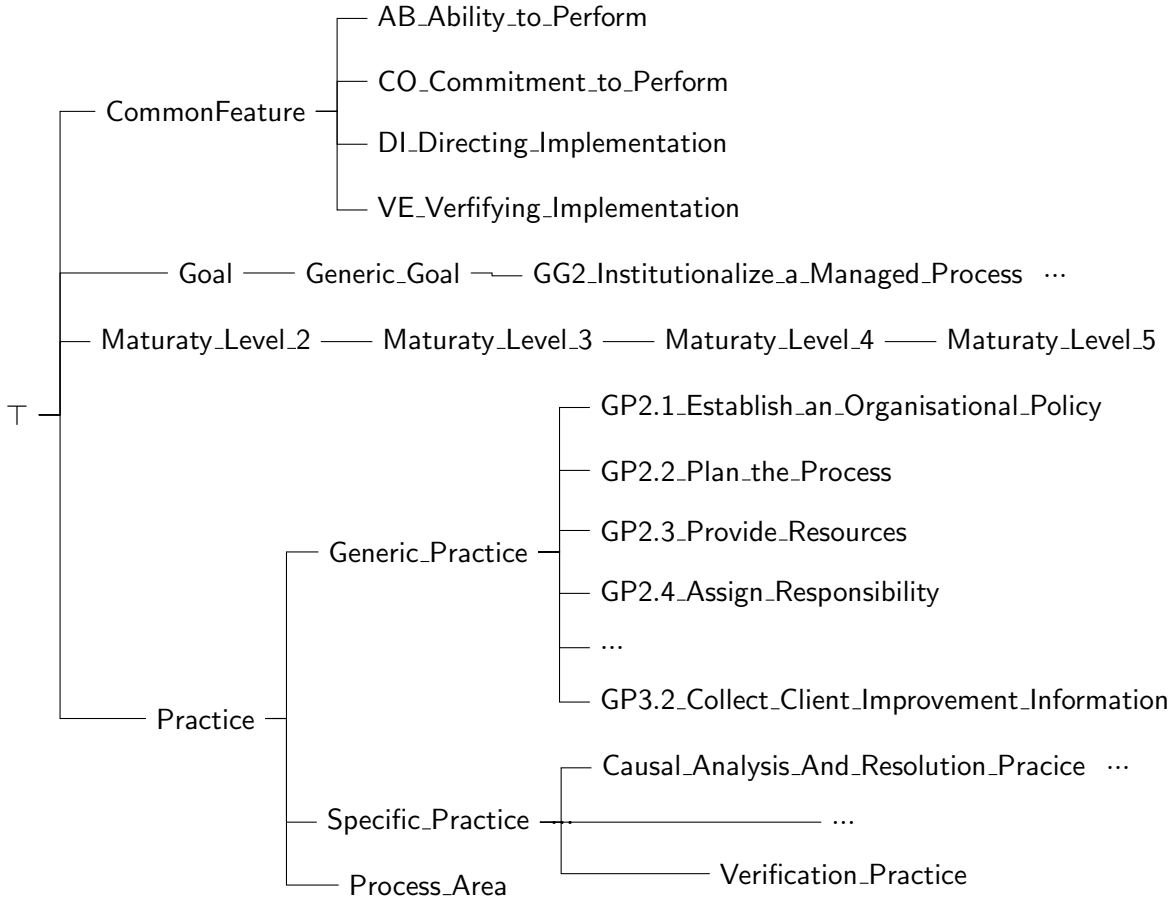
DataMiningDomain(DataMiningDomain_Inst)

dealsWith(ClusteringApproach_Inst, ClusteringProblem_Inst)

isRelatedTo(ClusteringApproach_Inst, DataMiningDomain_Inst)

CMMI.owl

213 individuals, 309 concepts, 4 roles, 304 concept inclusions, 97 equivalences and 1030 disjoint classes.



This ontology includes no embellishments. The relationships are defined through equivalences.

For example:

$\text{Maturity_Level_2} \equiv \exists \text{consistsOf}.\text{Configuration_Management} \sqcap \exists \text{consistsOf}.\text{Measuremant_and_Analysis} \sqcap$
 $\exists \text{consistsOf}.\text{Process_and_Product_Quality_Assurance} \sqcap$
 $\exists \text{consistsOf}.\text{Project_and_Monitoring_and_Control} \sqcap \exists \text{consistsOf}.\text{Project_Planning} \sqcap$
 $\exists \text{consistsOf}.\text{Requirements_Management} \sqcap \exists \text{consistsOf}.\text{Supplier_Agreement_Management}$
 $\text{GP2.5_Train_People} \equiv \exists \text{organisedBy}.\text{AB_Ability_to_Perform}$

Example assertions:

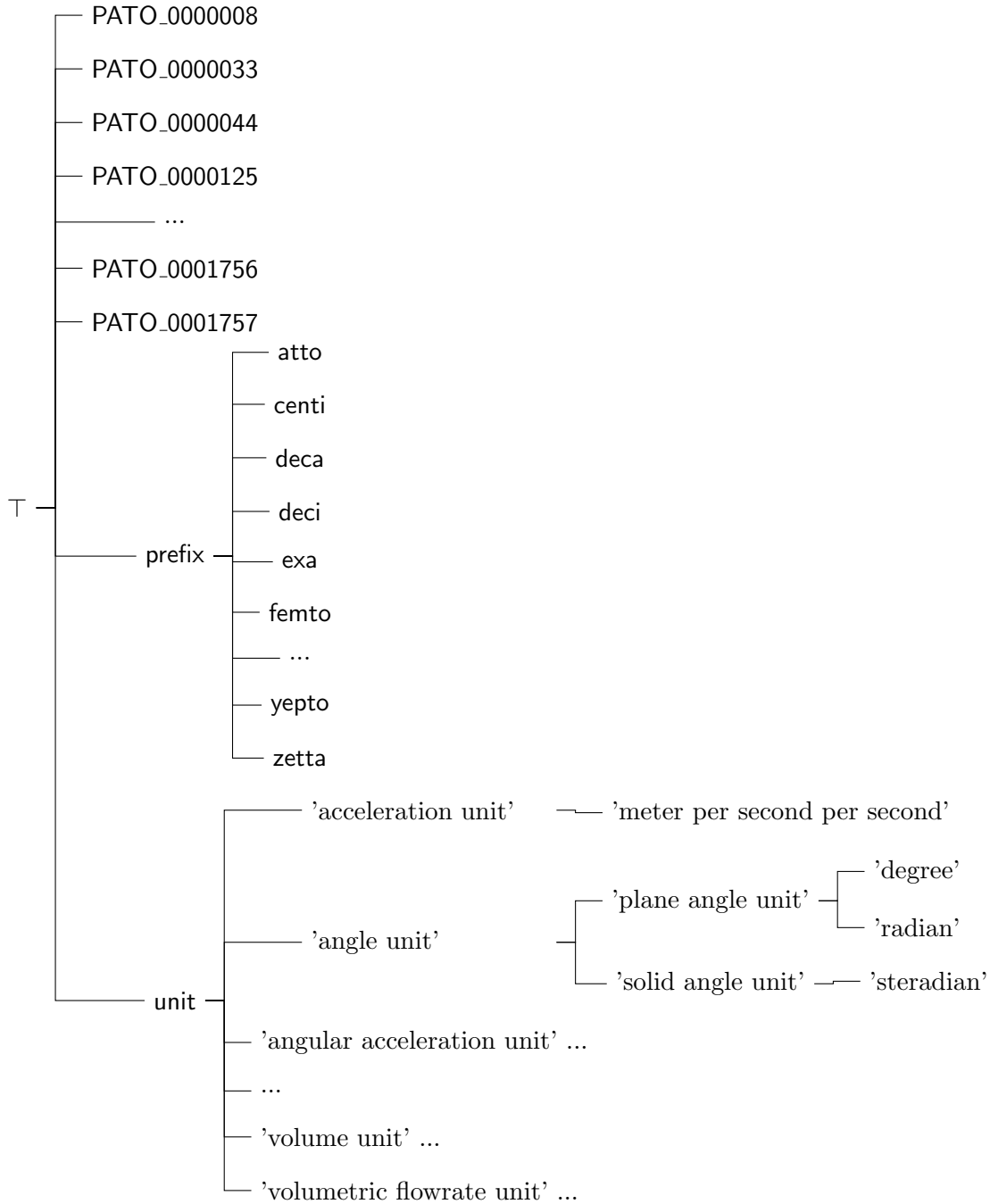
GP2.5_Train_People(O1_GP2.5_Train_People)

AB_ABility_to_Perform(O1_AB_ABility_to_Perform)

organisedBy(O1_GP2.5_Train_People, O1_AB_ABility_to_Perform)

unit-ontology.owl

217 individuals, 510 concepts, 2 roles, 650 concept inclusions, 67 equivalences and 0 disjoint classes.



Bibliography

- [AB02] Guray Alsac and Chitta Baral. Reasoning in description logics using declarative logic programming. In *Proceedings of AAAI*, 2002.
- [AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [ABW88] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In J Minker, editor, *Foundations of deductive databases and logic programming*, volume 1988, chapter 2, pages 89–148. Morgan Kaufmann, 1988.
- [ACKZ09] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyashev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36(1):1–69, 2009.
- [Bak95] Andrew B. Baker. Intelligent backtracking on constraint satisfaction problems: experimental and theoretical results. *PhD Thesis / Dissertation*, 1995.
- [BBH96] Franz Baader, Martin Buchheit, and Bernhard Hollander. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1-2):195–213, 1996.
- [BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. *IJCAI International Joint Conference on Artificial Intelligence*, 5:364–369, 2005.

- [BCM⁺07] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-schneider. *The Description Logic Handbook*. Cambridge University Press, 2007.
- [BCMM13] Katarina Britz, Giovanni Casini, Thomas Meyer, and Kody Moodley. Ordered Interpretations and Entailment for Defeasible Description Logics. Technical report, CAIR, CSIR Meraka and UKZN, South Africa, 2013.
- [BCM^V13] Katarina Britz, Giovanni Casini, Thomas Meyer, and Ivan Varzinczak. Preferential Role Restrictions. In *Proceeding of the 26th International Workshop on Description Logics (DL2013)*, 2013.
- [BCS⁺16] Elena Boteva, Diego Calvanese, Valerio Santarelli, Domenico Fabio Savo, Alessandro Solimando, and Guohui Xiao. Virtual OBDA over Expressive Ontologies: Rewritings and Approximation. In *Proceedings of the 29th International Workshop on Description Logics (DL2016)*, 2016.
- [BED94] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1-2):53–87, 1994.
- [BFH⁺09] Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Michael Smith. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2009. URL: <http://www.w3.org/TR/owl2-syntax/>.
- [BFPS15] Piero A. Bonatti, Marco Faella, Iliana M. Petrova, and Luigi Sauro. A new semantics for overriding in description logics. *Artificial Intelligence*, 222:1–48, 2015.
- [BH95] Franz Baader and Bernhard Hollunder. Embedding Defaults into Terminological Knowledge Representation Formalisms. *Journal of Automated Reasoning*, 14(1):149–180, 1995.

- [BHM08] Katarina Britz, Johannes Heidema, and Thomas Meyer. Semantic Preferential Subsumption. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference (KR-08)*, pages 476–484, 2008.
- [BHS08] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 135–169. Elsevier, 2008.
- [Bie12] Meghyn Bienvenu. Inconsistency-tolerant conjunctive query answering for simple ontologies. In *Proceedings of the 25th International Workshop on Description Logics*, page 59. Citeseer, 2012.
- [BKM99] Franz Baader, Ralf Kiisters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 1, pages 96–101, 1999.
- [BL03] Tim Berners-Lee. WWW Past, present and future, 2003. URL: <http://www.w3.org/2003/Talks/0922-rsoc-tbl/slide1-0.html>.
- [BLR00] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.
- [BLW06] Piero A. Bonatti, Carsten Lutz, and Frank Wolter. Description logics with circumscription. In Patrick Doherty, John Mylopoulos, and Christopher Welty, editors, *Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 400–410. AAAI Press, 2006.
- [BMV11] Katarina Britz, Thomas Meyer, and Ivan Varzinczak. Semantic Foundation for Preferential Description Logics. *AI 2011: Advances in Artificial Intelligence*, 7106:491–500, 2011.
- [BS01] Franz Baader and Ulrike Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69(1):5–40, 2001.

- [Cal96] Diego Calvanese. Finite Model Reasoning in Description Logics. *Proceedings of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 1996)*, 96:292–303, 1996.
- [CCD⁺12] Diego Calvanese, Jeremy Carroll, Giuseppe De Giacomo, Jim Hendler, Ivan Herman, Bijan Parsia, Alan Ruttenberg, Uli Sattler, and Michael Schneider. OWL 2 Web Ontology Language Profiles (Second Edition), 2012. URL: <https://www.w3.org/TR/owl2-profiles/>.
- [CCIL08] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in ASP: Theory and implementation. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming*, volume 5366 LNCS, pages 407–424. Springer Berlin Heidelberg, 2008.
- [CDL⁺07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.
- [CFG⁺12] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-Core-2 Input language format, 2012. URL: <https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf>.
- [Che76] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
- [CMMN14] Giovanni Casini, Thomas Meyer, Kodylan Moodley, and Riku Nortjé. Relevant Closure : A New Form of Defeasible Reasoning for Description Logics. *JELIA*, 8761:92–106, 2014.
- [CMMV13a] Giovanni Casini, Thomas Meyer, Kody Moodley, and Ivan Varzinczak. Non-monotonic reasoning in description logics: Rational Closure for the ABox. In

- CEUR Workshop Proceedings*, volume 1014, pages 600–615. Centre for Artificial Intelligence Research, 2013.
- [CMMV13b] Giovanni Casini, Thomas Meyer, Kody Moodley, and Ivan Varzinczak. Towards practical defeasible reasoning for Description Logics. In *CEUR Workshop Proceedings*, volume 1014, pages 587–599, 2013.
- [CRP⁺93] Roger A. Cote, DJ Rothwell, JL Palotay, RS Beckett, and L Brochu. *The systematized nomenclature of human and veterinary medicine: SNOMED international*. College of American Pathologists; Schaumburg, IL: American Veterinary Medical Association, 1993.
- [CS10] Giovanni Casini and Umberto Straccia. Rational Closure for Defeasible Description Logics. *Lecture Notes in Computer Science*, 6341:77–90, 2010.
- [CS12] Giovanni Casini and Umberto Straccia. Lexichographic Closure for Defeasible Description Logics. In *Proceedings of Australasian Ontology Workshop*, volume 969, 2012.
- [CS13] Giovanni Casini and Umberto Straccia. Defeasible Inheritance-Based Description Logics. *Journal of Artificial Intelligence Research*, 48:415–473, 2013.
- [D’A99] Marcello D’Agostino. *Handbook of Tableau Methods*. Kluwer Academic Publishers, 1999.
- [DLP94] D Dubois, J Lang, and H Prade. Possibilistic logic. In Dov M Gabbay, C J Hogger, and J A Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 3)*, pages 439–513. Oxford University Press, Inc., New York, NY, USA, 1994.
- [DNR02] Francesco M Donini, Daniele Nardi, and Riccardo Rosati. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.

- [DPPY04] Zhongli Ding, Yun Peng, Rong Pan, and Yang Yu. A Probabilistic Extension to Ontology Language OWL. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, volume 00, pages 1–10, 2004.
- [DtEK09] Minh Dao-tran, Thomas Eiter, and Thomas Krennwallner. Realizing Default Logic over Description Logic Knowledge Bases. In Claudio Sossai and Gaetano Chemello, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty: 10th European Conference, ECSQARU 2009, Verona, Italy, July 1-3, 2009. Proceedings*, LNCS/LNAI, pages 602–613. Bolsano, Springer Berlin Heidelberg, 2009.
- [EFFW07] Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):123–165, 2007.
- [EL16] Thomas Eiter and Thomas Lukasiewicz. Generalized Consistent Query Answering under Existential Rules Generalized Inconsistency-Tolerance. In *Proceedings, Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, pages 359–368, 2016.
- [FMK⁺08] G. Flouris, D. Manakanatas, H. Kondylakis, Dimitris Plexousakis, and Grigoris Antoniou. Ontology change: Classification and survey. *The Knowledge Engineering Review*, 23(2):117–152, 2008.
- [FPL10] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2010.
- [GGOP09] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. ALC+T : a Preferential Extension of Description Logics. *Fundamenta Informaticae*, 96(3):1–32, 2009.

- [GGOP13] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. A non-monotonic Description Logic for reasoning about typicality. *Artificial Intelligence*, 195(0):165–202, 2013.
- [GGOP15] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. Semantic characterization of rational closure: From propositional logic to description logics. *Artificial Intelligence*, 226:1–33, 2015.
- [GH08] Stephan Grimm and Pascal Hitzler. Defeasible Inference with Circumscribed OWL Ontologies. *ARea: Scalability and Commonsense*, 2008.
- [GHM⁺14] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.
- [GHVD03] Benjamin N Grosz, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs : Combining Logic Programs with Description Logic Categories and Subject Descriptors. *Representations*, (May):48–57, 2003.
- [Gin93] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993. URL: <http://arxiv.org/abs/cs/9308101>.
- [GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–385, 1991.
- [Gov04] Guido Governatori. Defeasible description logics. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3323 LNCS:98–112, 2004.
- [GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. In *Web Semantics*, volume 3, pages 158–182, 2005.
- [HB11] Matthew Horridge and Sean Bechhofer. The OWLAPI: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

- [HMS07] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.
- [Hol94] Bernhard Hollunder. An Alternative Proof Method for Possibilistic Logic and its Application to Terminological Logics. *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence*, pages 327–335, 1994.
- [HPH15] Ioana Hulpu, Narumol Prangnawarat, and Conor Hayes. Introducing Defeasibility into OWL Ontologies. *The Semantic Web - ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, 9367:409–426, 2015.
- [HPS08] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and Precise Justifications in OWL. *The Semantic Web ISWC 2008*, 5318:323–338, 2008.
- [HPS09] Ian Horrocks, Bijan Parsia, and Uli Sattler. OWL 2 Web Ontology Language Direct Semantics, 2009. URL: <http://www.w3.org/TR/owl2-direct-semantics/>.
- [HRG96] Ian Horrocks, Alan L Rector, and Carole A Goble. A Description Logic Based Schema for the Classification of Medical Data. In *KRDB*, volume 96, pages 24–28. Citeseer, 1996.
- [HS99] Ian R. Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.
- [HT00] Ian Horrocks and Stephan Tobies. Reasoning with Axioms: Theory and Practice. *arXiv preprint cs/0005012*, pages 1–23, 2000. URL: <http://arxiv.org/abs/cs/0005012>.
- [HW09] Hui Hou and Jinzhao Wu. Quasi-classical semantics and tableau calculus of description logics for paraconsistent reasoning in the semantic web. In *Proceedings - 12th IEEE International Conference on Computational Science and Engineering, CSE 2009*, volume 2, pages 703–708, 2009.

- [Kal06] Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, University of Maryland, 2006.
- [KLM90] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic Reasoning, Preferential Models. *Artificial Intelligence*, 44:167–207, 1990.
- [KLP97] Daphne Koller, Alon Levy, and Avi Pfeffer. P-CLASSIC: A tractable probabilistic description logic. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, (August):390–397, 1997.
- [KMH11] Adila Krisnadhi, Frederick Maier, and Pascal Hitzler. OWL and Rules. *Language*, (September):382–415, 2011.
- [KPK06] Vladimir Kolovski, Bijan Parsia, and Yarden Katz. Implementing OWL Defaults. Technical report, DTIC Document, 2006.
- [Lif91] Vladimir Lifschitz. Nonmonotonic Databases and Epistemic Queries. *Proceedings of the 12th International Conference on Artificial Intelligence- Volume 1*, pages 381–386, 1991.
- [LLR⁺10] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant Semantics for Description Logics. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Web Reasoning and Rule Systems*, volume 6333 of *Lecture Notes in Computer Science*, pages 103–117, Berlin Heidelberg, 2010. Springer.
- [LLR⁺15] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant query answering in ontology-based data access. *Journal of Web Semantics*, 33:3–29, 2015.
- [LP14] Michel Ludwig and Rafael Pe. Error-Tolerant Reasoning in the Description Logic EL. *Logics in Artificial Intelligence*, 8761:107–121, 2014.
- [LPF⁺06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge repre-

- sensation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.
- [LT94] Vladimir Lifschitz and Hudson Turner. Splitting a Logic Program. *International Conference on Logic Programming (ICLP)*, 19(1):23–37, 1994.
- [LTW04] Thomas Linke, Hans Tompits, and Stefan Woltran. On Acyclic and Head-Cycle Free Nested Logic Programs . In *Logic Programming: 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004. Proceedings*, pages 225–239. Springer Berlin Heidelberg, 2004.
- [Luk08] Thomas Lukasiewicz. Expressive probabilistic description logics. *Artificial Intelligence*, 172(6-7):852–883, 2008.
- [Mat] Nicolas Matentzoglou. The Manchester OWL Repository. URL: <http://mowlrepo.cs.manchester.ac.uk/>.
- [MBP13] Nicolas Matentzoglou, Samantha Bail, and Bijan Parsia. A Snapshot of the OWL Web. In *The Semantic Web - ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 331–346, Berlin, Heidelberg, 2013. Springer.
- [McC86] John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116, 1986.
- [MH09] Yue Ma and Pascal Hitzler. Paraconsistent reasoning for OWL 2. In *3rd International Conference on Web Reasoning and Rule Systems, RR 2009, co-located with the International Semantic Web Conference ISWC 2009*, volume 5837 LNCS, pages 197–211, 2009.
- [MHL08] Yue Ma, Pascal Hitzler, and Zuoquan Lin. Paraconsistent reasoning for expressive and tractable description logics. In *CEUR Workshop Proceedings*, volume 353, 2008.

- [MMH13] Frederick Maier, Yue Ma, and Pascal Hitzler. Paraconsistent OWL and related logics. *Semantic Web*, 4:395–427, 2013.
- [MMV12] Kody Moodley, Thomas Meyer, and Ivan José Varzinczak. A Defeasible Reasoning Approach for Description Logic Ontologies. Technical report, Technical report, CAIR, CSIR Meraka and UKZN, South Africa, 2012.
- [Mot06] Boris Motik. Reasoning in Description Logics using Resolution and Deductive Databases. *PhD thesis University Karlsruhe Germany*, 2(3):351–384, 2006.
- [NS13] Nadeschda Nikitina and Sven Schewe. Simplifying description logic ontologies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8218(1):411–426, 2013.
- [NSD⁺01] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Ferguson, and Mark A. Musen. Creating semantic web contents with protégé-2000. *IEEE Intelligent Systems and Their Applications*, 16(2):60–71, 2001.
- [Nut87] Donald Nute. Defeasible Reasoning. In *Proceedings of the 20th Hawaii International Conference on Systems Science*, pages 470–477. IEEE Press, 1987.
- [oP12] University of Potsdam. Potassco, the Potsdam Answer Set Solving Collection, 2012. URL: <http://potassco.sourceforge.net/>.
- [OVD⁺08] Femke Ongenaë, Stijn Verstichel, Filip De Turck, Tom Dhaene, Bart Dhoedt, and Piet Demeester. OTAGen: A tunable ontology generator for benchmarking ontology-based agent collaboration. In *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, pages 529–530. IEEE, 2008.
- [Prz88] T C Przymusiński. On the declarative Semantics of Deductive Databases and Logic Programming. In *Foundations of deductive databases and logic programming*, chapter 5, pages 193 – 216. Morgan Kaufmann, 1988.

- [PS89] Peter F Patel-Schneider. A four-valued semantics for terminological logics. *Artificial Intelligence*, 38(3):319–351, 1989.
- [QJPD11] Guilin Qi, Qiu Ji, Jeff Z. Pan, and Jianfeng Du. Extending description logics with uncertainty reasoning in possibilistic logic. *International Journal of Intelligent Systems*, 26(4):353–381, 2011.
- [QP08] Guilin Qi and Jeff Z. Pan. A tableau algorithm for possibilistic description logic ALC. In *Lecture Notes in Computer Science: The Semantic Web*, volume 5367, pages 61–75. Springer Berlin Heidelberg, 2008.
- [QZ13] Guilin Qi and Zhizheng Zhang. Preferential semantics for plausible subsumption in possibility theory. *Minds and Machines*, 23(1):47–75, 2013.
- [Rei80] Raymind Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, apr 1980.
- [Rei87] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
- [Rei07] James Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O’Reilly Media, 2007.
- [RNG93] Alan Rector, W A Nowlan, and Andrzej Glowinski. Goals for concept representation in the GALEN project. *Proceedings of the ... Annual Symposium on Computer Application [sic] in Medical Care. Symposium on Computer Applications in Medical Care*, pages 414–418, 1993.
- [RRGM12] Riccardo Rosati, Marco Ruzzi, Mirko Graziosi, and Giulia Masotti. Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7650 LNCS, pages 337–349, 2012.
- [SHJ15] Kunal Sengupta, Pascal Hitzler, and Krzysztof Janowicz. Revisiting Default Description Logics – and Their Role in Aligning Ontologies. In *Semantic Technology:*

4th Joint International Conference, JIST 2014, Chiang Mai, Thailand, November 9-11, 2014. Revised Selected Papers, pages 3–18. Springer International Publishing, Cham, 2015.

- [Sho87] Yoav Shoham. Nonmonotonic Logics : Meaning and Utility. *IJCAI International Joint Conference on Artificial Intelligence*, 10:388–393, 1987.
- [Šim14] Alexander Šimko. *Logic Programming with Preferences on Rules*. PhD thesis, Comenius University in Bratislava, 2014.
- [Str97] Umberto Straccia. A sequent calculus for reasoning in four-valued description logics. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 343–357. Springer, 1997.
- [Swi04] Terrance Swift. Deduction in Ontologies via ASP. *Logic Programming and Non-monotonic Reasoning*, 2923:275–288, 2004.
- [TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the Third International Joint Conference (IJCAR 2006)*, pages 292–297. Springer Berlin / Heidelberg, 2006.
- [TSL15] Dmitry Tsarkov, Renate A Schmidt, and Zhen Li. Dynamic Backtracking in Description Logic Tableaux. In *Poster at the 28th International Workshop on Description Logics (DL2015)*, 2015.
- [Var82] Moshe Y Vardi. The Complexity of Relational Query Languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.
- [Vor14] Andrei Voronkov. AVATAR: The architecture for first-order theorem provers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8559 LNCS:696–710, 2014.

- [Zol] Evgenty Zolin. Complexity of reasoning in Description Logics. URL: <http://www.cs.man.ac.uk/~ezolin/dl/>.
- [ZQS13] Jinfan Zhu, Guilin Qi, and Boontawee Suntisrivaraporn. Tableaux Algorithms for Expressive Possibilistic Description Logics. In *International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 227–232. IEEE, 2013.
- [ZWWQ15] Xiaowang Zhang, Kewen Wang, Zhe Wang, and Guilin Qi. A Distance-based Paraconsistent Semantics for DL-Lite. In *Knowledge Science, Engineering and Management: 8th International Conference, KSEM 2015, Chongqing, China, October 28-30, 2015, Proceedings*, pages 15–27. Springer International Publishing, 2015.
- [ZXL09] Xiaowang Zhang, Guohui Xiao, and Zuoquan Lin. A tableau algorithm for handling inconsistency in OWL. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5554 LNCS, pages 399–413, 2009.
- [ZXLV14] Xiaowang Zhang, Xiao Xiao, Zuoquan Lin, and Jan Van den Bussche. Inconsistency-tolerant reasoning with OWL DL. *International Journal of Approximate Reasoning*, 55(2):557–584, 2014.