

KU LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

APPLICATIONS OF FEASIBLE INFERENCE FOR EXPRESSIVE LOGICS

Hanne Vlaeminck

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

December 2012

APPLICATIONS OF FEASIBLE INFERENCE FOR EXPRESSIVE LOGICS

Hanne Vlaeminck

Jury:
Prof. Dr. ir. Pierre Verbaeten, president
Prof. Dr. Marc Denecker, promotor
Dr. Joost Vennekens, co-promotor
Prof. Dr. ir. Maurice Bruynooghe
Prof. Dr. Bart Demoen
Prof. Dr. Dave Clarke
Prof. Dr. Vladimir Lifschitz
(University of Texas, Austin)
Prof. Dr. Gerhard Lakemeyer
(Aachen University of Technology)

Dissertation presented in
partial fulfillment of the
requirements for the degree
of Doctor in Engineering

December 2012

© Katholieke Universiteit Leuven – Faculty of Engineering
Address, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotocopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2012/7515/144
978-94-6018-610-3

Preface

How many times have I dreamt of writing this preface, have I composed it in my head? It is still hard for me to grasp that I'm *actually* writing it now, that I've done it! Now that I'm here, looking back I can say that it has been quite a journey, and sometimes a though one too. Sometimes you feel lost, you've lost your motivation, or you simply don't know where your research is going. At times like this, a small thing like the thought of writing these acknowledgements, is what keeps you going.

What also helps of course, is to have friends. Friends who have gone through the same challenges, who know what you're going through, what you're talking about, and are willing to listen to you. Álvaro, Maarten, Fabian and Stephen, you have been such friends, and more. Thank you![1]

While doing a PhD might be difficult at times, fortunately there are highlights too, like the exciting times when a paper gets accepted after hard work, or when you're exploring new ideas, your brains are working at high speed and things are finally starting to come together. Many of these ideas would not have been there without you, Marc, and discussing them, even though tedious at times, was often fun and inspiring! Dankjewel!

I remember that at some point, Marc, you said to me that you thought I was doing too much things aside from my PhD, and that I would never be able to finish it if I kept on doing them. Well, I guess I've proven you wrong now `;-)`. I even think that I've never would have been able to write this dissertation, if it wasn't for those precious moments of guitar, badminton,

[1] Joost and Bart, no worries, I didn't forget about you.

running, climbing, cooking,… that allowed me to unwind my mind, and let my 'frontal lobe', as you like to say, do the work. So to everyone that shared one of these moments with me, dankjewel!

Most days of the past five years, I cycled to work, and doing so, I usually passed the beautiful Arenberg castle. Oftentimes in the evening, there would be signs with the names of my predecessors on, and every time I would imagine how it would be to see *my* name on such a sign, how nice it would be to share that moment with my family. So mama, papa, broer en zusjes, opa en oma, even though you might not have been aware of it, you were there for me, when cycling home in the evening, giving me a little push in the back, telling me to work a bit harder the next day, to reach my goal. Dankjewel!

I think it's fair to say that I would not be writing these words right now, if it wasn't for two people. Joost, you have this amazing ability of listening to what I have to say (or better, trying to say) and then just transforming that seemingly effortlessly into a couple of short sentences that actually make sense and sound convincing! Apart from being, for example, an office mate, a running buddy and a Warcraft team member, most of all you gave me invaluable guidance and advice throughout this journey. I know you're not a fan of sentences without verbs, but I hope you can appreciate the next one nevertheless. Dankjewel!

Finally, my last words here are directed at my very special friend, Bart. Two years ago, it was hard for me to imagine that I would ever be where I am today. You were about to move to Switzerland, I still had a whole lot of work to do before even dreaming of finishing my PhD. What followed was an seemingly endless series of night train trips, easyjet flights and hours of skype calls. All along you kept on telling me to have patience, have faith, work hard, and that everything would work out in the end. And so it did.

Hanne Vlaeminck

Abstract

Knowledge Representation and Reasoning is the area of artificial intelligence that is concerned with how knowledge can be represented symbolically in a formal language, and how computer programs can reason about knowledge in an automatic way. One candidate for such a knowledge representation language is *first-order logic* (FO). Historically people have been doubtful, however, about the usefulness of first-order logic as a knowledge representation language. Indeed, the discovery that first-order logic is undecidable led a whole part of the field into believing that in order to be of any practical use, first-order logic needs to be severely restricted. However, recent trends in the field of computational logic have shown that there do exists *feasible* inference methods for first-order logic, many of which have been demonstrated to be practically useful.

On a different theme, fields such as non-monotonic reasoning and answer set programming have their origin in a conviction that first-order logic is not expressive enough as a KR language. Indeed, several important concepts can not —or at least not naturally— be represented in first-order logic. Instead of turning away from FO and taking a different approach as was done, for example, by the answer set programming community, several researchers have tried to extend FO with different constructs, in order to overcome some of these shortcomings. The language FO(·) is the extension of FO with a number of such constructs. It has been demonstrated that a large variety of practical problems can be represented in FO(·) in an intuitive way.

In this text, we explore in a number of settings how the relevant problems can be solved in a feasible way, while relying on FO(·) as the main knowledge representation language. Often,

the answer will be to use cheap forms of inference, such as *propagation*. In the first part of this dissertation we study a propagation method for first-order logic, and extend it to first-order logic with inductive definitions, which is denoted by FO(ID). In the rest of this dissertation we start by an exploration of a *Knowledge Base System* based on FO(·), through a case study about configuration software. We catalogue a number of key reasoning tasks that arise naturally within the context of configuration software, and investigate how they can be handled efficiently.

When inference tasks become too hard to be of practical use, a common solution is to approximately solve that task. In the literature, approximative approaches to a number of problems can be found. We propose a framework to approximately solve finite domain $\exists\forall SO(ID)$ satisfiability problems. Our approach provides a general framework for these approximative approaches in the literature, and can also be used for solving useful practical problems.

Next we turn our attention to reasoning about and representing sensing actions and the knowledge of an agent. We extend FO(·) even further so that we can intuitively represent how the knowledge of an agent changes through sensing actions. This leads to a very general representation language. We show how the propagation method for FO(ID) can be used to define an approximative inference method for solving the projection problem.

We conclude this dissertation by looking at Ordered Epistemic Logic. Instead of further extending a language, here we do the opposite. We observe that many existing epistemic languages do not maintain the stratified structure that is inherent in many practical examples. This complicates the semantics and reasoning procedures considerably. By defining a logic that keeps this inherent structure explicit, we have the benefit of still having a very expressive logic, in which most of these examples can be naturally represented, while inference tasks have a lower complexity.

Samenvatting

Kennisrepresentatie en Redeneren is het deelgebied in het onderzoeksdomein van artificiële intelligentie waar onderzocht wordt hoe kennis symbolisch kan worden voorgesteld in een formele taal, en hoe computerprogramma's automatisch kunnen redeneren over deze kennis. Een mogelijke kandidaat voor zo'n kennisrepresentatietaal is *eerste orde logica* (FO). Historisch gezien trekken mensen het nut van eerste orde logica als kennisrepresentatietaal echter vaak in twijfel. De ontdekking dat eerste orde logica onbeslisbaar is, zorgde ervoor dat een een gans deel van de onderzoeksgemeenschap van mening was dat, om van enig nut te zijn, de expressiviteit van FO sterk moet beperkt worden. Recente trends in computationele logica tonen echter aan dat er haalbare inferentiemethodes voor FO bestaan, en vaak blijken deze praktisch zeer nuttig te zijn.

Onderzoeksgebieden zoals non-monotonic reasoning en answer set programming zijn langs de andere kant ontstaan uit een overtuiging dat eerste orde logica niet expressief genoeg is als kennisrepresentatietaal. Verschillende belangrijke concepten kunnen immers niet —of op zijn minst niet op een natuurlijke manier— worden voorgesteld in FO. In plaats van zich af te keren van FO en een andere aanpak te volgen, zoals bijvoorbeeld gedaan werd in de answer set programming gemeenschap, hebben een aantal onderzoekers verschillende uitbreidingen van FO voorgesteld, die een aantal van deze tekortkomingen aanpakken. Het resultaat hiervan is de taal FO(·), in welke een grote variëteit van praktische problemen kan worden voorgesteld op een intuïtieve manier.

In deze tekst onderzoeken we, voor een aantal verschillende situaties, hoe de relevante problemen kunnen opgelost worden op een praktisch haalbare manier, met FO(·) als kennis-

representatietaal. Vaak zullen we hierbij goedkope vormen van inferentie gebruiken, zoals *propagatie*. In het eerste deel van deze dissertatie bestuderen we een propagatiemethode voor eerste orde logica en breiden we ze verder uit naar eerste orde logica met inductieve definities, een kennisrepresentatietaal die genoteerd wordt met FO(ID). In de rest van deze dissertatie starten we met een theoretische studie van een *kennisbank systeem* gebaseerd op FO(·). We doen dit aan de hand van een case-study over configuratie software. We identificiëren een aantal belangrijke inferentietaken die naar voor komen in de context van zo'n configuratie software en onderzoeken op welke manier deze efficiënt kunnen worden opgelost.

Wanneer inferentietaken te moeilijk worden om nog praktisch van nut te zijn, is het een vaak gebruikte oplossing om die taak approximatief op te lossen. In de literatuur zijn dan ook een aantal approximatieve aanpakken voor verschillende problemen te vinden. We stellen een raamwerk op voor het approximatief oplossen van eindige domein $\exists\forall SO(ID)$ satisfieerbaarheidsproblemen. We tonen aan dat onze aanpak kan gezien worden als een algemeen kader voor een aantal zo'n approximatieve methodes in de literatuur, en bovendien ook kan gebruikt worden voor een aantal praktische applicaties.

Vervolgens bestuderen we hoe sensing acties en de kennis van een agent in dynamische systemen kunnen worden voorgesteld en hoe daarover kan worden geredeneerd. We breiden FO(·) nog verder uit zodat we op een intuïtieve manier kunnen voorstellen hoe de kennis van een agent verandert door middel van sensing acties. Dit levert ons een zeer algemene representatietaal op. We tonen ook hoe we de propagatiemethode voor FO(ID) kunnen gebruiken om een approximatieve inferentiemethode voor het projectieprobleem te definiëren.

We beëindigen deze dissertatie met het bestuderen van Geordende Epistemische Logica. In plaats van opnieuw een taal te gaan uitbreiden, doen we nu het tegenovergestelde. We observeren dat veel bestaande epistemische talen de inherente gestratifieerde structuur die aanwezig is in veel praktische voorbeelden, niet bewaren. Dit maakt zowel de semantiek als de redeneermethoden zeer complex. Door een logica te definiëren die die inherente structuur wel expliciet bewaart, krijgen we een logica die nog steeds zeer expressief is en waarin de meeste van deze voorbeelden op een natuurlijke manier kunnen worden voorgesteld, terwijl inferentietaken een lagere complexiteit hebben.

Contents

Preface i

Abstract iii

Samenvatting v

Contents vii

List of symbols xi

1 Introduction 1

1.1 Structure of the text 6

2 Preliminaries 7

2.1 First-order logic 7

2.2 Approximation theory 17

2.2.1 Sets, functions, orders, lattices and operators . . . 17
2.2.2 Approximation fixpoint theory . . . 18
2.3 FO(·) . . . 19
2.3.1 Inductive definitions . . . 20
2.3.2 Other extensions of FO . . . 24
2.3.3 Representing dynamic domains in FO(·) . . . 25
2.4 FO(FP) . . . 28
2.5 Autoepistemic Logic . . . 31

3 Propagation for FO(ID) 33
3.1 Propagation for FO . . . 35
3.2 Propagation for FO(ID) . . . 43
3.3 Conclusions . . . 54

4 Knowledge Base Systems: a case study for configuration software 57
4.1 Case study: a course selection system . . . 59
4.1.1 Representing the domain knowledge . . . 61
4.1.2 Using the Knowledge Base . . . 62
4.1.3 Implementation . . . 69
4.2 Related work . . . 72
4.3 Conclusions . . . 73

5 Approximating ∃∀SO satisfiability problems 77
5.1 Approximating $\exists\forall SO$-satisfiability problems . . . 80
5.1.1 A naive method . . . 81
5.1.2 A more complete method . . . 82
5.2 Approximating $\exists\forall SO(ID)$-satisfiability problems . . . 86
5.3 Experimental evaluation . . . 89

5.4 Applications and related work . . . 90
5.4.1 Conformant Planning . . . 90
5.4.2 Querying and reasoning in open databases . . . 96
5.5 Conclusions . . . 100

6 Representing and reasoning with sensing actions 101
6.1 Representing knowledge and sensing actions . . . 102
6.1.1 Syntax . . . 103
6.1.2 Semantics . . . 105
6.2 An approximative algorithm for the projection problem . . . 117
6.3 Conclusions and related work . . . 120

7 Ordered epistemic logic: semantics, complexity and applications 123
7.1 Syntax and semantics . . . 125
7.1.1 Ordered Epistemic Logic . . . 126
7.1.2 Distributed Ordered Epistemic Logic . . . 131
7.2 Problem Solving . . . 134
7.2.1 From OEL and d-OEL to FO . . . 134
7.2.2 Complexity of model checking, satisfiability checking and query answering . . . 136
7.2.3 Approximative reasoning for OEL and d-OEL . . . 139
7.3 Related work . . . 142
7.3.1 Answer Set Programming . . . 143
7.3.2 Default Logic . . . 145
7.3.3 Other Hierarchical Approaches . . . 147
7.3.4 More Related Work . . . 148
7.4 Conclusions . . . 150

8 Conclusions **151**

A Example 3.9 **155**

B Nested Fixpoint Approximations **157**

B.1 Proof of Theorem 3.25 157

B.2 A sound FO(FD) approximation of $\Delta \wedge \psi$ 161

Bibliography **165**

Curriculum vitae **175**

List of publications **177**

List of symbols

$I, J, \ldots$	Interpretations for a vocabulary, page 10
$\mathcal{I}^{tf}$	The two-valued Σ^{tf} structure associated to $\mathcal{I}$, page 14
$\Sigma^{ct/pt}$	The vocabulary extending Σ symbols P^{ct} and P^{pt} for every symbol P in Σ, page 47
Σ^{tf}	The vocabulary that contains symbols P^{ct} and P^{cf} for each predicate symbol P in Σ, page 14
$\mathcal{I}$	A four-valued or three-valued interpretation, page 12
$Mod(\mathcal{T})$	The unique belief system of an OEL or d-OEL theory, page 129
$\models_{OEL}$	The satisfaction relation for OEL , page 128
$\models_{dist}$	The satisfaction relation for d-OEL , page 134
Σ	A first-order vocabulary, page 8
Σ^F	The function symbols of vocabulary Σ, page 8
Σ^P	The predicate symbols of vocabulary Σ, page 8
$\leq_p$	The precision order, page 12
$\leq_t$	The truth order, page 11

φ^{+I-J}	The pos-neg evaluation of φ in I and J, page 12
φ^{cf}	The formula $(\neg\varphi[\bar{x}])^{ct}$, page 14
$\varphi^{ct/pt}$	The formula obtained by replacing in φ all positive occurrences of P by P^{ct} and negative occurrences by P^{pt}, page 47
φ^{ct}	The Σ^{tf}-formula obtained by replacing all positive occurrences of atoms $P(\bar{t})$ by $P^{ct}(\bar{t})$ and all negative occurrences by $\neg P^{cf}(\bar{t})$, page 14
$sv_{(I,J)}(\varphi)$	The supervaluation of φ in the pair (I, J), page 14
$C_A^{\downarrow}$	The lower stable operator of A, page 19
$C^{\uparrow}$	The upper stable operator of A, page 19
T_Δ^O	The two-valued immediate consequence operator of Δ given O, page 24
$\mathcal{T}_\Delta^{\mathcal{O}}$	The four-valued immediate consequence operator for a definition Δ, page 23
$\mathcal{C}_A$	The partial stable operator associated to A, defined as $(C_A^{\downarrow}(y), C_A^{\uparrow}(x))$, page 19
τ	Isomorphism between pairs of interpretations and four-valued interpretations, page 13
$Compl(\Delta)$	The completion of a definition Δ, page 24
$Def(\Delta)$	The defined symbols of definition Δ, page 21
$Open(\Delta)$	The set of symbols that are open with respect to definition Δ, page 21
$WFM_{\mathcal{O}}(\Delta)$	The well-founded model of a definition Δ given $\mathcal{O}$, page 23
Δ	A least fixpoint definition in the context of FO(FP), page 30
Δ	An inductive definition, page 21
$\Delta_{\mathcal{I}}$	The definition associated to the four-valued interpretation $\mathcal{I}$, page 42
$\mathcal{D}$	A fixpoint definition in the context of FO(FP), page 30
$\mathcal{R}$	A rule set in the context of FO(FP), page 30
∇	A greatest fixpoint definition in the context of FO(FP), page 30
$Mod(T)$	The set of all models of theory T, page 24
$\mathcal{I}_\Sigma$	The set of all possible interpretations of a propositional vocabulary Σ, page 32

$\mathcal{W}_\Sigma$	The set of possible world structures for a propositional vocabulary Σ, page 32
$(I, \mathbb{K})$	A knowledge line structure, page 111
$(I, \mathcal{K})$	A knowledge structure, consisting of an interpretation I and a set of interpretations $\mathcal{K}$ over a vocabulary Σ_{ltc}, page 107
$Upd_I(i)$	All the information an agent learns at timepoint i, page 108
$Upd_I^{\leqslant}$	All the information the agent has learned up to timepoint i, page 113
Σ_{ltc}	A vocabulary in the context of the linear time calculus, page 26
$\mathcal{S} \circ Upd_I(i)$	The i-update of a knowledge structure $\mathcal{S} = (I, \mathcal{K})$, page 109
$\mathbb{K}$	A knowledge line, page 111
$\mathcal{K} \circ \Psi$	The set of all $J \in \mathcal{K}$ for which $J \models \Psi$, page 108
$Eff_I(A(\bar{d}, i))$	The set of all ground formulas that the agent learns by performing the sensing action $A(\bar{d}, i)$ in the real world I, page 108
$\mathcal{H}_{Q,I}(\varphi)$	The truth function in autoepistemic logic, page 33
$App_\sigma^{ct/pt}(\Delta)$	Inductive definition encoding the three-valued well-founded model of Δ in terms of σ, page 47
$Approx_\sigma(\psi)$	Inductive definition encoding valid propagations for a first-order formula ψ, page 41
$\mathcal{D}_\sigma^{\Delta \wedge \varphi}$	Inductive definition encoding valid propagations sanctioned by an FO(ID) theory $\Delta \wedge \varphi$, page 53

1 Introduction

This dissertation is situated in the field of *Knowledge Representation and Reasoning* (KRR), the area of artificial intelligence (AI) that is concerned with how knowledge can be represented symbolically in a formal language, and how computer programs can reason about knowledge in an automatic way. But what do we mean exactly by "knowledge", "representation" and "reasoning", and why are these concepts relevant to AI and computer science (CS) in general?

The question "what is knowledge" has been debated about since the ancient Greeks. According to Plato, knowledge is a "justified true belief". However, in general, knowledge is a complex matter. For example, not everything is either true or false, not all true beliefs are justified. It will come as no surprise that numerous philosophers throughout history have tried to define knowledge, and—perhaps as an even lesser surprise—no single agreed upon definition exists.

What people in the field of KRR usually mean by the term knowledge (and therefore also how we will interpret it throughout this dissertation), is a piece of *information*, a *proposition about some property of the world* that can be either *true* or *false*, right or wrong and that can be expressed by simple declarative sentences such as "all humans are mortal". Such pieces of information could be knowledge in the philosophical sense, belief, or even disbelief.[1]

Possessing knowledge (i.e., information) is obviously an important aspect of intelligence, but, some might say, what really makes us human is the ability to think and *reason* about

[1] Confusing, no? 'Information Representation and Reasoning' would probably be a better name for the field.

our knowledge. Imagine that we have the following two pieces of knowledge.

1. Socrates is human.
2. All humans are mortal.

Most people would agree that it is indeed *reason* that allows us to conclude that Socrates, as any other human being, is mortal. In fact, this type of inference, called *deductive reasoning*, where we try to prove that a certain fact is entailed from a set of premises, has been studied since antiquity. But just as with the question of what knowledge is, reasoning is a complex matter, and does not end with deduction. Imagine, for example, a chess-board and a number of queens which have to be placed on that board such that none of them is attacking another queen along a row, column or diagonal. It is also *reasoning*, albeit different from deductive reasoning, that allows us to come up with a correct solution.

One might ask himself why this topic, that for such a long period in time, seems to have mostly been a playground for philosophers, is relevant for AI and CS in general. Obviously, the interest of the AI community in knowledge and reasoning is that we want to build systems that know things about the world and do not act unreasonably. But KRR is also relevant outside the AI field. Indeed, *any* system, AI-based or not, can be said to have knowledge about its world. For example, it is obvious that a Java compiler needs to know a lot about the Java language. We also see that in more mundane CS applications various forms of (automated) reasoning become more and more important, such as intelligent query answering in the context of the semantic web, automated planning and proving termination and other critical properties of programs.

As the field of possible KRR applications grew bigger (and still grows), researchers have been investigating how to formally write down all the different forms of knowledge that can be found in these application domains in a natural way. As a result, there is now a whole array of knowledge representation languages. Some of these languages differ only superficially in some syntactic sugar, while others differ more significantly on the *semantic* level. So which knowledge representation language should we choose?

In the early days of AI, hopes where high that *first-order logic* (FO), while originally invented for the formalization of mathematical knowledge and inference, could be co-opted for more general KRR purposes. Indeed, the ultimate goal in the KRR field is to represent knowledge formally and reason automatically with it, and after all, logic was one of the first serious attempts to formalize knowledge and inference. However, gradually, many lost this hope, and the conviction that first-order logic was not useful for declarative problem solving as it was, led to the creation of, for example, the Description Logics and Logic Programming communities, which study restricted variants of FO, in order to make theorem proving decidable and efficient. However, it is interesting to see that researchers have turned away from deduction as the main reasoning task, and focussed on 'cheaper'

forms of reasoning, such as finite model checking, finite or bounded model generation, or approximate reasoning, and moreover, have shown that many real-world problems can indeed be solved by such forms of inference.

On the other hand, researchers discovered a number of shortcomings of FO as a knowledge representation language. This was one of the main motivations behind the non-monotonic reasoning field. While, for example, answer set programming (ASP) took a different way and left first-order semantics behind, other researchers such as Oberschelp (1989); Denecker & Ternovska (2008); Pelov et al. (2007) have all proposed *extensions* to FO to overcome some of these shortcomings. While formalisms such as ASP certainly have their own merits, we feel that one of the strong points of FO (or extensions of FO) as a knowledge representation language, is that there is a clear mapping from its formal to its informal semantics. The definition of the formal semantics of FO contains, amongst others, the following rule,

$$I \models A \wedge B \text{ iff } I \models A \textit{ and } I \models B,$$

which essentially is also a *translation rule* that says that we should read the formal symbol $\wedge$ as *and*.[2] Also, as many practical examples have demonstrated, the connectives $(\wedge, \neg, \forall, \ldots)$ from first-order logic (FO) seem almost indispensable for any serious knowledge representation. Indeed, subsets of them are present in some form in nearly all knowledge representation languages.

On a different theme, while they all have their own strong points, many declarative formalisms and knowledge representation languages forsake another old dream of the AI community, that is, that one specification can be re-used for different tasks and forms of inference. Indeed, we see that most formalisms are all fine-tuned towards one particular form of inference, for example, SQL for query answering, description logics for theorem proving, ASP for answer set generation, constraint programming languages for constraint programming, etc. This is not the case for FO and its extensions.

All this motivates our belief that FO with a number of these extensions, which we will denote by FO(·), is a good knowledge representation language, and it will serve as the main knowledge representation language throughout this dissertation. We argue that as such, FO(·) not only offers the important fundamental constructs of FO, but also the many extensions that are necessary for practical applications, while still being able to re-use the same representation for different tasks. The main goal of this dissertation is to add further proof to the claim that it *is* possible to have feasible forms of inference for expressive logics such as FO(·). We will do this by exploring in a number of settings how the relevant problems can be solved in a feasible way, while relying on FO(·) as the main knowledge representation language. Often, the answer to this question will be to use cheap forms of

[2] One might argue that natural language is ambiguous. However, it might be more correct to say that natural language connectives are *overloaded*. Nevertheless, it appears that in practice, people are more than capable of disambiguating such overloading. In mathematical definition as above, for example, most people would agree that the 'and' does not refer to a temporal sequential conjunction, such as 'I got up and brushed my teeth'.

inference, such as *propagation*. This propagation method was first presented in Wittocx et al. (2010) for FO and in Chapter 3 we present this method and extend it to FO(ID), the extension of FO with a construct to represent inductive definitions.

In Chapter 4 we investigate the feasibility of a *Knowledge Base System* (KBS), a system storing (declarative) domain knowledge and able to solve a *range of tasks* and problems in that domain, by applying various forms of inference on its knowledge base. It is a fundamental goal in the field of Knowledge Representation and Reasoning to develop a KBS system. As an example, imagine a KBS storing a specification of course scheduling at a university, and able to solve or support tasks of generating schedules at the start of the year, but also of verifying correctness of hand-made or revised schedules, of updating or revising the current schedule under additional or changed requirements, etc., all using the same knowledge base. The difference between the KBS paradigm and declarative programming frameworks such as logic programming, answer set programming (ASP) or constraint logic programming (CLP), lies in the reuse of the knowledge base for solving different problems and tasks requiring different forms of inference: deduction, model checking, model generation, update and revision, abduction, learning, etc. Thus, a knowledge base does not encode a solution for a specific problem, nor is it a declarative program with an operational semantics induced by one specific form of inference. The knowledge base is "only" a formal representation of declarative properties of the domain.

We will investigate the feasibility of a such KBS system, based upon the knowledge representation language FO(·), from a theoretical point of view. We do this through a logical analysis of one particular configuration software application. While this case study is just a beginning, and leaves many research questions, in particular towards the real-world scalability and applicability, open, we nevertheless show that, at least for this case study, the domain knowledge can be expressed naturally and modularly in FO(·), and more importantly, we categorize a number of inference tasks that arise naturally within the context of configuration software, and show how these can be implemented.

A common way to tackle the expressivity-efficiency trade-off, is to solve inference tasks that have a high complexity *approximately*, that is, through a method that works fast(er), when it returns a solution it is guaranteed to be correct (i.e., the method is *sound*), but solutions are not always guaranteed to be found, even though they might exist (i.e., the method is *incomplete*). Many such approximative methods can be found in the literature, and in Chapter 5 we provide a *general framework* for a number of these approaches in the form of an approximative reasoning method for $\exists\forall SO(ID)$ satisfiability problems (a fragment of second-order logic extended with inductive definitions). We use a syntactic representation of the propagation method for FO(ID) to transform such an $\exists\forall SO(ID)$ satisfiability problem to an $\exists SO(ID)$ satisfiability problem. The finite domain satisfiability problem for the latter can then be handled by several existing solvers. We also show how we can use this framework for solving useful problems, such as conformant planning, in an effective way.

Many temporal domains can be naturally represented in FO(·) and tasks such as planning, conformant planning and projection can be formulated as inference tasks for FO(·). In temporal domains that involve an agent operating in a partially known environment however, the knowledge of an agent is just as important as the state of the world. The agent needs to reason about his knowledge and acquire extra knowledge through sensing actions. In Chapter 6 we therefore extend the syntax and semantics of FO(·) even further such that the evolution of the knowledge of such an agent in a temporal domain can be modeled. This leads to a very general framework for representing actions and sensing. Since our semantics uses a Kripke-style representation of the agent's knowledge, inference tasks become computationally challenging. Fortunately, our framework allows us to apply the propagation method from Chapter 3 to obtain a sound but incomplete method to solve the projection problem in polynomial time.

In the last chapter of this dissertation we turn our attention to epistemic reasoning. Many examples of epistemic reasoning in the literature exhibit a stratified structure. In autoepistemic logic, default logic and answer set programming (ASP) this inherent stratification is not preserved as theories may refer to their own knowledge or logical consequences. Defining the semantics of such logics requires a complex mathematical construction. Ordered Epistemic Logic (Konolige, 1988a; Denecker et al., 2010b) (OEL) was introduced precisely with the goal of maintaining this stratification, which allows us to define an easy to understand semantics. Again, this logic integrates seamlessly into classical logic and its other extensions. We prove that inference tasks have a lower complexity than in autoepistemic logic and default logic and we investigate how model generation and query answering can be approximately solved.

To summarize, in this dissertation we showed for a number of different settings, that it is possible to have feasible forms of inference, while having an expressive knowledge representation language. More precisely, our contributions are the following.

1. We extend the FO propagation method from (Wittocx et al., 2010) to FO(ID) theories.
2. We investigate the feasibility of an FO(·) KBS through a logical analysis of a piece of configuration software.
3. We present a general framework for approximately solving $\exists\forall SO(ID)$ satisfiability problems, which generalizes a number of existing approximative approaches in the literature, and which can be used to solve a number of useful problems, such as conformant planning.
4. We propose a new framework for representing and reasoning with sensing actions, and give a sound but incomplete inference method to solve the projection problem.
5. We further extend the initial work by (Konolige, 1988a; Denecker et al., 2010b) about OEL in several ways. We prove complexity results, investigate the relationship with

autoepistemic logic, illustrate the use of model generation for OEL, and define a distributed variant of the logic.

1.1 Structure of the text

The text is structured as follows:

- Chapter 2 introduces some of the background required for the rest of text: first-order logic, several extensions of it such as, e.g., inductive definitions, and auto-epistemic logic.
- Chapter 3 presents propagation for FO(ID). The propagation method in Section 3.1 was first published in (Wittocx, Denecker, & Bruynooghe, 2010), but can be found under this form, together with most of the rest of this chapter in (Vlaeminck, Vennekens, Denecker, & Bruynooghe, 2012c).
- In Chapter 4 we investigate the feasibility of an FO(·) KBS, through a configuration software case study. The results in this chapter were published in (Vlaeminck, Vennekens, & Denecker, 2009).
- Chapter 5 presents a general framework for approximating $\exists\forall SO(ID)$ problems. The contributions in this chapter have been published in (Vlaeminck, Vennekens, Denecker, & Bruynooghe, 2012c).
- In Chapter 6 we propose a new framework for representing and reasoning with sensing actions. Most of the work presented in this chapter was published in (Vlaeminck, Vennekens, & Denecker, 2012b).
- Chapter 7 studies the logic OEL and its distributed variant d-OEL. Most of this work was published in (Vlaeminck, Vennekens, Bruynooghe, & Denecker, 2012a).
- Finally, we summarize our results and explore possibilities for future work in Chapter 8.

2 Preliminaries

In this chapter we introduce the technical background and some of the notations used throughout this thesis. We review first-order logic (FO), also called *classical logic*, and a number of useful extensions of this logic. In order to define the semantics of some of these extensions, we need some concepts from approximation fixpoint theory. We finish this chapter with a short summary of autoepistemic logic (AEL).

2.1 First-order logic

In order to have a *formal language* we first need a set of words or a set of symbols of some sort (a *vocabulary*). But symbols on their own are not enough. We also need a *syntax*, that is, a specification of how to build sentences of the language, and a *semantics*. The semantics of a formal language expresses in a mathematically precise way what the sentences of the language mean. In this section we review syntax and semantics of first-order logic (from here on denoted as FO). For a full exposition, see, e.g., (Enderton, 2001).

Syntax

In first-order logic, there are two types of symbols: *logical* and *non-logical* symbols. Intuitively, the logical symbols are the ones that have a fixed meaning in the language.

1. Logical symbols:
   (a) *punctuation symbols*: '(', ')' and ',';
   (b) *logical connectives*: '¬', '∧', '∨' and '='. These connectives are interpreted as follows: ¬ is the negation ("not"), ∧ is the conjunction ("and"), ∨ is the disjunction ("or") and = is equality;
   (c) *quantifiers*: '∃' ("there exists") and '∀' ("for all");
   (d) an infinite supply of variable symbols, which we will often denote using x, y and z.
2. Non-logical symbols:
   (a) countably many *function* symbols: $F_1, F_2, \ldots$, each with an associated arity $n \geq 0$;
   (b) countably many *predicate* symbols: $P_1, P_2, \ldots$, each with an associated arity $n \geq 0$.

We will denote a symbol S with arity n by S/n. 0-ary function symbols are called *constants*.

Definition 2.1. We define a *vocabulary* Σ as any countable set of non-logical symbols, that is, a countable set of predicate symbols Σ^P and a countable set of function symbols Σ^F.

We denote that a vocabulary Σ' is a *subvocabulary* of Σ by $\Sigma' \subseteq \Sigma$. There are two types of legal syntactic expressions in FO: *terms* and *formulas*. Intuitively, terms will refer to objects in the world, while formulas express propositions about the world.

Definition 2.2. Given a vocabulary Σ, a *term* over Σ is inductively defined by

1. each variable is a term;
2. if $t_1, \ldots, t_n$ are n terms, and F is a function symbol of arity n, then $F(t_1, \ldots, t_n)$ is a term.

The set of *first-order formulas* over Σ is the least set of expressions closed under the following rules:

1. if $t_1, \ldots, t_n$ are n terms and P is a predicate symbol of arity n, then $P(t_1, \ldots, t_n)$ is a formula;
2. if t_1 and t_2 are terms, then $t_1 = t_2$ is a formula;
3. if φ and ψ are formulas and x is a variable, then $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\exists x\, \varphi$ and $\forall x\, \varphi$ are formulas.

Following standard practice, to increase readability we will often add or omit parentheses, and we assume that '$\neg$' binds stronger than '$\wedge$' and '$\vee$' which in turn bind stronger than '$\forall$' and '$\exists$'. We use $(\varphi \Rightarrow \psi)$, $(\varphi \Leftrightarrow \psi)$ and $(t_1 \neq t_2)$ as shorthands for $(\neg\varphi \vee \psi)$, $((\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi))$ and $(\neg(t_1 = t_2))$, respectively. We use $\bar{t}$ to denote tuples of terms. If $\bar{x}$ denotes the the tuple of variables $(x_1, \ldots, x_n)$, then $(\forall \bar{x}\, \varphi)$ and $(\exists \bar{x}\, \varphi)$ denote the formulas $(\forall x_1 \ldots \forall x_n\, \varphi)$ and $(\exists x_1 \ldots \exists x_n\, \varphi)$ respectively.

An *atomic formula*, also called an *atom*, is a formula of the form $P(\bar{t})$ or $t_1 = t_2$. A *literal* is an atom $P(\bar{t})$ (a *positive* literal) or its negation $\neg P(\bar{t})$ (a *negative* literal). In a formula of the form $(\forall x\, \varphi)$ or $(\exists x\, \varphi)$, the subformula φ is called the *scope* of '$\forall x$', respectively '$\exists x$'. Every occurrence of the variable x in φ is said to be *bound*. We say that x is bound by the innermost occurrence of a quantifier $\exists x$ or $\forall x$ in which scope it occurs. A variable that is not bound by any quantifier is called a *free* variable. We will use the notation $\varphi[\bar{x}]$ to denote that the free variables of φ are a subset of $\bar{x}$. A *sentence* is a formula without free variables. A formula or term that contains *no variables* is called *ground*. We call a set of FO formulas T over a vocabulary Σ a *theory* if every formula $\varphi \in T$ is a sentence. We say that a formula φ is in *negation normal form* if φ contains no implications or equivalences, and all negations occur directly in front of atoms.

Semantics

As explained before, the purpose of a semantics for a formal language is to express in a mathematically precise way what the sentences of the language *mean*. Before being able to give meaning to complex FO formulas, we first need to have an interpretation for the basic predicate and function symbols of the vocabulary. This can be made mathematically precise in the following way.

Definition 2.3. Given a vocabulary Σ, a Σ-*interpretation* (also called a Σ-*structure*) I consists of

- a non-empty set D^I (sometimes also denoted as $dom(I)$), called the *domain* of I. Its elements are called *domain elements*;
- for every predicate symbol $P/n \in \Sigma^P$, a relation $P^I \subseteq (D^I)^n$;
- for every function symbol $F/n \in \Sigma^F$, a function $F^I : (D^I)^n \to D^I$.

We call I a *finite interpretation* if the domain D^I is finite. A *pre-interpretation* of Σ consists of a domain and an interpretation of the function symbols. If I is a Σ-interpretation and $\Sigma' \subseteq \Sigma$, we denote by $I|_{\Sigma'}$ the restriction of I to the symbols of Σ'. Vice versa, I is called an *expansion* of $I|_{\Sigma'}$ to Σ. If Σ_1 and Σ_2 are two disjoint vocabularies, I a Σ_1-interpretation with domain D^I and J a Σ_2-interpretation with the same domain, then $I + J$ denotes the unique $(\Sigma_1 \cup \Sigma_2)$-interpretation with domain D^I such that $(I + J)|_{\Sigma_1} = I$ and $(I + J)|_{\Sigma_2} = J$.

A Σ-interpretation I already tells us how to interpret any variable-free term over Σ. To deal with terms including variables, we need a *variable assignment* over D^I, that is, a mapping θ from variables to the elements of D^I. If x is a variable and $d \in D^I$, then we denote by $\theta[x/d]$ the variable assignment that assigns d to x and corresponds to θ on all other variables. This notation is extended to tuples of variables and tuples of domain elements of the same length. The *value of a term t in a Σ-interpretation I under the variable assignment θ* is denoted by $I\theta(t)$ and inductively defined by

- $I\theta(t) = \theta(t)$ if t is a variable;
- $I\theta(F(t_1, \ldots, t_n)) = F^I(I\theta(t_1), \ldots, I\theta(t_n))$.

Note that according to these rules, $I\theta(t)$ is always an element of D^I.

Once we have an interpretation I and a variable assignment θ, we specify which FO sentences are true and which are false. Formally, the satisfaction relation is defined as follows.

Definition 2.4. Given a Σ-interpretation I, a variable assignment θ and a formula φ, the *satisfaction relation* $I\theta \models \varphi$ is inductively defined through the following rules.

- $I\theta \models P(\bar{t})$ if $(I\theta(\bar{t})) \in P^I$;
- $I\theta \models (t_1 = t_2)$ if $I\theta(t_1) = I\theta(t_2)$;
- $I\theta \models \neg\varphi$ if $I\theta \not\models \varphi$;
- $I\theta \models \varphi \wedge \psi$ if $I\theta \models \varphi$ and $I\theta \models \psi$;
- $I\theta \models \varphi \vee \psi$ if $I\theta \models \varphi$ or $I\theta \models \psi$;
- $I\theta \models \forall x\, \varphi$ if $I\theta[x/d] \models \varphi$ for every $d \in D^I$;
- $I\theta \models \exists x\, \varphi$ if there exists a $d \in D^I$ such that $I\theta[x/d] \models \varphi$.

A formula φ is *satisfiable* if there exists an interpretation I and variable assignment θ such that $I\theta \models \varphi$.

Property 2.5. *Let I be a Σ-interpretation, θ a variable assignment, and φ be a Σ-formula with free variables $\bar{x}$ such that $I\theta \models \varphi[\bar{x}]$. Then it holds for every variable assignment θ' for which $\theta(x) = \theta'(x)$ for all $x \in \bar{x}$, that $I\theta' \models \varphi[\bar{x}]$.*

A consequence of this property is that if for a domain tuple $\bar{d}$ it holds that $I\theta[\bar{x}/\bar{d}] \models \varphi[\bar{x}]$, this does not depend on θ, but only on $\bar{d}$, since it holds for *any* variable assignment θ' that $I\theta'[\bar{x}/\bar{d}] \models \varphi[\bar{x}]$. Therefore, to make notation somewhat lighter, we often use $I \models \varphi[\bar{d}]$ as a shorthand for $I\theta[\bar{x}/\bar{d}] \models \varphi[\bar{x}]$. When φ contains no free variables, we write $I \models \varphi$, and say

that φ is *true in* I or *false in* I otherwise. Given a theory T, we will use the notation $I \vDash T$ to denote that all of the sentences of T are true in I. We will call an interpretation that satisfies T a *model* of T. If T_1 and T_2 are two theories, we use $T_1 \vDash T_2$ to say that every model of T_1 is also a model of T_2. If $T_1 \vDash T_2$ and $T_2 \vDash T_1$, we say that T_1 and T_2 are equivalent theories.

Given an interpretation I, a formula φ with n free variables, and a tuple of domain elements $\bar{d} \in (D^I)^n$, we define the truth evaluation function $(\varphi[\bar{d}])^I$ as follows: $(\varphi[\bar{d}])^I = \mathbf{t}$ iff $I \vDash \varphi[\bar{d}]$ and $(\varphi[\bar{d}])^I = \mathbf{f}$ otherwise. We define the inverse on truth values as follows: $(\mathbf{f})^{-1} = \mathbf{t}$ and $(\mathbf{t})^{-1} = \mathbf{f}$. We also define the following strict order on the truth values: $\mathbf{f} <_t \mathbf{t}$. As usual, we denote the corresponding non-strict order by $\leq_t$. This *truth order* point-wise extends to interpretations *extending the same pre-interpretation*: if I and J are two Σ-interpretations extending the same pre-interpretation, then we say that $I \leq_t J$ if for every predicate symbol P and tuple of domain elements $\bar{d}$ it holds that $P^I(\bar{d}) \leq_t P^J(\bar{d})$.

Often in this thesis, we will assume that vocabularies do not contain function symbols. This assumption can be made without loss of generality, since any theory T can be transformed in an equivalent function-free theory by replacing each function symbol by a new predicate symbol and adding some extra constraints. Formally, let $\mathcal{G}(\Sigma)$ be the vocabulary obtained from a vocabulary Σ by replacing every function symbol F/n by a new predicate $P_F/n+1$. If I is a Σ-interpretation, then I can be accordingly transformed into a $\mathcal{G}(\Sigma)$-interpretation $\mathcal{G}(I)$ that assigns $(P_F)^{\mathcal{G}(I)} = \{(\bar{d}, d') | F^I(\bar{d}) = d'\}$, for every $F \in \Sigma^F$ and $P^{\mathcal{G}(I)} = P^I$ for every $P \in \Sigma^P$. Using these new predicates, we can now transform a Σ–theory T into a function free $\mathcal{G}(\Sigma)$-theory T'. Now all that is left, is to make sure that in every model J of T' the predicates P_F indeed represent functions, that is, that for every predicate $P_F \in \mathcal{G}(\Sigma)$ and domain tuple $\bar{d} \in D^n$ there is exactly one $d' \in D$ such that $(\bar{d}, d') \in (P_F)^J$. This can be done by adding the following sentences to T' for every $F \in \Sigma^F$.

$$\forall \bar{x} \exists y \, P_F(\bar{x}, y).$$
$$\forall \bar{x} \forall y_1 \forall y_2 \, (P_F(\bar{x}, y_1) \wedge P_F(\bar{x}, y_2)) \Rightarrow y_1 = y_2.$$

Approximating interpretations

Similar to how a real number r can be approximated by an interval $[l, u]$ such that $l \leq r \leq u$, it is possible to approximate a Σ-interpretation K by a pair (I, J) of Σ-interpretations extending the same pre-interpretation, such that $I \leq_t K \leq_t J$. We denote by $[I, J]$ the interval of all such interpretations K. This interval is empty if and only if $I \not\leq_t J$. It follows easily from well-known monotonicity results, that if we evaluate all positive occurrences (i.e., in the scope of an even number of negations) of atoms in some formula φ by I, and all negative occurrences (i.e., in the scope of an odd number of negations) by J, we are underestimating the truth of φ in the interval $[I, J]$. Conversely, if we evaluate positive occurrences in J and negative occurrences in I, we are overestimating the truth of φ in $[I, J]$. To state this property more formally, we introduce the following notation.

Definition 2.6 (Pos-neg evaluation relation φ^{+I-J})**.** Let φ be a Σ-formula and let I and J be Σ-interpretations extending the same pre-interpretation. We define the *pos-neg evaluation* of φ in I and J, denoted by φ^{+I-J}, by induction over the size of φ:

- for an atom $\varphi = P(\bar{t})$, $\varphi^{+I-J} = \varphi^I$;
- for $\varphi = \neg\psi$, $\varphi^{+I-J} = (\psi^{+J-I})^{-1}$;
- for $\varphi = \psi_1 \wedge \psi_2$, $\varphi^{+I-J} = \mathbf{t}$ iff $\psi_1^{+I-J} = \mathbf{t}$ and $\psi_2^{+I-J} = \mathbf{t}$;
- for $\varphi = \exists x\ \psi$, $\varphi^{+I-J} = \mathbf{t}$ iff there is a $d \in D$ such that $\psi[\bar{d}]^{+I-J} = \mathbf{t}$.

We now indeed have that, for each $K \in [I, J]$, $\varphi^{+I-J} \leq \varphi^K \leq \varphi^{+J-I}$. Also, we have that $\varphi^K = \varphi^{+K-K}$.

There is an intimate connection between the approximation of an interpretation by a pair of interpretations and Belnap's four-valued logic (Belnap, 1977). We denote the truth values *true, false, unknown* and *inconsistent* of four-valued logic by respectively $\mathbf{t}, \mathbf{f}, \mathbf{u}$ and $\mathbf{i}$. On these truth values, the *truth order* $\leq_t$ and *precision order* $\leq_p$ are defined as shown in Figure 2.1.

A four-valued Σ-interpretation $\mathcal{I}$ consists of a pre-interpretation and for every $P/n \in \Sigma^P$ a function $P^{\mathcal{I}} : D^n \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}, \mathbf{i}\}$ [1]. Again, the precision order point-wise extends to interpretations: if $\mathcal{I}$ and $\mathcal{J}$ are two Σ-interpretations extending the same pre-interpretation, then we say that $\mathcal{I} \leq_p \mathcal{J}$ if for every predicate symbol P and tuple of domain elements $\bar{d}$ it holds that $P^{\mathcal{I}}(\bar{d}) \leq_p P^{\mathcal{J}}(\bar{d})$. Similarly, also the truth order is extended to interpretations.

There is a natural isomorphism between Belnap's four truth values and pairs of the two standard truth values:

$$\begin{aligned} \tau(\mathbf{t}, \mathbf{t}) &= \mathbf{t}; \\ \tau(\mathbf{f}, \mathbf{t}) &= \mathbf{u}; \\ \tau(\mathbf{t}, \mathbf{f}) &= \mathbf{i}; \\ \tau(\mathbf{f}, \mathbf{f}) &= \mathbf{f}. \end{aligned}$$

Intuitively, this mapping τ interprets its first argument as an underestimate to the "real" truth value, and its second argument as an overestimate: if the underestimate is $\mathbf{f}$ and the overestimate is $\mathbf{t}$, then the real truth value is indeed unknown; whereas, if the underestimate is $\mathbf{t}$ and the overestimate is $\mathbf{f}$, then there cannot exist a real truth value, since $\mathbf{t} \not\leq_t \mathbf{f}$, so we end up with inconsistency. This isomorphism τ extends in an obvious way to an isomorphism between pairs (I, J) of two-valued interpretations and four-valued interpretations $\mathcal{I}$ which

[1] It is possible to give a more general definition of four-valued interpretations, that is, a definition where interpretations of functions are not necessarily two-valued. However, since, defining the truth value of formulas in such general four-valued interpretations is not without its problems (e.g., formulas are required to be in a certain normal form), we have opted here for a definition where functions are interpreted two-valued.

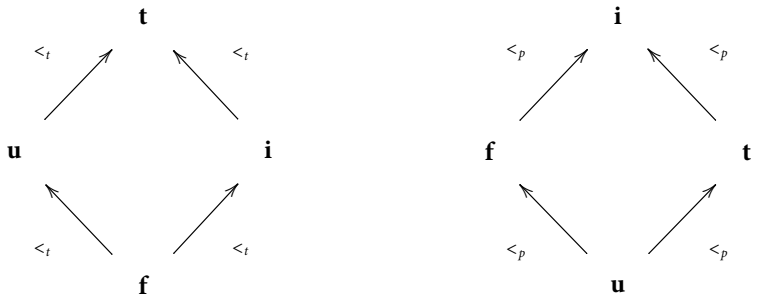

Figure 2.1: The truth and precision order

all share the same pre-interpretations: (I, J) and $\mathcal{I}$ are isomorphic iff, for each predicate P/n and tuple $\bar{d} \in D^n$, $P^{\mathcal{I}}(\bar{d}) = \tau(P^I(\bar{d}), P^J(\bar{d}))$. We also denote this isomorphism by τ.

There is a tight link between the pos-neg evaluation function φ^{+I-J} and the Belnap evaluation $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} = \tau(\varphi^{+I-J}, \varphi^{+J-I}), \text{ where } \tau(I, J) = \mathcal{I}.$$

When $\mathcal{I}$ is a three-valued structure (i.e., it never assigns **i**) this corresponds to the standard Kleene evaluation (Kleene, 1952). In the rest of this thesis, we will often omit the isomorphism τ, and, e.g., simply denote the four-valued truth value of a formule φ in a pair of interpretations (I, J) as $\varphi^{(I,J)}$. An important property, that we already stated above in different notation, is that $\varphi^{(I,J)} \leq_p \varphi^K$ for all $K \in [I, J]$.

There is a natural and well-known alternative way of using an interval $[I, J]$ for which $I \leq_t J$ to assign a truth value to a formula φ: the *supervaluation* (van Fraassen, 1966).

Definition 2.7 (Supervaluation $sv_{(I,J)}(.)$)**.** The supervaluation $sv_{(I,J)}(\varphi)$ of a sentence φ in a pair of interpretations (I, J) (or equivalently, a three-valued interpretation $\tau(I, J)$) is defined as

$$sv_{(I,J)}(\varphi) = glb_{\leq_p}(\{\varphi^K | K \in [I, J]\}).$$

It is easy to see that always $sv_{(I,J)}(\varphi) \geq_p \varphi^{(I,J)}$. This inequality may be strict. For instance, if we take $\varphi = Q \vee \neg Q$ and interpretations I and J such that $Q^{(I,J)} = \mathbf{u}$, then $sv_{(I,J)}(\varphi) = \mathbf{t}$, but $\varphi^{(I,J)} = \mathbf{u}$.

A property that we will often use in this thesis is that we can simulate the four-valued truth evaluation in pairs of interpretations by encoding what is certainly true and certainly false,

using a single two-valued structure $\mathcal{I}^{tf}$ over a new vocabulary Σ^{tf}. The new vocabulary Σ^{tf} contains the function symbols Σ^F of Σ and, for each predicate $P \in \Sigma^P$, two symbols P^{ct} and P^{cf}. The interpretations of P^{ct} and P^{cf} in $\mathcal{I}^{tf}$ contain, respectively, those tuples $\bar{d}$ for which $P^{\mathcal{I}}(\bar{d})$ is certainly true and those for which it certainly false. Formally, for a vocabulary Σ and a four-valued Σ-interpretation $\mathcal{I} = \tau(I, J)$, the Σ^{tf}-interpretation $\mathcal{I}^{tf}$ has the same pre-interpretation as $\mathcal{I}$, and is defined by:

$$(P^{ct})^{\mathcal{I}^{tf}} = \{\bar{d} | P^{\mathcal{I}}(\bar{d}) \geq_p \mathbf{t}\} = P^I,$$

$$(P^{cf})^{\mathcal{I}^{tf}} = \{\bar{d} | P^{\mathcal{I}}(\bar{d}) \geq_p \mathbf{f}\} = D^n \setminus P^J.$$

An interpretation $\mathcal{I}$ is three-valued iff $(P^{ct})^{\mathcal{I}^{tf}}$ and $(P^{cf})^{\mathcal{I}^{tf}}$ are disjoint for any $P \in \Sigma$. $\mathcal{I}$ is two-valued iff $(P^{ct})^{\mathcal{I}^{tf}}$ and $(P^{cf})^{\mathcal{I}^{tf}}$ are each others complement in D^n. Also, if $\mathcal{I} \leq_p \mathcal{J}$, then, for each P, $(P^{ct})^{\mathcal{I}^{tf}} \subseteq (P^{ct})^{\mathcal{J}^{tf}}$ and $(P^{cf})^{\mathcal{I}^{tf}} \subseteq (P^{cf})^{\mathcal{J}^{tf}}$.

Definition 2.8 (φ^{ct} and φ^{cf})**.** For any given Σ-formula $\varphi[\bar{x}]$, let $\varphi^{ct}[\bar{x}]$ be the Σ^{tf}-formula obtained by replacing all positive occurrences of atoms $P(\bar{t})$ by $P^{ct}(\bar{t})$ and all negative occurrences by $\neg P^{cf}(\bar{t})$ and reducing to negation normal form, and let $\varphi^{cf}[\bar{x}]$ be the formula $(\neg\varphi[\bar{x}])^{ct}$.

An interesting property of the formulas φ^{ct} and φ^{cf} is that they do not contain negations. Also, the following proposition is well-known.

Proposition 2.9. *[(Feferman, 1984)] For every Σ-formula φ and interpretation $\mathcal{I}$, it holds that $\varphi[\bar{d}]^{\mathcal{I}} \geq_p \mathbf{t}$ if and only if $\varphi[\bar{d}]^{+I-J} = \mathbf{t}$ if and only if $(\varphi^{ct}[\bar{d}])^{\mathcal{I}^{tf}} = \mathbf{t}$. Also, $\varphi[\bar{d}]^{\mathcal{I}} \geq_p \mathbf{f}$ if and only if $\varphi[\bar{d}]^{+J-I} = \mathbf{f}$ if and only if $(\varphi^{cf}[\bar{d}])^{\mathcal{I}^{tf}} = \mathbf{t}$, where I and J are the two Σ^{tf}-interpretations such that $\mathcal{I} = \tau(I, J)$.*

Sorted and propositional logics

It is straightforward to define a *many-sorted* variant of FO. Intuitively, sorts (hereafter also called *types*) denote *classes of objects*. It is often more convenient to model some domain knowledge using many-sorted FO, than in the standard one-sorted FO we described in the previous section. The main difference between many-sorted FO and standard FO is that a vocabulary in many-sorted FO now also contains a set Σ^S of sorts. Every predicate now has, apart from its arity n, also an associated tuple $(S_1, \ldots, S_n)$ of sorts from Σ^S, and likewise for functions. The definition of an interpretation can be easily adapted to account for these sorts, for example, instead of one domain D^I , an interpretation for many-sorted FO contains a domain S^I for every sort $S \in \Sigma^S$.

Throughout this thesis we often implicitly use many-sorted logic in our examples, while in the definitions and theorems we use standard FO. We can safely do this, since Oberschelp

(1962) showed that many-sorted FO can be reduced to one-sorted FO by introducing a predicate of arity one for each sort of a many-sorted vocabulary.

Propositional logic is an important fragment of FO. In a propositional vocabulary all predicate symbols have arity 0 and there are no function symbols. A *propositional clause* is a disjunction $L_1 \vee \ldots \vee L_n$, where all L_i with $1 \leq i \leq n$ are propositional literals. A theory is in *conjunctive normal form* (CNF) if all its sentences are clauses. The *boolean satisfiability problem* (SAT) is the problem of deciding for a propositional theory whether it has a model. This problem is **NP**-complete (Cook, 1971).

Herbrand interpretations

In logic programming, one often looks only at a particular class of interpretations, namely, the class of *Herbrand interpretations*. Herbrand interpretations are interpretations that have a specific domain, called the *Herbrand universe*, which consists of all the ground terms that can be constructed using the constant and function symbols in the vocabulary. A *Herbrand pre-interpretation* is a pre-interpretation that has the Herbrand universe as its domain and interprets each ground term (that is, each term that can be constructed using constant and function symbol in the vocabulary) *by itself*. A Herbrand interpretation is then an interpretation that extends some Herbrand pre-interpretation.

Given a set of constant and/or function symbols τ with the same type S, the *unique name axiom* for τ (Reiter, 1980a) is the informal proposition that different ground terms of τ represent different objects. In general the unique name axiom can be expressed by the following set of formulas, which we will denote by $UNA(\tau)$:

$$\forall \bar{x}\bar{y}(F(\bar{x}) \neq G(\bar{y})),$$
$$\forall \bar{x}\bar{y}(F(\bar{x}) = F(\bar{y}) \Rightarrow \bar{x} = \bar{y}),$$

for all distinct function and constant symbols $F, G \in \tau$.

Given a set of constant and/or function symbols τ with the same type S, the (typed) *domain closure axiom* (Reiter, 1982) for τ is the (informal) proposition that every object of type S is represented by a ground term of τ. In general the domain closure axiom cannot be expressed in FO. If τ contains only constants $C_1, \ldots, C_n$, the domain closure axiom for τ is expressible in FO by the sentence $DCA(\tau)$:

$$\forall x\,(x = C_1 \vee \ldots \vee x = C_n).$$

In Section 2.3, we will show how to express the domain closure axiom in general, using *inductive definitions*.

For a type S and τ the set of all function and constant symbols of this type, we write $UNA(S)$ (resp. $DCA(S)$) to denote $UNA(\tau)$ (resp. $DCA(\tau)$). The Herbrand models of a theory are

isomorphic to models of the extension of that theory with the *unique names axioms* (UNA) and the *domain closure axioms* (DCA) for the Herbrand universe.

Second-order logic

Throughout this text, we sometimes refer to *second-order logic* (SO). Second-order logic is simply the language of first-order logic augmented with *second order variables*, that is, variables ranging over relations and functions of all arities. This quantification over relations and/or functions greatly enhances the expressive power of second-order logic compared to first-order logic.

Given a (first-order) vocabulary Σ, Σ-terms and Σ-formulas are defined in second-order logicas as in first-order logic, but using also function-variables and relation-variables in complete analogy to the variables of FO. While we refer to (Barwise, 1977) for a precise definition of the semantics, the standard semantics of second-order Σ-formulas is defined with respect to standard first-order Σ-interpretations. The truth of a formula in such an interpretation is straightforward: function-variables of arity n range over functions of n arguments over the domain of the interpretation, and relation-variables of arity n over n-ary relations. Let ψ be a SO formula over Σ. The *finite domain satisfiability problem* for ψ, is the problem of deciding whether there exists a Σ-interpretation I with a *finite domain*, such that $I \models \psi$.

In this dissertation we are interested in some specific *fragments* of SO. The first such fragment is *existential second-order logic*, which we denote by $\exists SO$. All Σ-formulas in this fragment are of the form $\exists P_1 \dots P_n \ \psi$, where ψ is a regular FO formula over $\Sigma \cup \{P_1, \dots, P_n\}$. The *universal second-order logic* fragment ($\forall SO$) is defined analogous but such that formulas only have universal quantification over predicate or function symbols. The $\forall SO$ fragment has an interesting connection with the *supervaluation* (Definition 2.7). Let I be an interpretation for the vocabulary Σ of a $\forall SO$ formula $\varphi = \forall \bar{Q} \ \psi$, and let (J_1, J_2) be the least precise pair of interpretations for $\bar{Q}$ in the domain D of I (i.e., $Q^{(J_1,J_2)}(\bar{d}) = \mathbf{u}$ for all $Q/n \in \bar{Q}$ and $\bar{d} \in D^n$). We then have that $sv_{(I+J_1, I+J_2)}(\psi) = \mathbf{t}$ if and only if $I \models \varphi = \mathbf{t}$. The last fragment that we will refer to in this text is $\exists\forall SO$. In this fragment, every Σ-formula is of the form $\exists P_1 \dots P_n \ \forall Q_1 \dots Q_m \ \psi$, where ψ is again an FO-formula over $\Sigma \cup \{P_1, \dots, P_n, Q_1, \dots, Q_m\}$.

It is a famous result by Fagin (1974) that every language in NP has an existential second-order characterization. This result was generalized to the polynomial-time hierarchy by Stockmeyer (1976).

2.2 Approximation theory

In order to define the semantics of some of the extensions of FO that we introduce in the next sections, we need some concepts of approximation theory (Denecker et al., 2004). Approximation fixpoint theory is an algebraic fixpoint theory for arbitrary (non-monotone) operators that generalizes all main semantics of a number of non-monotonic logics. First, we recall some of the well known definitions and results from lattice theory.

2.2.1 Sets, functions, orders, lattices and operators

A binary relation θ on a set S is *well-founded* if it has no infinite descending chains, that is, if there exists no infinite sequences $(x_i)_{i\in\mathbb{N}}$ such that, for all $i \in \mathbb{N}$, $x_i \theta x_{i-1}$ and $x_i \neq x_{i-1}$. A binary relation $\leq$ on a set S is called a *partial order* if it is reflexive, transitive and anti-symmetric. If $\leq$ is a partial order, then the pair $\langle S, \leq\rangle$ is called a partially order set, or in short *poset*. A poset for which $\leq$ is well-founded is called a *well-founded set*.

For a subset R of a poset S, an element $m \in R$ is *minimal* in R if for every $m' \in R$, if $m' \leq m$ then $m \leq m'$. It can be shown that a poset S is well-founded if and only if every non-empty subset of S has a minimal element. A partial order on S is a *total order* if every two elements $x, y \in S$ are comparable, that is, $x \leq y$ or $x \geq y$. For each subset R of S, an element $l \in S$ is called a *lower bound* of R if for all $r \in R$ it holds that $l \leq r$. An element $g \in S$ such that g is a lower bound of R and for each other lower bound l of R it holds that $l \leq g$, is called the *greatest lower bound* or *infimum* of R, denoted $glb(R)$. Similarly, an element $u \in S$ is called an *upper bound* of R if for all $r \in R$, it holds that $r \leq u$, and if u is less or equal then each other upper bound of R, u is called the *least upper bound*, or *supremum* of R, denoted $lub(R)$.

A pair $\langle L, \leq\rangle$ is called a *lattice* if $\leq$ is a partial order on the non-empty set L and if every finite subset S of L has a least upper bound and a greatest lower bound. A lattice $\langle L, \leq\rangle$ is *complete* if each (not necessary finite) subset L' of L has a greatest lower bound and least upper bound. Consequently, a complete lattice has a *least element* ($\bot$) and a maximal element ($\top$).

An *operator* is a function $O : L \to L$ from a lattice L to itself. An element $x \in L$ is a *pre-fixpoint* of O if $x \geq O(x)$, a *fixpoint* if $x = O(x)$ and a *post-fixpoint* if $x \leq O(x)$. An operator O is *monotone* if for each $x, y \in L$ such that $x \leq y$, it holds that $O(x) \leq O(y)$. The following famous theorem was originally obtained by Tarski (1955), and is one of the most fundamental results used in studies of semantics of logics and programming languages.

Theorem 2.10 (Knarster-Tarski)**.** *Every monotone operator O on a complete lattice $\langle L, \leq\rangle$ has a complete lattice of fixpoints and hence a least fixpoint (*$\mathrm{lfp}(O)$*) and greatest fixpoint (*$\mathrm{gfp}(O)$*). Moreover, the least fixpoint is the least pre-fixpoint of O and the greatest fixpoint is the greatest post-fixpoint of O.*

A consequence of this theorem is that the least upper bound of the sequence $(O^n(\bot))_{n\in\mathbb{N}}$ is the least fixpoint of O, and the greatest lower bound of the sequence $(O^n(\top))_{n\in\mathbb{N}}$ is the greatest fixpoint of O.

An operator O is *anti-monotone* if $x \leq y$ implies $O(x) \geq O(y)$. An *oscillating pair* of an operator O is a pair (x, y) such that $O(x) = y$ and $O(y) = x$. An anti-monotone operator O on a complete lattice has a *maximal oscillating pair* (x, y), that is, for any oscillating pair (x', y'), it holds that $x \leq x'$ and $y' \leq y$.

2.2.2 Approximation fixpoint theory

Approximation fixpoint theory is a general fixpoint theory for arbitrary operators, which generalizes ideas found in, amongst others, (Baral & Subrahmanian, 1991) and (Fitting, 2002). In this section we present a short overview of approximation theory. We refer to (Denecker et al., 2004), (Denecker et al., 2000) and (Vennekens, 2007) for more details.

Let $\langle L, \leq\rangle$ be a complete lattice. An element (x, y) of the square L^2 can be seen as denoting a (possibly empty) interval $[x, y] = \{z \in L | x \leq z \leq y\}$. This interval is non-empty iff $x \leq y$. Such pairs are called *consistent*. Intervals corresponding to pairs (x, x) contain precisely one element, namely x itself. Such pairs are called *exact*. The order $\leq$ induces two natural orders on L^2:

- the *product* order $\leq$ is the point-wise extension of $\leq$, that is, for all $x, x', y, y' \in L$, $(x, y) \leq (x', y')$ iff $x \leq x'$ and $y \leq y'$;
- the *precision* order $\leq_p$ is defined as follows: for all $x, x', y, y' \in L$, $(x, y) \leq_p (x', y')$ iff $x \leq x'$ and $y \geq y'$.

It can easily be shown that both $\langle L^2, \leq\rangle$ and $\langle L^2, \leq_p\rangle$ are complete lattices. The lattice L^2 together with its two orders is called the *bilattice* corresponding to L.

Operators on bilattices L^2 that are monotone w.r.t. the precision order $\leq_p$ are called *approximations*. For an approximation A and elements $x, y \in L$, we denote by $A_1(x, y)$ and $A_2(x, y)$ the unique elements of L for which $A(x, y) = (A_1(x, y), A_2(x, y))$. An approximation *approximates* an operator O on L if for each $x \in L$, $A(x, x)$ contains $O(x)$, that is, $A_1(x, x) \leq O(x) \leq A_2(x, x)$. An approximation is *symmetric* if for each pair $(x, y) \in L^2$, if $A(x, y) = A(x', y')$, then $A(y, x) = (y', x')$.

For an approximation A on L^2, the following two classes of operators on L can be defined:

- for each $y \in L$, the operator $A_1(\cdot, y)$ maps an element $x \in L$ to $A^1(x, y)$, that is, $A_1(\cdot, y) = \lambda x.A_1(x, y)$;

- for each $x \in L$, the operator $A_2(x, \cdot)$ maps an element $y \in L$ to $A^2(x, y)$, that is, $A_2(x, \cdot) = \lambda y.A_2(x, y)$.

For every lattice L, all of these operators are *monotone* and thus have a least fixpoint.

Definition 2.11. Let L be a complete lattice and A an approximation on L^2. The operators $C_A^{\downarrow}$ and $C_A^{\uparrow}$ on L are defined as follows. The operator $C_A^{\downarrow}$ maps each $y \in L$ to $\mathrm{lfp}(A_1(\cdot, y))$ and $C_A^{\uparrow}$ maps each $x \in L$ to $\mathrm{lfp}(A_2(x, \cdot))$. The operator $C_A^{\downarrow}$ is called the *lower stable operator* of A, and $C_A^{\uparrow}$ the *upper stable operator* of A. Note that if an approximation A is symmetric, its upper and lower stable operators are equal, that is, $C_A^{\downarrow} = C_A^{\uparrow}$.

Both these operators are anti-monotone. We can combine these two operators, to form another, monotone operator.

Definition 2.12. The *partial stable operator* $\mathcal{C}_A$ on L^2 is defined as the operator that maps each pair (x, y) to $(C_A^{\downarrow}(y), C_A^{\uparrow}(x))$.

An approximation A defines a number of different fixpoints:

- the least fixpoint of an approximation A is called its *Kripke-Kleene fixpoint*;
- fixpoints of its partial stable operator $\mathcal{C}_A$ are *partial stable fixpoints*;
- the least fixpoint of the partial stable operator $\mathcal{C}_A$ is called the *well-founded fixpoint* of A.

As shown in (Denecker et al., 2004) and (Denecker et al., 2000), these fixpoints correspond to various semantics of logic programming, autoepistemic logic and default logic.

2.3 FO(·)

Although FO is an expressive logic on its own, it is well-known that there are several concepts that cannot be expressed in it. An example of such a concept is the transitive closure of a relation. Other concepts, such as certain aggregates, arithmetic, can be expressed *in principle*, but not in an easy, concise or natural way. Yet, many of these concepts are often very useful in practice. For example, in the context of databases, query languages have been extended with aggregates, fixpoint constructs to represent transitive closure, etc. The lack of expressive power of FO to represent these concepts has motivated the definition of the logic FO(·), an extension of FO with, among others, inductive definitions, aggregates and arithmetic. In this section we recall the most important of these extensions. For a detailed description of all extensions we refer to (Wittocx, 2010).

2.3.1 Inductive definitions

Informally, an inductive definition is a 'recipe' to construct a mathematical object, usually a set, by describing when to add elements given the presence or absence of other elements. In mathematical texts, inductive definitions are usually represented as collections of rules representing a set of base cases and inductive cases. Inductive rules may be *monotone* or *non-monotone*. As an example, let us look at the following definition of the satisfaction relation for propositional logic.

- $I \models P$ if $P^I = \mathbf{t}$;
- $I \models \varphi \wedge \psi$ if $I \models \varphi$ and $I \models \psi$;
- $I \models \varphi \vee \psi$ if $I \models \varphi$ or $I \models \psi$;
- $I \models \neg\varphi$ if $I \not\models \varphi$.

The first three rules are monotone but the last one is non-monotone. Indeed, it adds $(I, \neg\varphi)$ to the satisfaction relation if (I, φ) does not belong to it.

While inductive definitions can be found in many mathematical texts, they also occur in common sense knowledge. An example thereof is the definition of *reachability* in a graph:

- a node v_2 is reachable from v_1 if there is an edge between v_1 and v_2;
- a node v_2 is reachable from v_1 if there is some intermediate node v such that d is reachable from v_1 and v_2 is reachable from d.

While this definition is *monotone*, it is well-known that the concept of reachability is not expressible in FO (Libkin, 2004). The logic FO(ID) (Denecker & Ternovska, 2008) is an the extension of FO with a construct to respresent some of the most common forms of inductive definitions, such as monotone induction, induction over a well-founded order and iterated induction.

Syntax

A *definition* Δ over Σ is a finite set of rules of the form

$$\forall \bar{x}(P(\bar{t}) \leftarrow \varphi[\bar{x}]) \tag{2.1}$$

where $P(\bar{t})$ is an atomic formula over Σ, φ an FO Σ-formula, and $\bar{t}$ a tuple of terms with free variables among $\bar{x}$. The symbol $\leftarrow$ is a new connective, called the *definitional implication*, to be distinguished from the FO material implication symbol $\Leftarrow$ (or its inverse $\Rightarrow$). A predicate

symbol P in the head of a rule of Δ is called a *defined predicate of* Δ; all other predicate and function symbols in Δ are called the *open symbols* or the *parameters* of the definition; the set of defined predicates is denoted $Def(\Delta)$, the remaining symbols $Open(\Delta)$. Note that $Open(\Delta)$ therefore also includes Σ^F.

Example 2.13. In order to express reachability in a graph, let us take the vocabulary $\{Reach/2, Edge/2\}$. The following definition defines the predicate $Reach$ in terms of the open predicate $Edge$.

$$\left\{ \begin{array}{l} \forall x \forall y\, Reach(x,y) \leftarrow Edge(x,y). \\ \\ \forall x \forall y\, Reach(x,y) \leftarrow \exists z(Reach(x,z) \wedge Reach(z,y)). \end{array} \right\}$$

Example 2.14. Recall that in Section 2.1 we mentioned that in general the domain closure axiom is not expressible in FO. However, it can be expressed in FO(ID). Given a finite set of constant and/or function symbols τ, let $S/1$ be a new predicate representing all elements of type S (for simplicity we assume that all arguments of all constant and function symbols have type S). Then $DCA(\tau)$ can be expressed as follows:

$$\left\{ \begin{array}{l} \ldots \\ S(C) \leftarrow . \\ S(F(t)) \leftarrow S(t). \\ \ldots \\ \ldots \end{array} \right\}$$
$$\forall x S(x).$$

A definition Δ where none of the defined predicates occur negatively in the bodies of the rules of Δ, is called *positive*. The definition of Example 2.13 above is such a positive definition. Another important class of definitions is the class of *stratified definitions*.

Definition 2.15. A definition Δ is stratified if there exists a mapping l from $Def(\Delta)$ to $\mathbb{N}$ such that if P and Q are two predicates such that P occurs positively in a body of a rule defining Q, it holds that $l(P) \leq l(Q)$, and if P occurs negatively in a body of a rule defining Q, it holds that $l(P) < l(Q)$.

An FO(ID) formula is defined as a standard FO formula, augmented with one extra case:

- A definition Δ over Σ is an FO(ID) formula (over Σ).

FO(ID) formulas are quantified boolean combinations of atoms and definitions. Notice that rule bodies do not contain definitions, and that rules only occur inside definitions and are not FO(ID) formulas themselves.

From now on, we assume without loss of generality that for any definition Δ, it holds that every defined predicate $P \in Def(\Delta)$ is defined by exactly one rule, denoted by $\forall \bar{x}(P(\bar{x}) \leftarrow \varphi_P[\bar{x}])$. Indeed, any definition Δ can be brought into this form by a process similar to predicate completion. The transformation consists of first transforming rules of the form $\forall \bar{x}(P(\bar{t}) \leftarrow \varphi)$ into equivalent rules $\forall \bar{y}(P(\bar{y}) \leftarrow \exists \bar{x}(\bar{y} = \bar{t} \wedge \varphi))$. Next, we merge all rules with the same head predicate P into one rule:

$$\begin{array}{lll} \forall \bar{x}(P(\bar{x}) \leftarrow \varphi_1[\bar{x}]) & & \\ \quad \vdots & \text{becomes} & \forall \bar{x}(P(\bar{x}) \leftarrow \varphi_1[\bar{x}] \vee \ldots \vee \varphi_n[\bar{x}]). \\ \forall \bar{x}(P(\bar{x}) \leftarrow \varphi_n[\bar{x}]) & & \end{array}$$

Semantics

In general, the model of an inductive definition is defined as the *well-founded fixpoint* of a certain approximation, as defined in Section 2.2. For a set of predicates σ, we denote by $\mathcal{L}^F_\sigma$ the class of all two-valued $(\Sigma^F \cup \sigma)$-interpretations that extend some given pre-interpretation F. It is easy to see that $\langle \mathcal{L}^F_\sigma, \leq_t \rangle$ (where $\leq_t$ is the truth-order) is a complete lattice. The idea is now to define an operator on the bilattice $(\mathcal{L}^F_{Def(\Delta)})^2$ that is an *approximation*. Recall that in Section 2.1 we saw that there is an isomorphism between pairs of interpretations and four-valued interpretations, and denoted this isomorphism by τ. We therefore sometimes abuse notation and use the two, that is, pairs of interpretations and four-valued interpretations, interchangeably, to make notation easier.

Definition 2.16 (Operator $\mathcal{T}^{\mathcal{O}}_\Delta$)**.** For a definition Δ and a given (potentially 4-valued) *Open* (Δ)-interpretation $\mathcal{O}$ extending a pre-interpretation F, we define the operator $\mathcal{T}^{\mathcal{O}}_\Delta$ on $(\mathcal{L}^F_{Def(\Delta)})^2$ such that $\mathcal{T}^{\mathcal{O}}_\Delta(\mathcal{I}) = \mathcal{J}$ iff for each defined predicate P/n and n-tuple $\bar{d} \in D^n$, it holds that

$$P^{\mathcal{J}}(\bar{d}) = \varphi_P^{\mathcal{O}+\mathcal{I}}[\bar{d}]$$

Phrased in terms of pairs of interpretations, this operator works as follows. If we give this operator a pair (I, J) approximating some interpretation $I \leq K \leq J$, it will produce a new, more precise estimate (I', J'), by using (I, J) to under-estimate the truth of the bodies of rules in Δ, and using (J, I) to over-estimate the truth of the bodies.

It is easy to see that the operator $\mathcal{T}^{\mathcal{O}}_\Delta$ is a *symmetric approximation*, which means the operator has an associated upper and lower stable operator, which are equal, so only one of them is needed. We denote this operator by $ST^{\mathcal{O}}_\Delta$, and recall that this operator is defined as the operator $\lambda J.(\text{lfp}(\lambda K.(\mathcal{T}^{\mathcal{O}}_\Delta(K, J)_1))$.

Again, let us examine this operator from a bit closer to see what it concretely does. First look at the operator $\lambda K.(\mathcal{T}^{\mathcal{O}}_\Delta(K, J)_1)$. This operator takes a $Def(\Delta)$-structure K, turns it

into a 4-valued one by combining it with J, applies the operator $\mathcal{T}_\Delta^{\mathcal{O}}$, and projects the result on its first argument. We can see that $\mathcal{T}_\Delta^{\mathcal{O}}(K,J)_1 = L$ iff for each defined predicate P/n and n-tuple $\bar{d} \in D^n$, it holds that

$$P^L(\bar{d}) = \mathbf{t} \text{ iff } (\varphi_P[\bar{d}])^{+(O_1+K)-(O_2+J)} = \mathbf{t}$$

In other words, positive occurrences of atoms are evaluated in $O_1 + K$, negative occurrences in $O_2 + J$. This confirms that for each J, this operator $\lambda K.(\mathcal{T}_\Delta^{\mathcal{O}}(K,J)_1)$ is indeed monotone, as expected, and has a least fixpoint. The operator $ST_\Delta^{\mathcal{O}}$ now maps J to this least fixpoint, and it can be proven that this operator is indeed anti-monotone. We will use $\mathcal{ST}_\Delta^{\mathcal{O}}$ to denote the partial stable operator associated to $\mathcal{T}_\Delta^{\mathcal{O}}$ on $(\mathcal{L}^F_{Def(\Delta)})^2$.

The *well-founded model* of a definition Δ given $\mathcal{O}$ (denoted by $WFM_{\mathcal{O}}(\Delta)$) is now defined as the well-founded model of $\mathcal{T}_\Delta^{\mathcal{O}}$, i.e., the least fixpoint of the stable operator. Similarly, a pair (I,J) is a *partial stable model* of Δ given $\mathcal{O}$ iff (I,J) is a fixpoint of this stable operator. An interpretation I for which (I,I) is a partial stable model is called an *stable model*.

The above definitions generalize in a rather obvious way the standard well-founded semantics of propositional logic programs and are strongly linked to the stable semantics of Gelfond & Lifschitz (1988). In the case of a propositional logic program Δ, where $Open(\Delta) = \{\}$, the operator $\lambda K.(\mathcal{T}_\Delta(K,J)_1)$ is nothing else than the immediate consequence operator T_{Δ_J} of the Gelfond Lifschitz reduct Δ_J of Δ, and the operator that maps J to $\text{lfp}(\lambda K.(\mathcal{T}_\Delta(K,J)_1)$ is the stable operator of Δ. As shown by Van Gelder (1993), its maximal oscillation pair is indeed the well-founded model of Δ.

Definition 2.17 (Satisfaction relation for definitions)**.** Let Δ be a definition over Σ and I a Σ-interpretation. The satisfaction relation is defined as follows: $I \models \Delta$ iff $(I|_{Def(\Delta)}, I|_{Def(\Delta)})$ is the well-founded model of Δ in $I|_{Open(\Delta)}$.

Note that when the definition Δ has a three-valued well-founded model for every possible interpretation of the open predicates $Open(\Delta)$, the definition has no model (i.e., there does not exists an interpretation I such that $I \models \Delta$). We call a definition *total* if it has a two-valued well-founded model for every possible two-valued interpretation of its open predicates. Intuitively, total definitions are *well-formed* definitions, since they define for every defined predicate P and domain tuple $\bar{d}$, whether that tuple belongs to the relation P or not. While in general whether a definition is total is undecidable (Schlipf, 1995), almost all of the definitions that occur in practice are total, for example all definitions that occur in FO(ID) theories used in the ASP competition (Calimeri et al., 2011) are total. Also, the classes of all monotone and stratified definitions are total (Van Gelder et al., 1991).

In the case of a *positive definition*, it can be easily shown that the well-founded model coincides with the *least fixpoint* of the immediate consequence operator.

Definition 2.18 (Operator T_Δ^O)**.** With a definition Δ and a given $Open(\Delta)$-interpretation O, we define the immediate consequence operator T_Δ^O on two-valued $Def(\Delta)$-interpretations

such that $T_\Delta^O(I) = J$ iff for each defined predicate P/n and tuple $\bar{d} \in D^n$, it holds that $P^J(\bar{d}) = \mathbf{t}$ iff $\varphi_P^{(O+I)}[\bar{d}] = \mathbf{t}$.

A well-known concept regarding definitions, is the *completion* of a definition.

Definition 2.19. The *completion* of a definition Δ, denoted by $Comp(\Delta)$, is the FO theory that contains for every $P \in Def(\Delta)$ the sentence

$$\forall \bar{x}(P(\bar{x}) \Leftrightarrow \varphi_P[\bar{x}]).$$

If T is a theory, then we denote by $Comp(T)$ the result of replacing all definitions in T by their completion. Denecker & Ternovska (2008) showed that the completion of a definition Δ is weaker than Δ itself, that is, $\Delta \vDash Comp(\Delta)$.

For an FO(ID) theory T we will denote the class of all models of T by $Mod(T)$. For pure FO, it is straightforward to see that informal and formal semantics correspond. For inductive definitions however, this is maybe less obvious. Nevertheless, Denecker & Ternovska showed that the informal and formal semantics of inductive definitions do coincide for the most common types of inductive definitions: monotone induction, induction over a well-founded order and iterated inductive definitions. In the same paper it is also illustrated that inductive definitions cannot only be used to represent mathematical concepts, but also the sort of common sense knowledge that is often represented by logic programs, such as (local forms of) the closed world assumption, inheritance, exceptions, defaults and causality.

2.3.2 Other extensions of FO

In this section we briefly consider some other useful extensions of FO. For example, the modeling of many real world domains involves *integer arithmetic*. We can extend FO with integer arithmetic (denoted by FO(IA)) in a rather obvious way. In this logic, every vocabulary contains at least a base sort $\mathbb{Z}$, a predicate symbol $\leq$, functions symbols $+, -, \cdot, /$, mod, abs and for each $a \in \mathbb{Z}$ a constant symbol a. Then, in each structure I, this part of the vocabulary is interpreted in the obvious way.

In many real-world applications *aggregates* allow for a more compact and intuitive modeling. In the extension FO(Agg), FO extended with aggregates, an aggregate expression can be of one of two forms. The first is

$$F(\{x_1 \ldots x_n \mid \varphi(x_1, \ldots, x_n)\})$$

Here F is an aggregation function such as *cardinality*, *sum*, or *average*, and the expression denotes the result of applying this function to the set A of all tuples $(x_1, \ldots, x_n)$ for which $\varphi(x_1, \ldots, x_n)$ holds; in case of functions such as *sum*, which operate on numerical values

instead of tuples, the projection on the first element x_1 is used, i.e.,

$$sum\,(\{x_1 \dots x_n \mid \varphi(x_1, \dots, x_n)\}) = \sum_{(x_1,\dots,x_n)\in A} x_1$$

The second kind of aggregate expression is of the form:

$$cardinality\,(\{\varphi_1, \dots, \varphi_n\}).$$

This expression denotes the number of true formulas among the φ_i. To see when such an expression might be useful, we can consider government regulations concerning things such as scholarship grants; often, there are a number of different criteria that apply, and to be eligible for the grant, a student has to satisfy at least i of the n criteria. If we represent each criterion by some formula $\varphi_i(x)$, we can concisely write this down as:

$$\forall x\ cardinality\,(\{\varphi_1(x), \dots, \varphi_n(x)\}) \geq i \Leftrightarrow Eligible\,(x)$$

In this section we have presented a number of useful extensions of FO and will denote the logic resulting from combining these extensions by FO(·). While several other extensions of FO can be thought of, such as the extension FO(FP) of FO with fixpoint definitions which we will come back to further in this section, the language FO(·) is the language that is currently supported by the finite model generator IDP (idp).

2.3.3 Representing dynamic domains in FO(·)

Often in this thesis, we assume that we have an FO(·) theory that represents a *dynamic domain*. While there exists a number of languages specifically aimed for representing such domains (e.g., the situation calculus (Reiter, 1991), event calculus (Shanahan, 1997), the family of action languages (Gelfond & Lifschitz, 1998)), our motivation to use an FO(·)-based calculus is twofold. First of all, inductive definitions offer an elegant way of specifying the evolution of fluents in a domain. In (Denecker & Ternovska, 2007), this was studied in the context of the situation calculus, but we will define here a *linear time* variant. Our second reason for using FO(·), is that this allows us to use generic reasoning methods and off-the-shelf reasoners to solve problems in these dynamic domains.

Definition 2.20. A *Linear Time Calculus (LTC) vocabulary* Σ_{ltc} consists of

- a set of types, including a type *Time*,
- for every type S a set of constants Σ_S of type S,
- a set Σ_{stat} of *static* predicate and function symbols without an argument of type *Time*,

- a set Σ_{dyn} of *dynamic* predicate and function symbols that have exactly one temporal argument. The set Σ_{dyn} is further divided into a set Σ_{act} of predicate symbols representing *actions* and Σ_{flu} of predicate and function symbols representing *fluents*,
- a set Σ_{init} containing for every predicate $P/n \in \Sigma_{flu}$ a predicate *Init*$\_P/n$–1 having only the non-temporal arguments of P and for every function symbol $F/n \in \Sigma_{flu}$ a predicate symbol *Init*$\_F/n$–1, again with the same arguments as F but with the temporal argument dropped. These predicates represent the initial state of the world.
- a set Σ_{caus} of auxiliary causal predicates containing for every n–ary predicate $P(\bar{x}, t) \in \Sigma_{flu}$ two new predicates: $C_P(\bar{x}, t)$, which denotes that $P(\bar{x}, t)$ is caused to become true and $C_{\neg P}(\bar{x}, t)$, which denotes that $P(\bar{x}, t)$ is caused to become false, and for every n–ary function symbol $F(\bar{x}, t) \in \Sigma_{flu}$ one new predicate $C_F(\bar{x}, y, t)$, representing that $F(\bar{x}, t)$ is caused to become equal to y.

A *state formula* $\varphi[t]$ over Σ_{ltc} is an FO formula in which t occurs free and is the only term of sort $Time$. We say that φ *refers to* t. A *bi-state formula* $\varphi[t]$ over Σ_{ltc} is an FO formula in which T occurs free and t and $t+1$ are the only terms of sort $Time$.

In the rest of this text, whenever we want to represent dynamic domains, we assume that every type S apart from $Time$ has only a finite number of constants and restrict attention to Herbrand interpretations of S (that is, we assume that each LTC theory implicitly contains $UNA(\Sigma_S)$ and $DCA(\Sigma_S)$). This allows us to identify domain elements and constants. We will fix the interpretation of $Time$ to the natural numbers $\mathbb{N}$, and use constants $0, 1, \ldots$, the arithmetic functions $+$, and the comparison operator $\leq$, all fixed to their usual interpretation in $\mathbb{N}$.

As argued in (Denecker & Ternovska, 2007), inductive definitions offer an elegant way to specify the evolution of fluents in a domain, taking care of *inertia*. Inertia is the property of most fluents that their values do not change unless there is an explicit cause. It is relatively easy to represent what effects of such an explicit cause are. What is more difficult is expressing that everything else stays the same. This problem is commonly known as the *frame problem* (Reiter, 1991). Let Σ_{ltc} be a LTC vocabulary. A *fluent definition* Δ_{fluent} is the union of the following definitions: for every predicate symbol $P/n \in \Sigma_{flu}$, the definition

$$\left\{ \begin{array}{rcl} P(\bar{x}, t+1) & \leftarrow & C_P(\bar{x}, t+1) \vee (P(\bar{x}, t) \wedge \neg C_{\neg P}(\bar{x}, t+1)). \\ P(\bar{x}, 0) & \leftarrow & \mathit{Init}\_P(\bar{x}). \\ C_P(\bar{x}, t+1) & \leftarrow & \varphi_P[\bar{x}, t]. \\ C_{\neg P}(\bar{x}, t+1) & \leftarrow & \varphi_{\neg P}[\bar{x}, t]. \end{array} \right\},$$

and for every function symbol $F/n \in \Sigma_{flu}$, the definition

$$\left\{ \begin{array}{rcl} F(\bar{x}, t+1) = y & \leftarrow & C_F(\bar{x}, y, t+1) \vee (F(\bar{x}, t) = y \wedge \neg \exists z (C_F(\bar{x}, z, t+1))). \\ F(\bar{x}, 0) = y & \leftarrow & \mathit{Init}\_F(\bar{x}, y). \\ C_F(\bar{x}, y, t+1) & \leftarrow & \varphi_F[\bar{x}, y, t]. \end{array} \right\},$$

where the formulas $\varphi_P[\bar{x}, t]$, $\varphi_{\neg P}[\bar{x}, t]$ and $\varphi_F[\bar{x}, y, t]$ in the bodies of the rules above need to be state or bi-state formulas. Intuitively the first rule in the first definition above expresses that an atom $P(\bar{x}, t+1)$ is true if and only if there is some direct cause for it to be true, or it was true in the previous timepoint and there is no cause for it to be terminated. The second rule defines the value of the fluents for the initial situation, and the last rules define the causality predicates.

We do not always need the auxiliary causal symbols C_P, $C_{\neg P}$, etc. Indeed, when the formulas φ_P, $\varphi_{\neg P}$, etc. in the definition Δ_{fluent} do not contain symbols from Σ_{caus}, we can replace their occurrences in rules defining the successor state by their definitions. When the causes are simple and few, we will often use this simplification.

In case there exists a well-founded order on the atoms of the definition Δ_{fluent}[2], the definition can be equivalently replaced by its completion. This does not hold in general however, for example, when there are (immediate) cyclic causations. It is known that pure FO formalizations of these domains have unintended models, e.g., a prototypical temporal domain where this is the case is a domain with two gear wheels. If one wheel turns (due to some external cause) that causes the other to turn and vice versa. In FO, there are models where the wheels 'self-supportingly' cause each other to turn, which is clearly an undesired model. In (Denecker & Ternovska, 2007) it was shown that the solution above, using an inductive definition to define causal predicates, does give the correct models.

Example 2.21. Consider a temporal domain where there are a number of plates, say three. Plates can either be clean or dirty. Using a plate makes it dirty, wiping a plate obviously makes it clean. Since we don't want to eat from a dirty plate, a precondition for using a plate is that it is clean. Also, we can only do one action at a time. We know that initially the only clean plate is plate number 2. We use the following vocabulary to represent this domain:

- Types: *Plate* , *Time*
- Static constants: *Plate1, Plate2, Plate3* , and the natural numbers.
- Fluent predicates: *Clean* (*Plate,Time*),
- Action predicates: *WipePlate* (*Plate,Time*) and *UsePlate* (*Plate,Time*),
- Auxiliary predicates: *Init_Clean* (*Plate*), C_{Clean}(*Plate,Time*), and $C_{\neg Clean}$(*Plate,Time*).

The following FO(·) theory T (assuming that we implicitly include the necessary UNA's and DCA's) now is a correct representation of this temporal domain:

[2] in the sense that for every atom $Q[\bar{d}]$ in a body $\varphi_P[\bar{d}]$ of a of a definitional rule defining the predicate P, it holds that $Q[\bar{d}] < P(\bar{d})$.

$$\left\{\begin{array}{rcl} Clean(p,t+1) & \leftarrow & C_{Clean}(p,t+1) \vee (Clean(p,t) \wedge \neg C_{\neg Clean}(p,t+1)). \\ Clean(p,0) & \leftarrow & Init\_Clean(p). \\ C_{Clean}(p,t+1) & \leftarrow & WipePlate(p,t). \\ C_{\neg Clean}(p,t+1) & \leftarrow & UsePlate(p,t). \end{array}\right\}$$

$\forall p \forall t(UsePlate(p,t) \Rightarrow Clean(p,t))$. (precondition for UsePlate)
$\forall p_1 p_2 \forall t(UsePlate(p_1,t) \wedge UsePlate(p_2,t) \Rightarrow p_1 = p_2)$. (no concurrency)
$\forall p_1 p_2 \forall t(WipePlate(p_1,t) \wedge WipePlate(p_2,t) \Rightarrow p_1 = p_2)$.
$\forall t \neg \exists p_1 p_2(UsePlate(p_1,t) \wedge WipePlate(p_2,t))$.

$\forall p\, Init\_Clean(p) \Leftrightarrow p = Plate2$. (description of initial situation)

In conclusion, LTC is a simple but quite general formalism to represent temporal systems. It is based on FO(·) and allows for concurrent actions, dynamic functions and complex preconditions (e.g., potentially even referring to a future state), and can be easily extend to also allow non-determinism (by allowing non-inertial fluents), features which are often not available in other formalisms.

2.4 FO(FP)

In this section we recall an alternative extension of FO, namely, the logic FO(FP), which is an extension of classical logic with *fixpoint definitions*. This logic was first defined in (Hou et al., 2010), and adds *mixed induction and co-induction* to FO (compared to FO(ID), where we extended FO with inductive definitions). FO(FP) allows one to express a number of concepts that cannot (or only very unpleasantly) be expressed in FO(ID). The notion of fixpoint definitions in FO(FP) is a syntactic variant of the notion of nested least and greatest fixpoint expressions (Park, 1969). A difference is that in FO(FP) we define predicate symbols that represent the fixpoint expression, compared to directly using fixpoint expressions denoting relations.

Syntax

In FO(FP), fixpoint definitions are again built out of rules. As before, a *rule* over a vocabulary Σ is an expression of the form: $\forall \bar{x}(P(\bar{x}) \leftarrow \varphi_P[\bar{x}])$, where P is a predicate symbol of Σ and $\varphi_P[\bar{x}]$ is an arbitrary first-order formula over Σ. We say that a rule is *positive in a set of predicate symbols* σ if each predicate of σ has only positive occurrences in the body of the rule (i.e., is in the scope of an even number of $\neg$).

A *fixpoint definition* $\mathcal{D}$ is defined inductively as either a *least fixpoint definition* (denoted by Δ) $\lfloor \mathcal{R}, \Delta_1, \ldots, \Delta_n, \nabla_1, \ldots, \nabla_m \rfloor$ or a *greatest fixpoint definition* (denoted by ∇) $\lceil \mathcal{R}, \Delta_1, \ldots, \Delta_n, \nabla_1, \ldots, \nabla_m \rceil$, where in both cases $\mathcal{R}$ is a set of rules, and $\Delta_1, \ldots, \Delta_n$ are least fixpoint definitions and $\nabla_1, \ldots, \nabla_m$ are greatest fixpoint definitions. The defined predicates $Def(\mathcal{D})$ of a fixpoint definition $\mathcal{D}$ are $Def(\mathcal{R}) \cup Def(\Delta_1) \cup \cdots \cup Def(\Delta_n) \cup Def(\nabla_1), \ldots, Def(\nabla_m)$; as before, all other symbols belong to $Open(D)$, the set of open symbols of D. We can now formulate the following additional requirements on a fixpoint definition $\mathcal{D}$.

- Every defined symbol of $\mathcal{D}$ has only positive occurrences in $\mathcal{D}$,
- $Def(\mathcal{R}), Def(\Delta_1), \ldots, Def(\Delta_n), Def(\nabla_1), \ldots, Def(\nabla_m)$ are all mutually disjoint,
- For every subdefinition $\mathcal{D}'$ of $\mathcal{D}$ (i.e., a least fixpoint definition Δ_i or greatest fixpoint definition ∇_i), $Open(\mathcal{D}') \subseteq Open(\mathcal{D}) \cup Def(\mathcal{R})$. In particular, a symbol defined in another subdefinition $\mathcal{D}'' \neq \mathcal{D}'$, does not occur in $\mathcal{D}'$.

An FO(FP) formula is defined as in FO with one extra case:

- A fixpoint definition $\mathcal{D}$ over Σ is an FO(FP) formula (over Σ).

Semantics

With each fixpoint definition $\mathcal{D}$, and an $Open(\mathcal{D})$-interpretation I, we can define a monotonic operator $\Gamma^I_{\mathcal{D}}$ on the set of $Def(\mathcal{D})$-interpretations.

Definition 2.22 (Operator $\Gamma^I_{\mathcal{D}}$)**.** Define $\Gamma^I_{\mathcal{D}}(J)$ through induction over the nested structure of the fixpoint definition $\mathcal{D}$, as the interpretation $K + K'$ where

- K is the $(Def(\mathcal{D}) \setminus Def(\mathcal{R}))$-interpretation such that, for $J' = I + J|_{Def(\mathcal{R})}$:
  - $K|_{Def(\Delta_i)} = \mathrm{lfp}(\Gamma^{J'}_{\Delta_i})$,
  - $K|_{Def(\nabla_j)} = \mathrm{gfp}(\Gamma^{J'}_{\nabla_j})$.

  Note that J' is an interpretation of the open symbols in the nested definitions.
- K' is the $Def(\mathcal{R})$-interpretation $T^{I+K}_{\mathcal{R}}(J|_{Def(\mathcal{R})})$. Note that $I + K$ is an interpretation of the open symbols in $\mathcal{R}$.

When $\mathcal{D}$ contains no subdefinitions, we get that $\Gamma^I_{\mathcal{D}}$ simply becomes the immediate consequence operator $T^I_{\mathcal{R}}$, since then $Def(\mathcal{D}) = Def(\mathcal{R})$. This operator is now used to define the model of a nested fixpoint definition.

Definition 2.23 (Satisfaction relation for fixpoint definitions)**.** Let $\mathcal{D}$ be a fixpoint definition and I a two-valued Σ-interpretation such that Σ contains all symbols in $\mathcal{D}$.

- If $\mathcal{D}$ is a least fixpoint definition, then I is a model of $\mathcal{D}$, denoted $I \models \mathcal{D}$, iff $I|_{Def(\mathcal{D})} = \mathrm{lfp}(\Gamma_{\mathcal{D}}^{I|_{Open\ \mathcal{D}}})$.
- If $\mathcal{D}$ is a greatest fixpoint definition, then I is a model of $\mathcal{D}$, denoted $I \models \mathcal{D}$, iff $I|_{Def(\mathcal{D})} = \mathrm{gfp}(\Gamma_{\mathcal{D}}^{I|_{Open\ \mathcal{D}}})$.

The satisfaction relation $I \models \varphi$ of FO(FP) is now again defined using the standard inductive rules of FO, augmented with one extra rule, namely the one above that defines when $I \models \mathcal{D}$, for a nested fixpoint expresssion $\mathcal{D}$.

An example of a nested fixpoint definition, taken from (Hou et al., 2010), is the following.

Example 2.24. Given is a transition graph on a set of vertices representing states by a binary predicate T. Let R represent a property of the states, i.e., it is a unary predicate on vertices of the graph. We want P to represent the set of states which are on a path that passes through an infinite number of states with property R. This set is defined by:

$$\left[\begin{array}{l} \forall x\,(P(x) \leftarrow Q(x)) \\ \left[\begin{array}{lll} \forall x\,(Q(x) & \leftarrow & R(x) \wedge \exists y(T(x,y) \wedge P(y))) \\ \forall x\,(Q(x) & \leftarrow & \exists y(T(x,y) \wedge Q(y))) \end{array}\right] \end{array}\right]$$

This nested fixpoint definition consists of an outer greatest fixpoint definition and a nested least fixpoint definition. The least fixpoint definition has two rules defining the predicate Q. The first one can be seen as the 'base case'. Since the predicate P is defined in the greatest fixpoint definition and is thus initially assumed to be true for all nodes, Q will be true for all nodes that have the property R and are not the end point of a path. The second rule is the 'inductive rule' and causes Q to be true for all nodes of the graph that are on a path that leads to a node for which Q was already true, i.e., to a node with property R that is not an end node. Now the rule in the greatest fixpoint causes the predicate P to be 'updated' from being true for all nodes, to being true for the nodes that Q is true for. So then the nested least fixpoint is recomputed with this smaller interpretation for P. This time the base case rule will only add nodes that satisfy R but also they have to be on a path where the next node is still in P. We can see that this process will indeed 'throw out' all nodes that are not on a path that contains an infinite number of states with property R.

We have now recalled two extensions of FO, namely, FO(ID) and FO(FP), and an obvious question to ask is how they relate to each other. In (Hou et al., 2010) an equivalence preserving transformation from non-monotone inductive definitions in FO(ID) to nested fixpoint definitions is presented. We now repeat this transformation.

Definition 2.25 (Transformation from FO(ID) to FO(FP))**.** For an inductive definition Δ, we define $\mathcal{D}_\Delta$ as $\lfloor\ \mathcal{R}^{ct}, \lceil\ \mathcal{R}^{cf}\ \rceil\ \rfloor$ where $\mathcal{R}^{ct}$ consists of the rules

$$\forall \bar{x}(P^{ct}(\bar{x}) \leftarrow \varphi^{ct})$$

and $\mathcal{R}^{cf}$ consists of the rules

$$\forall \bar{x}(P^{cf}(\bar{x}) \leftarrow \varphi^{cf})$$

for every definitional rule $\forall \bar{x}\ P(\bar{x}) \leftarrow \varphi \in \Delta$. Here φ^{ct} and φ^{cf} are as defined by Definition 2.8.

In (Hou et al., 2010) it is proven that this transformation from Δ to $\mathcal{D}_\Delta$ is equivalence preserving, in the sense that there exists a one-to-one mapping between the models of Δ and $\mathcal{D}_\Delta$. To be more precise, given a two-valued interpretation I, then $I \vDash \Delta$ iff $I^{tf} \vDash \mathcal{D}_\Delta$.

2.5 Autoepistemic Logic

In this section, we give a short overview of autoepistemic logic (AEL). Autoepistemic logic was first introduced by Moore (1984) and proposes a way to complete collections of base facts about the world, possibly containing facts about an agent's belief or disbelief, to belief sets, called *expansions*, that the agent might hold given the base theory. Let $\mathcal{L}$ be the language of propositional logic based on a set of atoms Σ. Extending this language with a modal operator K gives a language $\mathcal{L}_K$ of modal propositional logic. An autoepistemic theory is a set of formulas in this language $\mathcal{L}_K$. An *interpretation* or *world* is a subset of the vocabulary Σ. The set of all interpretations of Σ is denoted by $\mathcal{I}_\Sigma$, i.e., $\mathcal{I}_\Sigma = 2^\Sigma$. A *possible world structure* is a set of interpretations. Intuitively, a possible world structure sums up all 'situations' which are possible. We can order possible world structures according to the inverse set inclusion to get a *knowledge order*, denoted by $\leq_k$, that is, for two possible world structures Q, Q', $Q \leq_k Q'$ iff $Q \supseteq Q'$. Indeed, if a a possible world structure contains more possibilities, it actually contains less knowledge. The set of all possible world structures is denoted by $\mathcal{W}_\Sigma$ and defined as $2^{\mathcal{I}_\Sigma}$. It is not hard to see that $\langle \mathcal{W}_\Sigma, \leq_k \rangle$ is a complete lattice. Figure 2.2 shows part of the lattice $\mathcal{W}_\Sigma$ for $\Sigma = \{p, q\}$.

A possible-world structure can be viewed as a universal Kripke model with a total accessibility relation (Chellas, 1980; Hughes & Cresswell, 1996). Possible-world structures were used by Moore (Moore, 1984) and later by Levesque (Levesque, 1990) in their investigations of autoepistemic logic. For a formula $\varphi \in \mathcal{L}_K$, the truth function $\mathcal{H}_{Q,I}$, where Q is a possible-world structure, and I an interpretation, is defined as follows.

- For an atom p, $\mathcal{H}_{Q,I}(p) = \mathbf{t}$ iff $p \in I$.

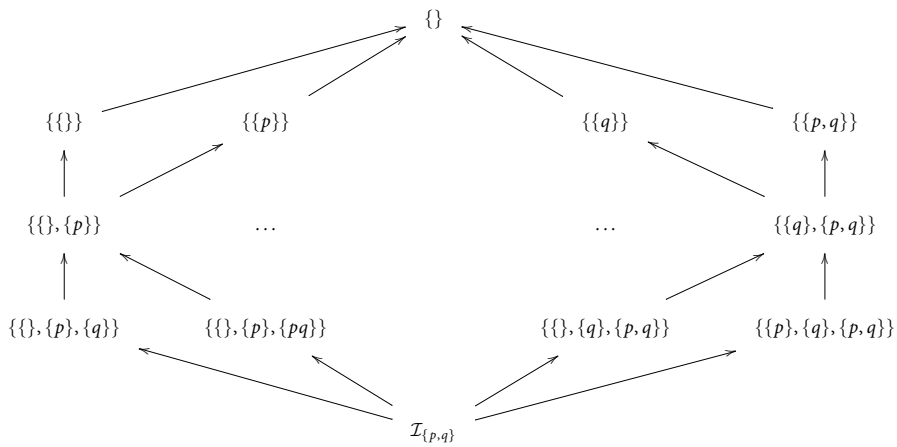

Figure 2.2: Part of the lattice $\mathcal{W}_{\{p,q\}}$, ordered according to $\leq_k$.

- $\mathcal{H}_{Q,I}(\varphi_1 \wedge \varphi_2) = \mathbf{t}$ iff $\mathcal{H}_{Q,I}(\varphi_1) = \mathbf{t}$ and $\mathcal{H}_{Q,I}(\varphi_2) = \mathbf{t}$.
- $\mathcal{H}_{Q,I}(\varphi_1 \vee \varphi_2) = \mathbf{t}$ iff $\mathcal{H}_{Q,I}(\varphi_1) = \mathbf{t}$ or $\mathcal{H}_{Q,I}(\varphi_2) = \mathbf{t}$.
- $\mathcal{H}_{Q,I}(\neg\varphi) = \mathbf{t}$ iff $\mathcal{H}_{Q,I}(\varphi) = \mathbf{f}$.
- $\mathcal{H}_{Q,I}(K\varphi) = t$, iff for every interpretation $J \in Q$, $\mathcal{H}_{Q,J}(\varphi) = \mathbf{t}$.

Note that the value of a modal atom $K\varphi$ given by $\mathcal{H}_{Q,I}$ does not depend on I. Thus, it is entirely determined by the possible-world structure Q. For every modal theory T, Moore defined an operator D_T on $\mathcal{W}_\Sigma$:

$$D_T(Q) = \{I | \mathcal{H}_{Q,I}(\varphi) = \mathbf{t}, \text{ for every } \varphi \in T\}. \tag{2.2}$$

The intuition behind this definition is as follows. The possible-world structure $D_T(Q)$ is a revision of a possible-world structure Q. This revision consists of the worlds that make all the formulas in T true in the context of the current belief state Q. Fixpoints of the operator D_T represent "stable" belief sets, i.e., they cannot be revised any further. Though originally Moore used the term *stable expansion* to denote all *formulas* φ such that $K\varphi$ is true in such a fixpoint of the operator D_T, we will use the term stable expansion here to refer to the fixpoints themselves.

In (Denecker et al., 2003) a more general semantic approach is developed, based on the concepts from approximation theory, making it possible to define a whole *family* of semantics, such as a well-founded semantics.

3

Propagation for FO(ID)

In a number of practical applications, we encounter the following problem: given a theory and some information about a part of the vocabulary, we would like to find out what the *logical implications* are of this information together with the theory. We start off by illustrating this with an example.

Example 3.1. The *n-Queens problem* is a well-known problem where the goal is to put n queens on a chess-board of size $n \times n$ so that none of them can capture another queen along a row, column or diagonal. The following theory T_{queens} is one possible representation of valid solutions of this problem.

$$\forall r\ \exists c\ (Queen(r,c) \land \forall c'\ (Queen(r,c') \Rightarrow c = c')).$$
$$\forall c\ \exists r\ (Queen(r,c) \land \forall r'\ (Queen(r',c) \Rightarrow r = r')).$$
$$\forall r\ c\ r'\ c'\ (Diagonal(r,c,r',c') \land Queen(r,c) \land Queen(r',c')) \Rightarrow r = r' \land c = c'.$$

Assume that together with this theory, we are given extra information. We know that the chessboard has dimensions 4×4 and one queen has already been placed on square $(2,1)$ (we don't know anything about the other queens). We are also given an interpretation for the predicate $Diagonal/4$ (such that it effectively represents the diagonal of the chessboard). If we now assume that all the constraints in theory T_{queens} have to be satisfied, this, together with our extra information, has a number of implications. First of all, since we know that that there is a queen on square $(2,1)$, the first sentence of T_{queens} implies that there *can not* be another queen in the same row, that is, on squares $(2,2)$, $(2,3)$ and $(2,4)$. Similarly, the second sentence implies that there can not be a queen on $(1,1)$, $(3,1)$ and $(4,1)$, and the

last sentence, together with the interpretation for *Diagonal*, that there cannot be a queen on $(3,2)$, $(1,2)$ and $(4,3)$. But now the second sentence says that each column should have a queen. Therefore, there *must be* a queen on $(4,2)$, since this is the only possibility left. Similarly, there must be a queen on $(1,3)$, and so on. This *propagation* process is illustrated in Figure 3.1.

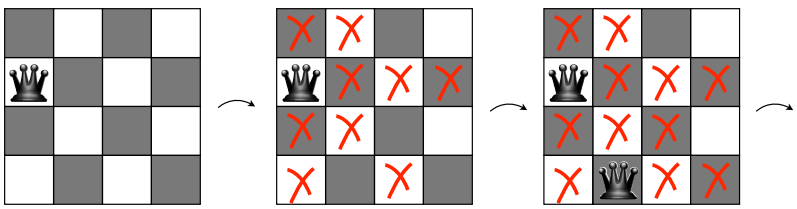

Figure 3.1: Step by step, more logical implications are derived.

More formally, we would like to do the following. Suppose we have an FO(ID) theory T in vocabulary Σ, a pre-interpretation of Σ, and a finite three-valued interpretation $\mathcal{I}$ that represents some (incomplete) knowledge about the predicates of Σ. We would like to know the implications of this knowledge, assuming the theory T is satisfied in the context of $\mathcal{I}$. To find this out, we can look at the set $\mathcal{M}$ of all models of T that complete this three-valued interpretation, i.e., $\mathcal{M} = \{M \mid M \models T \text{ and } \mathcal{I} \leq_p M\}$. Given the partial information $\mathcal{I}$, everything that is true in all $M \in \mathcal{M}$ must certainly be true according to T, while everything that is false in all such M must certainly be false according to T. In other words, all the information that T allows us to derive from $\mathcal{I}$ can be captured by the greatest lower bound $\mathcal{G} = \text{glb}_{\leq_p}\mathcal{M}$.

In general, computing this greatest lower bound may be too expensive (the data complexity is in Δ_2^P) to be of practical use. However, we may still achieve useful results by computing (more efficiently) some approximation $\tilde{\mathcal{M}}$ such that $\mathcal{I} \leq_p \tilde{\mathcal{M}} \leq_p \mathcal{G}$. Such an approximation can be computed using a *constraint propagation* method for FO(ID). Intuitively, the idea behind this propagation method is to see FO(ID)-formulas as *constraints* on a certain set of interpretations, and, just like in constraint programming approaches, associate with each type of constraint (that is, type of formula) a number of propagation rules to reduce the number of possible interpretations. Obviously, these propagation rules have to be efficiently computable. Such constraint propagation was first introduced for FO in (Wittocx et al., 2010) as a tool to make the grounding of first-order theories smaller, but, as we will show in the next chapters, has many other interesting applications as well. Moreover, the *propagation process* using these propagation rules has the interesting property that it can be *syntactically represented in the logic* FO(ID) itself, as an inductive definition, and the model of such an inductive definition can be computed in polynomial time.

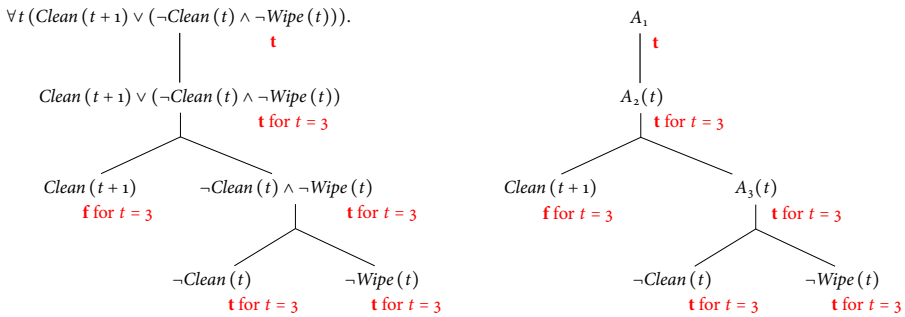

Figure 3.2: Propagation for FO.

In Section 3.1, we start with a slightly generalized presentation of the work in (Wittocx et al., 2010). We come back to the precise relation between the material presented in this section and the work in that paper at the end of the section. Next, in Section 3.2, we extend the propagation method for first-order logic to FO(ID).

3.1 Propagation for FO

We start this section with an example to illustrate propagation for first-order logic.

Example 3.2. Consider the sentence ψ: $\forall t\, Clean\,(t+1) \Leftarrow Clean\,(t) \vee Wipe\,(t)$. Rewriting this into negation normal form, it becomes:

$$\forall t\, Clean\,(t+1) \vee (\neg Clean\,(t) \wedge \neg Wipe\,(t)).$$

Now, assume that ψ is satisfied, and that we know that *Clean* is false at timepoint 4. From the knowledge that ψ is satisfied, it immediately follows that, for all timepoints t, the disjunctive formula $Clean\,(t+1) \vee (\neg Clean\,(t) \wedge \neg Wipe\,(t))$ is satisfied. Using the fact that *Clean* is false at timepoint 4, we can now deduce that the conjunction $(\neg Clean\,(t) \wedge \neg Wipe\,(t))$ is true for timepoint 3. Therefore, in all models of ψ where *Clean* is false at timepoint 4, *Wipe* and *Clean* have to be false at timepoint 3. This reasoning process is illustrated on the left part of Figure 3.2.

We now construct a symbolic representation of this propagation process. First, we introduce some additional vocabulary Σ_{Aux} to refer to the different nodes of the parse tree on which this process operates. We then use this additional vocabulary to transform an FO formula into *equivalence normal form*. This is similar to the Tseitin transformation for propositional logic (Tseitin, 1968).

Definition 3.3 ($ENF(\psi)$)**.** For an FO formula ψ in negation normal form, we introduce a new predicate symbol A_φ of arity n for each non-literal subformula $\varphi[\bar{x}]$ of ψ with n free variables. We denote the set of these new predicates by $Aux(\psi)$. Each of these new predicate symbols is defined by a formula $Eq(A_\varphi)$ as follows. To make notation simpler we assume that each $\varphi_1, \ldots, \varphi_n$ is a non-literal subformula. The definitions are analogous whenever a φ_i is a literal, but instead of A_{φ_i} the literal φ_i itself is used in the body of the definition.

- If $\varphi[\bar{x}]$ is a subformula of the form $\varphi_1[\bar{x_1}] \wedge \varphi_2[\bar{x_2}] \wedge \ldots \wedge \varphi_n[\bar{x_n}]$, then $Eq(A_\varphi)$ is $\forall \bar{x}\,(A_\varphi(\bar{x}) \Leftrightarrow A_{\varphi_1}(\bar{x_1}) \wedge A_{\varphi_2}(\bar{x_2}) \wedge \ldots \wedge A_{\varphi_n}(\bar{x_n}))$.
- If $\varphi[\bar{x}]$ is a subformula of the form $\varphi_1[\bar{x_1}] \vee \varphi_2[\bar{x_2}] \vee \ldots \vee \varphi_n[\bar{x_n}]$, then $Eq(A_\varphi)$ is $\forall \bar{x}\,(A_\varphi(\bar{x}) \Leftrightarrow A_{\varphi_1}(\bar{x_1}) \vee A_{\varphi_2}(\bar{x_2}) \vee \ldots \vee A_{\varphi_n}(\bar{x_n}))$.
- If $\varphi[\bar{x}]$ is a subformula of the form $\forall \bar{y}\, \varphi_1[\bar{x}]$, then $Eq(A_\varphi)$ is $\forall \bar{x}\,(A_\varphi(\bar{x}) \Leftrightarrow \forall \bar{y}\, A_{\varphi_1}(\bar{x}, \bar{y}))$.
- If $\varphi[\bar{x}]$ is a subformula of the form $\exists \bar{y}\, \varphi_1[\bar{x}, \bar{y}]$, then $Eq(A_\varphi)$ is $\forall \bar{x}\,(A_\varphi(\bar{x}) \Leftrightarrow \exists \bar{y}\, A_{\varphi_1}(\bar{x}, \bar{y}))$.

We define the *equivalence normal form* of ψ as the set of all such $Eq(A_\varphi)$, and denote it as $ENF(\psi)$.

Example 3.4. According to this definition, the $ENF(\psi)$ theory from Example 3.2 is:

$$\begin{array}{rcl} A_1 & \Leftrightarrow & \forall t\, A_2(t). \\ \forall t\, A_2(t) & \Leftrightarrow & \mathit{Clean}\,(t+1) \vee A_3(t). \\ \forall t\, A_3(t) & \Leftrightarrow & \neg \mathit{Clean}\,(t) \wedge \neg \mathit{Wipe}\,(t). \end{array}$$

This is illustrated in the right side of Figure 3.2.

Using the auxiliary vocabulary, we can now write down the propagations shown in Figure 3.2 as the following implications.

$$\begin{array}{lcl} A_1 & \Rightarrow & A_2(3). \\ A_2(3) \wedge \neg \mathit{Clean}\,(4) & \Rightarrow & A_3(3). \\ A_3(3) & \Rightarrow & \neg \mathit{Clean}\,(3). \\ A_3(3) & \Rightarrow & \neg \mathit{Wipe}\,(3). \end{array}$$

Note that these rules are all *top-down* rules, that is, implications that propagate information about a subformula down the parse tree, to a component of that subformula (possibly using also information about other components of the subformula, as in the implication $A_2(3) \wedge \neg \mathit{Clean}\,(4) \Rightarrow A_3(3)$). In general, also *bottom-up* propagations are of course possible. For instance, from $\mathit{Clean}\,(4)$ we could derive $A_2(3)$. For every predicate A_ψ, we

$\varphi = Eq(A_\psi)$	$Imp(\varphi)$		
$\forall \bar{x}\ (L \Leftrightarrow L_1 \wedge \ldots \wedge L_n).$	$\forall \bar{x}\ (L_1 \wedge \ldots \wedge L_n \Rightarrow L).$		BU
	$\forall \bar{x}\ (\neg L_i \Rightarrow \neg L).$	$1 \leq i \leq n$	BU
	$\forall \bar{x}\ (L \Rightarrow L_i).$	$1 \leq i \leq n$	TD
	$\forall \bar{x}\ (\neg L \wedge L_1 \wedge \ldots \wedge L_{i-1} \wedge L_{i+1} \wedge \ldots \wedge L_n \Rightarrow \neg L_i).$	$1 \leq i \leq n$	TD
$\forall \bar{x}\ (L \Leftrightarrow L_1 \vee \ldots \vee L_n).$	$\forall \bar{x}\ (\neg L_1 \wedge \ldots \wedge \neg L_n \Rightarrow \neg L).$		BU
	$\forall \bar{x}\ (L_i \Rightarrow L).$	$1 \leq i \leq n$	BU
	$\forall \bar{x}\ (\neg L \Rightarrow \neg L_i).$	$1 \leq i \leq n$	TD
	$\forall \bar{x}\ (L \wedge \neg L_1 \wedge \ldots \wedge \neg L_{i-1} \wedge \neg L_{i+1} \wedge \ldots \wedge \neg L_n \Rightarrow L_i).$	$1 \leq i \leq n$	TD
$\forall \bar{x}\ (L[\bar{x}] \Leftrightarrow \forall \bar{y}\ L'[\bar{x}, \bar{y}]).$	$\forall \bar{x}\ ((\forall \bar{y}\ L'[\bar{x}, \bar{y}]) \Rightarrow L[\bar{x}]).$		BU
	$\forall \bar{x}(\exists \bar{y}\ \neg L'[\bar{x}, \bar{y}]) \Rightarrow \neg L[\bar{x}]).$		BU
	$\forall \bar{x} \forall \bar{y}\ (L[\bar{x}] \Rightarrow L'[\bar{x}, \bar{y}]).$		TD
	$\forall \bar{x} \forall \bar{y}\ ((\neg L[\bar{x}] \wedge \forall \bar{z}\ (\bar{y} \neq \bar{z} \Rightarrow L'[\bar{x}, \bar{y}][\bar{y}/\bar{z}])) \Rightarrow \neg L'[\bar{x}, \bar{y}]).$		TD
$\forall \bar{x}\ (L[\bar{x}] \Leftrightarrow \exists \bar{y}\ L'[\bar{x}, \bar{y}]).$	$\forall \bar{x}\ ((\forall \bar{y}\ \neg L'[\bar{x}, \bar{y}]) \Rightarrow \neg L[\bar{x}]).$		BU
	$\forall \bar{x}(\exists \bar{y}\ L'[\bar{x}, \bar{y}]) \Rightarrow L[\bar{x}]).$		BU
	$\forall \bar{x} \forall \bar{y}\ (\neg L[\bar{x}] \Rightarrow \neg L'[\bar{x}, \bar{y}]).$		TD
	$\forall \bar{x} \forall \bar{y}\ ((L[\bar{x}] \wedge \forall \bar{z}\ (\bar{y} \neq \bar{z} \Rightarrow \neg L'[\bar{x}, \bar{y}][\bar{y}/\bar{z}])) \Rightarrow L'[\bar{x}, \bar{y}]).$		TD

Table 3.1: From ENF to INF

can derive from $Eq(A_\psi)$ a set of implications $\varphi_1 \Rightarrow \varphi_2$, such that each such propagation corresponds to deriving the consequent φ_2 from the antecedent φ_1 (so, different implications can be logically equivalent). This is defined in Table 3.1. The last column of this table indicates whether the rule is top-down (TD) or bottom-up (BU).

Definition 3.5 ($INF(\psi)$)**.** Given an equivalence $\varphi \in ENF(\psi)$ for a certain formula ψ, we denote with $Imp(\varphi)$ the set of all implications obtained through Table 3.1. We define the *implication normal form* of ψ, denoted by $INF(\psi)$, as follows: $INF(\psi) = \bigcup_{\varphi \in ENF(\psi)} Imp(\varphi)$.

Obviously, the transformation from a formula ψ to $INF(\psi)$ is linear in the size of ψ. In (Wittocx et al., 2010) it is proven that models of ψ and models of $INF(\psi)$ where A_ψ is true correspond, in the sense that the restriction of a model of $INF(\varphi) \wedge A_\psi$ to Σ is also a model of ψ, and vice versa, every model of ψ can be extended to a model of $INF(\psi) \wedge A_\psi$. These implications will form the core of our propagation method. While the propagation could be made more complete by adding more implications to $INF(\psi)$, the above definition tries to strike a balance between completeness and the ease of automatically deriving the implications.

Example 3.6. For each of the three formulas φ in $ENF(\psi)$ in Example 3.4, the following table shows the corresponding set of implications $Imp(\varphi)$. The complete theory $INF(\psi)$ consists of the union of these three sets.

$$\left\{\begin{array}{l}(\forall t\, A_2(t)) \Rightarrow A_1.\\ (\exists t\, \neg A_2(t)) \Rightarrow \neg A_1.\\ \forall t\,(A_1 \Rightarrow A_2(t)).\\ \forall t\,((\neg A_1 \wedge \forall t'\,(t \neq t' \Rightarrow A_2(t'))) \Rightarrow \neg A_2(t)).\end{array}\right\} \qquad (A_1 \Leftrightarrow \forall t\, A_2(t).)$$

$$\left\{\begin{array}{l}\forall t\,(\neg Clean\,(t+1) \wedge \neg A_3(t) \Rightarrow \neg A_2(t)).\\ \forall t\,(Clean\,(t+1) \Rightarrow A_2(t)).\\ \forall t\,(A_3(t) \Rightarrow A_2(t)).\\ \forall t\,(\neg A_2(t) \Rightarrow \neg Clean\,(t+1)).\\ \forall t\,(\neg A_2(t) \Rightarrow \neg A_3(t)).\\ \forall t\,(A_2(t) \wedge \neg A_3(t) \Rightarrow Clean\,(t+1)).\\ \forall t\,(A_2(t) \wedge \neg Clean\,(t+1) \Rightarrow A_3(t)).\end{array}\right\} \qquad \forall t\,(A_2(t) \Leftrightarrow Clean\,(t+1) \vee A_3(t)).$$

$$\left\{\begin{array}{l}\forall t\,(\neg Clean\,(t) \wedge \neg Wipe\,(t) \Rightarrow A_3(t)).\\ \forall t\,(Clean\,(t) \Rightarrow \neg A_3(t)).\\ \forall t\,(Wipe\,(t) \Rightarrow \neg A_3(t)).\\ \forall t\,(A_3(t) \Rightarrow \neg Clean\,(t)).\\ \forall t\,(A_3(t) \Rightarrow \neg Wipe\,(t)).\\ \forall t\,(\neg A_3(t) \wedge \neg Wipe\,(t) \Rightarrow Clean\,(t)).\\ \forall t\,(\neg A_3(t) \wedge \neg Clean\,(t) \Rightarrow Wipe\,(t)).\end{array}\right\} \qquad \forall t\,(A_3(t) \Leftrightarrow (\neg Clean\,(t) \wedge \neg Wipe\,(t))).$$

The reader can verify that the four implications representing the propagation in Example 3.2 all indeed belong to $INF(\psi)$.

The propagation process in Example 3.2 can now be described as a least-fixpoint computation, where we 'apply' the implications (i.e., infer the head when the body is already inferred), until we no longer can infer any new information. We will represent this fixpoint computation as an inductive definition in the syntax of FO(ID). However, there are two complications.

First, in some applications of this propagation method, we do not always need all of the implications in $INF(\psi)$. Indeed, there will typically be some subset $\sigma \subseteq \Sigma$ of symbols about which we already know all there is to know. For example, when using this propagation method to solve conformant planning problems (see Chapter 5), this will be the case for the action predicates, for which we will let a model expansion system guess a complete interpretation. The job of the propagation process is then to figure out the consequences of each particular guess. For this, the implications with a predicate from σ in their head are obviously not needed.

Second, the fixpoint computation not only needs to infer that atoms are true but also that they are false. However, the syntax of FO(ID) does not allow negative literals in the heads of rules. Therefore, our definition will not contain rules with the predicates P of the original vocabulary Σ in their head, but will instead use predicates P^{ct} and P^{cf} from the Σ^{tf}-vocabulary. Since we do not need rules with the fully known predicates σ in the head, we will only introduce these P^{ct} and P^{cf} predicates for those P that are in $\Sigma \setminus \sigma$. For a given formula φ, we therefore define φ^{ct}_{σ} as the formula φ^{ct} (see Definition 2.8) but with

P^{ct} replaced by P and P^{cf} by $\neg P$ for every predicate $P \in \sigma$.

Definition 3.7 ($Approx_\sigma(\psi)$)**.** For a formula ψ and $\sigma \subseteq \Sigma$, we define $Approx_\sigma(\psi)$ as the inductive definition that contains, for every sentence $\forall \bar{x}\ (\varphi \Rightarrow L[\bar{x}])$ of $INF(\psi)$ in which L is a literal of a predicate not in σ, the definitional rule $\forall \bar{x}(L[\bar{x}]^{\mathrm{ct}}_\sigma \leftarrow \varphi^{\mathrm{ct}}_\sigma)$. We also define $Approx^{BU}_\sigma(\psi)$ (and $Approx^{TD}_\sigma(\psi)$) in the same way as $Approx_\sigma(\psi)$, but only containing definitional rules coming from the *bottom-up* (respectively, *top-down*) rules of $INF(\psi)$.

We will sometimes be able to assume that $\sigma = \varnothing$. Whenever this is the case we drop the σ and use $Approx(\psi)$ rather than $Approx_\sigma(\psi)$, to denote the approximative definition.

Example 3.8. Using the implications $INF(\psi)$ of Example 3.6, we obtain the definition shown in Figure 3.3 for $Approx(\psi)$. If we take $\sigma = \{Wipe\}$, we get the same definition for $Approx_\sigma(\psi)$ as in Figure 3.3, apart from the last seven definitional rules that are replaced by the following five definitional rules.

$$\left\{\begin{array}{lcl} & \vdots & \\ A_3^{ct}(t) & \leftarrow & Clean^{\,cf}(t) \wedge \neg Wipe(t). \\ A_3^{cf}(t) & \leftarrow & Clean^{\,ct}(t). \\ A_3^{cf}(t) & \leftarrow & Wipe(t). \\ Clean^{\,cf}(t) & \leftarrow & A_3^{ct}(t). \\ Clean^{\,ct}(t) & \leftarrow & A_3^{cf}(t) \wedge \neg Wipe(t). \end{array}\right\}$$

In contrast with $Approx(\psi)$ this definition no longer approximates the predicate *Wipe*. The definition $Approx_\sigma(\psi)$ can be used to find out what certainly holds or not holds given a *two valued* interpretation for the predicates in σ.

Example 3.9. For a larger example, let T be the following theory:

$$T: \begin{array}{l} \forall t\,(Clean(t+1) \Leftrightarrow Clean(t) \vee Wipe(t)). \\ Clean(0) \Leftrightarrow InitiallyClean. \end{array}$$

Let us again take $\sigma = \{Wipe\}$. Then the definition $Approx_\sigma(T)$ can be found in Appendix A.

This approximative definition has some useful properties, which we formulate in the next two theorems. The first property is that, when using the approximative definition together with an encoding of a three-valued interpretation $\mathcal{I}$ of the original vocabulary, we can give an exact characterization of what the approximative definition computes. Indeed, in this setting, $Approx^{BU}(\psi)$ actually encodes the three-valued Kleene evaluation of ψ in $\mathcal{I}$. Moreover, adding the top-down rules does not change this, since they will not compute actually anything, as long as only information about the original vocabulary is provided as input. Before we can formally state this property, we need to define how we encode a

$$
\left\{
\begin{array}{lcl}
A_1^{ct} & \leftarrow & \forall t\, A_2^{ct}(t).\\
A_1^{cf} & \leftarrow & \exists t\, A_2^{cf}(t).\\
A_2^{ct}(t) & \leftarrow & A_1^{ct}.\\
A_2^{cf}(t) & \leftarrow & A_1^{cf} \wedge \forall t'\,(t \neq t' \Rightarrow A_2^{ct}(t')).\\
 & & \\
A_2^{cf}(t) & \leftarrow & Clean^{cf}(t+1) \wedge A_3^{cf}(t).\\
A_2^{ct}(t) & \leftarrow & Clean^{ct}(t+1).\\
A_2^{ct}(t) & \leftarrow & A_3^{ct}(t).\\
Clean^{cf}(t+1) & \leftarrow & A_2^{cf}(t).\\
A_3^{cf}(t) & \leftarrow & A_2^{cf}(t).\\
Clean^{ct}(t+1) & \leftarrow & A_2^{ct}(t) \wedge A_3^{cf}(t).\\
A_3^{ct}(t) & \leftarrow & A_2^{ct}(t) \wedge Clean^{cf}(t+1).\\
 & & \\
A_3^{ct}(t) & \leftarrow & Clean^{cf}(t) \wedge Wipe^{cf}(t).\\
A_3^{cf}(t) & \leftarrow & Clean^{ct}(t).\\
A_3^{cf}(t) & \leftarrow & Wipe^{ct}(t).\\
Clean^{cf}(t) & \leftarrow & A_3^{ct}(t).\\
Wipe^{cf}(t) & \leftarrow & A_3^{ct}(t).\\
Clean^{ct}(t) & \leftarrow & A_3^{cf}(t) \wedge Wipe^{cf}(t).\\
Wipe^{ct}(t) & \leftarrow & A_3^{cf}(t) \wedge Clean^{cf}(t).
\end{array}
\right\}
$$

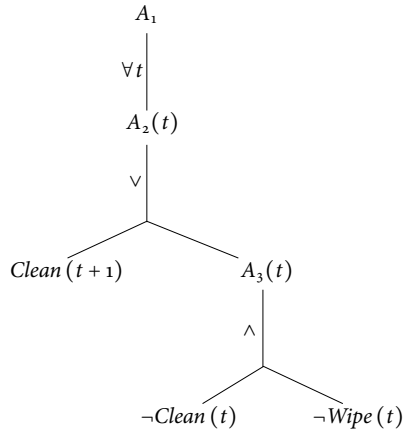

Figure 3.3: Example of an approximative definition

four-valued interpretation as a definition. In the rest of this chapter, we assume that for any vocabulary Σ and Σ-pre-interpretation I, Σ contains a constant symbol C_d for every domain element d in the domain D of I, and that for the pre-interpretation I it holds that $(C_d)^I = d$. This allows us to identify C_d and d and therefore, abusing notation, we will use d to denote C_d in what follows.

Definition 3.10. Given a four-valued Σ-interpretation $\mathcal{I}$, the *definition associated to* $\mathcal{I}$ is denoted by $\Delta_{\mathcal{I}}$ and is defined by

$$
\begin{array}{rcl}
\Delta_{\mathcal{I}} & = & \{P^{ct}(\bar{d}) \leftarrow \mathbf{t} | P^{\mathcal{I}}(\bar{d}) \geq_p \mathbf{t}\}\\
 & & \cup \{P^{cf}(\bar{d}) \leftarrow \mathbf{t} | P^{\mathcal{I}}(\bar{d}) \geq_p \mathbf{f}\}
\end{array}
$$

Theorem 3.11. *Given a Σ-formula ψ and a four-valued Σ-interpretation $\mathcal{I}$, the following holds:*

a) Let M be $Mod(Approx(\psi) \cup \Delta_{\mathcal{I}})$, v_1 the truth value of A_ψ^{ct} in M and v_2 be the truth value of $\neg A_\psi^{cf}$ in M. Similarly, let M' be $Mod(Approx^{BU}(\psi) \cup \Delta_{\mathcal{I}})$, v_1' the truth value of A_ψ^{ct} in M' and v_2 the truth value of $\neg A_\psi^{cf}$ in M'. Then $(v_1, v_2) = (v_1', v_2')$ and both correspond to the four-valued truth value $\psi^{\mathcal{I}}$, i.e., $\psi^{\mathcal{I}} = \tau(v_1, v_2)$.

b) In the case that $\mathcal{I}$ is three-valued it holds that $Approx(\psi) \cup \Delta_{\mathcal{I}}$ is logically equivalent to $Approx^{BU}(\psi) \cup \Delta_{\mathcal{I}}$, that is, $Mod(Approx(\psi) \cup \Delta_{\mathcal{I}}) = Mod(Approx^{BU}(\psi) \cup \Delta_{\mathcal{I}})$.

Proof. First, remark that Feferman (1984) showed that the four-valued evaluation of a formula φ in an interpretation $\mathcal{I}$ can be simulated by computing the standard two-valued evaluation of φ^{ct} and φ^{cf} in $\mathcal{I}^{tf}$. It is easy to verify that the bottom-up rules in $Approx(\psi)$ inductively encode this evaluation. We split the proof in two parts. First we assume that $\mathcal{I}$ is tree-valued. We show that in this case only the bottom-up rules are used, i.e., leaving out the top-down rules does not change the model of the definition. This proves the second part of the theorem, and together with the above remark also proves the first part of the theorem for the case that $\mathcal{I}$ is three-valued. Then, all that is left to prove, is that the first part of the theorem also holds for four-valued $\mathcal{I}$.

So let us assume that $\mathcal{I}$ is tree-valued. We prove that $Mod(Approx^{BU}(\psi) \cup \Delta_{\mathcal{I}}) = Mod(Approx(\psi) \cup \Delta_{\mathcal{I}})$ by contradiction. Assume there is a predicate A^{ct}_{φ} (the proof goes analogous for A^{cf}_{φ}) such that $Mod(Approx(\psi) \cup \Delta_{\mathcal{I}}) \vDash A^{ct}_{\varphi}$ but $Mod(Approx^{BU}(\psi) \cup \Delta_{\mathcal{I}}) \nvDash A^{ct}_{\varphi}$. In the preliminaries we recalled that the model of a positive inductive definition is the least-fixpoint of the immediate consequence operator T^{O}_{Δ}. The model of such a definition is thus the limit of a sequence of applications of the immediate consequence operator. One can prove (see e.g., (Denecker & Ternovska, 2004)) that we do not have to apply the immediate consequence operator of the *complete* definition in every step. I.e., applying the immediate consequence operator of a subset of the definition, until there no longer exists such a immediate consequence operator that will give something new, gives the same model. Suppose we take such a sequence where we first apply all the bottom-up rules, and only after no bottom-up rules are applicable we try to apply the top-down rules. Suppose A^{ct}_{φ} is the first atom we infer with the top down rules in this sequence. Obviously A^{ct}_{φ} cannot be the top-level atom A^{ct}_{ψ}, since there are no top-down rules for this. We can now do a case study on the type of (sub)formula φ occurs in, e.g., assume that φ is a subformula of the formula ψ where $\psi = \varphi \vee \varphi'$. From the fact that A^{ct}_{φ} is true, it follows that the body of the top down rule $A^{ct}_{\varphi} \leftarrow A^{ct}_{\psi} \wedge A^{cf}_{\varphi'}$ has to be true, and thus dat A^{ct}_{ψ} and $A^{cf}_{\varphi'}$ are true. Since A^{ct}_{φ} was the first atom to be inferred by a top-down rule, we have that since A^{ct}_{ψ} is true, also $A^{ct}_{\varphi} \vee A^{ct}_{\varphi'}$ must be true. Now since A^{ct}_{φ} only became true in the last step of the sequence, we have that $A^{ct}_{\varphi'}$ must have been true already. This means that after applying the bottom-up rules $A^{ct}_{\varphi'}$ and $A^{cf}_{\varphi'}$ are both true, which is a contradiction with the fact that $\mathcal{I}$ was tree-valued and that the bottom-up rules encode the four-valued evaluation. The proof is analogous for the other types of subformulas.

Now in the case that $\mathcal{I}$ is four-valued, and not three-valued it is no longer the case that only the bottom-up rules contribute to the model (i.e., $Mod(Approx^{BU}(\psi) \cup \Delta_{\mathcal{I}}) \neq Mod(Approx(\psi) \cup \Delta_{\mathcal{I}})$). To see this, consider the following formula $P \wedge Q$, and take for $\mathcal{I}$ the four-valued interpretation such that $P = \mathbf{i}$ and $Q = \mathbf{t}$. Then one can verify that the bottom-up rules in $Approx(\psi) \cup \Delta_{\mathcal{I}}$ will infer that both $A^{ct}_{P \wedge Q}$ and $A^{cf}_{P \wedge Q}$ are true. However, now the top-down rules can also infer that Q^{cf} has to be true. What happens is that once an inconsistency is inferred for a certain subformula, this propagates back down the parse-

tree. However, similar to above, we can again do a case study on the structure of ψ to prove that (for the 'top' formula ψ) $Mod(Approx^{BU}(\psi) \cup \Delta_{\mathcal{I}}) \vDash A^{ct}_{\psi} \wedge A^{cf}_{\psi}$ if and only if $Mod(Approx(\psi) \cup \Delta_{\mathcal{I}}) \vDash A^{ct}_{\psi} \wedge A^{cf}_{\psi}$. Now since since $Approx^{BU}(\psi)$ is clearly a direct encoding of the four-valued evaluation, this concludes the proof. □

In summary, what this theorem says is that, first of all, the approximation always computes the four-valued Belnap evaluation of ψ in the four-valued structure $\mathcal{I}$. Moreover, this computation is done by the bottom-up rules of the approximation alone. If $\mathcal{I}$ is three-valued, then the top-down rules actually have no effect at all. If $\mathcal{I}$ is four-valued, then they may still serve a purpose, however: once the bottom-up rules have derived that some subformula is inconsistent, they can then propagate this information to derive that smaller formulas are also inconsistent.

Recall that in the preliminaries we saw that in general for a formula ψ supervaluation and Kleene evaluation are not equal. However, for some formulas ψ they *are* equal. In the literature, several *classes of formulas* for which they agree have been proposed, e.g., in the context of locally closed databases (Denecker et al., 2010a), or in the context of reasoning with incomplete first-order knowledge (Liu & Levesque, 1998). The latter introduces a normal form $\mathcal{NF}$ for first-order formulas, for which the supervaluation coincides with the Kleene evaluation, and proves for certain classes of formulas that they are in the normal form $\mathcal{NF}$. Once such class is that of all CNF formulas in which every two literals are *conflict-free*: a pair of literals is conflict-free if they either have the same polarity, or they use different predicates, or they use different constants at some argument position.

In the theorem above the only information we add to the approximative definition is in the form of the definition $\Delta_{\mathcal{I}}$, i.e., we only assert the truth, resp. falsity of *domain atoms*. The following definition now allows us to assert the truth or the falsity of any grounded subformula $\varphi[\bar{d}]$.

Definition 3.12. Given a Σ-formula ψ, a Σ-pre-interpretation I, and a set Φ of formulas $(\neg)\varphi[\bar{d}]$, such that $\varphi[\bar{x}]$ is a subformula of ψ of arity n and $\bar{d} \in D^n$ where D is the domain of I, we then define Δ_{Φ} as follows:

$$\Delta_{\Phi} = \{A^{ct}_{\varphi}(\bar{d}) \leftarrow \mathbf{t} \mid \varphi[\bar{d}] \in \Phi\} \cup \{A^{cf}_{\varphi}(\bar{d}) \leftarrow \mathbf{t} \mid \neg\varphi[\bar{d}] \in \Phi\}.$$

If we assert in this way the truth (or the falsity) of a set of grounded subformulas Φ, then we will obtain an approximation of everything that holds (respectively, does not hold) in all models of Φ. However, as opposed to the theorem above, the next theorem does not give an exact characterization of the approximation we get.

Theorem 3.13. *Given a Σ-formula ψ, a set Φ as defined above and a subformula $\varphi'[\bar{x}']$. Let M be $Mod(Approx(\psi) \cup \Delta_{\Phi})$. If $M \vDash A^{ct}_{\varphi'}(\bar{d}')$ (resp. $A^{cf}_{\varphi'}(\bar{d}')$), then $\Phi \vDash \varphi'[\bar{d}']$ (resp. $\Phi \vDash \neg\varphi'[\bar{d}']$).*

Note that an interesting special case of this theorem is where we take Φ equal to $\{\psi\}$ and thus add $A^{ct}_{\psi} \leftarrow \mathbf{t}$ to $Approx(\psi)$. Then this definition gives an approximation of everything that is *certainly true* resp. *certainly false* in *all models* of ψ.

Unfortunately, in general it is difficult to characterize precisely how complete the method is. For instance, apart from the fact that the Kleene evaluation is incomplete, another source of loss in completeness comes from the fact that our current translation to ENF cannot recognize multiple occurrences of the same subformula, and will introduce a different Tseitin predicate for each occurrence.

Example 3.14. Let ψ be the formula $\psi_1 \Rightarrow \psi_2$, where we take $\psi_1 = P \Rightarrow R \vee Q$ and $\psi_2 = R \vee Q$, and let $\Phi = \{\psi_1, P\}$. Clearly the whole formula ψ is certainly true in this case. However, our method will not be able to infer this, since the transformation to ENF will introduce two new Tseitin predicates, one for the formula $R \vee Q$ in ψ_1 and one for $R \vee Q$ in ψ_2. The propagation will infer that ψ_1 has to be certainly true, but obviously it cannot infer anything about R or Q, and since the only way to infer that ψ_2 has to be certainly true is via R or Q, the propagation will not find out that ψ_2 has to be certainly true, and thus that ψ is certainly true. A smarter translation to ENF could recognize that the subformula $R \vee Q$ occurs multiple times and substitute all of these occurrences by the same auxiliary predicate.

Returning now to the exact relationship between the work in (Wittocx et al., 2010) and the contents of this section, we see that Wittocx et al. are only interested in this special case, i.e., in approximating all models of a theory. For this reason their transformation from a formula ψ to $ENF(\psi)$ already includes a formula $A_{\psi} \Leftrightarrow \mathbf{t}$, which will cause the rule $A^{ct}_{\psi} \leftarrow \mathbf{t}$ to always be included in the approximating definition. All their soundness results have also been formulated and proven in this setting. However, it is not difficult to see that the proofs can be trivially adapted to the more general setting used in this section.

3.2 Propagation for FO(ID)

In this section we extend the method from the previous section to FO(ID) theories. That is, we would like a way to transform an FO(ID) theory T into a definition that is a syntactic representation of sound inferences that can be made from T in a three-valued context. To improve readability we will assume without loss of generality that an FO(ID) theory only contains a single definition, and thus is of the form $\Delta \wedge \varphi$, where φ is a pure FO formula. We present three ways to obtain such an approximation. A first step is of course to find a sound approximation of the inferences that are sanctioned by the definition Δ.

Using the completion of a definition

Our first approach is based on the fact that a model of a definition Δ is also a model of its FO completion $compl(\Delta)$ (Clark, 1978). The completion of a definition Δ is the conjunction of the equivalences $\forall \bar{x}\, P(\bar{x}) \Leftrightarrow \varphi_P(\bar{x})$ for all predicates $P \in Def(\Delta)$, where $\varphi_P(\bar{x})$ is the body of the unique rule with P in its head. A useful property is that each definition Δ implies its completion $compl(\Delta)$. Moreover, if Δ is non-recursive, then the two are actually equivalent. Instead of directly approximating $\Delta \wedge \varphi$, we can therefore just approximate the FO formula $\gamma = compl(\Delta) \wedge \varphi$. Since every model of $\Delta \wedge \varphi$ is also a model of $compl(\Delta) \wedge \varphi$, an approximation of the set of all models of $compl(\Delta) \wedge \varphi$ is also an approximation of the set of all models of $\Delta \wedge \varphi$. Therefore, it directly follows that $Approx_\sigma(\gamma) \cup \{A^{ct}_\gamma \leftarrow \mathbf{t}\}$ is a sound approximation of all models of $\Delta \wedge \varphi$

The disadvantage of using the completion is that it will never be able to infer something that follows from Δ but not from $compl(\Delta)$. For instance, the inductive definition $\{P \leftarrow P\}$ entails $\neg P$, but its completion $P \Leftrightarrow P$ does not.

In (Denecker & Ternovska, 2008) it is proven that, in addition to non-recursive definitions, a particular class of recursive definitions *are* equivalent to their completion. In particular, this is the case for definitions over a strict *well-founded order* $<$[1]. We can therefore replace those definitions by their completion without losing precision.

Using a certainly true/possibly true approximation

In this section, we present a more complete approximation for an inductive definition Δ. The idea is to construct an approximative definition Δ' whose two-valued well-founded model encodes the potentially four-valued well-founded model of the original definition Δ, given a potentially four-valued interpretation for the predicates of Δ. We will therefore again represent a four-valued interpretation of the orginal vocabulary Σ of Δ by a two-valued interpretation of a larger vocabulary Σ'. However, instead of introducing, for each predicate P of Σ, a predicate P^{ct} (P is certainly true) and P^{cf} (P is certainly false), as we did before, we will now introduce predicates P^{ct} and P^{pt} (P is possibly true, i.e., P is *not* certainly false). Let $\Sigma^{ct/pt}$ denote the vocabulary $\Sigma^F \cup \{P^{ct} \mid P \in \Sigma\} \cup \{P^{pt} \mid P \in \Sigma\}$. For a four-valued Σ-interpretation $\mathcal{I}$, we define the corresponding $\Sigma^{ct/pt}$-interpretation $\mathcal{I}^{ct/pt}$ as the interpretation with the same pre-interpretation as $\mathcal{I}$ such that $(P^{ct})^{\mathcal{I}^{ct/pt}} = \{\bar{d} \mid P^{\mathcal{I}}(\bar{d}) \geq_p \mathbf{t}\}$ and $(P^{pt})^{\mathcal{I}^{ct/pt}} = \{\bar{d} \mid (P^{\mathcal{I}}(\bar{d}) \leq_p \mathbf{t})\}$.

Also, for a Σ-formula φ, we define the formula $\varphi^{ct/pt}$ as the formula that we obtain after replacing all positive occurrences of a predicate P in φ by P^{ct}, and all negative occurrences by P^{pt}, and finally reducing to negational normal form. It is easy to see that $\varphi^{ct/pt}$ can also be

[1] An order $<$ is well-founded if there are no infinite descending chains $\ldots < x_n < x_{n-1} < \ldots < x_1$.

obtained from φ^{ct} by replacing, for every predicate P, all occurrences of P^{cf} by $\neg P^{pt}$. Unlike φ^{ct}, $\varphi^{ct/pt}$ is therefore no longer a positive formula, i.e., it contains negations. In particular, P^{ct} occurs only positively while P^{pt} occurs only negatively. For a subvocabulary $\sigma \subseteq \Sigma$, $\varphi_\sigma^{ct/pt}$ again denotes $\varphi^{ct/pt}$ but with both P^{ct} and P^{pt} replaced by P for every predicate $P \in \sigma$. Again, in what follows we will use σ to denote predicates that do not need to be approximated because we have two-valued information about them.

Definition 3.15. For a definition Δ, we define $App_\sigma^{ct/pt}(\Delta)$ as the definition $\{\mathcal{R}^{ct} \cup \mathcal{R}^{pt}\}$ where $\mathcal{R}^{ct}$ consists of the rules

$$\forall \bar{x}(P^{ct}(\bar{x}) \leftarrow \varphi_\sigma^{ct/pt})$$

and $\mathcal{R}^{pt}$ consists of the rules

$$\forall \bar{x}(P^{pt}(\bar{x}) \leftarrow \neg(\neg\varphi)_\sigma^{ct/pt})$$

for every definitional rule $\forall \bar{x}\ (P(\bar{x}) \leftarrow \varphi)$ in Δ.

Note that $\neg(\neg\varphi)^{ct/pt}$ actually denotes the formula φ where every positively occurring atom P is replaced by P^{pt} and every negatively occurring atom P by P^{ct}. It is easy to see that this gives us an overestimation of φ. As before, whenever we assume that σ is empty, we drop it in the notation of $App_\sigma^{ct/pt}$.

Example 3.16. Consider the following inductive definition Δ.

$$\left\{ \begin{array}{lcl} B & \leftarrow & B \vee \neg A. \\ A & \leftarrow & \neg D. \end{array} \right\}$$

Assume $\sigma = \{\}$. Then

$$App^{ct/pt}(\Delta) = \left\{ \begin{array}{lcl} B^{ct} & \leftarrow & B^{ct} \vee \neg A^{pt}. \\ A^{ct} & \leftarrow & \neg D^{pt}. \\ B^{pt} & \leftarrow & B^{pt} \vee \neg A^{ct}. \\ A^{pt} & \leftarrow & \neg D^{ct}. \end{array} \right\}.$$

We see that for a three-valued interpretation $\{D = \mathbf{u}\}$, which translates to $\{D^{ct} = \mathbf{f}, D^{pt} = \mathbf{t}\}$, the approximative definition will correctly infer that B^{ct} is false and B^{pt} is true. If we take $\{D = \mathbf{f}\}$ as an interpretation for the open predicate D, we see that the approximative definition correctly infers that both B^{ct} and B^{pt} are false. This is an example of *unfounded set* reasoning: once A is known to be true, the approximation detects that B could only be derived from B itself and therefore must be false. This kind of reasoning could not be done by our previous, completion-based approximation method, since it is only sound w.r.t. the semantics of the definition itself, and not w.r.t. its weaker completion.

This example also demonstrates why we have to use the vocabulary $\Sigma^{ct/pt}$ instead of $\Sigma^{ct/cf}$, since the latter would have yielded a definition:

$$\left\{ \begin{array}{lcl} B^{ct} & \leftarrow & B^{ct} \vee A^{cf}. \\ A^{ct} & \leftarrow & D^{cf}. \\ B^{cf} & \leftarrow & B^{cf} \wedge A^{ct}. \\ A^{cf} & \leftarrow & D^{ct}. \end{array} \right\}.$$

Because of the loop, for $\{D = \mathbf{f}\}$ these rules can never derive the truth of B^{cf}, while $App^{ct/pt}$ can derive that B^{pt} is false (and hence that B^{cf} is true) for $\{D = \mathbf{f}\}$.

Each two-valued interpretation of the "double" vocabulary $\Sigma^{ct/pt}$ corresponds to a four-valued interpretation of the original vocabulary Σ. We again want to establish a link between the well-founded model of the original definition Δ and that of $App_\sigma^{ct/pt}(\Delta)$. A complicating factor here is that, as the above example shows, the definition $App_\sigma^{ct/pt}(\Delta)$ is no longer monotone and therefore it is no longer guaranteed to have a two-valued well-founded model. Because a three-valued interpretation of $\Sigma^{ct/pt}(\Delta)$ no longer corresponds to even a four-valued interpretation of the original vocabulary Σ, we can only prove an exact correspondence if the well-founded model of $App_\sigma^{ct/pt}(\Delta)$ is two-valued.

Theorem 3.17. *Let $\mathcal{O}$ be a four-valued interpretation for the open predicates of a definition Δ. $App^{ct/pt}(\Delta)$ has a two-valued well-founded model given $\mathcal{O}^{ct/pt}$ if and only if Δ has a unique four-valued stable fixpoint given $\mathcal{O}$. Moreover, if $App^{ct/pt}(\Delta)$ has a two-valued well-founded model I, then the unique four-valued stable fixpoint of Δ is the unique interpretation $\mathcal{I}$ for which $\mathcal{I}^{ct/pt} = I$.*

A key ingredient in the proof of Theorem 3.17 is the following property of $App^{ct/pt}(\Delta)$. Its immediate consequence operator $T^{O^{ct/pt}}_{App^{ct/pt}(\Delta)}$ on two-valued interpretations simulates the immediate consequence operator $\mathcal{T}^{(O_1,O_2)}_\Delta$ on four-valued interpretations of the original definition. This is made more precise in the following lemma.

Definition 3.18. For a pair of Σ-interpretations (I, J), we use $t(I, J)$ to denote the $\Sigma^{ct/pt}$-interpretation $(I, J)^{ct/pt}$.

Lemma 3.19. *For each (O_1, O_2) and (I, J),*

$$\mathcal{T}^{(O_1,O_2)}_\Delta(I, J) = t^{-1}(T^{t(O_1,O_2)}_{App^{ct/pt}(\Delta)}(t(I, J))).$$

Proof. Let (I', J') be $\mathcal{T}^{(O_1,O_2)}_\Delta(I, J)$ and let $F = T^{t(O_1,O_2)}_{App^{ct/pt}(\Delta)}(t(I, J))$. We first show that $F|_{\Sigma^{ct}} = I'$. Since $F|_{\Sigma^{ct}}$ depends only on the rules of $App^{ct/pt}(\Delta)$ with a predicate from Σ^{ct}, we can discard all rules with a head from Σ^{pt}. As a result, we are left with a single copy of Δ in which positive occurrences of atoms have been replaced by their $\cdot^{ct}$ variant and negative

ones by their $\cdot^{pt}$ variant. This implies that the evaluation of the bodies of these remaining rules according to $t(O_1 \cup I, O_2 \cup J)$ will be identical to the evaluation of the bodies of the original rules by $(O_1 \cup I, O_2 \cup J)$ in the construction of I', thus proving the equality. The proof of the remaining equality $F|_{\Sigma^{pt}} = J'$ is analoguous. □

Proof of Theorem 3.17. First, recall that, given some partial knowledge (O_1, O_2), the four-valued well-founded models of Δ, resp. $App^{ct/pt}(\Delta)$ are the least fixpoints of operators $\mathcal{ST}_{\Delta}^{(O_1,O_2)}$ resp. $\mathcal{ST}_{App^{ct/pt}(\Delta)}^{t(O_1,O_2)}$ (note that since $t(O_1, O_2)$ is two-valued, we abuse notation here and in the rest of the proof and denote the two-valued pair $(t(O_1, O_2), t(O_1, O_2))$ by $t(O_1, O_2)$).

Now, the latter operator is rather peculiar, in the sense that it is actually juggling four different interpretations of the original alphabet Σ. In more detail, each element in its domain looks like this:

$$(I, J) = \begin{pmatrix} I_{ct} & & J_{ct} \\ \cup & , & \cup \\ I_{pt} & & J_{pt} \end{pmatrix}.$$

where I_{ct} and J_{ct} interpret the alphabet Σ^{ct}, and I_{pt} and J_{pt} interpret Σ^{pt}. If we now apply the operator $\mathcal{ST}_{App^{ct/pt}(\Delta)}^{t(O_1,O_2)}$, we obtain a new such pair:

$$(I', J') = \begin{pmatrix} I'_{ct} & & J'_{ct} \\ \cup & , & \cup \\ I'_{pt} & & J'_{pt} \end{pmatrix}.$$

From the general definition of the $\mathcal{ST}_{\Delta}^{\mathcal{O}}$ construction, it is obvious that $I'_{ct} \cup I'_{pt}$ depends only on $J_{ct} \cup J_{pt}$. However, in this particular case, the operator exhibits even more structure. The operator $ST_{App(\Delta)}^{(O_1,O_2)}(J)$ uses its argument J as a *fixed* interpretation for the negative occurrences, which remains constant throughout the least fixpoint computation over the positive occurrences. Now, $App^{ct/pt}(\Delta)$ contains two copies of Δ which interact only through negative occurrences (that is, all occurrences of a $\cdot^{pt}$ predicate in the body of a rule with a $\cdot^{ct}$ predicate in its head are always negative ones, and vice versa). This means that as long as we keep the interpretation of the negative occurrence fixed to a constant value J, these two copies of Δ do not interact at all. Consequently, to construct I'_{ct}, we can discard all rules with a $\cdot^{pt}$ predicate in its head. This means we are left with rules whose head is a $\cdot^{ct}$ predicate and whose body contains only positive occurrences of $\cdot^{ct}$ predicate and negative occurrences of $\cdot^{pt}$ predicates. Therefore, I'_{ct} depends only on J_{pt}. Moreover, the value of I'_{ct} will be such that if we map its symbols back to the original alphabet Σ (let $orig(\sigma^{ct}) = orig(\sigma^{pt}) = \sigma$ for all $\sigma \in \Sigma$), then $orig(I'_{ct}) = ST_{\Delta}^{(O_1,O_2)}(orig(J_{pt}))$. Similarly,

we also obtain that:

$$orig(I'_{pt}) = ST_{\Delta}^{(O_1,O_2)}(orig(J_{ct})),$$

$$orig(J'_{ct}) = ST_{\Delta}^{(O_1,O_2)}(orig(I_{pt})),$$

$$orig(J'_{pt}) = ST_{\Delta}^{(O_1,O_2)}(orig(I_{ct})).$$

In other words,

$$(orig(I'_{ct}), orig(J'_{pt})) = \mathcal{ST}_{\Delta}^{(O_1,O_2)}(orig(I_{ct}), orig(J_{pt})),$$

$$(orig(I'_{pt}), orig(J'_{ct})) = \mathcal{ST}_{\Delta}^{(O_1,O_2)}(orig(I_{pt}), orig(J_{ct})).$$

Now, if we consider the construction of the well-founded model of $App^{ct/pt}(\Delta)$, we have a sequence of this form:

$$\begin{pmatrix} I^0_{ct} & , & J^0_{ct} \\ \cup & & \cup \\ I^0_{pt} & & J^0_{pt} \end{pmatrix} \mapsto \begin{pmatrix} I^1_{ct} & , & J^1_{ct} \\ \cup & & \cup \\ I^1_{pt} & & J^1_{pt} \end{pmatrix} \mapsto \begin{pmatrix} I^2_{ct} & , & J^2_{ct} \\ \cup & & \cup \\ I^2_{pt} & & J^2_{pt} \end{pmatrix} \mapsto \cdots \mapsto \begin{pmatrix} I^\infty_{ct} & , & J^\infty_{ct} \\ \cup & & \cup \\ I^\infty_{pt} & & J^\infty_{pt} \end{pmatrix}.$$

Let $(I^i, J^i)_{i\geq 0}$ be the well-founded model construction for the original definition Δ, i.e., for all predicates $P/n \in Def(\Delta)$ and domain tuples $\bar{d} \in D^n$ we have that $(P(\bar{d}))^{I^0} = \mathbf{f}$ and $(P(\bar{d}))^{J^0} = \mathbf{t}$, and $(I^{n+1}, J^{n+1}) = \mathcal{ST}_{\Delta}^{(O_1,O_2)}(I^n, J^n)$ for $n \geq 0$. It is easy to see that we have that $t^{-1}(I^0_{ct} \cup J^0_{pt}) = (I_0, J_0)$. This provides a base case, while the above equation provide the inductive step to prove that, for each i, $(I^i, J^i) = t^{-1}(I^i_{ct} \cup J^i_{pt})$. In other words, the well-founded model construction for the original definition Δ is tracked by these elements of the well-founded model construction for $App^{ct/pt}(\Delta)$:

$$\begin{pmatrix} I^0_{ct} & , & \cdot \\ \cup & & \cup \\ \cdot & & J^0_{pt} \end{pmatrix} \mapsto \begin{pmatrix} I^1_{ct} & , & \cdot \\ \cup & & \cup \\ \cdot & & J^1_{pt} \end{pmatrix} \mapsto \begin{pmatrix} I^2_{ct} & , & \cdot \\ \cup & & \cup \\ \cdot & & J^2_{pt} \end{pmatrix} \mapsto \cdots \mapsto \begin{pmatrix} I^\infty_{ct} & , & \cdot \\ \cup & & \cup \\ \cdot & & J^\infty_{pt} \end{pmatrix}.$$

What about the other diagonal? There we have that $t^{-1}(J^0_{ct} \cup I^0_{pt}) = (\top, \bot)$, which is the most precise element $\top_p$ in the lattice of pairs of interpretations. Therefore, we find that the other diagonal actually tracks the construction of the *greatest* fixpoint of $\mathcal{ST}_{\Delta}^{(O_1,O_2)}$. Combining these two results, we see that if (L_1, L_2) and (G_1, G_2) are these least and greatest fixpoint, respectively, the well-founded model of $App^{ct/pt}(\Delta)$ looks like this:

$$\begin{pmatrix} L_1 & , & G_1 \\ \cup & & \cup \\ G_2 & & L_2 \end{pmatrix}.$$

Note that a unique three-valued stable fixpoint is both the least and greatest stable fixpoint and hence also the well-founded fixpoint. This immediately concludes the proof.

□

In order for $App^{ct/pt}(\Delta)$ to have a guaranteed two-valued well-founded model, Theorem 3.17 requires that Δ has a unique four-valued stable model for each four-valued input interpretation $\mathcal{O}$. This is a stronger requirement than the more common condition of *totality*, which only requires a definition to have a two-valued well-founded model given a *two-valued* input interpretation O. As the following example shows, this stronger condition is indeed necessary.

Example 3.20. Consider the following definition:

$$\left\{\begin{array}{l} A \leftarrow \neg B. \\ \\ B \leftarrow \neg A \wedge C. \\ \\ C \leftarrow O \wedge \neg O. \end{array}\right\}$$

This definition is total, because, for each two-valued interpretation for the open predicate O, it has $(\{A\},\{A\})$ as its two-valued well-founded model. However, for the three-valued interpretation $(\{\},\{O\})$ (i.e., "O is unknown") for the open predicate O, the three-valued well-founded model $(\{\},\{O,A,B,C\})$ is *not* the unique three-valued stable fixpoint, since $(\{A\},\{O,A,C\})$ is also such a fixpoint. And indeed, we find that

$$App^{ct/pt}(\Delta) = \left\{\begin{array}{l} A^{ct} \leftarrow \neg B^{pt}. \\ \\ B^{ct} \leftarrow \neg A^{pt} \wedge C^{ct}. \\ \\ C^{ct} \leftarrow O^{ct} \wedge \neg O^{pt}. \\ \\ A^{pt} \leftarrow \neg B^{ct}. \\ \\ B^{pt} \leftarrow \neg A^{ct} \wedge C^{pt}. \\ \\ C^{pt} \leftarrow O^{pt} \wedge \neg O^{ct}. \end{array}\right\}$$

does not have a two-valued well-founded model given $\{O^{pt}\}$. The easiest way to see this is to fill in the fact that we know that O^{pt} is $\mathbf{t}$ and O^{ct} is $\mathbf{f}$ and propagate this information:

$$\left\{\begin{array}{l} A^{ct} \leftarrow \neg B^{pt}. \\ \\ B^{ct} \leftarrow \neg A^{pt} \wedge C^{ct}. \\ \\ C^{ct} \leftarrow \mathbf{f} \wedge \neg \mathbf{t}. \\ \\ A^{pt} \leftarrow \neg B^{ct}. \\ \\ B^{pt} \leftarrow \neg A^{ct} \wedge C^{pt}. \\ \\ C^{pt} \leftarrow \mathbf{t} \wedge \neg \mathbf{f}. \end{array}\right\} \rightsquigarrow \left\{\begin{array}{l} A^{ct} \leftarrow \neg B^{pt}. \\ \\ B^{ct} \leftarrow \neg A^{pt} \wedge \mathbf{f}. \\ \\ \\ A^{pt} \leftarrow \neg B^{ct}. \\ \\ B^{pt} \leftarrow \neg A^{ct} \wedge \mathbf{t}. \\ \\ C^{pt} \leftarrow \end{array}\right\} \rightsquigarrow \left\{\begin{array}{l} A^{ct} \leftarrow \neg B^{pt}. \\ \\ \\ \\ \\ A^{pt} \leftarrow \neg \mathbf{f}. \\ \\ B^{pt} \leftarrow \neg A^{ct}. \\ \\ C^{pt} \leftarrow \end{array}\right\} \rightsquigarrow \left\{\begin{array}{l} A^{ct} \leftarrow \neg B^{pt}. \\ \\ \\ \\ \\ A^{pt} \leftarrow \\ \\ B^{pt} \leftarrow \neg A^{ct}. \\ \\ C^{pt} \leftarrow \end{array}\right\}$$

So, we are left with a loop over negation, which means that A^{ct} and B^{pt} will remain unknown in the three-valued well-founded model $(\{A^{pt}, C^{pt}\}, \{A^{ct}, A^{pt}, C^{pt}, B^{pt}\})$ of the definition.

By computing the three-valued well-founded model of Δ given $\mathcal{O}$, the approximative definition $App^{ct/pt}(\Delta)$ can produce more precise results than the approximation of $compl(\Delta)$; in particular it can detect that atoms in an unfounded set must be false, as illustrated in Example 3.16 above.

To use $App^{ct/pt}(\Delta)$ in an approximation of an FO(ID) formula $\Delta \wedge \varphi$, we still need to show how it can be combined with the approximation $Approx_\sigma(\varphi)$ of φ to produce a sound approximation of $\Delta \wedge \varphi$. To do this, we need to combine one definition of ct/cf-predicates with another definition of ct/pt-predicates. We achieve this by first merging the two definitions and then adding rules that copy information from the one vocabulary to the other.

Definition 3.21. Given a vocabulary Σ, a subvocabulary $\sigma \subset \Sigma$, an inductive definition Δ and first-order formula φ. We define $\mathcal{D}_\sigma^{\Delta\wedge\varphi}$ as the following inductive definition.

$$
\begin{gathered}
App_\sigma^{ct/pt}(\Delta) \\
\cup \\
Approx_\sigma(\varphi) \cup \{A_\varphi^{ct} \leftarrow \mathbf{t}\} \\
\cup \\
\{O^{pt} \leftarrow \neg O^{cf} \mid \text{for every predicate } O \in Open(\Delta) \setminus \sigma\} \\
\cup \\
\{P^{cf} \leftarrow \neg P^{pt} \mid \text{for every predicate } P \in Def(\Delta) \setminus \sigma\} \\
\cup \\
\{O^{cf} \leftarrow \mathbf{f}, O^{ct} \leftarrow \mathbf{f} \mid \text{for every predicate } O \in Open(\Delta) \setminus \sigma \text{ that does not occur in } \varphi\}
\end{gathered}
$$

This definition indeed contains both the rules from the approximation of Δ and the rules from the approximation of φ, that is, $App^{ct/pt}(\Delta)$ and $Approx(\varphi) \cup \{A_\varphi^{ct} \leftarrow \mathbf{t}\}$, respectively, but also a number of extra rules that make a connection between these two approximations. To approximate a defined predicate Q the approximation of Δ uses the pair of predicates Q^{ct} and Q^{pt} while the approximation of φ uses Q^{ct} and Q^{cf}. Hence, a number of extra rules are needed to transfer information between the predicates Q^{pt} and Q^{cf}. The rules $\{O^{pt} \leftarrow \neg O^{cf}\}$ transfer information that the approximation of φ has derived about the truth of an open predicate O (by means of O^{cf}) to the corresponding predicate O^{pt} of the approximation of the definition. The rules $\{P^{cf} \leftarrow \neg P^{pt}\}$ in turn propagate information derived about the truth of a defined predicate in the approximation of the definition to the corresponding predicate P^{cf} of the approximation of φ. Finally, the rules $\{O^{cf} \leftarrow \mathbf{f}, O^{ct} \leftarrow \mathbf{f}\}$ make sure that O^{cf} and O^{ct} are defined atoms (instead of open ones) and that their default value is **u**. The following proposition states that $\mathcal{D}_\sigma^{\Delta\wedge\varphi}$ gives indeed a sound approximation of all models of $\Delta \wedge \varphi$.

Proposition 3.22. *Given a vocabulary Σ, a subvocabulary σ and an FO(ID) formula $\Delta \wedge \varphi$. Then, for every σ-interpretation I, if $WFM_I(\mathcal{D}_\sigma^{\Delta\wedge\varphi}) \vDash P^{ct}(\bar{d})$ (resp. $P^{cf}(\bar{d})$), then it holds for every model M of $\Delta \wedge \varphi$ extending I that $M \vDash P(\bar{d})$ (resp. $M \vDash \neg P(\bar{d})$).*

In order to prove this proposition, we first need the following lemma.

Lemma 3.23. *Given a Σ-definition Δ, $\sigma \subset \Sigma$ a subset of Σ, and an FO-formula ψ. Let Σ', $\sigma' \subseteq \Sigma'$ and Δ' be renamed copies of Σ, resp. σ and Δ. Then define $\mathcal{D}$ as the following definition.*

$$\begin{aligned}\mathcal{D} = App^{ct/pt}(\Delta) \cup \{P^{ct} \leftarrow P'^{ct}\}_{P\in\sigma'} \cup \{O^{pt} \leftarrow O'^{pt}\}_{O\in\sigma'\cap Open(\Delta')} \\ \cup \{O^{ct} \leftarrow \mathbf{f}.\ O^{pt} \leftarrow \mathbf{f}\}_{O\in Open(\Delta')\setminus\sigma'}.\end{aligned}$$

Assume that we have a three-valued Σ interpretation $\mathcal{I}$ that approximates all models of $\Delta \wedge \psi$. Let $\mathcal{I}'$ be the Σ'-interpretation that interprets Σ' corresponding to how $\mathcal{I}$ interprets Σ. Then it holds that $WFM_{\mathcal{I}'^{ct/pt}|_{\sigma'}}(\mathcal{D})$ also approximates $\Delta \wedge \psi$, i.e.,

- *if $P^{ct}(\bar{d}) \in WFM_{\mathcal{I}'^{ct/pt}|_{\sigma'}}(\mathcal{D})$, then $\vDash (\Delta \wedge \psi) \Rightarrow P(\bar{d})$,*
- *if $P^{pt}(\bar{d}) \notin WFM_{\mathcal{I}'^{ct/pt}|_{\sigma'}}(\mathcal{D})$, then $\vDash (\Delta \wedge \psi) \Rightarrow \neg P(\bar{d})$.*

Note that, slightly different from how we use σ elsewhere in this chapter, in the above lemma, σ denotes a set of predicates about which we have extra (three-valued) information, for example, coming from approximating a first order theory, which not only soundly approximates all models of Δ but also of ψ. Now the lemma states that if we 'plug in' this extra information into the definition $App^{ct/pt}$, we again get a sound approximation of $\Delta \wedge \psi$. Also note that $Open(\mathcal{D})$ is indeed σ'.

Proof. For this proof, we assume without loss of generality that $\sigma = \Sigma$. We prove this through induction over the construction of the well-founded model. However, we use here an alternative construction method, which we will not go in detail about here, but can be found in (Denecker & Vennekens, 2007). Assume we have an induction sequence $(\mathcal{I}_i)_{0\leq i}$, of four-valued $Def(\mathcal{D})$-interpretations such that $\mathcal{I}_0$ is the interpretation in which everything is completely unknown, and $\mathcal{I}_{wfm}$ is the limit of the sequence, equal to the well-founded model $WFM_{\mathcal{I}'^{ct/pt}|_{\sigma'}}(\mathcal{D})$. We prove for every i that $\mathcal{I}_i$ is a sound approximation of $\Delta \wedge \psi$. This is trivially the case for $n = 0$. So now assume that $\mathcal{I}_k$ is a sound approximation of $\Delta \wedge \psi$. We have to prove that this is also the case for $\mathcal{I}_{k+1}$. We need to prove two things for this: first, an atom $P^{ct}(\bar{d})$ cannot be true in $\mathcal{I}_{k+1}$ if $P(\bar{d})$ is not true in all models of $\Delta \wedge \psi$, and second, that an atom $P^{pt}(\bar{d})$ cannot be false in $\mathcal{I}_{k+1}$ if it is not false in all models of $\Delta \wedge \psi$. We prove both cases by contradiction.

We start with the first case. So assume that in the k-th well-founded induction step, for a certain predicate P and domain tuple $\bar{D}$, $P^{ct}(\bar{d})$ is incorrectly deduced, i.e., there exists a model $M \vDash \Delta \wedge \psi$, s.t. $M \nvDash P(\bar{d})$. Since $P^{ct}(\bar{d})$ was inferred in the k-th step, this means

that the body of the rule defining $P^{ct}(\bar{d})$, that is, $(\varphi_P)^{ct}[\bar{d}]$ was already made true in a previous step. The induction hypothesis then tells us that in every model M of $\Delta \wedge \psi$ it holds that $M \models \varphi_P[\bar{d}]$, but according to the semantics of inductive definitions for every such M obviously also $M \models P$, which is a contradiction with our assumption.

Next, we consider the second case. This time, assume that in the k-th well-founded induction step, $P^{pt}(\bar{d})$ is inferred to be false, while there exists a model M of $\Delta \wedge \psi$, s.t. $M \models P(\bar{d})$. Now, using this alternative version of the well-founded semantics, there are two ways for a domain atom to become false. Indeed, a domain atom can become false because the body of the defining rule was already false, or because it is part of an *unfounded set*.

If $P^{pt}(\bar{d})$ was made false because the body of the defining rule $(\varphi_P)^{pt}$ was already false, an argument completely analogous to the one above - using the induction hypothesis - again gives a contradiction with the assumption. So now, all that is left to prove, is that $P^{pt}(\bar{d})$ cannot be incorrectly made false through the application of an unfounded set rule. If $P^{pt}(\bar{d})$ is made false through the application of the unfounded set rule, this means that there is a set US of atoms, that are unknown in $\mathcal{I}_k$, such that when made false in the bodies of the rules defining these atoms, the Kleene evaluation of these bodies returns false. It is possible to verify that we can always find an US such that it only contains $\cdot^{pt}$-atoms.

Now let us take such a model M of $\Delta \wedge \psi$ such that $M \models P(\bar{d})$, while $WFM_{\mathcal{I}'^{ct/pt}|_{\sigma'}}(\mathcal{D}) \not\models P^{pt}(\bar{d})$. This M is obviously also a model of Δ. Consider the corresponding set US', consisting of domain atoms $P(\bar{d})$, such that $P^{pt}(\bar{d}) \in US$. For every atom $Q^{pt}[\bar{d}]$ in the body $(\varphi_P)^{pt}$ not in the set US, the induction hypothesis actually tells us that $(Q[\bar{d}])^M \leq_t (Q^{pt}[\bar{d}])^{\mathcal{I}_k}$. Similarly, for every atom Q^{ct} in the body of $(\varphi_P)^{pt}$, it says that $(Q^{ct}[\bar{d}])^{\mathcal{I}_k} \leq_t (Q[\bar{d}])^M$. Now, since Q^{pt} atoms occur only positively and and Q^{ct} only negatively in $(\varphi_P)^{pt}$, it follows that M interprets literals that are not in US' in a more 'false' way than $\mathcal{I}_k$. Thus, US' is also an unfounded set (indeed, turning all the atoms in US' to false will make all the bodies of the defining rules false), and thus $M \models \neg P(\bar{d})$, which is again a contradiction with the assumption, and which concludes the proof of this lemma. □

Proof of Proposition 3.22. The proof of this proposition is now an easy proof over induction on the construction of the well-founded model of $\mathcal{D}^{\Delta\wedge\psi}$, using the lemma above, and the soundness of $Approx(\psi)$. □

Proposition 3.22 is the analogue for inductive definitions of the result that Theorem 3.13 states for FO formulas. One difference between the two results is that the above proposition always assumes that the definition Δ itself holds, while Theorem 3.13 makes no such assumption about the FO formula ψ that is approximated. A second difference is that Theorem 3.13 applies to arbitrary subformulas φ of ψ, while the above proposition only considers atoms $P(\bar{d})$. It is, however, an easy corollary of this proposition that the result in fact holds for each formula ψ that contains only predicates defined by $\mathcal{D}^{\Delta\wedge\varphi}$, i.e., whenever $WFM_I(\mathcal{D}_\sigma^{\Delta\wedge\varphi}) \models$

$\psi^{ct}(\bar{d})$ (or $\psi^{cf}(\bar{d})$), then for every model M of $\Delta \wedge \varphi$ extending I, it holds that $M \models \psi(\bar{d})$ (resp. $M \models \neg\psi(\bar{d})$).

We introduced the approximation $App^{ct/pt}(\Delta)$ with the aim of being more complete than the completion-based approximation. As long as only a single definition Δ was considered in isolation, we succeeded in this goal. However, now that we have also incorporated the additional formula φ, this is no longer the case. For instance, consider the following FO(ID) theory:

$$\{\ Q \leftarrow P\ \} \wedge Q.$$

Here, the approximation $\mathcal{D}^{\Delta\wedge\varphi}$ cannot derive that P is certainly true, simply because the ct/pt-approximation $App^{ct/pt}(\Delta)$ does not contain rules for *head-to-body* propagation, i.e., there are no rules to infer something about the body of a definitional rule, given information about its head. By contrast, the approximation of the completion does contain such rules and therefore has no problems reaching this conclusion. This motivates us to not just use $\mathcal{D}^{\Delta\wedge\varphi}$ but to use $\mathcal{D}^{\Delta\wedge(compl(\Delta)\wedge\varphi)}$ instead. Because each definition implies its completion, this is sound.

Nested fixpoint expressions

Our last approach to approximate FO(ID) theories, is based on another encoding of the three-valued model of a definition Δ, this time as an FO(FP) definition. Since there is at the time of writing at most a proof-of-concept solver available for FO(FP), this approach is less attractive from a practical point-of-view. However, the reason that we still present this approach here, is that the FO(FP) encoding of a definition Δ has the interesting property that it defines *always* a sound and complete representation of the three-valued well-founded model of Δ, that is, without having to require that Δ has a unique four-valued fixpoint, as is the case for the ct/pt-encoding.

Recall that in Chapter 2 we showed how the computation of the well-founded model extending a *two-valued Open* (Δ)-interpretation I can be encoded by a nested fixpoint expression in FO(FP). The following definition extends this to the case of three-valued *Open* (Δ)-interpretations $\mathcal{I}$ that we need in this context. Let $\sigma \subseteq Open(\Delta)$ such that σ contains all function symbols in Δ (i.e., both contain the same pre-interpretation). Assume again that $\mathcal{I}$ is three-valued only on symbols of $Open(\Delta) \setminus \sigma$ and that for any set of rules $\mathcal{R}$ it holds that $P \in Def(\mathcal{R})$ is defined by exactly one rule $\forall\bar{x}(P(\bar{x}) \leftarrow \varphi_P[\bar{x}])$.

Definition 3.24. For a definition Δ, we define $FP_\sigma(\Delta)$ as $\lfloor\ \mathcal{R}^{ct}, \lceil\ \mathcal{R}^{cf}\ \rceil\ \rfloor$ where $\mathcal{R}^{ct}$ consists of, the rules

$$\forall\bar{x}(P^{ct}(\bar{x}) \leftarrow \varphi^{ct}_\sigma)$$

and $\mathcal{R}^{cf}$ consists of the rules

$$\forall \bar{x}(P^{cf}(\bar{x}) \leftarrow (\neg\varphi)^{ct}_{\sigma})$$

for every definitional rule $\forall \bar{x}\ P(\bar{x}) \leftarrow \varphi \in \Delta$.

Intuitively, the idea here is that in order to capture the well-founded semantics, in particular the fact that in the well-founded semantics predicates are not allowed to 'keep each other true' (to be more precise: unfounded sets are false under the well-founded semantics), we need to minimize what is true (by using a least fixpoint for the rules that define when predicates are certainly true), and maximize falsity (by using a greatest fixpoint for the rules that define when predicates are certainly false).

The FO(FP) expression $FP_{\sigma}(\Delta)$ now does precisely what we want (i.e., it encodes the three-valued well-founded model of Δ extending a *three-valued* interpretation for $Open\,(\Delta)$).

Theorem 3.25. *Given Δ, σ and $\mathcal{I}$ as specified above, again, let $\tilde{I}$ be the encoding of $\mathcal{I}$ in terms of the symbols P^{ct}, P^{cf}. Then the unique model of $FP_{\sigma}(\Delta)$ extending $\tilde{I}$ encodes the well-founded model of Δ extending $\mathcal{I}$.*

Proof. See Appendix B.

□

Example 3.26. In the case of the definition $\{P \leftarrow P\}$, $FP(\Delta)$ is the following definition

$$\lfloor P^{ct} \leftarrow P^{ct}, \lceil P^{cf} \leftarrow P^{cf} \rceil \rfloor.$$

This definition has a unique model where $P^{ct} = \mathbf{f}$ and $P^{cf} = \mathbf{t}$. This correctly encodes the well-founded model of the original definition, and we see that $FP(\Delta)$ indeed lets us infer that P has to be false.

Finally, for an FO(ID) theory $\Delta \wedge \varphi$, we can again combine the approximation $Approx_{\psi}(\varphi)$ of φ with the $FP_{\psi}(\Delta)$ to produce a sound approximation of $\Delta \wedge \varphi$ in the form of one big nested fixpoint definition[2]. We refer the reader to Appendix B.2 for the details about how to do this.

3.3 Conclusions

In many applications we are interested in finding the logical implications that follow from a theory and some partial information. Computing these implications exactly is often

[2]Note that as defined in the previous section, for a first-order formula ψ, the approximating definition $Approx_{\sigma}(\psi)$ is a positive definition. This means that we can equally see them as a least fixpoint definition of FO(FP).

too expensive. Therefore, we are interested in approximate but efficient ways to do this. Wittocx et al. (2010) presents a way to do this for first-order theories. His method is based on propagation up and down the parse tree of a theory. The idea behind this propagation is in fact very intuitive, e.g., propagate that a disjunction is true from knowing that one of the disjuncts is true. What makes this propagation method even more interesting, is that the process can be syntactically represented as an inductive definition, which can be automatically derived from the original theory. An off-the-shelf FO(ID) reasoner, such as (idp), can then be used to compute logical implications, in polynomial time.

In this chapter we have presented the results from Wittocx et al. in a slightly different and more general way. We also extended the propagation method to FO(ID) theories. For this purpose, we defined three different ways to soundly represent inferences that are sanctioned by an inductive definition. The results from this chapter have many useful applications, as will be illustrated by the following chapters. It is an interesting topic for future work to see how this propagation method can be extended to full FO(·). Also, while a number of implementations of this propagation method exists (e.g., the one used in the grounder of the IDP system (idp), or the implementation we use for our approximation method for $\exists\forall SO(ID)$ in Chapter 5), it would certainly be interesting to have an optimized, robust, stand-alone implementation.

4

Knowledge Base Systems: a case study for configuration software

The idea of a *knowledge base paradigm* has been around in the AI community since the early days. Instead of counting on the programmer to "compile" his knowledge about the domain and the task to be performed into procedural code, the idea is to represent this knowledge explicitly and automatically derive the desired behavior of the program from it, by means of various forms of logical inference. For this approach to really work, we need to represent the domain knowledge in a *purely declarative* way. That is, it cannot suffice to just encode solutions to one specific problem in this domain. Instead, we need to be able to represent the domain knowledge *on its own*. Otherwise, we cannot hope to be able to *reuse* this same representation to solve the many different problems and tasks that a single software application might need to perform.

Such a KB paradigm makes a lot of sense if we look at the daily practice of software engineering. We see that computationally hard problems are vastly outnumbered by more mundane problems. A typical example is that of *interactive configuration software*, where the goal is to help a user fill out a form in accordance with certain constraints. Despite their apparent lack of computational hardness, such applications are not always easy to develop and maintain. An example famous in Belgium is the *tax-on-web* system, commissioned by

the Belgian government to allow citizens to fill in their tax return forms on-line and compute the amount of taxes due. Even though the system is now operational, its development took significantly more time and resources than originally planned. This problem is mainly hard because of the complexity of the *domain knowledge* involved: the first part of the forms alone already comes with an explanatory leaflet of no less than 96 pages; most of this knowledge must somehow find its way into the on-line application. Moreover, since tax laws change every year, this had better be done in such a way that it can easily be adapted later on. Another example, and the original motivation for the research in this chapter, is a piece of configuration software that the university of Brussels asked software companies to implement as a test before writing out a contract: a system that allows a student to select his study program for the next year. One of the requirements of the university was that it should be easy to change the rules for such a selection, and many of the candidates struggled with this requirement.

However, while the idea of a knowledge base system (KBS) seems very appealing from a theoretical point of view, in practice, building a KBS is still very challenging. A first big question that comes up is which knowledge representation language to use. Obviously, we need a knowledge representation language that is sufficiently expressive, not (only) in the formal sense of the word, but also informally. That is, for each natural language statement that we could find in, e.g., the Belgian tax laws, it should be possible to come up with some formula in the logic that is an "obviously" correct formalization thereof. The representation language should also not be tied to a specific form of inference. Indeed, we do not want the knowledge base to encode a solution to a specific problem, nor is it a program with an operational semantics. It is 'only' a representation of domain knowledge, and one should be able to use it to solve different problems and tasks, requiring different forms of inference, such as deduction, model checking, model generation, update and revision, abduction, learning, etc. Here, the knowledge base paradigm differs significantly and importantly from other strands of declarative programming such as logic programming, ASP or Constraint Logic Programming (CLP).

While first-order logic has been traditionally disregarded as a viable KBS language by many people in the AI community, as we already argued in the introduction of this dissertation, we do believe that FO(·) is a good candidate to serve as the knowledge representation language in a knowledge base system. This chapter is a first experiment in exploring the feasibility of an FO(·) based KBS. We formulate the functionality of a piece of configuration software in logical terminology, and implement a proof-of-concept study program selection application. While of course a real proof of the feasibility of an FO(·) KBS would require, to start with, a lot more tests and a range of case studies on a real-world scale, in-depth comparisons with other formalisms, etc., we do believe that the results in this chapter are valuable, since to the best of our knowledge, no full first-order KBS has been build thus far. Indeed, the knowledge base paradigm that is the topic of this chapter distinguishes itself by the following properties:

- We use an FO(·), an extension of first-order logic to represent the domain knowledge.
- We treat this language in an entirely declarative way: while writing the theory, the user does not need to be concerned about how this will later be used. This allows the same knowledge to be used to solve different tasks.
- We have algorithms that are able to efficiently perform different tasks for this expressive language.

4.1 Case study: a course selection system

In this section, we analyze one particular piece of configuration software: a system that allows a student to select his study program for the next year. Of course, the university imposes certain restrictions on a student's choice, which means that we need *domain knowledge* about what constitutes a valid study program. Typical examples of such restrictions include:

- Certain courses belong to a specific module, and can only be chosen if this module itself is chosen;
- If a module is chosen, then all courses that belong to it must be chosen;
- Compulsory courses have to be chosen;
- Each course is worth a number of credits and the student has to choose 60 credits in total.

This knowledge is *declarative* in nature: it is written down in university regulations, where it exists independently of any piece of software that we might want to develop.

The application now should use this declarative knowledge to perform certain tasks.

- When the user has selected all courses he wants to follow, the application should *check* whether this is valid.
- When the user selects a combination of courses that can never be part of a valid study program, the application should show an appropriate *error message*;
- The application should help the user fill out the form by indicating which other choices are already implied by the selection that the user has made so far.
- The application should be able to automatically *complete* the selection, potentially in some *optimized* way.

Conceptually, there is nothing complicated about these tasks. Moreover, given that the number of courses available to a student is never excessively large, there are also no stringent algorithmic requirements. Nevertheless, it would not be trivial to implement this application in a traditional imperative language. The reason for this is that the control flow of the program would need to take into account quite complex dependencies between different selections. For instance, consider the following situation:

1. There is a *Physics Module* which contains the courses *General Physics* and *Object-oriented Programming 1*;
2. There are three optional courses, e.g., *Introduction to Physics*, *Modeling and Simulation* and *Electrotechnics*, ...of which the student has to select precisely two;
3. Because of the sizable overlap between *General Physics* and *Introduction to Physics*, it is not allowed to take both.

Here are but a few of the propagations that the application should be able to perform:

- Selecting *Modeling and Simulation* and *Electrotechnics* implies that *Introduction to Physics* cannot be selected;
- Selecting *Modeling and Simulation* and choosing not to follow *Introduction to Physics* implies that *Electrotechnics* has to be selected;
- Selecting the module *Physics* implies that both *Electrotechnics* and *Modeling and Simulation* must be selected;
- Choosing not to take *Electrotechnics* implies that the *Physics* module cannot be selected;

It should be clear that the full list is rather long. A naive implementation in an imperative programming language such as C or Java might include an **if**-statement representing each of these possible propagations. An obvious downside of this approach is that it is hard to make sure that all possible propagations are accounted for. Moreover, such an implementation is especially unsuitable if we also take into account the fact that the regulations will most likely change every year. For instance, suppose that the university decides to allow each student a single exception to the rule that selecting a module means selecting *all* of its courses. From a declarative point-of-view, this is a small change, affecting only one of the many rules that govern the selection procedure. However, in a naively implemented system, it might be quite some work to figure out which of the originally implemented propagations should be removed and which should be added—it might even be easier to start over from scratch. An experienced programmer would therefore probably end up with a design in which each of the different regulations is represented by some object, which takes care of

all the propagations that should be performed because of this regulation. Essentially, such a design will lead to the implementation of an *ad hoc* constraint propagation system, which will take quite a bit of work to get right. Obviously, it is a superior solution to express this knowledge as a declarative theory, and use an existing solver or collection of solvers to provide the required inference.

4.1.1 Representing the domain knowledge

The central idea in a KBS approach is to start from a purely declarative representation of the domain knowledge. This means that we should be able to write down this knowledge in a way which is completely independent from how we might later want to use it. First, this simplifies the knowledge representation task, because we are free to put all considerations about the intended behavior of the application out of our mind for the time being. Second, this also makes the representation reusable: if in the future, we want to implement some other functionality in the same domain, we do not need to change our domain knowledge. For instance, we can imagine that in addition to our application for students, we also want to offer a decision support system for the program director, who has to decide whether a student is allowed certain exceptions to the general rules. One piece of functionality that such a system might offer is to pin-point precisely which of the student's desired choices violate which of the university regulations. Given the appropriate logical inference algorithm, we could completely reuse our representation of the domain knowledge to implement this.

So, in our approach, the first step towards implementing the course selection application as a KBS is to write down the relevant domain knowledge. As mentioned, we will do this in the language FO(·). It is rather obvious how this would go. In our typed logic, we first need to decide on the types that we will use. Let us choose types *Course* and *Module*, together with the "built-in" type *Integer*. We then have a vocabulary Σ consisting of predicates *Compulsory(Course)*, *NumberOfCredits(Course,Integer)*, *BelongsTo(Course,Module)*, *Selected(Course)* and *Selected(Module)*.

- Certain courses belong to a specific module, and can only be chosen if this module itself is chosen;

$$\forall x\ y : \mathit{BelongsTo}(x, y) \land \mathit{Selected}(x) \Rightarrow \mathit{Selected}(y).$$

- If a module is chosen, then all courses in this module must be chosen;

$$\forall x : \mathit{Module}(x) \land \mathit{Selected}(x) \Rightarrow$$

$$\forall y : \mathit{Belongs}(y, x) \Rightarrow \mathit{Selected}(y).$$

- Some courses are compulsory, which means they have to be chosen,

$$\forall x : \mathit{Compulsory}(x) \Rightarrow \mathit{Selected}(x).$$

- Each course is worth a number of credits and the student has to choose 60 credits in total.

$$sum\,(\{x\ y \mid NumberOfCredits\,(y, x) \wedge Selected\,(y)\}) = 60.$$

This last formula uses the sum construct from FO(·). Without such a construct, one would have to enumerate all possible combinations of courses that result in 60 credits or more. This would yield a large and unintuitive theory, which would have to change considerably each time the university changes the number of credits for even just a single course.

While for reasons of presentation we kept the amount of domain knowledge here deliberately small, we also experimented with, for example, other, more complicated, restrictions on valid selections, such as prerequisites on courses. It turned out that, for this case study at least, all such domain knowledge could be elegantly represented in FO(·).

4.1.2 Using the Knowledge Base

The central idea behind a KBS is to start from a purely declarative representation and reuse this to solve different tasks and problems. In the previous section we proposed a representation in FO(·) of the relevant domain knowledge of our case study. In this section, we look at the different problems that need to be solved in the context of our configuration program.

Obviously, it is not enough to simply describe these tasks informally. First, we *characterize* each such task as a *logical inference task*. Such inference task has a fixed input and output pattern. Both *input* and *output* for an inference task consist of any number of logical objects such as theories, interpretations, formulas, relations, etc. The output is defined in terms of the input, through well-defined logical relations and properties.

Example 4.1. The inference task *deduction* is characterized as follows.

Input:	an FO theory T and a formula ψ,
Output:	boolean value $r \in \{\mathbf{t}, \mathbf{f}\}$,
such that	$r = \mathbf{t}$ iff $T \models \psi$.

Of course, a mere description of all inference tasks relevant to our piece of configuration software, is not of much practical use. We also need *inference algorithms*, that is, algorithms that actually *solve* these inference tasks. Ideally, we of course want *sound* and *complete* algorithms. These are algorithms that solve the tasks at hand exactly. However, because some of these tasks have an inherent high complexity, we will also be interested in solving these task *approximately*[1]. Ideally, we also would like these algorithms to be implemented in an existing solver, which we can then use off-the-shelf.

[1]Note that the exact meaning of 'solving an inference task approximately' might be different from task to task.

In the rest of this section, we will look at the problems which need to be solved in our piece of configuration software, characterize them as inference tasks, and investigate which inference algorithms eventually can be used to effectively solve these tasks.

A simple validation system

Let us first assume that the user enters all of his choices manually, without any help from the application. Our job is merely to observe what happens and place this in a logical framework. A first observation is that we need to distinguish two different kinds of predicates in the vocabulary:

- The predicates *Compulsory(Course)*, *BelongsTo(Course,Module)* and *NumberOfCredits(Course,Credits)* are all given beforehand and are known to the system;
- *Selected(Course)* and *Selected(Module)* are not known beforehand and need to be filled in by the user.

We will refer to the first kind of predicates as *given* and to the second as *wanted*. Let Γ be the set of given predicates and Ω that of the wanted predicates. We assume, Γ and Ω partition the set of all predicates in Σ [2]. In logical terms, the fact that the given predicates are known up-front means that the application will have at its disposal some interpretation G for Γ. Note that since all courses, modules and number of credits are known up-front also means that the domain of this interpretation can be taken to be finite. The eventual end-state that the application should reach is one in which the user has chosen a complete interpretation for the wanted predicates such that all the university regulations are satisfied. That is, an interpretation W for Ω such that:

$$W \cup G \models T. \tag{4.1}$$

Here, $W \cup G$ denotes the interpretation for the vocabulary Σ that interprets all predicates $P \in \Gamma$ by P^G and all predicates $P \in \Omega$ by P^W. Thus we can identify a first task we want our KBS system to be able to handle.

Task 1 (model checking)**.** *Given a vocabulary Σ, an FO(·) theory T over Σ and a two-valued Σ-interpretation I with a finite domain, model checking is the problem of deciding whether $I \models T$.*

It is well-known that the data complexity of this problem is polynomial time for FO, and this is still the case for FO(·) (Mariën et al., 2006). Of course, the theory data complexity is perhaps not an ideal measure, here, since the theory itself may also be subject to change and

[2] This is not always the case necessarily, e.g., there might also be auxiliary predicates.

could grow quite large. However, as long as the nesting depth and formula width (i.e. the maximal number of free variables in a subformula) of additional formulas are bounded, the combined complexity of model checking for FO(·) remains in P. We believe that in practice, (comprehensible) formulas produced by human experts are bounded in this way. The IDP system (idp), for example, can be used to model checking for FO(·).

The user constructs his desired interpretation W only in a *gradual way*. Initially, the application knows nothing about it, but as it runs it obtains more and more information, checkbox by checkbox. This means the application goes through a sequence $\mathcal{W}_0, \ldots, \mathcal{W}_n$ of *three-valued* interpretations for Ω, that starts out with the least-precise interpretation $\bot_p$ as $\mathcal{W}_0$ (i.e., $\mathcal{W}_0$ assigns **u** to each atom), and grows increasingly more precise as the user fills in the form, i.e.:

$$\bot_p = \mathcal{W}_0 \leq_p \mathcal{W}_1 \leq_p \cdots \leq_p \mathcal{W}_n.$$

If the user has completed his selection, then $\mathcal{W}_n$ should correspond to a *two-valued* interpretation W. How the user precisely goes from $\mathcal{W}_i$ to $\mathcal{W}_{i+1}$ depends on the UI design. The most general case is that, for each course, he has both the option to indicate that he wants to follow it *and* to indicate explicitly that he doesn't. This means he can turn an atom which was **u** into either **t** or **f**. It is also common to see UI's in which the option to explicitly *not* select something is not present. Either way, the UI should also offer a button "done", to allow the user to indicate that he does not want to select any more courses; the last step to reach $\mathcal{W}_n$ is that of switching *all* atoms that are still **u** in $\mathcal{W}_{n-1}$ to **f**.

Completing the current selection

Let us now expand the functionality of this simple system. A first obvious improvement is to offer the user the possibility to *complete* the selection for him. Formally, assume that the user has made some choices and reached three-valued interpretation $\mathcal{W}$. We now want to find a two-valued interpretation W for which

$$\mathcal{W} \leq W \text{ and } W \cup G \models T.$$

This gives us a second task the KBS should be able to handle.

Task 2 (model expansion)**.** *Let T be an FO(·) theory over a vocabulary Σ and let Γ be a subset of the predicates in Σ. Let $\mathcal{S}$ be a three-valued interpretation with a finite domain, such that the projection of $\mathcal{S}$ on Γ is two-valued. A two-valued interpretation S is a solution to the model expansion problem with input $\langle T, \mathcal{S}\rangle$ if $S \models T$ and $S \geq_p \mathcal{S}$.*

Notice that if $\mathcal{S}$ is two-valued, then model expansion simply becomes model checking. It was shown in (Mariën et al., 2006) that model expansion for FO(·) is always in NP and,

moreover, that there exists input $\langle T, \mathcal{S} \rangle$ for which it is NP-complete. Again, this is speaking just of data-complexity, but the complexity bound continues to hold as long as nesting depth and width are bounded by a fixed constant, which we can expect to be the case in practice.

In some cases, we might want to compute not just *any* completion of the user's choice, but an *optimized* completion according to some measure. For example, in the case of the course selection case study, we might want to complete the user's choice such that a minimal number of courses is selected, or a minimal number of credits. Indeed, we wouldn't want a selection consisting of 200 credits, when the regulations say that a minimum of 120 credits is enough. The task of finding optimized models can be formalized as follows.

Task 3 (optimized model expansion)**.** *Let T be an FO(·) theory over a vocabulary Σ including a finitely totally ordered type $\mathcal{T}$, and let Γ be a subset of the predicates in Σ. Let $\mathcal{S}$ be a three-valued interpretation with a finite domain such that the projection of $\mathcal{S}$ on Γ is two-valued. Let t be a term of type $\mathcal{T}$. A two-valued interpretation S is a solution to the optimization problem with input $\langle T, \mathcal{S}, t \rangle$ if $S \models T$, $S \geq_p \mathcal{S}$ and there does not exists an S' such that $S' \models T$, $S' \geq_p \mathcal{S}$ and $t^{S'} < t^S$.*

Note that since an optimal model can be found by a polynomial calls to an NP oracle, this task is in Δ_2^P. Again, both model expansion and optimized model expansion problems can be solved by the IDP system (idp).

Detecting errors

Instead of waiting until the very end, we would like to flag errors as soon as they are made. To be more concrete, we know that the user has made a wrong choice, once it becomes impossible to fill out the remaining choices in such a way that the end result will be valid. So, we can report an error if the system reaches an interpretation $\mathcal{W}$ such that *there no longer exists* a two-valued interpretation W for which $W \geq_p \mathcal{W}$ and $G \cup W \models T$. Formally, in order to detect such errors, our KBS need to be able to handle the following task.

Task 4 (Detecting errors)**.** *Let T be an FO(·) theory over a vocabulary Σ and let Γ be a subset of the predicates in Σ. Let $\mathcal{S}$ be a three-valued interpretation with a finite domain, such that the projection of $\mathcal{S}$ on Γ is two-valued. The error detection task is the task of deciding whether there no longer exists a two-valued interpretation S such that $S \models T$ and $S \geq_p \mathcal{S}$.*

Note that detecting errors is the *dual* of the model expansion task. It is obvious that this way of reporting errors is sound and complete with respect to criterion (4.1), i.e., for each sequence $\mathcal{W}_0, \ldots, \mathcal{W}_n$, if we would get an error at some $\mathcal{W}_i$, then the two-valued interpretation $\mathcal{W}_n$ cannot satisfy (4.1) and, vice versa, if $\mathcal{W}_n$ does not satisfy (4.1), then we will get an error at some $\mathcal{W}_i$ (albeit that this may only happen for $i = n$).

Auto-completion

A third useful improvement is to help the user by filling out parts of the form for him. Indeed, if a certain new choice is implied by the choices made so far, then we can already fill this in automatically. Given a three-valued interpretation $\mathcal{W}_i$ and an atom A such that $A^{\mathcal{W}_i} = \mathbf{u}$, we say that the choice for $A = \mathbf{t}$ (or $A = \mathbf{f}$) is *forced* iff it is the case that for each two-valued W such that $W \geq_p \mathcal{W}_i$,

$$\text{if } G \cup W \models T \text{ then } A^W = \mathbf{t} \text{ (or, resp., } A^W = \mathbf{f}).$$

Note that if no such W exists (i.e., if the user has already made some invalid choice), then all atoms are both be forced to be true and forced to be false.

Task 5 (model completion)**.** *Given an FO(·) theory T over vocabulary Σ and a three-valued interpretation $\mathcal{S}$ - with a finite domain - for Σ, compute the greatest lower bound (w.r.t. $\leq_p$) of all solutions to the model expansion problem $\langle T, \mathcal{S} \rangle$, i.e., construct*

$$\overline{\mathcal{S}} = \cap_{\leq_p} \{S \mid S \geq_p \mathcal{S} \text{ and } S \models T\}.$$

Again, this way of assisting the user is sound and complete w.r.t. (4.1), i.e., for each W such that $G \cup W \models T$, $W \geq_p \mathcal{W}_i$ iff $W \geq_p \overline{\mathcal{W}_i}$. Every time the user makes a choice, we thus can replace the current three-valued interpretation $\mathcal{W}$ by its completion $\overline{\mathcal{W}}$ and update the form accordingly.

As we already showed in Chapter 3, this inference task is in general computationally hard. In this particular instance, the search space is of course bounded, and an efficient solver might be capable of computing all models and taking their intersection. However, it is clear that in general this solution is unfeasible. The solution that we have chosen is to use the propagation method from Chapter 3 to compute a polynomial time approximation $\tilde{\mathcal{S}}$ of $\overline{\mathcal{S}}$. The price to pay for this tractability is that some inferred literals are detected at later stages of the interactive model construction process and perhaps worse, that detection of inconsistency of the choices might be delayed. The 'ideal' method of computing the intersection of all models has indeed the pleasant property that as soon as the user's selection is inconsistent, this is detected immediately, and, vice versa, as long as the propagated structure is three-valued (as opposed to four-valued), the user has an option to complete it to a model.

In practice, the results of our experiment were positive. In almost all cases, the approximative algorithm was able to find all implications of the user's choices. Only in rare circumstances involving the aggregate expressions did the method miss some consequences. A possible solution is to combine this with the detection of errors described above. This also shows the importance of investigating the precision of our propagation method. Therefore, we likely want to combine this with the detection of errors described above. This also shows the importance of investigating the precision of our propagation method.

Undoing choices

Thus far, we have not yet considered what happens when the user changes his mind halfway through a run of the program. This can manifest itself in two ways: either the user tries to undo one of the choices he made himself, or he wants to deselect one of the courses that the application auto-selected.

The first case is easy. What happens here is that the user turns an atom that was **t** back into **u**. We thus obtain a new state which is *less* precise than the previous one, i.e., the applications goes from some $\mathcal{W}_i$ to a $\mathcal{W}_i' \leq_p \mathcal{W}_i$. Of course, the UI then should display $\overline{\mathcal{W}_i'}$ instead of $\overline{\mathcal{W}_i}$, so undoing a choice also undoes all other choices that were implied by it. Note that we can just use the algorithm of Task 5 to compute this, regardless of the order in which the user wants to undo his choices.

The second case is a bit trickier. Here, the user attempts to undo some choice $Selected(C)$ that was made automatically, i.e., one that is **t** in $\overline{\mathcal{W}_i}$ but not in $\mathcal{W}_i$. The problem is of course that $\mathcal{W}_i$ still contains whatever choices it were that implied $Selected(C)$ in the first place. So, switching $Selected(C)$ from **t** to **u** buys the user nothing: he will still either have to undo some of the choices from $\mathcal{W}_i$ that imply $Selected(C)$, or else be forced to reselect $Selected(C)$ in the future. Therefore, we might as well prevent the user from performing such actions, and only allow him to undo those choices that he has made itself. Of course the interface should visually make a difference between choices that are made by the user, which he can undo or change, and the ones we deduced for him, which the user cannot change directly.

An alternative is to try to automatically undo some of the user's choices so that $Selected(C)$ would no longer be implied. That is, we look for some $\mathcal{W}' \leq_p \mathcal{W}_i$ for which there still exists an interpretation W such that $W \geq_p \mathcal{W}'$, $W \cup G \models T$ and $Selected(C)$ is **f** in W. In order not to undo too much of the user's work, this $\mathcal{W}'$ should be as close as possible to $\mathcal{W}_i$, so we would be looking for those $\mathcal{W}'$ such that the set of atoms which are **u** in $\mathcal{W}'$ but **t** or **f** in $\mathcal{W}_i$ is minimal.

Task 6. *Let T be an FO(·) theory over a vocabulary Σ, $\mathcal{W}_i$ a three-valued Σ-interpretation and $L[\bar{d}]$ a Σ-literal. Find a maximal three-valued interpretation $\mathcal{W}' \leq \mathcal{W}_i$ such that there still exists a model $W \geq \mathcal{W}'$ such that $W \models \neg L[\bar{d}]$.*

There are some disadvantages to this approach when compared to the first approach. First, because of this minimality criterion it is computationally hard. Second, in this way we might undo more of the user's choices than he intended, which could cause behavior that is hard to understand and potentially frustrating. To the best of our knowledge, this inference task has not been implemented yet, and it is an interesting open topic to find an intelligent algorithm for this inference task.

Explaining errors

Above, we showed how we could see detecting errors formally boils down to the model expansion inference task. However, merely detecting an error is of course not very useful. It would greatly benefit the user if we could also explain why a certain selection is forced or disallowed.

Suppose that the application has reached a three-valued interpretation $\mathcal{S}$, such that no interpretation $S \geq_p \mathcal{S}$ is still a model of T. An explanation of the conflict should consist of two essential components:

- a number of choices made by the user that together cause the conflict.
- a number of sentences of the theory T that explain the conflict.

For instance, imagine that we represented the domain knowledge explained in Section 4.1 by means of a theory that contains the following sentence:

$$\neg(\mathit{SelectedCourse}(\mathit{IntroductionToPhysics}) \wedge \mathit{SelectedCourse}(\mathit{GeneralPhysics})). \tag{4.2}$$

If the user is not aware of this rule and tries to select both courses, a good configuration program would give him the following error report:

> It is not valid to make these choices:
>
> - Introduction to Physics, General Physics
>
> because they violate the following constraint:
>
> - It is not allowed to follow both Introduction to Physics and General Physics.

Here, the programmer should provide the natural language version of (4.2). Alternatively, this could be provided by some natural language interface (this is at the time of writing ongoing research). With this information, the user knows not only how to solve the conflict, but also why there was a conflict in the first place.

Task 7 (Explaining inconsistency)**.** *Given an FO(·) theory T over vocabulary Σ, and a three-valued interpretation $\mathcal{S}$ for Σ so that it is no longer the case that there exits a $S \geq_p \mathcal{S}$ for which $S \models T$. Find an explanation for the inconsistency, consisting of the following two minimal sets:*

1. *a minimal subset $T' \subseteq T$ such that for each $S \geq_p \mathcal{S}$, $S \not\models T'$,*

2. *a three-valued interpretation*

$$\cap_{\leq_p} \{\mathcal{S}' | \mathcal{S}' \leq_p \mathcal{S} \text{ and for all } S \geq_p \mathcal{S}',\ S \not\models T\}.$$

Again, this task is computationally hard. In (Wittocx et al., 2009) a model expansion algorithm is described based on the propagation method described in Chapter 3. This algorithm has the advantage that it can be traced. When the algorithm detects an inconsistency this means that some atom has to be both **t** and **f** at the same time. It is then possible to look at the trace and see which selections that were made by the user and which rules were used to infer this. Tracing the same algorithm could be used to give the user an explanation of *why a certain course has been inferred*. To the best of our knowledge, this algorithm has not been properly implemented (only as a very basic proof-of-concept), and we consider it an interesting topic for future work to do so.

4.1.3 Implementation

We have analyzed in Section 4.1.2 the desired behavior of the case study application and characterized it as a number of logical inference tasks. In this section, we will summarize how we implemented a proof-of-concept version of the application, using existing solvers to handle the tasks we identified.

Recall that we have a theory T in vocabulary Σ, where the predicates of Σ are partitioned into given predicates Γ and wanted predicates

$$\Omega = \{Selected(Module), Selected(Course)\}.$$

We are also given an interpretation G for the predicates Γ.

Our application has a simple UI, consisting of a list of checkboxes for all the courses and modules, as well as a button for verifying the current selection, completing the selection, completing the selection such that a minimal number of credits is selected, as well as a reset button, and finally undo and redo buttons. A screenshot of the application can be found in Figure 4.1.

The state of the application consists of a three-valued interpretation $\mathcal{S}$ for the vocabulary Σ. The UI *visualizes the approximation* $\tilde{\mathcal{S}}$ of the completion $\overline{\mathcal{S}}$ of $\mathcal{S}$, in a way which makes a clear distinction between the choices made by the user (i.e., those in $\mathcal{S}$) and those inferred by the application (i.e., those in $\tilde{\mathcal{S}}$ but not in $\mathcal{S}$). The interpretation $\tilde{\mathcal{S}}$ is visualized by coloring the courses that are certainly true according to $\tilde{\mathcal{S}}$ green, and those that are certainly false red. This can be seen in Figure 4.2.

At the start of the application, $\mathcal{S}$ interprets Γ by G, the predicate $Selected(Module)$ is interpreted by the function that maps all modules to **u**, and the predicate $Selected(Course)$ by the function that maps all courses to **u**.

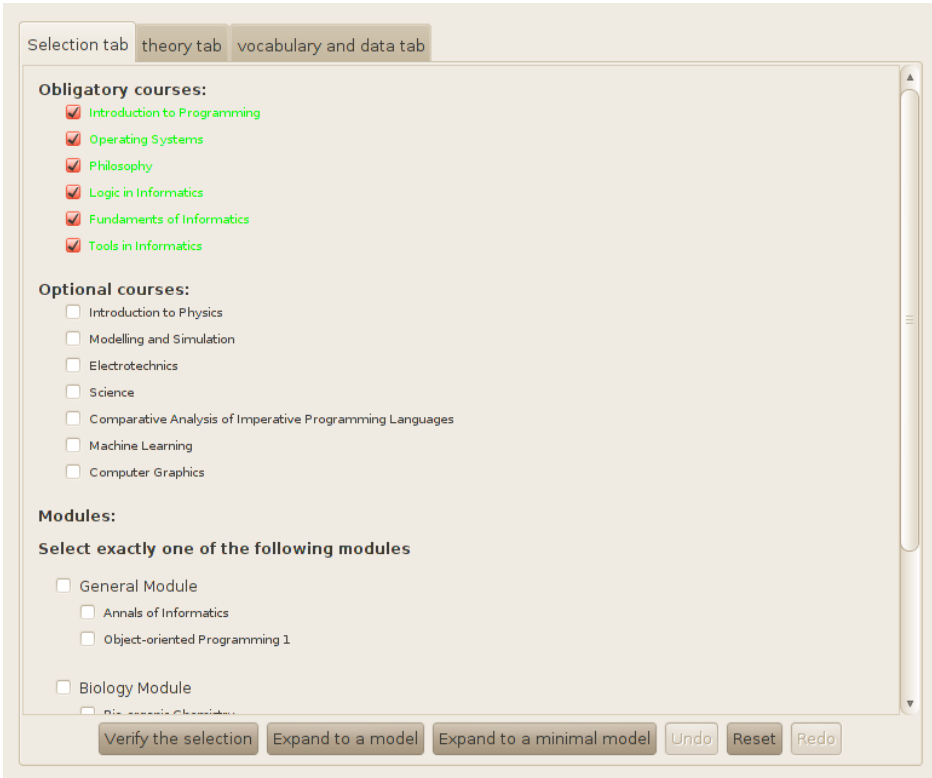

Figure 4.1: Screenshot of the proof-of-concept implementation of our case study.

The user can perform a number of actions, each of which affects the state $\mathcal{S}$ (and, hence, produces the corresponding change to the visualization of $\tilde{\mathcal{S}}$ on the screen):

- He can select a course (or module) c for which is $Selected^{\tilde{\mathcal{S}}}(c) = \mathbf{u}$. In this case, we change $\mathcal{S}$ by setting $Selected^{\mathcal{S}}(c) = \mathbf{t}$.
- He can unselect a course (or module) c for which $Selected^{\mathcal{S}}(c) = \mathbf{t}$. In this case, we change $\mathcal{S}$ by setting $Selected^{\mathcal{S}}(c) = \mathbf{u}$.
- He can try to select a course (or module) c for which $Selected^{\tilde{\mathcal{S}}}(c) = \mathbf{f}$. In this case, we produce an error message, explaining why this course cannot be selected. To do this, we used a proof-of-concept implementation of the debugging method described in (Wittocx et al., 2009).
- He can click the button "Verify the selection". We then construct a two-valued interpretation F representing his final choices: $Selected^{F}(c) = \mathbf{t}$ for all c such that

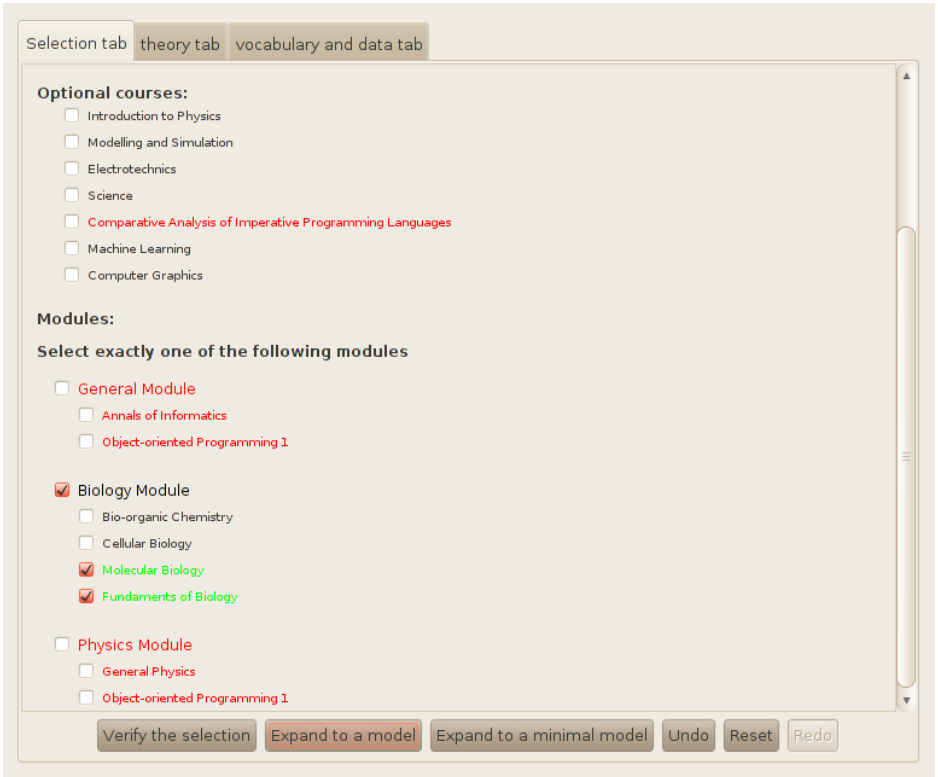

Figure 4.2: After the user selects the Biology Module, the UI visualizes that e.g., the Comparative Analysis can no longer be selected, and that the Molecular Biology course *has* to be selected.

$Selected^{\hat{S}}(c) = \mathbf{t}$; all other $Selected^{F}(c)$ are **f**. We then check whether $F \models T$. If not, the application produces an error message and does not change the state of the program; otherwise, we commit F to our student data base and terminate.

- He can click the button "Expand to a model", and similarly "Expand to a minimal model". Here we call the IDP system (idp) to find a model $S \geq_p \mathcal{S}$ such that $S \models T$ (and is a minimal model in the latter case).

Because of our use of the approximation algorithm, the first four of these tasks run in polynomial time. Moreover, for all of the theories we tried in our experiments, the approximation always managed to achieve optimal precision, apart from certain rare cases involving aggregates.

We have implemented this application by means of a simple Java program that provides a UI, which visualizes the underlies three-valued interpretation and calls the approximation method and the IDP-system (Mariën et al., 2006), which is a model expansion system for FO(·), as described above. The resulting proof-of-concept configuration program can be downloaded at the following URL:

`http://http://dtai.cs.kuleuven.be/krr/software`

4.2 Related work

We are not the first to apply AI research to the problem of developing configuration software. (Soininen & Niemelä, 1999) present a system to automatically compute valid configurations without user interaction. This is a slightly different setting to ours, since one of the things that interests us is precisely the demands that are imposed by the interactiveness of the system (responsiveness, error reporting, auto-completion). The language they use is a form of propositional logic programs. Another logic programming approach is that of (Axling & Haridi, 1994), which does focus on the interactive aspects. They represent domain knowledge in the SICStus Object System, using a representation which is specialized towards configuration tasks involving actual physical objects, such as computer components. It seems less suited to represent, for instance, the regulations for tax return forms. (Vanden Bossche et al., 2007) describes an *ontology based* development methodology using OWL (together with, as we understand it, several application-specific extensions), whereas (Subbarayan et al., 2004) describes *binary decision diagram* and *constraint programming* based approaches to developing configuration software. Probably the oldest interactive knowledge base systems are so-called *expert systems* (Nikolopoulus, 1997). These are typically rule based: the domain knowledge is represented as a set of implications, which are then used to reason either by forward chaining (deriving the consequence from its antecedents) or backward chaining (deriving that one of its possible antecedents has to have caused the consequence). Here, the interactions with the user are all governed by the same inference method and, moreover, getting the desired behavior often requires that this method of inference already be taken into account while constructing the theory.

A first thing that all of these approaches have in common when compared to our proposal is that they use a language that is less general than FO(·). By using FO(·), we hope that our method will be applicable to a significantly larger class of applications. Indeed, this language is not only expressive, but also general, in the sense that it is not geared towards any specific application area. Moreover, it is based on classical logic, the most studied and best understood logic in mathematics and computer science. We could consider, for instance, Answer Set Programming (Gelfond & Lifschitz, 1991) as an alternative (Baral, 2003), but we feel that this is going to be less familiar to people than classical logic. Moreover, we claim

that it is less suited to represent, e.g., the regulations for a valid study program, since this domain knowledge does not require any of the epistemic or minimal-model features of ASP, but does contain, e.g, nested quantifiers and equivalences.

A second difference is that it is precisely one of our goals to analyze the use of *different* methods of logical inference to implement the different tasks that arise within a software system, all using the *same* domain knowledge. The above approaches all consider a single form of inference: forward/backward chaining for the expert systems, SLD-resolution for the Prolog systems, model generation for ASP, and so on.

There also exist several more *ad hoc* KR systems that take an approach somewhat similar to ours. For instance, (Balduccini et al., 2006) uses a Java shell on top of an Answer Set Programming system to implement a decision support system for a particular diagnosis-and-repair problem in the context of NASA's space shuttle program. Again, the difference with our work is that we are trying to develop a generally applicable framework, that provides a range of domain-independent inference algorithms. By contrast, the aforementioned system uses a single inference algorithm and implements different tasks by changing, from within the imperative shell, the theory on which they are performed. The way in which these changes are made is particular to the application in question.

In as much as that our method also constructs a formal model of (certain aspects of) the behavior of a software program, it is related to research on *workflow languages* such as (van der Aalst & ter Hofstede, 2005). However, whereas they essentially regard the events/actions that occur in a software program as black boxes, we have a semantic model, which focuses precisely on describing the interactions with the user in terms of the domain knowledge. Therefore, our focus is significantly different. Nevertheless, more semantically oriented workflow languages also exist (Davulcu et al., 1998). These do not yet go as far, however, as to have a full representation of the domain knowledge, that can be considered separately from the behavior of the system.

4.3 Conclusions

One important source of complexity in software engineering is that declarative domain knowledge and procedural knowledge about the desired behavior of the system are delicately intertwined throughout a typical software program. In this chapter, we presented an approach that is aimed toward separating these two components completely. Our framework consists of the expressive language FO(·), together with a number of different inference tasks for this language.

The inspiration for the case study in this chapter came from a short internship that I did with a company that was struggling to implement this course selection application, which was part of a test from the university of Brussels before getting a contract. This company

was using OWL as a knowledge representation language and at-hoc reasoning procedures, but they encountered a number of problems. It turned out that the domain knowledge was not always easily representable in OWL, and moreover, they had to use different kinds of tricks in their representation, to get the reasoning procedures to work. This motived us to see if the domain knowledge for this application could be represented in FO(·), and analyze the different forms of reasoning that are needed to provide the required functionality.

The results in this chapter should be seen as a first promising result towards a proof of the feasibility of an FO(·) knowledge base system (but by no means as conclusive proof). Nevertheless, these are valuable results, since traditionally, many people have been doubtful about whether FO could be of any practical use. Indeed, most similar approaches to developing configuration software, such as (Subbarayan et al., 2004), are essentially propositional in nature. By offering full first-order quantification as well as aggregates, our approach makes the task of writing down the domain knowledge significantly easier. In the field of knowledge representation, there is a general trend towards more expressive languages, acknowledging the practical need for such features. This is seen in Answer Set Programming (e.g., aggregates, classical negation, disjunction in the head, (Leone et al., 2006)), Description Logic (e.g., rule languages such as SWRL) and Constraint Programming. The limiting factor is of course always the efficiency of the inference algorithms. In this chapter, we have catalogued a number of key reasoning tasks that are needed to implement configuration software, given an FO(·) knowledge base, and have shown that (at least from a theoretical point of view) all of these can be efficiently implemented or otherwise be adequately approximated in polynomial time.

Of course, building a KBS is a compound and ambitious research goal, consisting of many subgoals which are challenging in their own right, and many interesting research questions regarding this goal still remain unanswered. Indeed, in order to truly assess the real-world feasibility of a KBS system for configuration applications, for example, case-studies on a much larger scale would be required, as well as comparisons with other formalisms. The Java based API from Calus & Vennekens (2011), that allows to quickly make prototype GUI's for configuration software similar to the case study in this chapter, is perhaps a first step in this direction. This API offers support for linking an FO(·) knowledge base, input of the user, and the output of the different reasoning algorithms to a GUI with as little effort as possible. Moreover, it will be interesting to see how the KBS methodology can be applied to a larger class of software applications. While the inference tasks we have presented here seem adequate to handle simple configuration software, more complex applications might also require other forms of logical inference, which have yet to be defined and implemented. One particular issue here is that we have currently limited ourselves to finite domains only. There are many applications that require types with potentially infinite domains, such as strings, integers, and so on. Theoretically, our approximation algorithm easily accommodates such types, since, in principle, we do not need to be able to conclude anything about them until the user fills in their values. However, it would of course be more useful to propagate

information sooner. For instance, in a calendar system, if a meeting has to be finished by a certain time and the user schedules the start of the meeting at a time which is already past the deadline, we like to be able to flag this error. Such functionality could be achieved by integrating constraint propagation techniques into the approximation algorithm. This is currently the subject of ongoing research.

We would like to conclude this chapter with the note that while building a KBS is a indeed a complex task, it is still interesting to always keep the KBS paradigm in the back of our minds. First of all, recent trends in computational logic suggest that building a useful KBS may not be as impossible as many believe now after all. Perhaps more importantly, the KBS paradigm focusses our research by putting constraints on the languages and systems to be developed. It requires us to study what knowledge is required to solve practical tasks, and shows us both weaknesses and strengths of our KR language. It also helps to identify new useful inference tasks, or find new uses for already implemented ones. While a case study like the one in this chapter is by no means a rigid proof of the feasibility of a KBS, it does give us a good indication of what is possible and what not (yet).

5

Approximating ∃∀SO satisfiability problems

Finite model generation is a logical paradigm for solving constraint problems. A successful instance is the field of SAT, where efficient solvers for the low level CNF language are developed. Other instances, but for more expressive languages, are Answer Set Programming (ASP) (Baral, 2003) and model expansion (MX) for (extensions of) first order logic. In ASP, for example, finite Herbrand models of an answer set program are computed (Baral, 2003). Model expansion (MX) (Mitchell & Ternovska, 2005) generalizes Herbrand model generation and aims at computing one or more models of a theory T that expand a finite interpretation I_o for a (possibly empty) subset of symbols of T. MX for first-order logic (MX(FO)) is formally equivalent to the finite domain satisfiability checking problem for existential second-order logic ($SAT(\exists SO)$)[1] which is known from Fagin's celebrated theorem to *capture NP* (Fagin, 1974). That is, the problems in NP are exactly those that are in a precise sense equivalent to an ∃SO satisfiability problem, and hence an MX(FO) problem. A range of solvers exists for finite model generation. An overview of state-of-the-art ASP and MX(FO(·)) solvers can be found in (Denecker et al., 2009) or (Calimeri et al., 2011).

Example 5.1. Here is a bounded planning problem modeled as a finite model expansion problem. The problem is deliberately kept simple as it will serve as the running example in this chapter: *a glass may be clean or not, and can be cleaned by the action of wiping*. We

[1] Or more specifically, to the search problem for a *witness* of such a problem.

represent this dynamic domain by the following FO theory T_{act}:

$$\forall t : (\mathit{Clean}(t+1) \quad \Leftrightarrow \quad \mathit{Clean}(t) \vee \mathit{Wipe}(t)). \tag{5.1}$$

$$\mathit{Clean}(0) \quad \Leftrightarrow \quad \mathit{InitiallyClean}. \tag{5.2}$$

The bounded planning problem we are considering is then to turn a dirty glass into a clean one in n steps. This can indeed be formulated as a model expansion problem: find a model that satisfies T_{act} while *InitiallyClean* is false, and $\mathit{Clean}(n)$ true. We can formulate this problem equivalently as an $\exists SO$ finite domain satisfiability problem, namely as the satisfiability problem in the range $[0 \ldots n]$ of time points of the following $\exists SO$ formula:

$$\exists \mathit{Wipe, Clean, InitiallyClean} : (\psi_{act} \wedge \neg \mathit{InitiallyClean} \wedge \mathit{Clean}(n)), \tag{5.3}$$

where ψ_{act} denotes the conjunction of sentences in T_{act}. For $n > 0$, this formula is indeed satisfiable in the suitable interpretation of $0, n, +/2$, and, moreover, each *witness* W for its satisfiability provides a plan. For instance, wiping at time point 0 does the job, as is verified by the witness W for which $\mathit{Wipe}^W = \{0\}$ and $\mathit{Clean}^W = \{1, \ldots, n\}$.

While a large number of search problems can indeed be seen as finite model expansion problems, there are also a number of problems that are of a higher complexity than NP, and consequently cannot be formulated as an MX(FO) problem. Indeed, in this chapter we are not interested in NP, but in the next level Σ_2^P of the polynomial hierarchy. Perhaps the prototypical such problem is finite domain satisfiability for $\exists\forall SO$: satisfaction in finite interpretations is in Σ_2^P for every $\exists\forall SO$ sentence and is Σ_2^P-hard for some such sentences (Stockmeyer, 1976; Immerman, 1998). An interesting Σ_2^P problem is that of *conformant planning*, which we discuss in detail in Section 5.4 but which we already introduce in the next example.

Example 5.2. Extending Example 5.1, suppose that we do not know whether the object is initially clean or dirty, but still want a plan that is guaranteed to make it clean, *no matter what* the initial situation was. This is no longer a standard planning problem, but is called a conformant planning problem. We can formulate this as the following $\exists\forall SO$ satisfiability problem:

$$\exists\, \mathit{Wipe}\ \forall \mathit{InitiallyClean, Clean} : \quad (\psi_{act} \Rightarrow \mathit{Clean}(n)). \tag{5.4}$$

In words, we need an assignment to the action *Wipe* such that the goal $\mathit{Clean}(n)$ is satisfied for every initial situation *InitiallyClean* and evolution of the fluent *Clean* that satisfy the action theory.

Solving problems like this would require us to make a choice for the existentially quantified predicates and then check that the implication is satisfied for every interpretation of the

universally quantified predicates. While this can be done in principle, in practice it is often too expensive. In this chapter, we will again use the propagation method from Chapter 3 to compute in polynomial time, for a given FO(·) theory T and a partial interpretation $\mathcal{I}$, an approximation of what has to be *certainly true* (= true in all models of T that expand $\mathcal{I}$) and *certainly false* (= false in all such models). Recall that an interesting property of this propagation method is that it can be syntactically represented as a *monotone inductive definition* (Denecker & Ternovska, 2008) that defines (in an approximative way) these underestimates of the predicates in T and of their complements. Moreover, for any given theory T we can obtain this definition by a linear transformation.

Returning to the above example, we need to find an interpretation for the action predicates, such that *for every* interpretation of the other predicates, the implication $\psi_{act} \Rightarrow Clean(n)$ is satisfied, i.e., such that without knowing anything about these other predicates, we should already be *certain* that the implication is satisfied. The basic idea behind our method is to approximate an $\exists\forall SO$ problem of the form $\exists\bar{P}\forall\bar{Q}\,\psi$, using the approximate definition from Chapter 3 to check whether an interpretation for the existentially quantified predicates $\bar{P}$ has the property of making ψ true, regardless of the predicates $\bar{Q}$. Essentially, this reduces an $\exists\forall SO$ problem to an $\exists SO(ID)$ problem (where with SO(ID) we refer to SO extended with inductive definitions).

In Section 5.2, we extend our method to $\exists\forall SO(ID)$ problems. As we argued in Chapter 2, inductive definitions are a useful tool for knowledge representation. For example, many dynamic domains can be formulated naturally and in a modular way using inductive definitions, while this can be quite tedious in FO. We already mentioned conformant planning as a typical $\exists\forall SO$ satisfiability problem. Typically, these conformant planning problems require the modeling of a dynamic domain. Recall that in Chapter 2, we explained how dynamic domains can be modeled in FO(·). The dynamic domain of Example 5.1 can, as an alternative to the action theory T_{act}, be formulated as the following inductive definition Δ_{act}:

$$\left\{\begin{array}{lcl} Clean(t+1) & \leftarrow & Clean(t). \\ Clean(t+1) & \leftarrow & Wipe(t). \\ Clean(0) & \leftarrow & InitiallyClean. \end{array}\right\} \tag{5.5}$$

The conformant planning problem can then be formulated alternatively as the satisfiability problem of the formula $\exists\, Wipe\, \forall InitiallyClean, Clean : (\Delta_{act} \Rightarrow Clean(n))$. However, this is no longer a $\exists\forall SO$ satisfiability problem, but a $\exists\forall SO(ID)$ satisfiability problem. This motivates us to see how we can extend our approximation method to such $\exists\forall SO(ID)$ satisfiability problems.

Our approximation method has a number of benefits. First of all, it is a general method, that can be applied automatically to approximately solve any $\exists\forall SO(ID)$ problem. Second, the

required computation can be carried out by any off-the-shelf MX(FO(ID)) solver, allowing our method to benefit effortlessly from improvements in solver technology, such as (Mariën et al., 2006). Finally, as we show in Section 5.4, our method elegantly generalizes a number of approximate reasoning methods from the literature (Baral et al., 1998; Son et al., 2005; Denecker et al., 2010a; Doherty et al., 2006; Son et al., 2005).

5.1 Approximating $\exists\forall SO$-satisfiability problems

In this section, we will use the results from Chapter 3 to approximate the following problem. Take an $\exists\forall SO$ formula $F = \exists\bar{P}\forall\bar{Q} : \psi$. For ease of presentation, we assume throughout this chapter that second-order formulas contain no free predicate symbols, but all results generalize to the setting where there are free predicate symbols. We also assume that $\bar{P}$ and $\bar{Q}$ contain only predicate symbols. In what follows, we denote the vocabulary of ψ by Σ (that is, including the predicates $\bar{P}$ and $\bar{Q}$). The question we want to answer is whether the formula F is satisfied in a given finite-domain pre-interpretation I of the constant and function symbols of the formula. This satisfiability problem boils down to deciding whether we can find a witness for the satisfiability of this formula, in the following sense.

Definition 5.3. We call J a *witness* for the satisfiability of a formula $\exists\bar{P}\forall\bar{Q} : \psi$ given a finite Σ pre-interpretation I, if J is an interpretation of $\Sigma \setminus \bar{Q}$ extending I (i.e., J is an interpretation of the whole vocabulary without the universally quantified predicates) such that $\forall\bar{Q} : \psi$ is satisfied in J.

Equivalently, J is a witness if in the three-valued Σ-interpretation $\mathcal{J}$ that expands J by assigning **u** to each domain atom $Q(\bar{d})$, it holds that $sv_{\mathcal{J}}(\psi) = \mathbf{t}$.

Our goal in this section is now to approximate an $\exists\forall SO$ satisfiability problems by an $\exists SO(ID)$ satisfiability problem in the following sense.

Definition 5.4. Consider the $\exists\forall SO$ satisfiability problem for a formula $\exists\bar{P}\forall\bar{Q} : \psi$, where ψ is an FO formula in alphabet Σ. An $\exists SO(ID)$ formula of the form $\exists\bar{P}\bar{R} : \psi'$, where ψ' is an FO(ID) formula in the alphabet $(\Sigma \setminus \bar{Q}) \cup \bar{R}$, is a *sound approximation* of this satisfiability problem if, whenever J is a witness for the satisfiability of $\exists\bar{P}\bar{R} : \psi'$, then $J|_{\Sigma\setminus\bar{Q}}$ is a witness for the satisfiability of $\exists\bar{P}\forall\bar{Q} : \psi$.

In other words, a sound approximation G of the satisfiability problem for an $\exists\forall SO$ formula F is a stronger $\exists SO(ID)$ formula, i.e., one that has fewer witnesses for $\bar{P}$.

5.1.1 A naive method

We can now use the results of Theorem 3.11 to construct a sound approximation for a given $\exists\forall SO$ formula.

Definition 5.5. Given a formula $F = \exists\bar{P}\forall\bar{Q} : \psi$, take σ to be the alphabet of all function symbols in ψ and the predicates $\bar{P}$. We define $\mathcal{APP}(F)$ as the $\exists SO$ formula $\exists\bar{P}\exists\bar{R} : Approx_{\sigma}(\psi) \wedge A^{ct}_{\psi}$ in the vocabulary $\sigma \cup \bar{R}$, where $\bar{R} = (\bar{Q} \cup Aux(\psi))^{tf}$.

The intuition here is that for any σ-interpretation I, $Approx_{\sigma}(\psi)$ will give the result of the four-valued evaluation $\psi^{\mathcal{I}}$ in the Σ-interpretation $\mathcal{I}$ that expands I by assigning unknown to all universally quantified predicates $\bar{Q}$. If the entire FO formula ψ evaluates to true in this four-valued interpretation, we know that ψ will be satisfied in *any* Σ interpretation that expands I (in other words, for every interpretation of the $\bar{Q}$ predicates), and thus that I is a witness for the satisfiability of the entire formula F. The auxiliary predicates $Aux(\psi)$ — introduced by the transformation to ENF — are needed because of the way in which the propagation works, but their value is completely determined by that of $\bar{P}$.

Proposition 5.6. *For each $\exists\forall SO$ formula F of the form $\exists\bar{P}\forall\bar{Q} : \psi$, it holds that $\mathcal{APP}(F)$ is a sound approximation of F.*

Proof. This follows immediately from Theorem 3.11, where we take as three-valued interpretation, the interpretation $\mathcal{I}$ such that $(Q(\bar{d}))^{\mathcal{I}} = \mathbf{u}$ for all $Q \in \bar{Q}$ and $\bar{d} \in D^n$, and $(P(\bar{d}))^{\mathcal{I}} = (P(\bar{d}))^{I}$ for all $P \in \bar{P}$ and $\bar{d} \in D^n$, with D the domain of I. □

For example, if we take F to be the formula $\exists P\forall Q : \psi$ where $\psi = P \vee Q$, then $\mathcal{APP}(F)$ becomes:

$$\exists P, Q^{ct}, Q^{cf}, A^{ct}_{\psi}, A^{cf}_{\psi} : \left\{ \begin{array}{l} A^{ct}_{\psi} \leftarrow P \vee Q^{ct} \\ A^{cf}_{\psi} \leftarrow \neg P \wedge Q^{cf} \\ Q^{ct} \leftarrow A^{ct}_{\psi} \wedge \neg P \\ Q^{cf} \leftarrow A^{cf}_{\psi} \end{array} \right\} \wedge A^{ct}_{\psi}.$$

We start from an interpretation O of the open predicate P of this definition $Approx_{\{P\}}(\psi)$. Let us take the interpretation O that makes P true. The unique model I of the definition that extends O is then the interpretation that assigns true to A^{ct}_{ψ} and false to Q^{ct}, Q^{cf} and A^{cf}_{ψ}. Therefore, this I satisfies both $Approx_{\{P\}}(\psi)$ and A^{ct}_{ψ}. Hence, it is a witness for the satisfiability of $\mathcal{APP}(F)$, and, indeed, it is also a witness for the satisfiability of the original formula $\exists P\forall Q : P \vee Q$.

This approximation method is sound, but for many applications still too incomplete. Indeed, let us look at the following formula: $F = \forall Q : Q \vee \neg Q$. Then $\mathcal{APP}(F)$ becomes:

$$\exists Q^{ct}, Q^{cf}, A^{ct}_{\psi}, A^{cf}_{\psi} : \left\{ \begin{array}{l} A^{ct}_{\psi} \leftarrow Q^{ct} \vee Q^{cf} \\ A^{cf}_{\psi} \leftarrow Q^{cf} \wedge Q^{ct} \\ Q^{ct} \leftarrow A^{ct}_{\psi} \wedge Q^{ct} \\ Q^{cf} \leftarrow A^{cf}_{\psi} \\ Q^{cf} \leftarrow A^{ct}_{\psi} \wedge Q^{cf} \\ Q^{ct} \leftarrow A^{cf}_{\psi} \end{array} \right\} \wedge A^{ct}_{\psi}.$$

The definition does not entail that A^{ct}_{ψ}, so $\mathcal{APP}(F)$ is unsatisfiable, even though the original formula F is clearly always satisfied. The problem here is that, as we showed in Chapter 3, the definition $Approx_{\sigma}(\psi)$ encodes the three-valued Kleene evaluation of ψ, which is not strong enough to find out that the formula F is satisfied. To do this, we need the stronger supervaluation.

In Chapter 3 we saw that for a formula ψ supervaluation and Kleene evaluation are in general not equal, but that several *classes of formulas* for which they agree have been proposed, most importantly the normal form $\mathcal{NF}$ by (Liu & Levesque, 1998). It immediately follows that our approximation is complete for $\exists\forall SO$ formulas in which the first-order formula satisfies this condition.

Proposition 5.7. *Each $\exists\forall SO$ formula F of the form $\exists\bar{P}\forall\bar{Q} : \psi$, where ψ is in the $\mathcal{NF}$ normal form (according to (Liu & Levesque, 1998)) is satisfiable with respect to a given finite pre-interpretation I* if and only if *the $\exists SO$-formula $\mathcal{APP}(F)$ is satisfiable w.r.t. I.*

Proof. This follows immediately from the results in (Liu & Levesque, 1998) and Theorem 3.11. □

5.1.2 A more complete method

Unfortunately, many applications give rise to formulas in which the first-order part falls outside the class $\mathcal{NF}$, which means that completeness of our method is not guaranteed. Particularly troublesome in practice are formulas of the common form $\exists\bar{P}\forall\bar{Q} : \psi_1 \Rightarrow \psi_2$. For such formulas, the naive approximation method of the previous section tries to find interpretations for $\bar{P}$ such that the implication $\varphi = (\psi_1 \Rightarrow \psi_2)$ holds for all $\bar{Q}$. However, if we look at the details of the approximative definitions, we find that A^{ct}_{φ} is defined by a rule with a body $\psi_1^{cf} \vee \psi_2^{ct}$. In other words, the approximation will only derive that φ holds for all $\bar{Q}$ if it is either the case that ψ_1 is false for all $\bar{Q}$ or that ψ_2 is true for all $\bar{Q}$. However, this will rarely be the case. In most practical applications, the witnesses of interest will typically satisfy the implication $\psi_1 \Rightarrow \psi_2$ not because they always falsify ψ_1 or always satisfy ψ_2, but

rather because each interpretation for $\bar{Q}$ that satisfies ψ_1 also satisfies ψ_2. For instance, in the conformant planning example, there will always be interpretations for the fluents that do not satisfy the action theory ψ_{act}, because they arbitrarily assign some fluent a value that is wrong for its initial value and the actions that are performed. Even if a set of actions is a completely correct conformant plan, it therefore cannot make the goal certainly true, because it will still be unsatisfied in some of these wrong interpretations of the fluents. Of course, this should not bother a good method for finding conformant plans. The only thing that should matter is that the goal is satisfied in those interpretations of the fluents that do satisfy the action theory.

Luckily, our approximation method can also be used to discover this kind of witnesses. The only thing that is required is to add to the approximative definition $\Delta = \mathit{Approx}_\sigma(\varphi)$ a rule $A^{ct}_{\psi_1} \leftarrow \mathbf{t}$. By doing do, we seed our approximation with the assumption that ψ_1 holds. Starting from this assumption, the top-down rules will then derive properties of the predicates $\bar{Q}$ that are shared by all interpretations for $\bar{Q}$ that actually satisfy ψ_1. The bottom-up rules will then propagate this information upwards and discover whether these properties suffice to ensure that ψ_2 also holds. If they do, then we know that ψ_2 indeed must hold in every interpretation for $\bar{Q}$ that satisfies ψ_1 and that we therefore have found a witness for our formula.

If we want to find both witnesses of this kind and degenerate witnesses that either make ψ_1 false for all $\bar{Q}$ or ψ_2 true for all $\bar{Q}$, we could simply combine our new method with the old one and check either whether $A^{ct}_{\psi_2}$ holds according to $\Delta \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$ or whether A^{ct}_{φ} holds according to just Δ itself. However, it turns out that this is not necessary: we can achieve the same effect by just checking whether $\Delta \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$ implies A^{ct}_{φ}. This is because, first, the definition $\Delta \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$ will be able to derive A^{ct}_{φ} whenever Δ itself can: if Δ can derive that $A^{ct}_{\psi_2}$ then $\Delta \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$ will obviously still be able to do so; if Δ would be able to derive that $A^{cf}_{\psi_1}$, then $\Delta \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$ will also be able to do so, simply because our approximation has no flow of information between the ct and cf variants of the same formula, so the additional assumption that $A^{ct}_{\psi_1}$ holds will not change the original derivation of $A^{cf}_{\psi_1}$. Second, if $\Delta \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$ can derive $A^{ct}_{\psi_2}$ or $A^{cf}_{\psi_1}$, then it also derives A^{ct}_{φ}, simply because it contains the rule $A^{ct}_{\varphi} \leftarrow A^{cf}_{\psi_1} \vee A^{ct}_{\psi_2}$. Therefore, we can find both kinds of witnesses by checking whether A^{ct}_{φ} is implied by the single definition $\Delta \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$.

Definition 5.8. For an $\exists\forall SO$ formula $F = \exists\bar{P}\forall\bar{Q} : \psi$, where ψ is of the form $\psi_1 \Rightarrow \psi_2$, we define $\mathcal{APP}^{\Rightarrow}(F)$ as $\exists\bar{P}\exists\bar{R} : \Delta^{\Rightarrow} \wedge A^{ct}_{\psi}$, where $\Delta^{\Rightarrow} = \mathit{Approx}_\sigma(\psi_1 \Rightarrow \psi_2) \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$.

Note that if we take ψ_1 to be the trivial formula $\mathbf{t}$, we obtain Definition 5.5 as a special case.

This approximation method is still sound, as the following proposition states.

Proposition 5.9. *Given a formula F of the form $\exists\bar{P}\forall\bar{Q} : \psi$, where $\psi = \psi_1 \Rightarrow \psi_2$, the $\exists SO(ID)$ formula $\mathcal{APP}^{\Rightarrow}(F)$ is a sound approximation of F.*

Proof. Take a witness I for the satisfiability of $\mathcal{APP}^{\Rightarrow}(F)$. First let us remark that $Open\,(Approx_{\,\sigma}(\psi_1 \Rightarrow \psi_2) \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}) = \sigma$. From the fact that I is a witness of the satisfiability of $\mathcal{APP}^{\Rightarrow}(F)$ we know that the model M of this definition extending I contains A^{ct}_{ψ} and by construction of $Approx_{\,\sigma}(\psi_1 \Rightarrow \psi_2)$ it must also contain either $A^{cf}_{\psi_1}$ or $A^{ct}_{\psi_2}$.

Assume first that $A^{cf}_{\psi_1}$ is true in M. Then application of Theorem 3.13 (where we take $\Phi = \{\psi_1\}$ and $\varphi' = \psi_1$) gives: if M extends I and $M \vDash \psi_1$, then $M \nvDash \psi_1$, so the assumption that $M \vDash \psi_1$ results in a contradiction and hence $M \nvDash \psi_1$, in which case $\psi_1 \Rightarrow \psi_2$ holds for every M extending I, and thus is I a witness for the satisfiability of F.

Next, assume that $A^{ct}_{\psi_2}$ is true in M. Again applying Theorem 3.13 (where this time $\Phi = \{\psi_1\}$ and $\varphi' = \psi_2$) gives: if M extends I and $M \vDash \psi_1$ then $M \vDash \psi_2$, hence also in this case does $\psi_1 \Rightarrow \psi_2$ hold for every M extending I, and again this means that I is a witness of the satisfiability of F. □

Since the approximative definition $\Delta^{\Rightarrow}$ in $\mathcal{APP}^{\Rightarrow}(F)$ contains all the rules of $Approx\,(F)$ and because these are both monotone definitions, it is not hard to see that this new approximation method is at least as complete as the one using $\mathcal{APP}(F)$ (Definition 5.5). Moreover, as can be seen from the following example, it is also strictly more complete.

Example 5.10. Let us consider the following formula $F = \exists P \forall Q : (Q \Leftarrow P) \Rightarrow Q$. We have that $P = \mathbf{t}$ is clearly a witness for this satisfiability problem. If we denote $(Q \Leftarrow P) \Rightarrow Q$ by φ_1 and $(Q \Leftarrow P)$ by φ_2, then $\mathcal{APP}(F)$ is the following $\exists SO$ formula.

$$\exists P \bar{R} : \left\{ \begin{array}{lcl} A^{ct}_{\varphi_1} & \leftarrow & A^{cf}_{\varphi_2} \vee Q^{ct} \\ A^{cf}_{\varphi_1} & \leftarrow & A^{ct}_{\varphi_2} \wedge Q^{cf} \\ A^{ct}_{\varphi_2} & \leftarrow & \neg P \vee Q^{ct} \\ A^{cf}_{\varphi_2} & \leftarrow & P \wedge Q^{cf} \\ Q^{ct} & \leftarrow & A^{ct}_{\varphi_2} \wedge P \\ Q^{cf} & \leftarrow & A^{cf}_{\varphi_2} \end{array} \right\} \wedge A^{ct}_{\varphi_1}.$$

Now, even for $P = \mathbf{t}$, the definition in the body of this formula will not entail $A^{ct}_{\varphi_1} = \mathbf{t}$. Therefore, $\mathcal{APP}(F)$ is not satisfiable. On the other hand, $\mathcal{APP}^{\Rightarrow}(F)$ is the same formula as above, apart from that the definition contains one more rule, namely, the rule $A^{ct}_{\varphi_2} \leftarrow \mathbf{t}$. It is easy to verify that $\mathcal{APP}^{\Rightarrow}(F)$ *is* satisfiable, and indeed has $P = \mathbf{t}$ as a witness.

Obviously, the new method is still complete on formulas $\exists \forall \psi_1 \Rightarrow \psi_2$, where $\psi_1 \Rightarrow \psi_2$ satisfies the normal form $\mathcal{NF}$. However, our method also works for many formulas outside this class. Unfortunately, as before, it is difficult to characterize precisely how much more complete the new method is. For instance, as we showed in Chapter 3, one source of loss in completeness comes from the fact that our current translation to ENF cannot recognize multiple occurrences of the same subformula, and will introduce a different Tseitin predicate

for each occurrence. Even though we can not guarantee completeness for our method in general, it always found all solutions in the conformant planning benchmarks we considered in Section 5.3.

A final remark about this method is that the approximative definition $Approx_\sigma(\psi_1 \Rightarrow \psi_2)$ contains a number of rules that are superfluous in the context of $\mathcal{APP}^{\Rightarrow}$. Indeed, our method gives this definition as its inputs an interpretation for $\bar{P}$ and the assumption that ψ_1 is certainly true. It then uses the bottom-up and top-down rules derived from ψ_1 to compute the effect of these inputs on the predicates $\bar{Q}$. Finally, the rules derived from ψ_2 then compute whether the derived information about $\bar{Q}$ suffices to make ψ_2 certainly true. However, as we know from Theorem 3.11, only the bottom-up rules for ψ_2 are needed for this. Therefore, the top-down rules for ψ_2 actually contribute nothing and could just as well be removed. Adapting Definition 5.8 to use $\Delta^{\Rightarrow}_{BU} = \Delta^{\Rightarrow} \setminus Approx^{TD}_{\sigma}(\psi_2)$ instead of $\Delta^{\Rightarrow}$ leads to the following definition.

Definition 5.11. For an $\exists\forall SO$ formula $F = \exists\bar{P}\forall\bar{Q} : \psi$, where ψ is of the form $\psi_1 \Rightarrow \psi_2$, we define $\mathcal{APP}^{\Rightarrow}_{BU}(F)$ as $\exists\bar{P}\exists\bar{R} : \Delta^{\Rightarrow}_{BU} \wedge A^{ct}_{\psi}$, where

$$\Delta^{\Rightarrow}_{BU} = Approx_\sigma(\psi_1 \Rightarrow \psi_2) \setminus Approx^{TD}_{\sigma}(\psi_2) \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}.$$

It follows directly from Theorem 3.11 and Proposition 5.9 that this too is a sound approximation. Having removed the top-down rules from the approximation of ψ_2, the remaining rules just serve, as we already know, to compute the Kleene evaluation of ψ_2. They do this by computing the Kleene evaluation of each subformula γ of ψ_2, for which they use the Tseitin predicates A^{ct}_{γ} and A^{cf}_{γ}. An alternative is to avoid these Tseitin predicates by defining $A^{ct}_{\psi_2}$ directly by the single rule:

$$A^{ct}_{\psi_2} \leftarrow (\psi_2)^{ct}$$

This variant is summarized in the following definition.

Definition 5.12. For an $\exists\forall SO$ formula $F = \exists\bar{P}\forall\bar{Q} : \psi$, where ψ is $\psi_1 \Rightarrow \psi_2$, we define $\mathcal{APP}^{\Rightarrow}_{BU,Unf}(F)$ as $\exists\bar{P}\exists\bar{R} : \Delta^{\Rightarrow}_{BU,Unf} \wedge (A^{cf}_{\psi_1} \vee A^{ct}_{\psi_2})$, where

$$\Delta^{\Rightarrow}_{BU,Unf} = Approx_\sigma(\psi_1) \cup \{A^{ct}_{\psi_2} \leftarrow (\psi_2)^{ct}\} \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}.$$

The fact that all bottom-up rules for ψ_2 are positive and non-recursive, allows us to eliminate the Tseitin predicates introduced for the parse tree of ψ_2 by applying the *unfolding* procedure from (Tamaki & Sato, 1984). By iteratively applying this equivalence preserving procedure, we can reduce all the rules that were generated to approximate ψ_2 to just the single rule $A^{ct}_{\psi_2} \leftarrow (\psi_2)^{ct}$. A small difference between the approximation above and the previous approximations, however, is that we now no longer have a Tseitin predicate A^{ct}_{ψ}. Because

of this, we can no longer use our normal approximation of when the entire implication ψ is certainly true. However, the following reformulation of our original approximation for $\exists\forall SO$, which avoids the use of this A^{ct}_{ψ}, states that this is no issue, that the approximation above is equivalent to that of Def. 5.11

Proposition 5.13. *For an $\exists\forall SO$ formula $F = \exists\bar{P}\forall\bar{Q} : \psi$, where ψ is of the form $\psi_1 \Rightarrow \psi_2$, the approximation defined in Definition 5.8, i.e., the formula*

$$\exists\bar{P}\exists\bar{R} : (Approx_{\sigma}(\psi_1 \Rightarrow \psi_2) \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}) \wedge A^{ct}_{\psi}$$

is equivalent to

$$\exists\bar{P}\exists\bar{R} : (Approx_{\sigma}(\psi_1) \cup Approx_{\sigma}(\psi_2) \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}) \wedge (A^{cf}_{\psi_1} \vee A^{ct}_{\psi_2})$$

Proof. This is obvious from the fact that the difference between $Approx_{\sigma}(\psi_1 \Rightarrow \psi_2)$ and $Approx_{\sigma}(\psi_1) \cup Approx_{\sigma}(\psi_2)$ is precisely a set of rules that ensure that A^{ct}_{ψ} is equivalent to $A^{cf}_{\psi_1} \vee A^{ct}_{\psi_2}$. □

5.2 Approximating $\exists\forall SO(ID)$-satisfiability problems

We already argued in Chapter 2 that inductive definitions can make the task of representing a problem in logic considerably easier. An example of this is the use of inductively defined Situation Calculus for reasoning about actions. Recall that in the introduction of this chapter we showed how to represent T_{act} from Example 5.1 as an inductive definition Δ_{act}:

$$\left\{ \begin{array}{lcl} Clean(t+1) & \leftarrow & Clean(t). \\ Clean(t+1) & \leftarrow & Wipe(t). \\ Clean(0) & \leftarrow & InitiallyClean. \end{array} \right\}.$$

The associated conformant planning problem can then be expressed as an $\exists\forall SO(ID)$ satisfiability problem:

$$\exists Wipe\ \forall Clean, InitiallyClean : \Delta_{act} \Rightarrow Clean(n).$$

As we will show in more detail in Section 5.4, a general conformant planning problem can be seen as a satisfiability problem of the form

$$\exists\bar{A}\forall\bar{I}\forall\bar{F} : (\Delta_{act} \wedge \psi_{init}) \Rightarrow (\psi_{prec} \wedge \psi_{goal}),$$

where the predicates $\bar{A}$ represent the actions, $\bar{I}$ the initial fluents and $\bar{F}$ the other fluents. The definition Δ_{act} defines how the fluents change in terms of the actions, ψ_{init} is a first-order formula about the initial situation, ψ_{prec} describes the preconditions of the actions and

ψ_{goal} the goal. This motivates the extension of our approximation method to formulas including definitions. However, we do not analyze the general case where definitions may appear at arbitrary locations in a formula, but instead restrict attention to formulas of the form

$$\exists \bar{P} \forall \bar{Q} : (\Delta \wedge \psi_1) \Rightarrow \psi_2,$$

where Δ is a definition such that $Def(\Delta) \subseteq \bar{Q}$ and ψ_1 and ψ_2 are FO formulas. Even though these restrictions are not strictly necessary, they allow us to keep the technical details relatively simple (in particular, we avoid the need for approximation rules that infer that a definition as a whole is certainly true/false), while still covering the way in which definitions are typically used: under the assumption that all predicates indeed are what the definition Δ and the formula ψ_1 say they should be, ψ_2 then states what properties they should satisfy. All the applications in Section 5.4, such as conformant planning, are of this form.

The concepts of witness (Definition 5.3) and sound approximation (Definition 5.4) can straightforwardly be generalized for ψ being a FO(ID) formula. This allows us to develop three approaches. The first one replaces the definition by its completion and then applies the method of Section 5.1. However, the completion can be weaker than the definition. Therefore, we develop another approach, that uses results from Section 3.2 to approximate the valid propagations sanctioned by the conjunction of a definition with a FO formula.

Using the completion of a definition

First, let us recall that each definition Δ implies its completion $compl(\Delta)$. Replacing the definition by its completion in $(\Delta \wedge \psi_1) \Rightarrow \psi_2$ we obtain the formula $(comp(\Delta) \wedge \psi_1) \Rightarrow \psi_2$. As every model of Δ is a model of $comp(\Delta)$, every model of $(comp(\Delta) \wedge \psi_1) \Rightarrow \psi_2$ is a model of $(\Delta \wedge \psi_1) \Rightarrow \psi_2$ and every witness of the $\exists\forall\ (compl(\Delta) \wedge \psi_1) \Rightarrow \psi_2$ satisfiability problem is a witness of the $\exists\forall\ (\Delta \wedge \psi_1) \Rightarrow \psi_2$ satisfiability problem. Hence we can use the results of Section 5.1 and formulate the following proposition.

Proposition 5.14. *The formula* $\mathcal{APP}^{\Rightarrow}_{BU}((compl(\Delta) \wedge \psi_1) \Rightarrow \psi_2)$ *is a sound approximation of* $\exists \bar{P} \forall \bar{Q} (\Delta \wedge \psi_1) \Rightarrow \psi_2$.

Recall that the disadvantage of using the completion is that no matter how complete the approximation method defined in Definition 5.8 is, it will never be able to infer something that follows from Δ but not from $compl(\Delta)$. However, for definitions over a strict *well-founded order*, the definition is *equivalent* with its completion. The theory T_{act} in Example 5.1 is actually the completion of the definition Δ_{act}. Since Δ_{act} is a recursive definition over a strict well-founded order (we can make use of the time argument in the predicates to construct such a well-founded order), Δ_{act} and T_{act} are equivalent.

The Gaspipe conformant planning problem (Son et al., 2005), on the other hand, uses a dynamic domain for which the completion does not suffice. Summarized, the objective

of this conformant planning problem is to start a flame in a burner which is connected to a gas tank through a pipe line. The pipe line consists of sections connected to each other with valves. When a valve is opened with gas on one side and not on the other, the gas will spread as far as possible. This can be formalized by an inductive definition of the reachability relation on the pipe line:

$$\left\{ \begin{array}{l} \forall x, t\ Gas(x,t) \leftarrow \exists y\ Gas(y,t) \wedge \exists v\ Connected(x,y,v) \wedge Open(v,t). \\ \forall x, t\ Gas(x,t) \leftarrow Tank(x). \end{array} \right\}$$

Such reachability definitions are not equivalent to their completion. Therefore, the approximative method presented in this subsection will not always find a solution. In the next section we present a more complete approximation.

Using the certainly true/possibly true approximation

In the previous section, it was the property of $Approx_\sigma$, that it defines, for each subformula of ψ (including ψ itself), whether it is certainly true or certainly false, that allowed us to find witnesses by simply asserting that A^{ct}_ψ had to hold according to this definition. If we want to apply the same method to formulas ψ that contain a definition Δ, we have to construct an approximative definition that defines whether each of the subformulas of Δ (including Δ itself) is certainly true or certainly false. In the previous section, we managed to do this by simply replacing Δ by its completion. We now want to improve on this method by constructing an approximation that also takes into account the unfounded set reasoning that is performed by the well-founded semantics.

In Section 3.2, we proposed a ct/pt approximation $\mathcal{D}^{\Delta \wedge (compl(\Delta) \wedge \varphi)}$ for a definition Δ of all models of a definition that take unfounded set reasoning into account. However, it does not define when this definition as a whole is certainly true or certainly false. Luckily, this is not needed if we stick to our assumption that definitions appear only in the antecedent of the implication ψ. Indeed, because we approximate implications by assuming that their antecedent is certainly true (Definition 5.8), all that we really need is an approximation of the consequences of a definition.

To obtain a sound approximation for an $\exists\forall SO(ID)$ formula $\exists\bar{P}\forall\bar{Q}\,\psi$ with $\psi = (\Delta \wedge \varphi) \Rightarrow \psi_2$, we now just need to plug in our approximation $\mathcal{D}^{\Delta \wedge (compl(\Delta) \wedge \varphi)}$ for Δ into a suitable $\exists SO(ID)$ formula, similar to the one we defined in Definition 5.8 for an $\exists\forall SO$ formula $\exists\bar{P}\forall\bar{Q}\,\psi_1 \Rightarrow \psi_2$. Again we have that our approximation of a definition does not define predicates A^{ct}_Δ and A^{cf}_Δ that tell us when the definition Δ as a whole is certainly true or certainly false, and therefore, we can not use our normal approximation of when the entire implication ψ is certainly true. Instead, we have to use the reformulation from Proposition 5.13 again. Our approximation for $\exists\forall SO(ID)$ now essentially consists of just

replacing $Approx_\sigma(\psi_1) \cup \{A^{ct}_{\psi_1} \leftarrow \mathbf{t}\}$ and $A^{cf}_{\psi_1}$ respectively by $\mathcal{D}^{\Delta \wedge compl(\Delta) \wedge \varphi}_\sigma$ and A^{cf}_φ in the Proposition 5.13.

Proposition 5.15. *Given an $\exists\forall SO(ID)$ formula $F = \exists\bar{P}\forall\bar{Q}\,(\Delta \wedge \varphi) \Rightarrow \psi_2$. We define $\mathcal{APP}^{wf}(F)$ as the following $\exists SO(ID)$ formula.*

$$\exists\bar{P}\bar{R}:\ (\mathcal{D}^{\Delta \wedge compl(\Delta) \wedge \varphi}_\sigma \cup Approx_\sigma(\psi_2)) \wedge (A^{cf}_\varphi \vee A^{ct}_{\psi_2}).$$

Then $\mathcal{APP}^{wf}(F)$ is a sound approximation of F.

In some cases, using the approximating $\mathcal{D}^{\Delta \wedge compl(\Delta) \wedge \varphi}_\sigma$ instead of $\mathcal{D}^{\Delta \wedge \varphi}_\sigma$ will not gain us anything. For instance, consider an $\exists\bar{P}\forall\bar{Q}(\Delta \wedge \varphi) \Rightarrow \psi_2$ problem, where φ only contains *open predicates* of Δ (as is the case for the conformant planning problems we consider in the next sections). In this case, we will never need head-to-body propagation, and therefore $\mathcal{D}^{\Delta \wedge \varphi}_\sigma$ is just as complete as $\mathcal{D}^{\Delta \wedge compl(\Delta) \wedge \varphi}_\sigma$, and we are therefore better off using the former.

As was the case for our approximation method for $\exists\forall SO$, here as well, not all rules from the definition in $\mathcal{APP}^{wf}$ are necessary. Indeed, only the bottom-up rules from $Approx_\sigma(\psi_2)$ are needed, and can be unfolded into a single rule. Therefore, below we define two more variants of Definition 5.15.

Definition 5.16 ($\mathcal{APP}^{wf}_{BU}(F)$)**.** Given an $\exists\forall SO(ID)$ formula $F = \exists\bar{P}\forall\bar{Q}\,(\Delta \wedge \varphi) \Rightarrow \psi_2$.

We define $\mathcal{APP}^{wf}_{BU}(F)$ as the following $\exists SO(ID)$ formula,

$$\exists\bar{P}\bar{R}:\ (\mathcal{D}^{\Delta \wedge compl(\Delta) \wedge \varphi}_\sigma \cup Approx^{BU}_\sigma(\psi_2)) \wedge (A^{cf}_\varphi \vee A^{ct}_{\psi_2}),$$

and we define $\mathcal{APP}^{wf}_{BU,Unf}(F)$ as the following $\exists SO(ID)$ formula,

$$\exists\bar{P}\bar{R}:\ (\mathcal{D}^{\Delta \wedge compl(\Delta) \wedge \varphi}_\sigma \cup \{A^{ct}_{\psi_2} \leftarrow (\psi_2)^{ct}\}) \wedge (A^{cf}_\varphi \vee A^{ct}_{\psi_2}).$$

5.3 Experimental evaluation

In this chapter we have seen a number of methods to approximate $\exists\forall SO$ and $\exists\forall SO(ID)$ satisfiability problems. In this subsection, we explore, through a number of experiments, how we can use these methods to solve practically useful problems as fast as possible. We performed these experiments on a number of conformant planning benchmarks from the paper of Son et al. (2005). As we show in Section 5.4, all these benchmarks are of the form $F = \exists\bar{P}\forall\bar{Q}\Delta \Rightarrow \psi$, where Δ is a stratified definition, and is therefore equivalent to its completion. Therefore, F is equivalent to the $\exists\forall SO$ formula $\exists\bar{P}\forall\bar{Q}\,compl(\Delta) \Rightarrow \psi$, which we denote by F^{cp}. All experiments were run on a dual core 2.4 Ghz CPU, 2.8 Gb RAM

Linux machine, using the IDP model expansion system for FO(ID) (Mariën et al., 2006). A time-out of twenty minutes was used.

A first question we want to answer is whether for these definitions, the completion based approximation is faster than the ct/pt approximation. It is not hard to see that, even though $Approx(compl(\Delta))$ is linear in the size of the parse tree of $compl(\Delta)$, this definition may contain more rules than $App^{ct/pt}(\Delta)$, and moreover, these rules may contain a lot of recursion. This can pose a challenge for current solvers, and suggests that it is likely to be more efficient to use the ct/pt approximation for definitions. The first column in Table 5.1 shows times for using the completion of the definition Δ, that is, $\mathcal{APP}^{\Rightarrow}(F^{cp})$, while for the second column, the ct/pt-approximation of Δ was used, that is, $\mathcal{D}^{\Delta}$. As expected, the ct/pt-approximation was consistently faster.

Table 5.1 also compares solving times of the full completion-based approximative definition (in the first column) with the approximation $\mathcal{APP}^{\Rightarrow}_{BU}(F^{cp})$ (Def. 5.11), from which the top-down propagation rules for ψ have been removed (third column). We see that in the BT and BTC benchmarks we get an order of magnitude improvement. The fourth column of Table 5.1 shows timings for the unfolded approximation of ψ_2, $\mathcal{APP}^{\Rightarrow}_{BU,Unf}(F^{cp})$, in which the intermediate Tseitin predicates have been removed (Def. 5.12). We see that this unfolding consistently provides a speed-up.

These results suggest that combining the above techniques, that is, using the ct/pt approximation for Δ and the unfolding of the bottom-up approximation of ψ_2 together, will give us the fastest way of approximating $\exists\forall(\Delta \wedge \psi_1) \Rightarrow \psi_2$ satisfiability problems. Indeed, this is what formula $\mathcal{APP}^{wf}_{BU,Unf}(F)$ (Definition 5.16) does, and the results of this method are shown in the last column of Table 5.1. As expected, this is by far the fastest method.

5.4 Applications and related work

In the literature, many examples can be found of approaches that perform some kind of approximate reasoning about the models of a logical theory. Often, these approaches, which are specific to the problem at hand, boil down to an instantiation of the general methods presented here. In this section we give some examples.

5.4.1 Conformant Planning

In general, a *conformant planning problem* is a planning problem in a non-deterministic domain where the initial state may be not fully known. The goal is to come up with a plan (i.e., a sequence of actions) that is nevertheless guaranteed to work. This is a hard problem: the decision problem of deciding whether a conformant plan with a fixed length k exists is

Problem	$\mathcal{APP}^{\Rightarrow}(F^{cp})$	$\mathcal{APP}^{wf}(F)$	$\mathcal{APP}^{\Rightarrow}_{BU}(F^{cp})$	$\mathcal{APP}^{\Rightarrow}_{BU,Unf}(F^{cp})$	$\mathcal{APP}^{wf}_{BU,Unf}(F)$
BT(2,2)	0.597	0.111	0.083	0.050	**0.031**
BT(4,2)	3.38	3.521	0.226	0.099	**0.049**
BT(6,2)	39.0	14.74	0.785	0.308	**0.095**
BT(8,4)	—	—	15.3	1.643	**0.486**
BT(10,4)	—	—	662	—	**1.411**
BTC(2,2)	0.203	0.140	0.104	0.078	**0.048**
BTC(4,2)	—	—	7.89	4.92	**0.099**
BTC(6,2)	—	—	—	—	**0.321**
BTC(8,4)	—	—	—	—	**42.5**
Domino(100)	0.435	0.032	0.408	0.412	**0.031**
Domino(200)	1.123	0.035	1.13	1.13	**0.028**
Domino(500)	6.62	0.066	6.56	6.56	**0.050**
Domino(1000)	31.4	0.119	31.9	31.5	**0.087**
Domino(2000)	—	0.241	—	—	**0.170**
Ring(2)	7.042	0.363	2.908	3.401	**0.109**
Ring(4)	—	—	—	—	**0.283**
Ring(6)	—	—	—	—	**5.35**
Ring(8)	—	—	—	—	**156**
Ring(10)	—	—	—	—	**1139**

Table 5.1: The first column gives the name of the benchmark and the other ones different execution times. The second column gives the execution time in seconds for the approximation of the completion and the third for the cp/pt approximation. The fourth and fifth column use variants of the completion approximation. For the fourth column, the top-down rules for ψ_2 are removed while in addition, for the fifth column, the remaining bottom-up rules are unfolded. The last column combines the cp/pt approximation with both other changes. "-" means the execution was interrupted after 20 minutes.

Σ_2^P-complete[2] (Baral et al., 2000; Turner, 2002). Therefore, one typically attempts to solve it approximately. In this section, we show how we can apply our approximative methods to solve conformant planning problems.

Example 5.17. Let us consider the Clogged Bombs in the Toilet domain (McDermott, 1987; Son et al., 2005). There are a number of packages and a toilet. Each of the packages may contain a bomb which can be disarmed by dunking the package in the toilet. Dunking a package into a toilet also clogs the toilet and we can not throw a package in a clogged toilet. Flushing the toilet unclogs it. The effects of the actions on the fluents are modeled by the following definition Δ_{act}, and the preconditions by the conjunction ψ_{prec} of sentences in T_{prec}.

[2] For planning domains where the executability of actions in a given state cannot be determined polynomially, this is even Σ_3^P (Turner, 2002)

$$\Delta_{act} = \left\{ \begin{array}{l} \mathit{Clogged}(0) \leftarrow \mathit{Init\_Clogged}. \\ \mathit{Clogged}(t+1) \leftarrow \exists p : \mathit{Dunk}(p,t) \vee (\mathit{Clogged}(t) \wedge \neg \mathit{Flush}(t)). \\ \mathit{Armed}(p,0) \leftarrow \mathit{Init\_Armed}(p). \\ \mathit{Armed}(p,t+1) \leftarrow \mathit{Armed}(p,t) \wedge \neg \mathit{Dunk}(p,t). \end{array} \right\}$$

$$T_{prec} = \begin{array}{l} \forall\, p\, t : \mathit{Dunk}(p,t) \Rightarrow \neg \mathit{Clogged}(t). \\ \neg(\exists\, p\, t : \mathit{Dunk}(p,t) \wedge \mathit{Flush}(t)). \\ \forall\, p\, p_2\, t : \mathit{Dunk}(p,t) \wedge \mathit{Dunk}(p_2,t) \Rightarrow p = p_2. \\ \forall\, p\, t\, t_2 : \mathit{Dunk}(p,t) \wedge \mathit{Dunk}(p,t_2) \Rightarrow t = t_2. \end{array}$$

Now consider the following regular planning problem: given a completely specified initial situation (specified by a formula ψ_{init}), find a plan such that all packages are disarmed. We can formulate this problem as the following formula:

$$\exists \bar{A}, \bar{F}, \bar{I} : \Delta_{act} \wedge \psi_{prec} \wedge \psi_{init} \wedge (\exists t\, \forall p\, \neg \mathit{Armed}(p,t)),$$

where with $\bar{A}$, we denote the action predicates $\{\mathit{Dunk}/2, \mathit{Flush}/1\}$, with $\bar{F}$ we denote the fluent predicates $\{\mathit{Armed}/2, \mathit{Clogged}/1\}$ and with $\bar{I}$ we denote the predicates used to describe the initial situation $\{\mathit{Init\_Clogged}/0, \mathit{Init\_Armed}/1\}$. Now imagine that initial situation is not specified, and we want to find a plan that works for all possible initial situations, in other words a *conformant plan*. We can formulate the problem of finding such a plan as follows.

$$\exists \bar{A}\, \forall \bar{F}, \bar{I} : \Delta_{act} \Rightarrow (\psi_{prec} \wedge \exists t\, \forall p\, \neg \mathit{Armed}(p,t)).$$

All this can be formalized in general as follows.

Definition 5.18 (Conformant planning)**.** Let Σ be a vocabulary, consisting of a set of predicates $\overline{A}$, denoting *actions*, $\overline{I}$, denoting *initial fluents*, and $\overline{F}$ denoting *fluents*. Let T_{act} be an FO(ID) theory and T_{init}, T_{prec} and T_{goal} FO theories, all over Σ, such that T_{act} specifies the values of the fluents given an interpretation for the actions and initial fluents, T_{init} is a theory specifying the initial situation, T_{prec} contains preconditions for the actions, and T_{goal} specifies the goal of the planning problem. With ψ_{act} we denote the conjunction of the sentences and possibly definitions in T_{act} and similarly for the other theories. The problem of conformant planning is then to decide the satisfiability of the following formula:

$$\exists \overline{A}\, \forall \overline{I}\, \forall \overline{F} : (\psi_{act} \wedge \psi_{init}) \Rightarrow (\psi_{prec} \wedge \psi_{goal}). \tag{5.6}$$

$$
\exists \bar{A}\bar{R}: \left\{ \begin{array}{rcl}
Clogged^{ct}(0) & \leftarrow & Init\_Clogged^{ct}. \\
Clogged^{ct}(t+1) & \leftarrow & \exists p : Dunk(p,t) \vee (Clogged^{ct}(t) \wedge \neg Flush(t)). \\
Armed^{ct}(p,0) & \leftarrow & Init\_Armed^{ct}(p). \\
Armed^{ct}(p,t+1) & \leftarrow & Armed(p,t) \wedge \neg Dunk(p,t). \\
Clogged^{pt}(0) & \leftarrow & Init\_Clogged^{pt}. \\
Clogged^{pt}(t+1) & \leftarrow & \exists p : Dunk(p,t) \vee (Clogged^{pt}(t) \wedge \neg Flush(t)). \\
Armed^{pt}(p,0) & \leftarrow & Init\_Armed^{pt}(p). \\
Armed^{pt}(p,t+1) & \leftarrow & Armed(p,t) \wedge \neg Dunk(p,t). \\
Init\_Clogged^{ct} & \leftarrow & \mathbf{f}. \\
Init\_Armed^{ct}(p) & \leftarrow & \mathbf{f}. \\
Init\_Clogged^{pt} & \leftarrow & \neg Init\_Clogged^{cf}. \\
Init\_Clogged^{cf} & \leftarrow & \mathbf{f}. \\
Init\_Armed^{pt}(p) & \leftarrow & \neg Init\_Armed^{cf}(p). \\
Init\_Armed^{cf} & \leftarrow & \mathbf{f}. \\
Clogged^{cf}(t) & \leftarrow & \neg Clogged^{pt}(t). \\
Armed^{cf}(p,t) & \leftarrow & \neg Armed^{pt}(p,t). \\
A^{ct}_{\psi_2} & \leftarrow & \forall pt : Dunk(p,t) \Rightarrow Clogged^{cf}(t) \\
 & & \wedge \neg(\exists pt : Dunk(p,t) \wedge Flush(t)) \\
 & & \wedge \forall p_1 p_2 t : Dunk(p,t) \wedge Dunk(p_2,t) \Rightarrow p_1 = p_2 \\
 & & \wedge \forall p t_1 t_2 : Dunk(p,t_1) \wedge Dunk(p,t_2) \Rightarrow t_1 = t_2 \\
 & & \wedge \exists t \forall p : Armed^{cf}(p,t).
\end{array} \right\}
$$

$$\wedge$$

$$A^{ct}_{\psi_2}.$$

Figure 5.1: The complete approximation of the Clogged Bombs in the Toilet example.

In words, there must be a plan ($\exists \bar{A}$), such that no matter how the nondeterministic aspects turn out ($\forall \bar{I}, \bar{F}$), as long as the specification of the effects of the actions (ψ_{act}) and the (partial) specification of the initial situation (ψ_{init}) are obeyed, the plan will be executable (ψ_{prec}) and achieve the goal (ψ_{goal}).

Formula 5.6 is now exactly of the form we assumed above, and we can thus use one of our methods to approximate conformant planning problems.

Example 5.17. (continued) Continuing the Clogged Bombs in the Toilet example, by using the ct/pt-approximation for the definition, and unfolding the constraint $(\psi_{prec} \wedge \exists t \, \forall p \, \neg Armed(p,t))^{ct}$, we get the approximating $\exists SO$ formula $\mathcal{APP}^{wf}_{BU,Unf}$ (Definition 5.16), shown in Figure 5.1, where $\bar{R}$ are the ct- and cf-predicates introduced by the approximation method.

The result of applying our general approximation method to a conformant planning problem, as specified by T_{act}, T_{prec}, T_{goal} and T_{init} as above, is very similar to the approximation of an $\mathcal{AL}$ action theory by a logic program in the work of Son et al. (2005). However, there are some small differences in the details that make it difficult to formally compare the two. Nevertheless, for all experiments discussed in this section, our method always finds a correct solution (unless it times out), as does the method of Son et al. Moreover, the two approaches also found these solutions in comparable execution times.

In more detail, Table 5.2 presents the following results. We implemented a conformant planner by iteratively calling the IDP model generator for FO(ID) (Mariën et al., 2006) on our approximation, giving it an increasing number of timesteps until either a plan is found or a maximum number of timesteps is reached. We then compared this planner to the CP_{ASP} conformant planner (Son et al., 2005), using the same experimental setup as in Section 5.3. CP_{ASP} takes an action theory in the action language $\mathcal{AL}$, and encodes an approximation of the transition diagram corresponding to that action theory, by means of an answer set program. Then any answer set solver can be used to find conformant plans. As Son et al., we used as the ASP solver behind CP_{ASP} both CModels (E. Giunchiglia & Maratea, 2011) and SModels (Niemelä et al., 2000). As Table 5.2 shows, the combination of our approximation and the IDP system is comparable to the combination of CModels and Son et al.'s approximation. When compared to the same approximation given to SModels, our method tends to be a bit better. These results are in line with results from the ASP competition (Denecker et al., 2009) concerning the performance of SModels, CModels and IDP in general, suggesting that, for conformant planning, our approximation and that of Son et al. are of comparable quality.

Another approximative method for solving conformant planning problems can be found in the work of Palacios & Geffner (2009). In their paper, the authors consider conformant planning problems, specified in the language Strips extended with conditional effects and negation. They define a transformation K_0 that transforms such a conformant planning problem into a classical planning problem in a sound but incomplete way. For each fluent literal L in the conformant planning specification, two new literals KL and $K\neg L$ are created, denoting that L is known to be true, resp. known to be false, and the initial situation, action preconditions and effects are translated into an initial situation, preconditions and effects with reference to these new knowledge literals. It is not hard to verify that our approximation method generalizes this transformation: if we take an $\exists\forall SO$ encoding of a conformant planning problem P, the $\exists SO$ approximation obtained by our method can be interpreted as a classical planning problem in the ct/cf vocabulary. Apart from the initial situation, this planning problem will exactly be the planning problem specified by $K_0(P)$ (i.e., the action preconditions and effects correspond). The K_0 transformation does not do propagation on knowledge about the initial situation: given an initial situation I (specified as a set of clauses), $K_0(I)$ consists of only those literals KL where L is a *unit clause* in I. This means that, e.g., for an initial situation $I = \{P \vee Q, \neg P\}$, $K_0(I)$ will not include the literal KQ, while

Problem	IDP	Smodels	Cmodels
BT(2,2)	0.031	0.272	**0.028**
BT(4,2)	**0.049**	0.132	0.074
BT(6,2)	**0.095**	0.591	0.236
BT(8,4)	**0.486**	33.9	2.21
BT(10,4)	**1.41**	-	5.53
BTC(2,2)	0.048	0.138	**0.038**
BTC(4,2)	**0.099**	0.220	0.210
BTC(6,2)	**0.321**	3.71	0.997
BTC(8,4)	**42.5**	-	284
Cleaner(2,2)	**0.081**	0.199	0.116
Cleaner(2,5)	**0.313**	132	1.08
Cleaner(2,10)	-	-	-
Cleaner(4,2)	**0.455**	13.6	0.803
Cleaner(4,5)	-	-	-
Cleaner(4,10)	-	-	-
Cleaner(6,2)	6.35	-	**5.97**
Cleaner(6,5)	-	-	-
Domino(100)	0.031	0.050	**0.015**
Domino(200)	**0.028**	0.075	0.078
Domino(500)	**0.050**	0.143	0.148
Domino(1000)	**0.087**	0.700	0.723
Domino(2000)	**0.170**	1.40	1.49
Ring(2)	**0.109**	0.128	0.116
Ring(4)	**0.285**	1.837	0.681
Ring(6)	5.36	1134	**3.262**
Ring(8)	156	-	**20.0**
Ring(10)	1139	-	**240**

Table 5.2: Comparison IDP vs Cmodels vs Smodels. The results are given in seconds.

our method will be able to infer that Q^{ct} holds. This means our approximation method will be more complete than the K_o transformation.

Being a general method, ours does not only allow for solving conformant planning problems, but also allows for approximating a number of related problems in temporal domains. Consider, for example, the following problem: 'Given that some action A happens at timepoint t, will this certainly lead to a property ψ being true?' This can be formalized as the following $\forall SO$ satisfiability problem, to which our method applies again.

$$\forall \bar{A}\bar{I}\bar{F} : ((\Delta_{act} \wedge \psi_{init} \wedge \psi_{prec} \wedge A(t)) \Rightarrow \psi).$$

This formula is true if for *all possible plans* in which $A(t)$ happens, the property ψ holds. A variant of this problem is the so-called *projection problem*: 'Given that we exactly know which actions happened (we can thus assume that the preconditions were satisfied), does the property ψ hold?' In order to formulate this problem as a SO satisfiability problem, we need to express *that these and only these* actions happened. This can be done, for example, by using an inductive definition $\Delta_{\bar{A}}$. The projection problem can then be expressed as the $\forall \bar{A}\bar{I}\bar{F} : ((\Delta_{act} \wedge \psi_{init} \wedge \Delta_{\bar{A}}) \Rightarrow \psi)$ satisfiability problem. Another variant is the following problem: 'If the property ψ_1 holds for a certain plan, does property ψ_2 also hold?', which can be expressed as the $\forall \bar{A}\bar{I}\bar{F}((\Delta_{act} \wedge \psi_{init} \wedge \psi_{prec} \wedge \psi_1) \Rightarrow \psi_2)$ satisfiability problem.

5.4.2 Querying and reasoning in open databases

Approximate methods similar to ours have been used in the context of databases without complete information, in particular in databases without *closed world assumption* (CWA), such as *open* databases (Baral et al., 1998; Liu & Levesque, 1998) or databases that make forms of *local closed world assumptions* (Denecker et al., 2010a; Doherty et al., 2006). In most of these papers the goal is to compute certain or possible answers to queries. Because this task has a high complexity (from CoNP for a locally closed database without integrity constraints to possibly Δ_2^P for databases with first-order constraints - assuming a given finite domain), approximate methods are presented which translate an FO query into an approximate FO or FO(FP) query that can be solved directly against the database tables using standard (polynomial) query methods.

The method presented in this chapter can provide a similar functionality. Let DB be a set of ground literals, representing an incomplete database. Let Ψ be a background theory: it may contain integrity constraints, view definitions (datalog view programs are a special case of FO(ID) definitions), local closed world statements expressed in FO, etc. For a given FO query $\varphi_Q[\bar{x}]$, the goal is to find all tuples $\bar{d}$ such that $\varphi_Q[\bar{d}]$ holds in all models of $DB \cup \Psi$. The problem of deciding whether a *given* tuple $\bar{d}$ is an answer corresponds to the satisfiability problem of the formula

$$\forall \bar{R}(DB \wedge \Psi \Rightarrow \varphi_Q[\bar{d}]), \tag{5.7}$$

and we can directly use our approximation method on this problem. While this allows us to answer yes/no queries as well as to decide whether a given tuple $\bar{d}$ is a certain answer to a query, our approximation method does not directly provide a method to compute (an approximation of) all such tuples.

However, let us look at the following $\exists SO$ satisfiability problem.

$$\exists \bar{R}' : \mathcal{D}^{DB \wedge \Psi} \cup Approx^{BU}(\varphi_Q[\bar{x}]),$$

It looks very much like our approximation of $\exists\forall(\Delta \wedge \psi_1) \Rightarrow \psi_2$ satisfiability problems (as formulated in Proposition 5.15). We again have the definition $\mathcal{D}^{DB\wedge\Psi}$ approximating the database DB and background knowledge Ψ (note that Ψ possibly contains definitions), and the bottom up evaluation of the query, only now the constraint A^{ct}_Q has been dropped.

The definition $\mathcal{D}^{DB\wedge\Psi} \cup Approx^{BU}(\varphi_Q[\bar{x}])$ consists of rules describing propagations allowed by the database and the theory Ψ, and rules defining the predicate symbol $A^{ct}_{\varphi_Q}$, where A_{φ_Q} is the Tseitin predicate representing the query $\varphi_Q[\bar{x}]$. In the unique Herbrand model of this definition, the interpretation of $A^{ct}_{\varphi_Q}$ contains those tuples for which our propagation can derive that they certainly satisfy the query – a sound approximation of the full set of answers!

In (Denecker et al., 2010a), a *local closed database* ($LCDB$) is assumed. Such a locally closed database consists of a standard database (DB), that is, a set of atoms, together with a set of *local closed world assumptions* $\mathcal{LCWA}(P(\bar{x}), \Psi[\bar{x}])$. Each of these LCWA statements expresses that the database's knowledge about P is complete for those tuples $\bar{x}$ that satisfy the formula $\Psi[\bar{x}]$. An atom $P(\bar{d})$ is therefore true if it is in DB and it is false if it is not in DB and there is a $\mathcal{LCWA}(P(\bar{x}), \Psi[\bar{x}])$ such that $\Psi[\bar{d}]$ holds in the domain of discourse; otherwise it is unknown. The authors then present an approximate reasoning method for query answering in locally closed databases and show how this approximate query answering can be formulated as a fixpoint query. Basically, this boils down to the following. One constructs the following definition

$$\Delta_{\mathcal{LCWA}} = \left\{ \begin{array}{l} \cdots \\ P^{ct}(\bar{x}) \leftarrow P(\bar{x}) \\ P^{cf}(\bar{x}) \leftarrow \neg P^{ct}(\bar{x}) \wedge \Psi^{ct}_P[\bar{x}] \\ \cdots \end{array} \right\},$$

for every relation P in the database DB and for every local closed world assumption $\mathcal{LCWA}(P(\bar{x}), \Psi[\bar{x}])$. Although the authors do not phrase it in this form, their method for finding an approximation of the certain answers to a query $\varphi_Q[\bar{x}]$ actually boils down to solving the following satisfiability problem:

$$\exists \bar{R}' : DB \wedge CWA(DB) \wedge \Delta_{\mathcal{LCWA}} \wedge Approx^{BU}(\varphi_Q[\bar{x}]),$$

Here $\bar{R}'$ denotes all predicates and auxiliary predicates occurring in the body of the existential formula. With $CWA(DB)$, we denote the formula expressing *closed world assumption* for the database DB. The presence of this closed world assumption might seem strange at first sight, since the whole idea behind locally closed world databases is to not assume CWA per default. However, in order to correctly apply the local closed world assumptions, we need an exact specification of what is in the database and what is not, and this is precisely what is expressed by $DB \wedge CWA(DB)$. Indeed, given $DB \wedge CWA(DB)$, $\Delta_{\mathcal{LCWA}}$ can be seen as an approximative definition of what is certainly true and false in the context of the locally closed world assumptions. Again the predicate $A^{ct}_{\varphi_Q}$ will contain an approximate answer to the query $\varphi_Q[\bar{x}]$, i.e., a lower bound on all the tuples for which the query $\varphi_Q[\bar{x}]$ is certainly true. Similarly, the predicate $A^{cf}_{\varphi_Q}$ will contain a lower bound on all the tuples for which the query is false.

A limitation of the approach in (Denecker et al., 2010a) is that they extend the above method for only one type of integrity constraints, namely functional dependencies. The way these functional dependencies are handled is by extending $\Delta_{\mathcal{LCWA}}$ with extra propagation rules taking these functional dependencies into account. By contrast, our more general method can be used to easily extend this to *arbitrary* integrity constraints. This works as follows. Let T_{int} be a set of first-order integrity constraints. We can then approximate the problem of finding certain queries by the following satisfiability problem.

$$\exists \bar{R}' : Approx(T_{int}) \cup \{A_{T^{ct}_{int}} \leftarrow \mathbf{t}\} \wedge DB \wedge CWA(DB) \wedge \Delta_{\mathcal{LCWA}} \wedge Approx^{BU}(\varphi_Q[\bar{t}]).$$

Again, the predicate $A^{ct}_{\varphi_Q}$ will contain an approximate answer to the query $\varphi_Q[\bar{x}]$.

In (Doherty et al., 2006), another approach to asking queries to an incomplete database is presented. The authors use the term *approximate database* to denote a database, consisting of two layers: an extentional and an intensional layer. Both of these layers have an external representation towards the user, and an internal representation.

The *extensional* database consists of positive and negative literals, and is internally stored as a classical database, using the Feferman transformation (Feferman, 1984). For example, the extensional database (EDB), as entered by a user,

$$Color(Car1, Black), \neg Color(Car1, Red), Color(Car2, Red),$$

is internally stored as

$$Color^{ct}(Car1, Black), Color^{cf}(Car1, Red), Color^{ct}(Car2, Red).$$

The *intensional* database consists of rules to infer additional information from the facts in the EDB. The user can write down rules of the form $(\neg)P_1(\bar{x}_1) \wedge \ldots \wedge (\neg)P_n(\bar{x}_n) \rightarrow (\neg)P(\bar{x}))$, which are then internally stored as $((\neg)P_1(\bar{x}_1))^{ct} \wedge \ldots \wedge ((\neg)P_n(\bar{x}_n))^{ct} \rightarrow ((\neg)P(\bar{x})))^{ct}$. An example of such a IDB rule is the following rule

$$Color(x, y_1) \wedge y_1 \neq y_2 \rightarrow \neg Color(x, y_2),$$

which is internally stored as

$$Color^{ct}(x, y_1) \wedge y_1 \neq y_2 \rightarrow Color^{cf}(x, y_2).$$

To evaluate a query, a naive algorithm based on exhaustively applying all rules on the EDB is used.

The rules in the IDB resemble our INF formulas in the sense that both describe *valid inferences* that can be made from incomplete information. The internal representation of the IDB is indeed similar to our representation of INF formulas as definitional rules. However, a key difference is that in the approach of Doherty, when a user wants to add a property ψ to the database (e.g., 'a car can only have one color'), he has to write down all inferences that are valid according to that property, while in our approach these inference rules are automatically generated from the property itself. Manually writing down all valid inferences sanctioned by a property is not an easy task. For example, take the property "a car has to be inspected if and only if it was suspect and black" from (Doherty et al., 2006). This can be expressed in FO as the formula $\psi = \forall c(Suspect(c) \wedge Color(c, Black) \Leftrightarrow Investigate(c))$. While, in our method, $Approx(\psi)$ constructs an approximation of all valid inferences that can be made from this formula, the user has to write down the following rules in Doherty's approach:

$$\begin{gathered} Suspect(c) \wedge Color(c, Black) \rightarrow Investigate(c) \\ Suspect(c) \wedge \neg Investigate(c) \rightarrow \neg Color(c, Black) \\ \neg Suspect(c) \rightarrow \neg Investigate(c) \\ \ldots \end{gathered}$$

Our method therefore generalizes (Doherty et al., 2006) by deriving these rules automatically from a general first-order theory.

In (Liu & Levesque, 1998) another type of reasoning in open databases is proposed. They only consider a simple form of first-order knowledge bases, called *proper knowledge bases*. An interesting feature of these knowledge bases is that it is easy to obtain a *complete* characterization of what is certainly true, resp. certainly false. In our terminology, that means that one can construct a definition Δ, such that $KB \models P(\bar{d})$ if and only if $\Delta \models P^{ct}(\bar{d})$ and $KB \models \neg P(\bar{d})$ if and only if $\Delta \models P^{cf}$. Then Liu & Levesque use an evaluation procedure based on the three-valued Kleene-evaluation to check whether a query holds in the knowledge base. As mentioned earlier, they also define a normal form $\mathcal{NF}$ for queries, for which they prove that the Kleene-evaluation is complete. Our work extends their work, in the sense that we can take a general first-order knowledge base and approximately solve queries, as we have shown above. Of course, since in general we can no longer guarantee a complete characterization of what is certainly true/false, we can no longer guarantee completeness, even if the query is in the normal form $\mathcal{NF}$. Another difference between the work of (Liu & Levesque, 1998) and our work here, is that they assume a fixed countably *infinite* domain, while we assume a fixed finite domain. While this is indeed a theoretical

difference, in practice it does not make any difference, since their evaluation method only considers a *finite* set of domain elements that can be determined up-front.

5.5 Conclusions

Even if a problem is computationally hard in general, specific instances of it might still be solved efficiently. This is why approximate methods are important: they cannot solve every instance, but the instances they can solve, they solve quickly. In computational logic, hard problems arise quite readily. It is therefore not surprising that the literature contains numerous examples of algorithms that perform approximate reasoning tasks for various logical formalisms in various specific contexts. Since many of these algorithms share common ideas, it is a natural question whether they can be seen as instances of some more general method for a more general language.

This chapter presents such a method. We start from the propagation method presented in Section 3 and its symbolic expression and generalize this to a method for approximating the Σ_2^P-complete $\exists\forall SO(ID)$ satisfiability problem by solving an NP problem. Importantly, this is a syntactic method that transforms the $\exists\forall SO(ID)$ formula into an $\exists SO(ID)$ formula. This affords us the freedom to use any off-the-shelf solver for such a language to perform the approximative reasoning. Moreover, it also makes it significantly easier to update the method by adding (or removing) specific propagations.

Since our method is an approximation, it is necessarily incomplete. Nevertheless, our experiments have shown that, in practice, it often does manage to find a solution. In particular, we used our approximation method to implement a solver for conformant planning problems, and it managed to find a solution for all problems from (Son et al., 2005). An interesting topic for future work is to determine classes of problems, for which our method can be shown to be complete. Another interesting approach might be to make the method more complete by defining levels of 'cased-based-reasoning'. This could be seen as an generalization of our method from Section 5.1.2. A possible idea here is that the atoms or subformulas on which such cased based reasoning should be done, could be chosen interactively.

In summary, the contributions of this chapter are that (1) we have shown how to use the results from Chapter 3 to approximate a class of useful $SAT(\exists\forall SO(ID))$-problems; and (2) we have examined how existing approximation methods fit into our general framework.

6

Representing and reasoning with sensing actions

Many formalisms for reasoning about action and change assume that agents have complete information about the world. A more realistic assumption, however, is that they only partially know their environment, and must use *sensing actions* to gain additional knowledge. Several approaches exist that formalize such reasoning (Moore, 1985; Scherl & Levesque, 2003; Thielscher, 2000; Lakemeyer & Levesque, 1998; Son & Baral, 2001; Ditmarsch et al., 2007; Herzig et al., 2003). One thing that is lacking in these approaches, however, is that their semantics do not (explicitly) model that, at every point in time, an agent can also have knowledge about both the past and the future, and not just about the current time point itself. For example, an agent might initially already know that he will *never* be able to do a certain action and that he therefore will not be able to reach a certain state. Conversely, by observing the effects of one of its past actions, the agent might learn that some property held at the time of this action, even if it currently no longer does. In this chapter, we propose a more general semantic framework, based on the Linear Time Calculus (see Section 2.3.3), which supports this.

The aforementioned approaches, as well as ours, all use a Kripke-style representation of the agent's knowledge. While this is semantically the right way of doing it, it is also computationally quite challenging. Therefore, several other approaches use more limited representations of knowledge with better computational properties (Demolombe & del Pilar Pozos Parra, 2000; Liu & Levesque, 2005; Petrick & Bacchus, 2004; Tu et al., 2007).

Ideally, it would be possible to view such an approach as an incomplete approximation, in some sense, of a semantically correct approach. However, proving such a connection is often non-trivial (Petrick & Levesque, 2002). An interesting property of our framework is that it allows us to apply the propagation method for FO(ID) (see Chapter 3) to obtain an incomplete method for solving the projection problem in polynomial time. The soundness of this method follows immediately from the fact that this approximation is sound for FO(ID) in general. Even though the method is incomplete, it is able to reach conclusions about properties that must have held at an earlier point in time, even if they no longer hold now. This is something that other incomplete reasoning methods either cannot do (Tu et al., 2007; Liu & Levesque, 2005; Demolombe & del Pilar Pozos Parra, 2000), or can only do in an ad-hoc way (Petrick & Bacchus, 2004).

6.1 Representing knowledge and sensing actions

When an agent operates in a partially known environment, its actions will be constrained by its knowledge of the world. At the same time, its knowledge may also evolve through sensing actions. Let us start off with an example.

Example 6.1. An agent has a glass, which can be clean or not. A glass can be used for drinking, which makes the glass dirty, and it can be cleaned by wiping it. The agent's room has a light, which can be switched on and off.

We already showed in Section 2.3.3 how such a domain can be represented in FO(ID). The following inductive definition Δ_{flu} defines fluents $Clean(t)$ and $Light(t)$ in terms of their initial values $Init\_Clean$ and $Init\_Light$, and of the actions $Wipe(t), Drink(t)$ and $Switch(t)$.

$$\Delta_{flu} = \left\{ \begin{array}{lcl} Clean(t+1) & \leftarrow & (Clean(t) \wedge \neg Drink(t)) \vee Wipe(t). \\ Clean(0) & \leftarrow & Init\_Clean. \\ Light(t+1) & \leftarrow & (\neg Switch(t) \wedge Light(t)) \\ & & \quad \vee (Switch(t) \wedge \neg Light(t)). \\ Light(0) & \leftarrow & Init\_Light. \end{array} \right\}. \tag{6.1}$$

The agent may only drink if it *knows* the glass to be clean. If the light is on, then the sensing action of *inspecting* the glass will reveal if it is clean. In the next section we will extend FO(ID) and the linear time calculus so that we can represent such preconditions and effects of sensing actions.

6.1.1 Syntax

In this section, we introduce two new language constructs. The first is a dynamic modal operator $K(\cdot, t)$ that refers to the agent's knowledge at time t.[1]

Definition 6.2. For a LTC signature Σ_{ltc}, a *temporal modal atom* is an expression of the form $K(\varphi[\bar{x}], t)$, where $\varphi[\bar{x}]$ is a formula in Σ_{ltc} and $t \in \mathbb{N}$. We define $\mathcal{L}_K$ as the language that extends FO by allowing both atoms and temporal modal atoms as base cases and closing under the usual operators $\wedge, \vee, \neg, \forall, \exists$. An $\mathcal{L}_K$-formula is called *objective* if no modal operator $K(\cdot, t)$ occurs in it and *subjective* if *every* atom occurs in the scope of such a modal operator.

Recall that in Section 2.3.3 we assumed that an LTC vocabulary contains a type $Time$, and that every other type S has only a finite number of constants. Again, we restrict our attention to Herbrand interpretations of S and fix the interpretation of $Time$ to the natural numbers $\mathbb{N}$. We will use formulas of $\mathcal{L}_K$ to express that an agent may perform certain actions only if it knows that its preconditions are satisfied.

Definition 6.3. For a LTC vocabulary Σ_{ltc}, a *knowledge precondition* is a formula $\forall \bar{x} t\ (A(\bar{x}, t) \Rightarrow \varphi)$, where A is an action and φ a subjective $\mathcal{L}_K$-formula

These knowledge preconditions will actually be the only place in which we allow the modal operators $K(\cdot, t)$. On the one hand, this is motivated by the fact that the agent has no direct access to the real world, i.e., it can use only its own knowledge to decide upon its actions. On the other hand, the agent also only affects the state of the world through its actions: the real world does not respond to what the agent *knows* but only to what it *does*. For instance, if the agent enters the correct code into a safe, this will open regardless of whether the agent actually knew the code or was just guessing.

The precondition that an agent may drink from a glass only if he knows that it is clean can now be expressed by the $\mathcal{L}_K$-formula

$$\forall t\, Drink(t) \Rightarrow K(Clean(t), t). \tag{6.2}$$

Note that the formula φ in a temporal modal atom $K(\varphi, t)$ may contain its own temporal argument, not necessary equal to t. For example, if the agent only wants to drink from a glass that has never been used before, this can be expressed as:

$$\forall t\, Drink\,(t) \Rightarrow K(\forall t'\ (t' < t \Rightarrow \neg Drink\,(t')), t).$$

This is more general than most existing approaches, where it is typically not even syntactically possible in one situation to refer to the knowledge about another situation.

To represent the effects of sensing actions, we introduce the following construct.

[1] Note that we make the assumption that the beliefs of the agent are correct.

Definition 6.4. For a LTC signature Σ_{ltc}, a *sensing definition* is of the form:

$$A(\bar{x}, t) \textbf{ senses } \xi[\bar{x}, \bar{y}, t+1] \textbf{ if } \gamma[\bar{x}, \bar{y}, t+1], \tag{6.3}$$

where $A \in \Sigma_{act}$, and both $\xi[\bar{x}, \bar{y}, t+1]$ and $\gamma[\bar{x}, \bar{y}, t+1]$ are FO formulas in which $t+1$ is the only term of type $Time$. The action A is called a *sensing action*. When $\gamma = \mathbf{t}$, the sensing action is called *unconditional*.

Note that a sensing action performed at time t yields information about the state of the world at the next timepoint $t+1$. For our running example, the $Inspect$ action is described by the following sensing definition.

$$Inspect\,(t) \textbf{ senses } Clean\,(t+1) \textbf{ if } Light\,(t+1). \tag{6.4}$$

Putting this all together, we now arrive at the following.

Definition 6.5. Let Σ_{ltc} be an LTC vocabulary. An $LTC_{K,S}$ theory consist of

- a *total* inductive definition Δ_{flu} of the fluents,
- a constraint theory T_{constr} of FO formulas over $\Sigma_{act} \cup \Sigma_{stat}$ in which the action predicates appear only negatively,
- an FO theory T_{init} over Σ_{init} about the initial state of the fluents,
- a precondition theory T_{prec}, consisting of knowledge preconditions,
- a set of sensing definitions, denoted by T_{sense}.

The purpose of the theory T_{constr} is to express constraints on the occurrences of actions. A typical example of this, are constraints that express that a number of actions can not occur concurrently, e.g., the sentence

$$\forall t\, \neg(Drink\,(t) \wedge Wipe\,(t)), \tag{6.5}$$

expresses that an agent cannot drink from the glass while simultaneously wiping it. However, we only allow a theory to specify that actions *cannot occur* in certain circumstances, and not that they *must* sometimes occur. This will make it easier to define the semantics. The theory T_{init} specifies *all that is known* about the initial situation. In our running example, we take T_{init} to be empty, meaning that the agent knows *nothing* about the initial situation. Our Δ_{flu} is as in Example 6.1, T_{constr} is consists of the formula (6.5) above, T_{prec} consists of the single precondition (6.2) and T_{sense} contains expression (6.4). Putting everything

together, this gives us the following $LTC_{K,S}$-theory T_{ex}:

$$\Delta_{flu} = \left\{ \begin{array}{lll} Clean(t+1) & \leftarrow & (Clean(t) \wedge \neg Drink(t)) \vee Wipe(t). \\ Clean(0) & \leftarrow & Init\_Clean. \\ Light(t+1) & \leftarrow & (\neg Switch(t) \wedge Light(t)) \\ & & \vee (Switch(t) \wedge \neg Light(t)). \\ Light(0) & \leftarrow & Init\_Light. \end{array} \right\}$$

$$T_{constr} = \{\forall t\, \neg(Drink(t) \wedge Wipe(t)).\}$$

$$T_{prec} = \{\forall t\, Drink(t) \Rightarrow K(Clean(t), t).\}$$

$$T_{sense} = \{Inspect(t)\ \textbf{senses}\ Clean(t+1)\ \textbf{if}\ Light(t+1).\}$$

6.1.2 Semantics

Our semantics needs to consider both the real world I[2] and the agent's knowledge about I. To represent this knowledge, we use a set of interpretations $\mathcal{K}$, which we refer to as *possible worlds*. Since each possible world $K \in \mathcal{K}$ interprets the entire vocabulary, it specifies the value of all actions and fluents at *each* point in time. A schematic representation of this is given in Figure 6.1. In this figure each square box represents the projection of the fluents of a possible world K_i on a certain timepoint. The arrows in between boxes represent the actions that happen. What the agent knows at a certain timepoint about the fluents at timepoint t can therefore be seen as the lower bound of all projections on that timepoint t of his possible worlds.

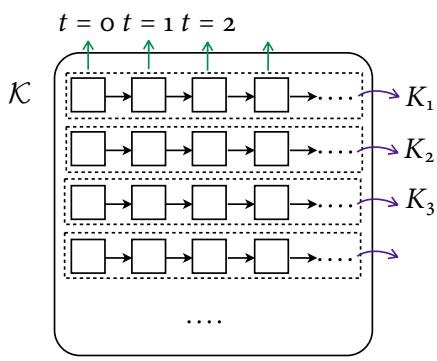

Figure 6.1: Schematic representation of a possible world set $\mathcal{K}$, representing the knowledge of the agent at one particular point in time, about past, current and future times.

[2]Note that with 'real world' we mean a description of all fluents, actions, etc. at all timepoints.

Definition 6.6. A *knowledge structure* $\mathcal{S}$ is a pair $(I, \mathcal{K})$ of an interpretation I and a set of interpretations $\mathcal{K}$, such that all these interpretations have the same vocabulary (and therefore also the same domain).

Because we only care about knowledge, and not beliefs that might be false, we require that knowledge structures be *consistent*.

Definition 6.7. A knowledge structure $\mathcal{S} = (I, \mathcal{K})$ is *consistent* if and only if $I \in \mathcal{K}$.

We first focus on the semantics of the sensing definitions. The purpose of these is to allow the agent to increase its knowledge. For a set Ψ of objective sentences, we denote by $\mathcal{K} \circ \Psi$ the set of all $J \in \mathcal{K}$ for which $J \models \Psi$. The smaller set of possible worlds $\mathcal{K} \circ \Psi$ now describes the agent's knowledge after it has learned Ψ in a situation where it already knew $\mathcal{K}$. We will now define a set of formulas Ψ that captures precisely what the agent learns from a sensing action. Without loss of generality, we assume that the theory T contains at most one sensing definition for each action $A \in \Sigma_{act}$. If the unique sensing definition for $A(\bar{x}, t)$ is of form (6.3), then we denote the formula ξ as ξ_A and the formula γ as γ_A.

Definition 6.8. Given an interpretation I, a sensing action $A(\bar{x}, t)$ and a time point $i \in \mathbb{N}$, the *scope* of $A(\bar{x}, i)$ in I is defined as the set of all tuples $(\bar{d}, \bar{e})$ of constants such that $I \models \gamma_A[\bar{d}, \bar{e}, i+1]$.

For the sensing action $Inspect$ in our running example, a timepoint i and an interpretation I, the scope of $Inspect(i)$ is either the empty tuple $\{()\}$ (i.e., "true") if $I \models Light(i+1)$ or the empty set $\varnothing$ (i.e., "false") if $I \not\models Light(i+1)$.

Definition 6.9. For a sensing action $A(\bar{x}, t)$, an interpretation I, a tuple $\bar{d}$ of constants, and a time point i, the *effect* of $A(\bar{d}, i)$ given I is denoted as $Eff_I(A(\bar{d}, i))$ and defined as the set of all formulas $\varphi[\bar{d}, \bar{e}, i+1]$ such that φ is either ξ_A or $\neg\xi_A$, the tuple $(\bar{d}, \bar{e})$ is in the scope of $A(\bar{x}, i)$, and $I \models \varphi[\bar{d}, \bar{e}, i+1]$.

Intuitively, the set $Eff_I(A(\bar{d}, i))$ contains all ground formulas that the agent learns by performing the sensing action $A(\bar{d}, i)$ in the real world I. To illustrate with our running example, consider the interpretation I such that $Light^I = \mathbb{N}$, $Inspect^I = \{0\}$ and all other predicates are interpreted by the empty relation. Then $Eff_I(Inspect(0)) = \{\neg Clean(1)\}$.

In addition to the results of its sensing actions, the agent also has a second source of information, namely, it will also know its own actions at each timepoint. At timepoint $i+1$, the agent will learn all information that the action and sensing actions at timepoint i gave him.

Definition 6.10. For a time point $i \geq 0$ and an interpretation I, the *action information* at i, denoted $Act_I(i)$, is defined as the set of all literals $L(\bar{d}, i)$ such that $I \models L(\bar{d}, i)$ and $L(\bar{d}, i)$

is either $A(\bar{d}, i)$ or $\neg A(\bar{d}, i)$ with $A \in \Sigma_{act}$. The $i+1^{th}$ *update* of I, denoted $Upd_I(i+1)$, is defined as the union of $Act_I(i)$ and all sets $Eff_I(A(\bar{d}, i))$ for which $I \models A(\bar{d}, i)$ and $A(\bar{x}, t)$ is a sensing action.

Note that this definition assumes that an agent knows its actions and its effects at the timepoint after performing them. For our running example, we have that $Upd_I(1) = \{\neg Drink(0), \neg Wipe(0), \neg Switch(0), Inspect(0), \neg Clean(1)\}$, and similarly $Upd_I(2) = \{\neg Drink(1), \neg Wipe(1), \neg Switch(1), \neg Inspect(1)\}$. We now arrive at the following update operation.

Definition 6.11. For a knowledge structure $\mathcal{S} = (I, \mathcal{K})$, the *i-update* of $\mathcal{S}$, denoted by $\mathcal{S} \circ i$, is defined as $(I, \mathcal{K} \circ Upd_I(i))$.

After performing such an update, the agent's new knowledge $\mathcal{K} \circ Upd_I(i)$ will contain all the information that was obtained as a direct result of its sensing actions at time $i-1$. In other words, the i-update removes from $\mathcal{K}$ all those worlds J for which the agent's actions at timepoint $i-1$ will allow him to conclude that J is not possible, either because his own actions in J at timepoint $i-1$ do not coincide with what he actually did in the real world I, or because one of his sensing actions at timepoint i directly contradicted J.

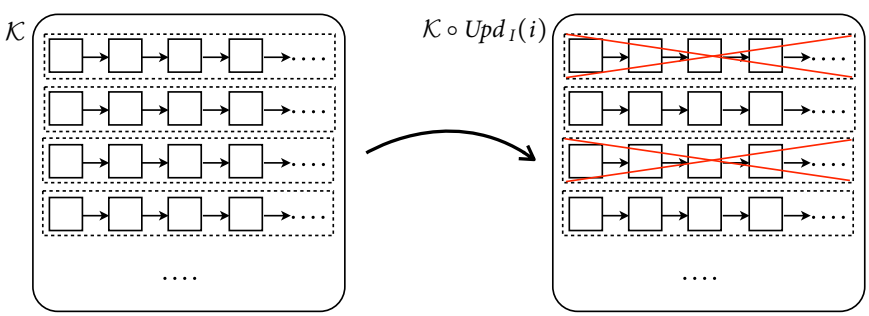

Figure 6.2: An agent updates his knowledge $\mathcal{K}$ by removing all possible worlds that do not coincide with what he did or sensed.

However, by *reflecting* on the difference between $\mathcal{K}$ and $\mathcal{K} \circ Upd_I(i)$, the agent may be able to deduce more information. For instance, if in our running example an $Inspect$ action fails to produce knowledge about whether the glass is clean, the agent could deduce that the light must be switched off. So, in addition to ruling out all worlds $J \notin Upd_I(i)$, the agent could also rule out all worlds $J \in \mathcal{K}$ in which the knowledge contained in $\mathcal{K} \circ Upd_J(i)$ would not have been the same as the knowledge $\mathcal{K} \circ Upd_I(i)$ that the agent actually gained in the real world.

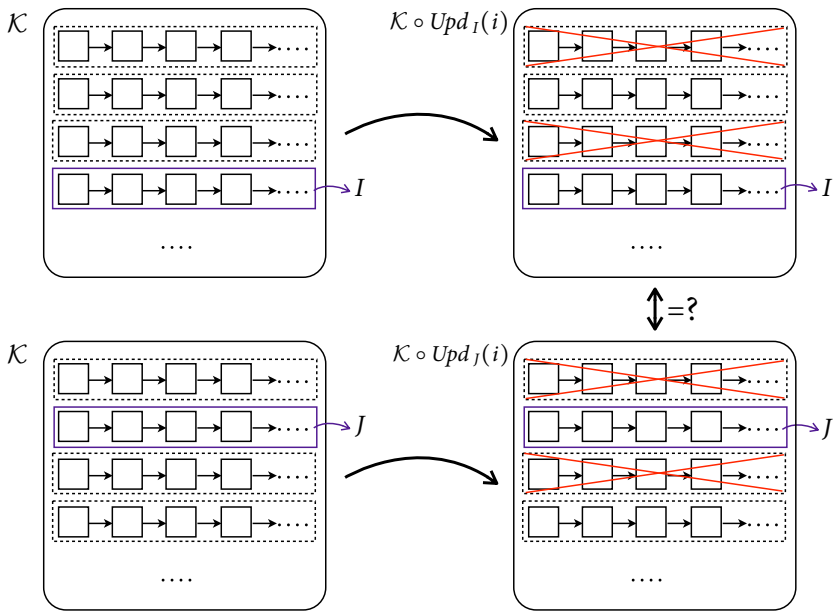

Figure 6.3: By reflecting on the update of his knowledge, an agent might be able to rule out even more possible worlds.

Let us define that two knowledge structures $\mathcal{S} = (I, \mathcal{K})$ and $\mathcal{S}' = (I', \mathcal{K}')$ are *epistemically equal*, denoted $\mathcal{S} =_k \mathcal{S}'$, if $\mathcal{K} = \mathcal{K}'$. We now define a *second-order* update operator $(I, \mathcal{K}) \circ^2 i$ as follows.

Definition 6.12. For a knowledge structure $\mathcal{S} = (I, \mathcal{K})$, if $\mathcal{S} \circ i = (I, \mathcal{K}')$, then $(I, \mathcal{K}) \circ^2 i$ is defined as $(I, \mathcal{K}'')$ with $\mathcal{K}'' = \{J \in \mathcal{K}' \mid (I, \mathcal{K}) \circ i =_k (J, \mathcal{K}) \circ i\}$.

Based on this principle, we can define an infinite sequence $(\circ^n)_{n\in\mathbb{N}}$ of ever more introspective update operators.

Definition 6.13. For a knowledge structure $\mathcal{S} = (I, \mathcal{K})$, if $\mathcal{S} \circ^n i = (I, \mathcal{K}_n)$, then $(I, \mathcal{K}) \circ^{n+1} i = (I, \mathcal{K}_{n+1})$ with $\mathcal{K}_{n+1} = \{J \in \mathcal{K}_n \mid (I, \mathcal{K}) \circ^n i =_k (J, \mathcal{K}) \circ^n i\}$. We define $\bullet$ as the limit of this sequence, i.e., $\mathcal{S} \bullet i = \bigcap_{n\in\mathbb{N}} \mathcal{S} \circ^n i$.

Note that the sequence $(\mathcal{K}_i)_{i\geq 0}$ is a descending sequence, that is, $\mathcal{K}_{i+1} \subseteq \mathcal{K}_i$. The limit $\bullet$ is a strong operator, which allows the agent to reason about its knowledge of its knowledge of its knowledge of … of its knowledge of the world. Luckily, however, it is never necessary to go more than two levels deep.

Proposition 6.14. *For any set of sensing definitions,* $\bullet = \circ^2$. *Moreover, if all of the sensing actions are* unconditional*, then even* $\bullet = \circ$.

Proof. For a given knowledge structure $\mathcal{S} = (I, \mathcal{K})$, we will use $\mathcal{K} \circ_I^n i$ to denote the $\mathcal{K}_n$ such that $\mathcal{S} \circ^n i = (I, \mathcal{K}_n)$. We first prove that $\bullet = \circ^2$.

Take any $J \in \mathcal{K} \circ_I^2 i$. As per definition, it then holds that $\mathcal{K} \circ_I i = \mathcal{K} \circ_J i$. Of course this means that $\mathcal{K} \circ_I^2 i = \{M \in \mathcal{K} \circ_I i \mid \mathcal{K} \circ_I i = \mathcal{K} \circ_M i\}$ is equal to $\mathcal{K} \circ_J^2 i = \{M \in \mathcal{K} \circ_J Ji \mid \mathcal{K} \circ_J i = \mathcal{K} \circ_M i\}$. Again, as per definition, this means that $J \in \mathcal{K} \circ_I^3 i$, and therefore that $\mathcal{K} \circ_I^2 i = \mathcal{K} \circ_I^3 i$. Since every operator $\circ_I^{i+1}$ is clearly completely determined by the operator $\circ_I^i$, this shows that $\circ_I^i = \circ_I^2$, for every $i \geq 2$.

Now we prove the second part of the proposition, that is, that $\circ^2 = \circ$ when there are no conditional actions. It immediately follows from the definition of the ith-update that, when there are no conditional sensing actions, if $J \vDash Upd_I(i)$, then $Upd_J(i) = Upd_I(i)$. Note that this does not hold when there are conditional sensing actions. Indeed, recall that $Upd_I(i)$ contains the sensing results and information about the actions that happened. Now, if in I a sensing action happened whose condition did not hold, then $Upd_I(i)$ will obviously not contain any sensed information from that action. However, in some other interpretation J for which $J \vDash Upd_I(i)$, it might be the case that the condition *does* hold, and therefore $Upd_I(i) \neq Upd_J(i)$, since J will contain extra sensed information.

If $J \in \mathcal{K} \circ_I i$, then obviously $J \vDash Upd_I(i)$. Therefore, $\mathcal{K} \circ_I^2 i = \{J \in \mathcal{K} \circ_I i \mid \mathcal{K} \circ_J i = \mathcal{K} \circ_I i\}$ simplifies to $\mathcal{K} \circ_I^2 i = \{J \in \mathcal{K} \circ_I i\}$. This concludes the proof. □

This operator $\bullet$ now allows us to define how the agent's knowledge will evolve over time, starting from an initial knowledge structure $\mathcal{S}_0 = (I, \mathcal{K})$.

Definition 6.15. A *knowledge line* $\mathbb{K}$ is a function that maps each time point $i \in \mathbb{N}$ to a set $\mathbb{K}(i)$ of possible worlds. For an initial knowledge structure $\mathcal{S}_0 = (I, \mathcal{K})$, we construct the sequence $(\mathcal{S}_i)_{i \geq 0}$ such that $\mathcal{S}_{i+1} = \mathcal{S}_i \bullet i + 1$. By $\overline{\mathcal{S}_0}$, we denote the associated knowledge line $\mathbb{K}$, i.e., for each i, $\mathbb{K}(i)$ is such that $\mathcal{S}_i = (I, \mathbb{K}(i))$.

If $\mathcal{K}$ represents the agent's initial knowledge (note again that $\mathcal{K}$ may contain knowledge about *any* point in time), then a knowledge line $\overline{(I, \mathcal{K})}$ represents how this knowledge evolves over time according to the real world I.

A *knowledge line structure* is now a pair $(I, \mathbb{K})$ of an interpretation I and a knowledge line $\mathbb{K}$. Since $I \vDash Upd_I(i)$, for each i, we have that for a consistent knowledge structure $(I, \mathcal{K})$, for each i, $I \in \mathbb{K}(i)$, where $\mathbb{K} = \overline{(I, \mathcal{K})}$. An illustration of this is given in Figure 6.4.

It is now straightforward to evaluate formulas of $\mathcal{L}_K$ in a knowledge line structure $(I, \mathbb{K})$.

Definition 6.16. For a formula φ of $\mathcal{L}_K$, a knowledge line structure $(I, \mathbb{K})$ and a variable assignment θ, we inductively define the relation $(I, \mathbb{K}), \theta \vDash \varphi$ as follows:

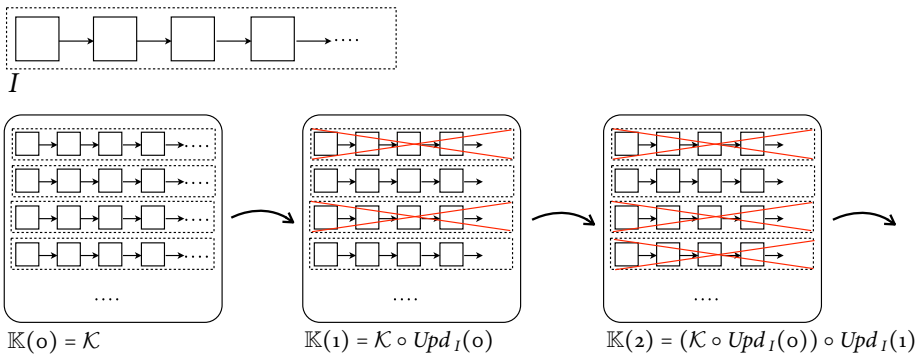

Figure 6.4: A knowledge line structure consists of a real world I and a knowledge line $\mathbb{K}$ representing the evolution of the knowledge of the agent.

- For an atom $P(\bar{t})$, we define $(I, \mathbb{K}), \theta \models P(\bar{t})$ iff $I, \theta \models P(\bar{t})$;
- For $K(\varphi, t)$, we define $(I, \mathbb{K}), \theta \models K(\varphi, t)$ iff for each $J \in \mathbb{K}(t)$, $J, \theta \models \varphi$;
- The cases for $\neg, \wedge, \vee, \forall, \exists$ are defined as usual.

For sentences, the variable assignment θ is irrelevant and we omit it from notation.

This evaluation is a multi-modal extension of the standard Kripke evaluation, since each $K(\cdot, t)$ is evaluated w.r.t. the set $\mathbb{K}(t)$ of possible worlds. We now have the following obvious way of defining the semantics of a $LTC_{K,S}$ theory T.

Definition 6.17. A knowledge structure $\mathcal{S} = (I, \mathcal{K})$ is a *weak model* of T if $(I, \mathcal{K})$ is consistent, and for each $J \in \mathcal{K}$, it holds that $(J, \overline{\mathcal{S}}) \models T$.

Every possible world in $\mathcal{K}$ represents a world such that, when the agent has $\mathcal{K}$ as initial knowledge, the theory (that is, both non-epistemic and epistemic constraints) is satisfied by the combination of that world and the initial knowledge $\mathcal{K}$. Note that this definition of weak model is completely symmetric in the sense that each $J \in \mathcal{K}$ is just as plausible for the agent, i.e., if $(I, \mathcal{K})$ is a weak model of T, then also $(J, \mathcal{K})$ is a weak model for each $J \in \mathcal{K}$.

A limitation of weak models is that they do not restrict the agent's knowledge about the initial situation: the agent must know *at least* the theory T_{init}, but is not prevented from arbitrarily knowing more. In our running example, the theory T_{init} is empty, so the agent cannot know whether the glass is initially clean or not (or, to be more precise, the agent *initially* does not know this, even though it is possible that later sensing actions will reveal that the glass was actually clean or dirty all along). Our theory now indeed correctly has

a weak model $(I, \mathcal{K})$ in which $\mathcal{K}$ contains both interpretations in which $Clean(0)$ holds and does not hold. However, there is also a weak model $(I, \mathcal{K}')$ where $\mathcal{K}'$ contains only interpretations J in which the glass is always clean, even though the agent should not know this.

As a final step, our semantics will therefore select precisely those weak models in which the agent knows *only* what it should, and nothing more. The agent's initial knowledge about the initial situation should consist of all models of T_{init}. Of course, the agent initially also has knowledge about the future timepoint 1. This should consist of all that is possible given the agent's initial knowledge about timepoint 0 and the theories Δ_{flu}, T_{constr} and T_{prec}. And so on for the initial knowledge about the next timepoints $i \geq 2$. For our running example, the agent initially has no knowledge at all about what holds at time point 0. According to his initial knowledge, since he doesn't know whether the glass is clean, the only actions he can do at time point 0 are an inspect or wipe action, and he can do these in every possible initial setting. Since the agent knows the effects of his actions, he knows that after doing a wipe action, the glass will be clean, so his initial knowledge should contain interpretations in which both $Wipe(0)$ and $Drink(1)$ hold, but no interpretations in which $\neg Wipe(0)$ and $Drink(1)$ hold. It is clear that continuing this process, for this example at least, gives us the initial knowledge that we wanted.

To define this concept properly, we first need a couple of definitions.

Definition 6.18. We define $Upd_I^{\leq}(i)$ as $\cup_{j \leq i} Upd_I(j)$. We write $I \sim_i J$ to say that J coincides with I on these formulas $Upd_I^{\leq}(j)$, that is, for each $\varphi \in Upd_I^{\leq}(i)$, $I \vDash \varphi$ iff $J \vDash \varphi$.

Worlds I and J for which $I \sim_i J$ are actually indistinguishable to the agent up to time point i. Note also that while exactly the same actions and sensing results happen in I and J, the values of the fluents that are not sensed can be totally different in I and J. Next, we define when two sets of possible worlds contain the same knowledge about the time points up to some $i \in \mathbb{N}$.

Definition 6.19. We write $I \overset{f}{\approx}_i J$ if I and J coincide on the static predicates (i.e., for each static predicate P, $P^I = P^J$) and on the fluents for all $t \leq i$ (i.e., for each fluent predicate P and each correctly typed tuple $\bar{d}$ whose temporal argument t is such that $t \leq i$, it holds that $\bar{d} \in P^I$ iff $\bar{d} \in P^J$). Similarly, we write $I \overset{a}{\approx}_i J$ if I and J coincide on the static predicates and on the actions for all $t \leq i$. We say that $I \approx_i J$ iff $I \overset{f}{\approx}_i J$ and $I \overset{a}{\approx}_{i-1} J$.

Given two possible world sets $\mathcal{K}_1$ and $\mathcal{K}_2$, we write $\mathcal{K}_1 \approx_i K_2$ iff for each $I \in \mathcal{K}_1$ there is a $J \in \mathcal{K}_2$ such that $I \approx_i J$, and vice versa.

If $I \approx_i J$, this means that interpretations I and J are exactly the same up to timepoint i considering the fluents and up to timepoint $i-1$ considering the actions. As a consequence, we have that if $I \approx_i J$, then $I \sim_i J$. The other way does not hold, since if $I \sim_i J$, then I and J can still differ on the fluents that are not sensed.

With these extra concepts, we now formalize the concept of 'minimal knowledge' for a possible world set $\mathcal{K}$, that we already informally described in the paragraph above.

Definition 6.20. Given an $LTC_{K,S}$ theory T, an interpretation I and a possible world set $\mathcal{K}$, such that $(I, \mathcal{K})$ is a weak model of T, we say that $\mathcal{K}$ has *minimal knowledge according to* T if the following three properties hold.

- $\mathcal{K} \approx_0 \mathcal{M}$, where $\mathcal{M} = \{I \mid I \vDash T_{init}\}$.
- There does not exist a $\mathcal{K}' \supset \mathcal{K}$ such that $(I, \mathcal{K}')$ is a weak model.
- For every $i \in \mathbb{N}$ and every I,J and I' in $\mathcal{K}$ for which $I \approx_i J$, $I' \sim_i I$ and $Act_I(i) = Act_{I'}(i)$, there must exists a $J' \in \mathcal{K}$ such that $I' \approx_i J'$, $J \sim_i J'$ and $Act_J(i) = Act_{J'}(i)$.

Some explanation and illustrations are in order here. The first two items of the above definition are fairly straightforward. The first says that $\mathcal{K}$ must contain exactly the same knowledge about timepoint 0 as the set of all models of T_{init}, and the second says that there cannot be a larger possible world set that also gives rise to a weak model. Now, the third item is a bit more complex. Let us take some fixed $i \in \mathbb{N}$. Then $I \approx_i J$ means that I and J are exactly the same up to i considering the fluents and up to $i - 1$ considering the actions. $I \sim_i I'$ means that I and I' are indistinguishable up to point i for the agent (i.e., $\overline{(I, \mathcal{K})}(j) = \overline{(I', \mathcal{K})}(j)$, $\forall j \leq i$), and finally $Act_I(i) = Act_{I'}(i)$ means that the actions that happen at timepoint i are the same in I and I'. Now this third part of the property says that for each such i, I, J and I' there must exist a J' that is exactly the same as I' up to i considering the fluents and to $i - 1$ considering the actions, such that J' is indiscernible from J up to timepoint i and such that the same actions at timepoint i happen as in J.

Take, for example, two interpretations I and J such that $I \approx_0 J$, that is, I and J interpret the fluents at time point 0 in the same way but different actions may happen at time point 0. Now take another interpretation I' that interprets the fluents at time point 0 differently from I, but in which the *same actions* happen, i.e., $Act_I(0) = Act_{I'}(0)$. Because for timepoint 0, $I \sim_0 I'$ is trivially satisfied, there must exists an interpretation J' that interprets the fluents in 0 in the same way as I', but such that the same actions as in J happen. Differently stated, this means that one requirement for a weak model to have minimal knowledge, is that if an action happens at time point 0 with one possible interpretation for the fluents at timepoint 0, it must happen with all other possible initial fluent interpretations. This is illustrated in Figure 6.5

In the context of our running example, we see that the weak model $(I, \mathcal{K})$ where $\mathcal{K}$ contains only interpretations in which the glass is always dirty does not satisfy the first item of the definition above. However, to see that the first two items alone are not enough, consider the following example. Let T be the following simple theory, consisting of only one precondition.

$$\forall t\, A(t) \Rightarrow K(F, t+1).$$

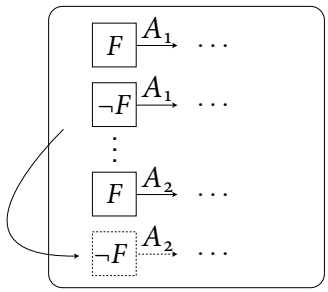

Figure 6.5: If an action happens at time point 0 with one possible interpretation for the fluents at timepoint 0, it must happen with all other possible initial fluent interpretations.

Here F is a constant, of which the initial value is unknown. Since F is unknown, and will clearly always stay unknown because there are no sensing actions or actions that change the value of F, we would obviously expect that no action A can ever happen.

However, one can verify that there does exists a weak model $(I, \mathcal{K})$ such that $\mathcal{K}$ satisfies the first two conditions of the definition above (where $I = \{F(0), A(0), F(1), F(2), \ldots\}$). This $\mathcal{K}$ contains both interpretations in which $F(0)$ is true and interpretations in which $F(0)$ is false. However, the action $A(0)$ only happens in interpretations in which $F(0)$ is true. The possible world set $\overline{(I, \mathcal{K})}(1)$ will thus contain only interpretations in which $A(0)$ is true. In all these interpretations $F(0)$ is true, and therefore the precondition $A(0) \Rightarrow K(F, 1)$ is satisfied in $(I, \overline{(I, \mathcal{K})})$. However, one can easily verify that the third condition of Definition 6.20 above is not satisfied.

We know define a *construction* of a possible world set that, as we show later, under some extra conditions represents the desired initial knowledge of an agent. Indeed, the following sequence of ever smaller possible world sets, gradually eliminates the weak models in which the agent has unwarranted knowledge by using, in each step, the knowledge that we have already gathered about timepoints $j < i$.

Definition 6.21. Given an $LTC_{K,S}$ theory T, we define the sequence $(\mathcal{K}_i)_{i \geq 0}$ as follows:

- $\mathcal{K}_0 = \{I \mid \exists \mathcal{K}$ such that $(I, \mathcal{K})$ is a weak model of $T.\}$
- $\mathcal{K}_{i+1} = \{I \mid \exists \mathcal{K} \approx_i \mathcal{K}_i$ such that $(I, \mathcal{K})$ is a weak model of $T.\}$

This sequence has a number of interesting properties.

Proposition 6.22. *For a theory T, the sequence $(\mathcal{K}_i)_{i \geq 0}$ defined above is descending, and thus has a limit, which we will denote by $\mathcal{K}_\infty$.*

Proof. We assume that $\forall i$, $\mathcal{K}_i$ is not empty. Whenever $\mathcal{K}_i$ is empty, it is clear that $\forall j > i$, $\mathcal{K}_j$ is empty too.

For any $i > 0$, take an interpretation $K \in \mathcal{K}_{i+1}$. We will prove that $K \in \mathcal{K}_i$. Since $K \in \mathcal{K}_{i+1}$, there must exist a possible world set $\mathcal{K}'$ such that $(K, \mathcal{K}')$ is a weak model and $\mathcal{K}' \approx_i \mathcal{K}_i$. We denote this $\mathcal{K}'$ as $\mathcal{P}_{i+1}(K)$.

In order to prove that $K \in \mathcal{K}_i$, it is sufficient to prove that for every $K' \in \mathcal{P}_{i+1}(K)$, there exists an $I \in \mathcal{K}_{i-1}$ such that $K' \approx_{i-1} I$ and vice versa. Indeed, then it follows that $\mathcal{P}_{i+1}(K) \approx_i K_{i-1}$, and thus that $K \in \mathcal{K}_i$.

Let us start with the first direction. Take such a $K' \in \mathcal{P}_{i+1}(K)$. Because of the definition of $\mathcal{K}_{i+1}$, there exists a $J \in \mathcal{K}_i$ such that $J \approx_i K'$. Now, with J, there is a corresponding $\mathcal{P}_i(J)$ and, because weak models are consistent, $J \in \mathcal{P}_i(J)$. Therefore, there must exist an $I \in \mathcal{K}_{i-1}$ such that $I \approx_{i-1} J$. But then it follows that $I \approx_{i-1} K'$.

The other direction is similar. Take any $I \in \mathcal{K}_{i-1}$ and any $J \in \mathcal{K}_i$. We now have that there must be a $J' \in \mathcal{P}_i(J)$ such that $J' \approx_{i-1} I$. Because of the symmetry of weak models also $J' \in \mathcal{K}_i$, it follows again that there must exist a $K' \in \mathcal{P}_{i+1}(K)$ such that $K' \approx_i J'$, and thus $K' \approx_{i-1} I$. □

The sequence $(\mathcal{K}_i)_{i \geq 0}$ also has the following property, which says that in the i-th step, $\mathcal{K}_i$, the actual knowledge about earlier timepoints $j < i$, $\mathcal{K}_j$, is preserved.

Proposition 6.23. *For all i, j such that $i > j$, if $\mathcal{K}_i$ is not empty, it holds that $\mathcal{K}_i \approx_j \mathcal{K}_j$.*

Proof. We will prove this by induction. First we will prove that $\mathcal{K}_{i+1} \approx_i \mathcal{K}_i$, and in a second step we will prove that if $\mathcal{K}_{i+2} \approx_{i+1} \mathcal{K}_{i+1}$ and $\mathcal{K}_{i+1} \approx_i \mathcal{K}_i$, then $\mathcal{K}_{i+2} \approx_i \mathcal{K}_i$.

To prove that $\mathcal{K}_{i+1} \approx_i \mathcal{K}_i$, we need to prove that

- $\forall I \in \mathcal{K}_{i+1}$, there exists a $J \in \mathcal{K}_i$, such that $I \approx_i J$, and
- $\forall J \in \mathcal{K}_i$, there exists an $I \in \mathcal{K}_{i+1}$, such that $I \approx_i J$.

Let us first note that in this proof we will make use again of the notation $\mathcal{P}_i(I)$, as introduced in the proof of Proposition 6.22. Now, take an $I \in \mathcal{K}_{i+1}$. Since I is clearly an element of $\mathcal{P}_{i+1}(I)$, it follows immediately that there must exist a $J \in \mathcal{K}_i$ such that $I \approx_i J$. Next, take a $J \in \mathcal{K}_i$, and any $I \in \mathcal{K}_{i+1}$. Now, there must exists an $I' \in \mathcal{P}_{i+1}(I)$ such that $J \approx_i I'$. Because of the symmetry of the definition of weak model, I' must be in $\mathcal{K}_{i+1}$ too, and this concludes the first step of the proof.

The second part follows immediately from the fact that if $I \approx_{i+1} J$, then also $I \approx_i J$. Indeed, if $I \approx_{i+1} J$ and $J \approx_i K$, we have that $I \approx_i K$. □

Finally, the last property says that, if $(I, \mathcal{K}_\infty)$ is consistent, it is a weak model.

Proposition 6.24. *For an $LTC_{K,S}$ theory T and a given interpretation I, if $(I, \mathcal{K}_\infty)$ is consistent, then this is a weak model.*

Proof. Assume that the knowledge structure $(I, \mathcal{K}_\infty)$ is consistent. First, notice that the following property holds. Let t_1 and t_2 be constants of type $Time$, and φ a first-order formula over Σ_{ltc}. Without loss of generality, we assume that φ contains no function symbols. Now let j be a constant such that $j > max\{t_1, t_2\}$. It then holds that

$$(I, \overline{(I, \mathcal{K}_\infty)}) \models K(\varphi[t_1], t_2) \text{ iff } (I, \overline{(I, \mathcal{K}_j)}) \models K(\varphi[t_1], t_2). \tag{6.6}$$

Indeed, this follows immediately from the fact that $j > max\{t_1, t_2\}$ and Proposition 6.23.

Now suppose that $(I, \mathcal{K}_\infty)$ is not a weak model. Then, there must exists a $J \in \mathcal{K}_\infty$ such that $(J, \overline{(J, \mathcal{K}_\infty)}) \not\models T_{prec}$ (Since $J \in \mathcal{K}_\infty$ obviously Δ_{flu},T_{init} and T_{const} must be satisfied by J). This means that there must exist a knowledge precondition ψ_{prec}, a minimal constant n of type $Time$ and a tuple of constants $\bar{d}$ of the correct arity and type such that $(J, \overline{(J, \mathcal{K}_\infty)}) \not\models \psi_{prec}[\bar{d}, n]$. If we now pick a j big enough, that is, strictly larger than any constant of type $Time$ occurring in $\psi_{prec}[\bar{d}, n]$, it follows from (6.6) that

$$(J, \overline{(J, \mathcal{K}_\infty)}) \not\models \psi_{prec}[\bar{d}, n] \text{ iff } (J, \overline{(J, \mathcal{K}_j)}) \not\models \psi_{prec}[\bar{d}, n],$$

and therefore that

$$(J, \overline{(J, \mathcal{K}_j)}) \not\models \psi_{prec}[\bar{d}, n]. \tag{6.7}$$

Next, we will prove that J cannot be in $\mathcal{K}_{j+1}$. Indeed, if $J \in \mathcal{K}_{j+1}$, then there would exist a possible world set $\mathcal{K}$ such that $(J, \overline{(J, \mathcal{K})}) \models \psi_{prec}[\bar{d}, n]$, and $\mathcal{K} \approx_j \mathcal{K}_j$, but since j is strictly larger than any constant of type $Time$ in $\psi_{prec}[\bar{d}, n]$ and $\mathcal{K}$ and $\mathcal{K}_j$ represent the same knowledge up to time point j, this would mean that $(J, \overline{(J, \mathcal{K}_j)}) \models \psi_{prec}[\bar{d}, n]$, which is a contradiction with (6.7). Therefore, such $\mathcal{K}$ cannot exists, and $J \notin \mathcal{K}_{j+1}$. But if $J \notin \mathcal{K}_{j+1}$, then $J \notin \mathcal{K}_\infty$, which contradicts our assumption. □

This now leads to the following definition of our semantics.

Definition 6.25. A weak model $(I, \mathcal{K})$ is a *model* of T if $\mathcal{K}$ is the limit of the sequence $(\mathcal{K}_i)_{i \geq 0}$ (as defined in Definition 6.21).

The question that remains now is whether models of a theory T indeed have minimal knowledge according to Definition 6.20. First let us look at the simple case where T_{prec} is empty. We then have that a knowledge structure $(I, \mathcal{K})$ is a weak model iff $I \models \Delta_{fu} \wedge$

$T_{constr} \wedge T_{init}$ and $I \in \mathcal{K}$. In other words, whether $(I, \mathcal{K})$ is a weak model only depends on I and whether $I \in \mathcal{K}$. From this it immediately follows that $\mathcal{K}_\infty = \mathcal{K}_o = \{I | I \models \Delta_{flu} \wedge T_{constr} \wedge T_{init}\}$ and it is easy to verify that this possible world set indeed satisfies Definition 6.20.

In its full generality, $LTC_{K,S}$ allows some unintuitive constraints to be expressed, such as stating that an agent's current action depends on its future knowledge. It turns out that such theories can lead to models that do not always have minimal knowledge. We therefore restrict attention to a more sensible fragment, in which the agent's current actions depend only on its current knowledge about current or past events.

Definition 6.26. An $LTC_{K,S}$ theory T is called a *basic sensing theory* if all of its knowledge preconditions are of the form $\forall \bar{x} t\, A(\bar{x}, t) \Rightarrow K(\varphi[\bar{x}, t'], t)$, where $t' \leq t$ and φ is a formula over $\Sigma_{ltc} \setminus \Sigma_{act}$.

For such basic action theories, models have the initial knowledge that we want.

Proposition 6.27. *If $(I, \mathcal{K})$ is a model of a basic sensing theory T, then $\mathcal{K}$ has minimal knowledge according to T.*

Proof. We start with proving the first part of the property in Definition 6.20, that is, that $\mathcal{K}_\infty \approx_o \mathcal{M}$, where $\mathcal{M} = \{I \mid I \models T_{init}\}$. Since we already know that $\mathcal{K}_\infty \approx_o \mathcal{K}_o$ (Proposition 6.23), and clearly $\mathcal{K}_o \subseteq \mathcal{M}$, all that is left to prove is that for each $I \in \mathcal{M}$ there exists a $J \in \mathcal{K}_o$ such that $I \approx_o J$. Now take an $I \in \mathcal{M}$ and let I' be the interpretation such that $I' \approx_o I$, $I' \models \Delta_{flu}$ and such that *no actions happen* in I'. Such an I' must exist. Since no actions happen T_{prec} and T_{constr} are irrelevant. T_{init} is certainly satisfiable, since $I \models T_{init}$. We can thus let I' interpret Σ_{init} in the same way as I. Now, since Δ_{flu} is a total definition, it must have a model, and we can take I' such that $I' \models \Delta_{flu}$. It immediately follows that $(I', \{I'\})$ is a weak model, and therefore $I' \in \mathcal{K}_o$.

The second property we have to prove, is that $\mathcal{K}_\infty$ is maximal, that is, there exists no interpretation I and possible world set $\mathcal{K} \supseteq \mathcal{K}_\infty$ such that $(I, \mathcal{K})$ is a weak model. Suppose there does exist such a $\mathcal{K}$. We will prove that $\mathcal{K} \approx_i \mathcal{K}_i$, $\forall i \in \mathbb{N}$. It then immediately follows from this (since then for every $I \in \mathcal{K}$, $(I, \mathcal{K})$ is a weak model such that $\mathcal{K} \approx_i \mathcal{K}_i$, and therefore $I \in \mathcal{K}_{i+1}$) that $\mathcal{K} \subseteq \mathcal{K}_i$, $\forall i > o$, and therefore that $\mathcal{K} \subseteq \mathcal{K}_\infty$. We will prove that $\mathcal{K} \approx_i \mathcal{K}_i$, $\forall i$ through induction.

First of all, for $i = o$, it holds that $\mathcal{K} \approx_o \mathcal{K}_o$. To prove this, we need to show that $\forall I \in \mathcal{K}$, there exists an $I' \in \mathcal{K}_o$ such that $I \approx_o I'$ and vice versa. Take an $I \in \mathcal{K}$. Since $(I, \mathcal{K})$ is a weak model of T, we have that $I \models T_{init}$, en thus $I \in \{I \mid I \models T_{init}\}$. But we already prove that $\mathcal{K}_\infty \approx_o \mathcal{K}_o \approx_o \{I \mid I \models T_{init}\}$. Therefore, there must exists an $I' \in \mathcal{K}_o$ such that $I \approx_o I'$. Since $\mathcal{K}_o \subseteq \mathcal{K}$, the other direction is trivial.

Now suppose that $\mathcal{K} \approx_i \mathcal{K}_i$ holds. We now have to prove that $\mathcal{K} \approx_{i+1} \mathcal{K}_{i+1}$. Since it immediately follows from $\mathcal{K} \approx_i \mathcal{K}_i$ that $\mathcal{K} \subseteq \mathcal{K}_{i+1}$, all that we need to prove is that for every

interpretation $I \in \mathcal{K}_{i+1}$ there exists an $I' \in \mathcal{K}$ such that $I \approx_{i+1} I'$. Now take an $I \in \mathcal{K}_{i+1}$. Since $\mathcal{K}_{i+1} \approx_{i+1} \mathcal{K}_\infty$, there exists a $J \in \mathcal{K}_\infty$ such that $I \approx_{i+1} J$, and since $\mathcal{K}_\infty \subseteq \mathcal{K}$, this concludes this part of the proof.

Now we prove that the third and last property in Definition 6.20 holds. To do this, take an $i \in \mathbb{N}$, $I, J \in \mathcal{K}_\infty$ such that $I \approx_i J$ and $I' \in \mathcal{K}_\infty$ such that $I \sim_i I'$ and $Act_I(i) = Act_{I'}(i)$. An important thing to note, is that for *basic action theories* (because of the constraints on the type of preconditions), if $I_1, I_2 \in \mathcal{K}_\infty$ and $I_1 \approx_i I_2$, then

$$(I_1, \overline{(I_1, \mathcal{K}_\infty)}) \models \psi_{prec}[\bar{d}, t_1] \text{ iff } (I_2, \overline{(I_2, \mathcal{K}_\infty)}) \models \psi_{prec}[\bar{d}, t_1], \ \forall t_1 \leq i. \tag{6.8}$$

So now, take an interpretation J' such that $J' \approx_i I'$, $J' \models \Delta_{flu}$, $Act_{J'}(i) = Act_{I'}(i)$ and such that for all $j > i$ no actions happen. It now follows easily from (6.8) and the assumptions on J' that $(J', \overline{(J', \mathcal{K}_\infty)}) \models T$. Now all that is left to prove is that $J' \in \mathcal{K}_\infty$. Since no actions happen after timepoint i, and $I' \approx_i J'$ with $I' \in \mathcal{K}_\infty$, we have that

$$(J', \overline{(J', \mathcal{K}_\infty)}) \models T \text{ iff } (J', \overline{(J', \mathcal{K}_\infty \cup \{J'\})}) \models T,$$

(or in other words, adding J' to $\mathcal{K}_\infty$ does not change the evaluation of the preconditions). Since we already proved that $(J', \overline{(J', \mathcal{K}_\infty)}) \models T$, it follows that $(J', \mathcal{K}_\infty \cup \{J'\})$ is a weak model. Now, similarly, it follows from $I' \approx_i J'$ that $(I', \overline{(I', (\mathcal{K}_\infty \cup \{J'\}))}) \models T$, and thus that $(I', \mathcal{K}_\infty \cup \{J'\})$ is also a weak model. But we already proved that $\mathcal{K}_\infty$ is maximal, and thus $J' \in \mathcal{K}_\infty$, which completes the proof. □

6.2 An approximative algorithm for the projection problem

A basic reasoning problem in temporal systems is the *projection problem*: given an action theory that formalizes the temporal domain, determine whether a formula holds after a sequence of actions is performed. In the context of incomplete information, we typically want to know whether the agent *knows* that a certain formula holds, since this will allow us to determine whether the preconditions to its actions are satisfied as well as whether it knows that it has reached its goal.

Recall that up to timepoint i, the information that the agent has so far learned about the real world I is precisely $Upd_I^{\leq}(i) = \bigcup_{j \leq i} Upd_I(j)$. The projection problem can now be defined as follows.

Definition 6.28. Given a $LTC_{K,S}$ theory T, an interpretation I, a timepoint i and a formula φ, the *projection problem* is the problem of deciding whether for all models $(J, \mathcal{K})$ of T such that $J \sim_i I$, it holds that $(J, \mathcal{K}) \models K(\varphi, i)$.

Typically, this problem is only of interest for interpretations I in which the agent does not perform impossible actions (i.e., there exists some $\mathcal{K}$ such that $(I, \mathcal{K}) \models T$). Indeed, one can iteratively solve for every action that happens a projection problem, to check whether the preconditions for that action are satisfied. We will therefore assume that the knowledge preconditions are satisfied and forget them. Once we know the information that the agent has sensed, as it is recorded in $Upd_I^{\leqslant}(i)$, also the sensing definitions become irrelevant. Our method will therefore only look at the following theory.

Definition 6.29. For an $LTC_{K,S}$ theory T, an interpretation I such that $(I, \mathcal{K}) \models T$ for some $\mathcal{K}$, and a timepoint $i > 0$, let $T_{proj}^{I,i}$ be the theory $\{\Delta_{flu}, T_{init}, T_{constr}, Upd_I^{\leqslant}(i)\}$.

As the following proposition shows, we can use the set of all models of $T_{proj}^{I,i}$ as a sound approximation of the agent's real knowledge at timepoint i.

Proposition 6.30. *Given a model $(I, \mathcal{K})$ of a basic sensing theory T, let $\mathcal{M}$ be the set of all FO(ID) models of $T_{proj}^{I,i}$. Then $\mathcal{M}$ is a superset of the agent's knowledge at time i according to $\mathbb{K} = \overline{(I, \mathcal{K})}$, i.e., $\mathcal{M} \supseteq \mathbb{K}(i)$.*

Proof. Since $(I, \mathcal{K})$ is a model, it immediately follows from the definition of a model, that for all $J \in \mathcal{K}$ it holds that $J \models \Delta_{flu} \wedge T_{init} \wedge T_{constr}$. We also have that for each $J \in \mathbb{K}(i)$, it holds that $J \models Upd_I^{\leqslant}(i)$. It immediately follows that $\mathbb{K}(i) \subseteq \mathcal{M}$. □

The theory $T_{proj}^{I,i}$ incorporates the knowledge that the agent has gained through his sensing actions by simply including $Upd_I^{\leqslant}(i)$. However, knowledge that the agent might gain through introspection is therefore not taken into account. In other words, this theory represents the $\circ$ operator instead of $\bullet$, and this is what accounts for its possible incompleteness. As shown earlier, however, these two operators are identical for theories containing only unconditional sensing actions.

Proposition 6.31. *Given a basic sensing theory T that contains only unconditional sensing actions, let $(I, \mathcal{K})$ be a model of T and $\mathcal{M}$ the set of all models of $T_{proj}^{I,i}$. Then $\mathcal{M}$ and $\mathbb{K}(i)$ have the same knowledge about all timepoints $j < i$, i.e., for each formula $\varphi[j]$ with $j < i$, we have: $\forall J \in \mathcal{M}, J \models \varphi[j]$ iff $\forall J \in \mathbb{K}(i), J \models \varphi[j]$.*

Proof. For the rest of this proof, let $\mathcal{M}_i$ denote the set of models of $T_{proj}^{I,i}$. What we actually have to prove boils down to proving for all $i \in \mathbb{N}$ that $\mathbb{K}(i) \approx_i \mathcal{M}_i$. We will prove this through induction.

As base case, we will prove that $\mathcal{K} \approx_0 \mathcal{M}_0$. Since $\mathcal{K} \subseteq \mathcal{M}_0$, all that is left to prove is that for all $M \in \mathcal{M}_0$, there exists a $J \in \mathcal{K}$ such that $M \approx_0 J$. So take an $M \in \mathcal{M}_0$. Now let M' be the interpretation such that $M' \approx_0 M$ (that is, M' interprets the fluents in timepoint 0 in the same way as M), $M \models \Delta_{flu} \wedge T_{constr}$ and such that *no* actions happen. Because of the way

we chose M' (indeed, no actions happen, so the preconditions and evaluation of modal literals become irrelevant), we have that

$$(M', \overline{(M', \mathcal{K})}) \models T \text{ iff } (M', \overline{(M', \mathcal{K} \cup \{M'\})}) \models T,$$

which means that $(M', \mathcal{K} \cup \{M'\})$ is a weak model of T. Moreover, since it also holds that $(M, \overline{(M, \mathcal{K} \cup \{M'\})}) \models T$, it follows immediately from Proposition 6.27 that $M' \in \mathcal{K}$.

Next, we will prove the induction step. Assume that $\mathbb{K}(i) \approx_i \mathcal{M}_i$. First of all, it follows from Proposition 6.14 that $\mathbb{K}(j) = \{J \in \mathcal{K} \mid J \models Upd_I^<(j)\}$ for all $j \in \mathbb{N}$, and therefore $\mathbb{K}(i+1) \subseteq \mathcal{M}_{i+1}$. We still need to prove that for all $M \in \mathcal{M}_{i+1}$ there exists a $J \in \mathbb{K}(i+1)$ such that $M \approx_{i+1} J$. As before, let M' be the interpretation such that $M' \approx_{i+1} M$, $M' \models \Delta_{flu} \wedge T_{constr}$ and such that no actions happen for time points $i+1$ or larger. Now, the only actions that happen in M' are the same actions as in the real world I for timepoints $t \leq i$. Since we assumed that these actions satisfy the preconditions and no other actions happen, we get that $(M', \overline{(M', \mathcal{K})}) \models T$. The induction hypothesis says that $\exists J \in \mathcal{K}$ such that $J \approx_i M \approx_i M'$. This means that using $\mathcal{K}$ or $\mathcal{K} \cup \{M'\}$ to evaluate the modal literals gives the same result for $t \leq i$ and therefore $(M', \overline{(M', \mathcal{K} \cup \{M'\})}) \models T$ and $(M, \overline{(M, \mathcal{K} \cup \{M'\})}) \models T$. As before, this in turn means that $M' \in \mathcal{K}$, which concludes the proof, that is, we have proven that $\mathbb{K}(i+1) \approx_{i+1} \mathcal{M}_{i+1}$. □

Note that both $\mathcal{M}$ and $\mathbb{K}(i)$ also contain knowledge about timepoints $j > i$. However, here the two differ, because $\mathbb{K}(i)$ already knows that the agent's future actions will satisfy the action preconditions, while $\mathcal{M}$ does not.

Finding out whether a formula holds in all models of a first-order theory is co-NP complete. Since applications such as planning need to solve the projection problem as an atomic step, a direct application of the above proposition would be infeasible in practice. However, as we have shown in Chapter 3, it is possible to efficiently approximate the set of all models of an FO(ID) theory. Since we only need natural numbers up to i to solve the projection problem, $Approx(T_{proj}^{I,i})$ therefore offers a polynomial time approximate solution to the projection problem. We will illustrate this with an example.

Example 6.32. We have a gun that can be loaded or not, and a turkey, which can be alive or dead. Shooting kills the turkey if the gun was loaded, and also unloads the gun. Finally, the agent can inspect the turkey to see whether it is dead.

$$\Delta_{flu} = \left\{ \begin{array}{lcl} Alive(t+1) & \leftarrow & Alive(t) \wedge \neg(Shoot(t) \wedge Loaded(t)). \\ Alive(0) & \leftarrow & Init\_Alive. \\ Loaded(t+1) & \leftarrow & Loaded(t) \wedge \neg Shoot(t). \\ Loaded(0) & \leftarrow & Init\_Loaded. \end{array} \right\}$$

$$T_{sense} = \{CheckAlive(t) \textbf{ senses } Alive(t+1) \textbf{ if } \mathbf{t}.\}$$

$$T_{init} = \{Init\_Alive.\}$$

If, after shooting a turkey that was known to be alive, the agent discovers that the turkey is dead, it should be able to conclude that the gun was previously loaded, even though it also knows that it is currently no longer loaded. We consider the real world I in which $Shoot^I = \{0\}$, $CheckAlive^I = \{1\}$, $Alive^I = \{0\}$, $Loaded^I = \{0\}$, $Init\_Alive^I = \mathbf{t}$ and $Init\_Loaded^I = \mathbf{t}$. Let us define $\psi[t] = \neg Shoot(t) \vee \neg Loaded(t)$ and $\varphi[t] = Alive(t) \wedge \psi[t]$. The definition $Approx(T^{I,3}_{proj})$ now contains, among others, the following rules.

$$\left\{\begin{array}{lcl}
Alive^{ct}(t+1) & \leftarrow & A^{ct}_{\varphi}(t). \\
A^{cf}_{\varphi}(t) & \leftarrow & Alive^{cf}(t+1). \\
A^{ct}_{\psi}(t) & \leftarrow & Shoot^{cf}(t) \vee Loaded^{cf}(t). \\
A^{cf}_{\psi}(t) & \leftarrow & A^{cf}_{\varphi}(t) \wedge Alive^{ct}(t). \\
Alive^{cf}(t) & \leftarrow & A^{cf}_{\varphi}(t) \wedge A^{ct}_{\psi}(t). \\
Loaded^{ct}(t) & \leftarrow & A^{cf}_{\psi}(t). \\
Loaded^{cf}(t+1) & \leftarrow & Loaded^{cf}(t) \vee Shoot^{ct}(t). \\
 & \ldots & \\
Init_Alive^{ct} & \leftarrow & \\
Shoot^{ct}(t) & \leftarrow & t = 0. \\
Shoot^{cf}(t) & \leftarrow & t = 1 \vee t = 2. \\
Alive^{cf}(t) & \leftarrow & t = 2.
\end{array}\right\}$$

This definition is able to derive that, because the turkey is certainly not alive at 2, the conjunction $\varphi(1)$ cannot hold. Since there is no *Shoot* at timepoint 1, the second disjunct $\psi(1)$ is certainly true and so the first disjunct $Alive(1)$ must be false. Again, this means that the disjunction $\varphi(0)$ must be false. Now, it is the first disjunct $Alive(0)$ that is certainly true, since *Init_Alive* is true. Hence, the second disjunct $\psi(0)$ cannot hold. Since $\psi(0)$ is itself again a disjunction, neither of its disjuncts hold, and one of these is $\neg Loaded(0)$. Therefore, this approximation is indeed able to reason backwards in time to conclude that initially the gun must have been loaded. Moreover, it will also be able to conclude that, because of the $Shoot(0)$ action, $Loaded(1)$ and $Loaded(2)$ no longer hold. It is not hard to verify that the above definition $Approx(T^{I,3}_{proj})$ effectively encodes this derivation, and therefore, entails $Loaded^{ct}(0)$, $Loaded^{cf}(1)$ and $Loaded^{cf}(2)$.

6.3 Conclusions and related work

Knowledge in the context of sensing actions was first investigated in (Moore, 1985), and combined with a solution for the frame problem in (Scherl & Levesque, 2003). Since then many other approaches have been developed. A large number of these cannot handle the *postdiction* reasoning of Example 6.32, i.e., they cannot draw the conclusion that at time point 2 the agent knows that the gun was initially loaded, but is not loaded anymore.

To handle such examples, it must be possible to trace back the current objective state of the world to the state from which it originated. This is not possible in the approaches to sensing that were developed in the context of the action language $\mathcal{A}$ (Son & Baral, 2001; Tu et al., 2007), since they only allow forward reasoning. An interesting feature of this approach, however, is that they also develop an approximation method, which transforms an $\mathcal{A}$ specification into an answer set program. This method has a very similar flavour to ours, and also to that of (Liu & Levesque, 2005), which defines a three-valued progression operator. Again, since only progression is considered, postdiction is not possible here. The approach of (Herzig et al., 2003) uses epistemic logic and has both progression and regression operations. However, the regression operator cannot take into account current knowledge about previous points in time, and therefore this semantics is also unable to handle our Example 6.32.

Some approaches do have the necessary semantic machinery in place, but simply lack the syntax to refer to knowledge about past or future events. This is the case for several approaches based on situation calculus or similar formalisms (Scherl & Levesque, 2003; Thielscher, 2000; Lakemeyer & Levesque, 1998), where the structure of the situation terms tracks the history of each state. While it would therefore be less problematic to extend these formalisms to handle postdiction, this has, as far as we know, not yet been done. The same can be said about the dynamic epistemic logic of (Ditmarsch et al., 2007).

Rather than extend an existing approach, however, we have chosen to develop a new framework. The main motivation for this was to be able to apply the approximation method of Chapter 3 to obtain an efficient but incomplete solution to the projection problem. The approaches of (Demolombe & del Pilar Pozos Parra, 2000; Petrick & Bacchus, 2004) try to achieve the same goal by abandoning a Kripke-style semantics in favour of a a more limited concept of knowledge. This is done by introducing, for every original fluent F, a new "knowledge fluent" K_F. The way in which such a K_F changes over time is then explicitly modeled by separate update axioms.The approach of (Petrick & Bacchus, 2004) is also able to handle knowledge about past or future events, and do postdiction reasoning. However, this is done in an algorithmic way, through a syntactical analysis of the specification. A disadvantage of these approaches in general is that it takes a substantial amount of work to prove a semantic connection between a knowledge fluent and the actual knowledge of that fluent (Petrick & Levesque, 2002). The semantics of our approach allowed us to apply a general approximation method for FO, which means that soundness follows almost immediately.

Finally, (Hunter & Delgrande, 2011) focuses mainly on belief change due to actions and observations where the agents may have erroneous beliefs. This is more general than our setting, since we assume that the agent's beliefs are always correct. As we do, they also allow beliefs about previous timepoints to change as a result of later (sensing) actions. However, their work does not focus on the knowledge representation side, and they assume a given transition system instead of a logical representation.

In summary, this chapter introduces a general framework for representing sensing actions, knowledge preconditions, conditional action effects, and knowledge about the past or the future. We also present a polynomial, sound but incomplete inference method for solving the projection problem, that we obtain directly by making use of a general approximation method for FO(ID). Even though this is an incomplete method, it still is capable to reach conclusions about properties that must have held at an earlier point in time, which is something that typical progression based reasoning methods cannot do.

7

Ordered epistemic logic: semantics, complexity and applications

Ordered Epistemic Logic (OEL) was first defined by Konolige (1988a) under the name *Hierarchic Autoepistemic Theories*. He observed that in non-monotonic reasoning the notion of inference from a *specific* body of knowledge often plays an important role. Recently, OEL was independently reintroduced by Denecker et al. (2010b) in order to merge the contributions of ASP (Brewka et al., 2011; Baral, 2003) on the level of Knowledge Representation into classical first-order logic. As many of the original motivating examples of ASP involve defaults and autoepistemic propositions (Gelfond & Lifschitz, 1991), this was done by adding an epistemic operator to FO.

Many of these motivating examples for ASP, as well as examples of Default Logic (DL) (Reiter, 1980b), often have a simple stratified structure: the goal is to reason on an existing incomplete knowledge base, for example, by adding default assumptions. More complex examples can have several levels of stratification. However, neither autoepistemic logic (AEL) (Moore, 1984), default logic, nor ASP preserve this inherent stratification. As a result, a theory in each of these logics is a theory that refers to its own information content through a reflexive epistemic operator (see (Denecker et al., 2011) for a recent account). This is a source of complexity that complicates both their semantics and their reasoning procedures.

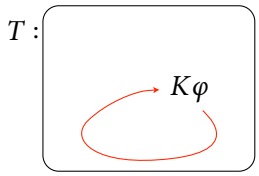

Figure 7.1: Autoepistemic semantics

By contrast, OEL maintains a stratified representation where each level extends the knowledge of the lower levels. This simplifies the logic considerably, while still being able to handle a lot of useful applications from AEL or DL, as we will show here. Contrary to AEL, DL or ASP, an OEL theory always defines a unique belief set, represented as a set of possible worlds. We will show that OEL solves some well-known problems of ASP in the context of epistemic applications. Indeed, ASP is a rather limited epistemic logic, in the sense that the reading of an ASP program as a definition of the beliefs of an agent is no longer valid in the presence of disjunctive information. Because of this, Gelfond (1991) introduced a strong introspection operator. However, this gives a language with two different epistemic operators, which we will claim is overly complex. OEL does not have this problem.

Syntactically, OEL extends FO; the only difference with FO is that OEL is a *closed domain* version of FO: all possible worlds share the same domain and interpretation of terms, like many first-order modal logics. With exception of this feature, OEL is a conservative extension of FO; its epistemic operator stands orthogonal to the other extensions of FO we discussed in Chapter 2, and hence seamlessly integrates with them. By combining them, a very rich KR language is obtained in which many of the motivating examples in DL, AEL and ASP have a natural expression. In some of the examples throughout this section we will make use of these extensions.

The work presented in this chapter extends the initial work in (Konolige, 1988a; Denecker et al., 2010b), in several ways. First, we prove that, in a given finite domain, the data complexity of model checking, satisfiability checking and query answering for OEL theories is in Δ_2^P, which is indeed lower than for AEL and DL, where some instances of satisfiability checking problems can be proven to be Σ_2^P-complete. We show how a model generator for OEL can be implemented. We define a number of classes of theories for which the model generation and query answering tasks can be solved approximately.

Second, we illustrate the use of OEL and of model generation in the context of a scheduling problem with an epistemic component. Third, we extend OEL to a logic for distributed epistemic agents, which we call *distributed ordered epistemic logic* (d-OEL). Knowledge bases are still hierarchically ordered, but now theories at one level no longer automatically

possess all the knowledge of lower levels.

Distributed ordered epistemic logic can cope with distributed knowledge, which makes it relevant for a number of new application areas. One example is the specification of access control policies. Formal specification languages used for this purpose (e.g., (Barker, 2009)) often include a construct allowing one policy manager to query the knowledge of another. Another potential application area is the Semantic Web[1], which can be seen as a huge network linking different sources of data and knowledge, often in the form of ontologies. Several proposals haven been made to combine data from such ontologies, for example, through the use of *bridge rules* (Bouquet et al., 2003). Others address the need for expressing defaults such as "if source x does not specify the color of the car, we assume its color is black". In yet other approaches, the web is seen as an open environment consisting of purely positive knowledge bases that do not contain negative information. This is to avoid potential inconsistencies between different sources (Berners-Lee et al., 2005). Here, a limited form of "negation-as-failure" called *scoped negation* (Polleres et al., 2006; Berners-Lee et al., 2005) has been proposed: this is an epistemic operator to query whether a specific source (e.g., some ontology) does not know a proposition. Common to all such applications is the presence of a number of knowledge sources that can query each other through some form of epistemic operator. We will argue that the logic d-OEL provides a simple formalism with a precise semantics to tackle some of these applications in a natural way.

The rest of this chapter is organized as follows. First we define syntax and semantics of both OEL and d-OEL, and prove a number of properties about both languages. After that, we have a closer look at model generation for OEL and d-OEL and prove our complexity results. Finally, we go into more detail about the relationship between OEL and other knowledge representation languages and we conclude.

7.1 Syntax and semantics

The logic OEL was defined first in (Konolige, 1988a) and later independently in (Denecker et al., 2010b). This logic has epistemic operators $K_{T'}$ that allow one theory T to query another theory T'. While T' may in turn query a third theory T'', cycles are not allowed. To enforce this hierarchical structure, the theories must be partially ordered. Moreover, in those papers, it was assumed that these partially ordered theories are increasingly expert in the domain; i.e., every T also knows everything that the theories below it know. In this section, we first recall the original definition of OEL and prove some additional properties of it. Then we generalize OEL to *distributed ordered epistemic logic* (d-OEL). In this logic a theory does not have all the knowledge of the theories below it. This is a useful extension, for example, it allows us to model agents with contradicting beliefs.

[1] http://www.w3.org/standards/semantic web/

7.1.1 Ordered Epistemic Logic

The syntax of OEL is defined as follows.

Definition 7.1. An ordered epistemic theory over vocabulary Σ is a pair $\langle \mathcal{T}, \leq \rangle$ of a finite set $\mathcal{T}$ of theories and a partial order $\leq$ on $\mathcal{T}$, such that each $T \in \mathcal{T}$ is a theory in the language $\mathcal{L}_T$, which is inductively defined as first-order logic with one extra formula-building rule: If φ is a formula in $\mathcal{L}_{T'}$ and $T' < T$, then $K_{T'}(\varphi)$ is in $\mathcal{L}_T$.

It is straightforward to extend the syntax (and the semantics we define further down) for OEL over first-order logic with other objective language constructs from FO(·) such as definitions (where the modal expressions $K_{T'}(\varphi)$ can be used in the bodies of the definition rules) and aggregates (see Section 2.3).

If the partial order $\leq$ is clear from context, we often identify an ordered epistemic theory with its set of theories $\mathcal{T}$. For each $T \in \mathcal{T}$, we denote by Σ_T the vocabulary that appears objectively (i.e., not in the scope of some $K_{T'}$) in T. Similarly, we also define $\Sigma_{\leq T}$ as the set of predicate symbols in $\mathcal{T}_{\leq T}$, where $\mathcal{T}_{\leq T}$ denotes the restriction of $\mathcal{T}$ to $\{T' \in \mathcal{T} \mid T' \leq T\}$.

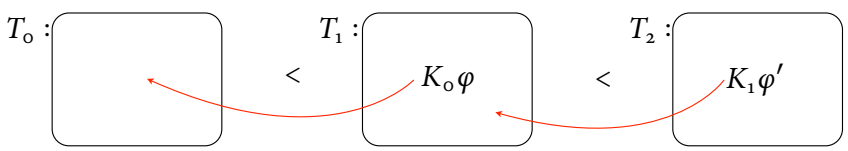

Figure 7.2: Ordered epistemic semantics

Self-referential statements such as $T = \{\neg K_T(P) \Rightarrow Q\}$ cannot be expressed in OEL. The benefit is that the semantics can be defined as an extension of FO semantics. The definition performs induction over the partial order $\leq$.

Definition 7.2. Let $\langle \mathcal{T}, \leq \rangle$ be an OEL theory over Σ and I_σ a pre-interpretation for a subset σ of Σ^F. We call the symbols of σ *rigid*. For an interpretation I of Σ extending I_σ, the relation I satisfies T, denoted $I \vDash_{OEL} T$, is defined as follows:

- For every $T' < T$, it holds that $I \vDash_{OEL} T'$;
- For an atom $P(\bar{t})$, $I \vDash_{OEL} P(\bar{t})$ iff $\bar{t}^I \in P^I$;
- For a modal literal $K_{T'}(\varphi)$, $I \vDash_{OEL} K_{T'}\varphi$ iff $J \vDash_{OEL} \varphi$ for all Σ-interpretations J extending I_σ such that $J \vDash_{OEL} T'$;
- The inductive cases for $\wedge, \vee, \neg, \exists, \forall$ are defined as usual.

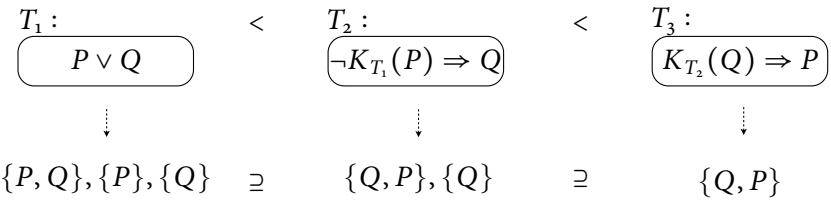

Figure 7.3: The unique belief system of OEL theory $\mathcal{T} = \{T_1, T_2, T_3\}$ is $\{\{P, Q\}\}$.

The unique *belief system* $Mod(\mathcal{T})$ of an OEL theory $\langle \mathcal{T}, \leq \rangle$ with $\mathcal{T} = \{T_1, \ldots, T_n\}$ is the set $\mathcal{M}$ of all interpretations I extending I_σ for which $I \vDash_{OEL} T_i$ for every $T_i \in \mathcal{T}$.

For a $\leq$-minimal theory T, the absence of modal operators means that $\vDash_{OEL}$ reduces to classical $\vDash$ with respect to interpretations extending I_σ. For simplicity, in the rest of this chapter we will only consider Herbrand interpretations of the vocabulary Σ. Because of this, all function and constant symbols are *rigid*: they have the same interpretation in all possible worlds. Moreover, because our complexity results need a *finite* domain, we will also assume that there are no function symbols and only a finite number of constants. However, apart from the complexity results and transformations to FO, our definitions and results can be generalized to interpretations extending any given pre-interpretation. Also, in what follows, we assume without loss of generality that modal operators are only applied to FO formulas. Indeed, nestings of modal operators can be removed by a simple generalization of such transformations for S_5[2].

Example 7.3. Let $\mathcal{T}$ be an OEL theory consisting of three theories $T_1 = \{P \vee Q.\}$, $T_2 = \{\neg K_{T_1}(P) \Rightarrow Q.\}$ and $T_3 = \{K_{T_2}(Q) \Rightarrow P.\}$ and assume these theories are ordered as follows: $T_1 < T_2 < T_3$. According to Definition 7.2, the unique belief system $Mod(\mathcal{T})$ is the set of all interpretations that satisfy all theories in T. Since T_1 contains no modal operators, the interpretations that satisfy this theory are $\{P\}, \{Q\}$ and $\{P, Q\}$. Theory T_2 contains the modal literal $K_{T_1}(P)$. This literal is true if and only if P is true in all models of T_1. This is clearly not the case, thus $K_{T_1}(P)$ is false. Thus, the only models of T_1 that also satisfy T_2 are those in which Q is true, that is, $\{Q\}$ and $\{Q, P\}$. Finally, theory T_3 contains another modal literal $K_{T_2}(Q)$. This literal is true if in all models of T_2, Q holds. This is the case and thus $K_{T_2}(Q)$ evaluates to true. The only model of T_2 (and T_1) that also satisfies T_3 is the interpretation $\{P, Q\}$. We can conclude that the unique belief system $Mod(\mathcal{T})$ is the set $\{\{P, Q\}\}$. An illustration of this is given in Figure 7.3.

Example 7.4. Consider the following scenario from (Gelfond & Lifschitz, 1991). Assume a database with complete knowledge about the grade point average (GPA) of students and

[2] To see this, note that $K_{T_1}((\neg)K_{T_2}(\varphi))$ is equivalent with $(\neg)K_{T_2}(\varphi)$, $K_{T_1}(\varphi_1 \vee K_{T_2}(\varphi_2))$ with $K_{T_1}(\varphi_1) \vee K_{T_2}(\varphi_2)$ and $K_{T_1}(\varphi_1 \wedge K_{T_2}(\varphi_2))$ with $K_{T_1}(\varphi_1) \wedge K_{T_2}(\varphi_2)$.

partial knowledge about whether they belong to a minority group. It can be represented in FO as:

$$DB = \left\{ \begin{array}{l} \forall x(FairGPA\,(x) \Leftrightarrow x = Bob \vee x = Ann\,). \\ \forall x(HighGPA\,(x) \Leftrightarrow x = John\,). \\ Minority\,(Ann\,). \end{array} \right\}$$

Now suppose the department decides that they want to interview students who may be, but are not necessarily, eligible for a grant, where eligible students are those with either a high GPA or a fair GPA and belonging to a minority. We now build on top of the database DB a second layer $T_I > DB$:

$$T_I = \left\{ \begin{array}{l} \forall x\; Interview\,(x) \Leftrightarrow \\ \qquad FairGPA\,(x) \wedge \neg K_{DB}(Minority\,(x)) \\ \qquad \wedge \neg K_{DB}(\neg Minority\,(x)). \end{array} \right\}$$

The theories T_{DB} and T_I both have several models (in which $Minority\,(Bob\,)$ and $Minority\,(John\,)$ can both be true and false). However, in every interpretation of $Mod(\mathcal{T})$, $Interview\,(Bob\,)$ is true.

We now illustrate how the *closed world assumption* (Reiter, 1977) can be represented in OEL. As an alternative to the database DB, we can decide to store in the database DB' only the positive facts about the relations for which we have complete information. We can then add an extra layer of expertise, adding the extra knowledge *on top* of DB', that if the DB' *does not say* that a student *has a particular GPA*, we conclude that he/she does not have that GPA. This extra layer T_{cwa} contains an ASP-style CWA:

$$DB' = \left\{ \begin{array}{l} FairGPA\,(Ann\,) \wedge FairGPA\,(Bob\,) \\ \qquad \wedge HighGPA\,(John\,) \wedge Minority\,(Ann\,) \end{array} \right\}$$

$$T_{cwa} = \left\{ \begin{array}{l} \forall x(\neg FairGPA\,(x) \Leftrightarrow \neg K_{DB'} FairGPA\,(x)). \\ \forall x(\neg HighGPA\,(x) \Leftrightarrow \neg K_{DB'} HighGPA\,(x)). \end{array} \right\}$$

Now in all $M \in Mod(\mathcal{T})$, where $\mathcal{T}$ is the OEL theory $\mathcal{T} = DB' \leq T_{cwa}$ it holds that $FairGPA\,(John\,)$ is false. We could now add another layer of expertise to the expertise of DB' and T_{cwa}, for example, again the theory T_I expressing when an interview should happen.

While a number of auto-epistemic features can be represented in OEL, OEL is in general less expressive than AEL, for example, one cannot express self-referential statements. The advantage this gives OEL is that the semantics are simpler, since we always have exactly one belief system (In contrast, a theory in AEL might have many or no stable expansions), and, as we show further on, OEL has a lower complexity than AEL. However, despite these differences, in some useful cases we can still view an ordered epistemic theory as an autoepistemic theory.

For a given OEL theory $\mathcal{T}$, define $AEL(\mathcal{T})$ as the modal theory obtained by taking the union of all $T \in \mathcal{T}$ and turning each occurrence of a modal operator K_T into the same operator K. Intuitively, an operator K_T expresses the knowledge of the source T and all sources lower in the hierarchy, while the operator K can be seen as a *distributed knowledge* operator, expressing the combined knowledge of *all sources*. This operation does not always preserve equivalence.

Example 7.5. Let $\mathcal{T} = \{T_1, T_2\}$, $T_1 < T_2$, and $T_1 = \{\}$, $T_2 = \{Q, \neg K_{T_1}(Q) \Rightarrow R\}$. In this case, $\{Q, R\}$ is the only possible world of $\mathcal{T}$, while the unique stable expansion of $AEL(\mathcal{T})$ contains a second possible world $\{Q\}$.

This difference is caused by the fact that T_2 itself possesses additional knowledge about the atom Q that occurs in its query to T_1. Therefore, while the system as a whole knows that Q is true, the theory T_1 in isolation does not. When all knowledge about a particular atom is contained within a single theory T_i, this cannot be the case and we would therefore expect $\mathcal{T}$ and $AEL(\mathcal{T})$ to be equivalent. However, in the translation to AEL, there is a subtle case where one theory T_i might still influence the interpretation of atoms that were described in a different theory T_j, and that is when T_i introduces an inconsistency that eliminates all models of T_j. To rule out such cases, we use the following concept.

Definition 7.6. An autoepistemic theory T is *permaconsistent* if every theory T' that can be constructed from T by independently replacing all non-nested occurrences of modal literals by **t** or **f** is consistent.

Proposition 7.7. *Given an OEL theory $\mathcal{T}$. If (1) each predicate symbol $P \in \Sigma$ has objective occurrences in at most one sub-theory $T \in \mathcal{T}$, (2) P appears only in modal literals $K_{T'}(\varphi)$ such that $T' < T$ and (3) $AEL(\mathcal{T})$ is a permaconsistent theory, then $AEL(\mathcal{T})$ has a unique autoepistemic expansion B equal to the unique belief system $Mod(\mathcal{T})$ of $\mathcal{T}$.*

Proof. This is an immediate consequence of the stratification results for AEL from (Vennekens et al., 2006). □

While permaconsistency may seem like a strong criterion, it is often met in practice. For instance, the entire subset of AEL that corresponds to ASP satisfies it[3].

While the theory (DB, T_I) from Example 7.4 also leads to a permaconsistent AEL theory, it fails to satisfy the other condition of Prop. 7.7, since both DB and T_I contain an objective occurrence of $FairGPA$. However, only DB actually provides information about $FairGPA$, because T_I does not pose any constraints on the interpretation of this predicate. Indeed, each model of the database DB can be extended with an interpretation of $Interview$ that satisfies the theory T_I. As the next proposition shows, this conditions suffices to ensure that the models of the OEL theory (DB, T_I) are also models of $AEL(DB, T_I)$.

Proposition 7.8. *Given an OEL theory $\mathcal{T}$. If for each $T \in \mathcal{T}$, every $\Sigma_{\leq T}$ model of $\mathcal{T}_{\leq T}$ can be extended to a model of $\mathcal{T}$ and for every literal $K_T\varphi$ in $\mathcal{T}$, φ contains only symbols of Σ_T, then the belief system $Mod(\mathcal{T})$ is an expansion of the theory $AEL(\mathcal{T})$.*

Proof. For simplicity, we assume that $\mathcal{T}$ is a propositional theory, that each $T_i \in \mathcal{T}$ has at most one occurrence of a modal operator K_{T_j}, and if so, that $j = i - 1$. The proof easily extends to the general case. If T is a theory containing $K_{T'}\varphi$ and $\mathcal{M}$ a set of interpretations, we denote by $T\langle\mathcal{M}\rangle$ the theory derived from T by evaluating $K_{T'}\varphi$ in $\mathcal{M}$, i.e., replacing each modal literal $K_{T'}\varphi$ by $\mathbf{t}$ if φ holds in all interpretations $I \in \mathcal{M}$ and by $\mathbf{f}$ otherwise. Also, for a FO theory T, we denote with $mod(T)$ the set of all models M of T.

We denote with $\mathcal{M}_i$ a set of $\Sigma_{\leq T_i}$ interpretations. It follows directly from the semantics of OEL that the modal operators $K_{T_{i-1}}$ that appear in a theory T_i are evaluated in the belief system $\mathcal{M}_{i-1}$ of the theory $\mathcal{T}_{\leq i-1}$. Therefore it holds that a set $\mathcal{M}_i$ is an OEL belief system of $\mathcal{T}_{\leq T_i}$ iff

$$\mathcal{M}_i = mod(T_0 \cup \bigcup_{1 \leq j \leq i} (T_j\langle\mathcal{M}_{j-1}\rangle)),$$

where for every $j < i$, $\mathcal{M}_j$ is the belief system of $\mathcal{T}_{\leq T_j}$. It now holds for every j such that $1 \leq j \leq i$ that $T_j\langle\mathcal{M}_{j-1}\rangle = T_j\langle\mathcal{M}_i\rangle$. Indeed, it holds that $\mathcal{M}_i|_{\leq\Sigma_{j-1}} \subseteq \mathcal{M}_{j-1}$, since each model of T_i is by definition also a model of each $T_j \leq T_i$. But also $\mathcal{M}_{j-1} \subseteq \mathcal{M}_i|_{\leq\Sigma_{j-1}}$, because every model $m \in \mathcal{M}_{j-1}$ is extendible to a model $m' \in \mathcal{M}_i$. Therefore,

$$\mathcal{M}_i = mod(T_0 \cup \bigcup_{1 \leq j \leq i} (T_j\langle\mathcal{M}_i\rangle)),$$

and thus that $\mathcal{M}_i$ is a fixpoint of the $D_{AEL(\mathcal{T}_{\leq T_i})}$ (see Section 2.5) operator. We can conclude that if $\mathcal{M}$ is the belief system of a OEL theory $\mathcal{T}$, then $\mathcal{M}$ is indeed a stable expansion of $AEL(\mathcal{T})$. □

[3] According to the original paper on stable semantics (Gelfond & Lifschitz, 1988), an ASP rule `p ← q, not r` corresponds to the AEL formula $p \Leftarrow (q \wedge \neg Kr)$. Since modal literals occur only in the body of logical implications, the ASP fragment of AEL is clearly permaconsistent.

The other direction of the proof does not hold because AEL theories can have self-supporting expansions, as is illustrated by the following example.

$$T_1 : \{K_{T_0}(R) \Rightarrow R.\}$$
$$T_0 : \{R \vee \neg R.\}$$

The OEL theory $\mathcal{T} = T_0, T_1$ has as its unique belief set $Mod(\mathcal{T})$ the set $\{\{\}, \{R\}\}$. This set is also an expansion. However, the autoepistemic theory $AEL(\mathcal{T})$ has a second expansion, that is, the self-supporting expansion $\{\{R\}\}$. It has been argued that these self-supporting expansions are in fact anomalies, and several attempts have been made to define autoepistemic expansions that have a stronger notion of 'groundedness'. Konolige (1988b) defined *moderately grounded expansions* (which are in fact maximal expansions) that eliminate some, but not all of these unwanted self-supporting expansions. In the same paper he also defined *strongly grounded* expansions[4], which are more restrictive then moderately grounded expansions, but do have the undesirable property of depending on the syntactic representation of the theory (in the sense that two equivalent theories can give rise to different expansions). The issue of self-supported expansions was resolved in a semantically appealing way by the introduction of stable and well-founded semantics for AEL by Denecker et al. (2003). It is worth noting that default logic (Reiter, 1980b) does not admit such self-supported extensions. As shown by Denecker et al., AEL under the stable semantics extends default logic in the sense that the so-called Konolige mapping $\frac{\alpha : M\beta}{\gamma} \longrightarrow K\alpha \wedge \neg K\neg\beta \Rightarrow \gamma$ is an equivalence preserving mapping from default logic to AEL under the stable semantics.

7.1.2 Distributed Ordered Epistemic Logic

In this section we define a generalization of OEL, called *distributed ordered epistemic* logic (d-OEL), where higher levels no longer possess the knowledge of lower levels, as is the case in the autoepistemic setting of OEL. This generalization has a number of interesting theoretical applications, namely, the class of problems where there are distributed epistemic agents that can query each other.

Definition 7.9. A *distributed ordered epistemic theory* over a set of vocabularies $\overline{\Sigma} = \{\Sigma_{T_1}, \ldots, \Sigma_{T_n}\}$ is a pair $\langle \mathcal{T}, \leq \rangle$ of a finite set $\mathcal{T} = \{T_1, \ldots, T_n\}$ of theories and a partial order $\leq$ on $\mathcal{T}$, such that each $T_i \in \mathcal{T}$ is a Σ_{T_i}-theory in the language $\mathcal{L}_T$ (as defined in Definition 7.1).

We again define the semantics by induction over the partial order $\leq$.

[4]The original definition later proved incorrect and a correct version was independently given by (Konolige, 1989) and (Marek & Truszczynski, 1989).

Definition 7.10. Let $\langle \mathcal{T}, \leq \rangle$ be an distributed ordered epistemic theory, and I_σ a pre-interpretation for a subset $\sigma \subseteq \Sigma^F_{T_1} \cup \ldots \Sigma^F_{T_n}$. For a Σ_T-interpretation I extending I_σ, the relation $I \models_{dist} T$ is defined as follows:

- For an atom $P(\bar{t})$, $I \models_{dist} P(\bar{t})$ iff $\bar{t}^I \in P^I$;
- For a modal literal $K_{T'}(\varphi)$, $I \models_{dist} K_{T'}(\varphi)$ iff $J \models \varphi$ for all $\Sigma_{T'}$-interpretations J such that $J \models_{dist} T'$;
- The inductive cases for $\wedge, \vee, \neg, \exists, \forall$ are defined as usual.

This semantics differs from that of OEL (Def. 7.2) by no longer requiring that a model of a theory $T \in \mathcal{T}$ has to satisfy every lower theory $T' < T$. As a consequence, this changes the way a modal operator K_T is evaluated. K_T now looks at all models of T, instead of only those models of T that are also models of the subtheories of T. As for OEL, in the rest of this chapter we will only consider Herbrand interpretations, and will therefore assume that all vocabularies $\Sigma_{T_i} \in \overline{\Sigma}$ share the same Herbrand universe. As a consequence, we can also assume that there are no nested modal operators.

It follows directly from Def. 7.10 that, for each theory $T \in \mathcal{T}$, we can construct the set $\mathcal{M}_T$ of all d-OEL models of T (i.e., the set of interpretations M such that $M \models_{dist} \mathrm{T}$) by means of a bottum-up construction along the partial order $\leq$. That is, for the minimal theories $T_1^{min}, \ldots, T_n^{min} \in \mathcal{T}$, each $\mathcal{M}_{T_i^{min}}$ is just the set of classical models of T_i^{min}; and once we have constructed the sets of models $\mathcal{M}_{T'}$ for all $T' < T$, we can construct the models of T itself, by interpreting all modal formulas $K_{T'}(\varphi)$ according to $\mathcal{M}_{T'}$ and then constructing the classical models of the remaining objective theory. It follows from this construction process that each $T \in \mathcal{T}$ has a unique set of models. Note that this construction remains valid in case the formula φ in $K_{T'}(\varphi)$ contains another modal operator as long as the nested one refers to a theory lower than T' in the partial order.

Definition 7.11. The unique *belief system* $Mod(\mathcal{T})$ of a d-OEL theory $\langle \mathcal{T}, \leq \rangle$ with $\mathcal{T} = \{T_1, \ldots, T_n\}$ is the tuple $(\mathcal{M}_{T_1}, \ldots, \mathcal{M}_{T_n})$ such that for each i, $\mathcal{M}_{T_i}$ is the set of all Σ_{T_i}-Herbrand interpretations I for which $I \models_{dist} T_i$.

Example 7.12. Let us again take the theory $\mathcal{T}$ from Example 7.3, but this time as an d-OEL theory. As illustrated in Figure 7.4, the sets of interpretations that satisfy T_1 and T_2 stay the same, but the interpretations that satisfy T_3 are $\{P\}$ and $\{P, Q\}$ as opposed to just $\{P, Q\}$ for the OEL theory. Indeed, in d-OEL, T_3 lacks the knowledge of the lower level theory T_2 that Q is true.

The next definition defines a syntactic transformation from an OEL theory to a d-OEL theory.

Definition 7.13. Given an OEL theory $\langle \mathcal{T}, \leq \rangle$ over the vocabulary Σ, we define the d-OEL theory $\langle \mathcal{T}^{\text{d-OEL}}, \leq \rangle$ as:

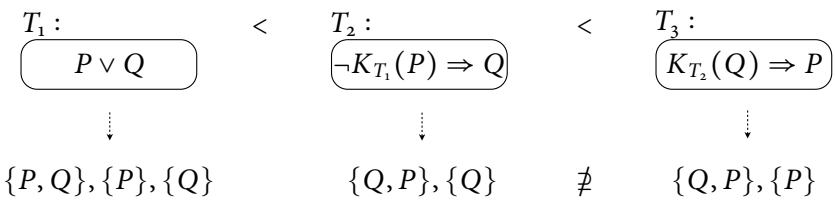

Figure 7.4: The unique belief system of d-OEL theory $\mathcal{T} = \{T_1, T_2, T_3\}$ is the tuple $\{\{\{P, Q\}, \{P\}, \{Q\}\}; \{\{P, Q\}, \{Q\}\}; \{\{P, Q\}, \{P\}\}\}$.

- For every $T \in \mathcal{T}$, $\mathcal{T}^{\text{d-OEL}}$ contains a corresponding element $T^{\text{d-OEL}} = T \cup \bigcup_{T' < T} T'$.
- $\mathcal{T}^{\text{d-OEL}}$ contains one maximal element $T^{\text{d-OEL}}_{max} = (T_1^{max})^{\text{d-OEL}} \cup \ldots \cup (T_n^{max})^{\text{d-OEL}}$, where $T_1^{max}, \ldots, T_n^{max}$ are the maximal elements of $\langle \mathcal{T}, \leq \rangle$.

The following proposition now states that an OEL theory $\langle \mathcal{T}, \leq \rangle$ is equivalent with the corresponding d-OEL theory $\langle \mathcal{T}^{\text{d-OEL}}, \leq \rangle$.

Proposition 7.14. *Given an OEL theory $\langle \mathcal{T}, \leq \rangle$ over the vocabulary Σ, then it holds for any Σ-interpretation I that $I \models_{dist} T^{d\text{-}OEL}_{max}$ iff $I \models_{OEL} \mathcal{T}$.*

The proof of this proposition is straightforward. Indeed, it is obviously the case that when the higher-up theories syntactically include the lower ones, information is guaranteed to increase along the partial order $\leq$, that is, the semantics of OEL are simulated. As a consequence of this proposition, we can see OEL as a sublogic of d-OEL.

Example 7.15. To illustrate how d-OEL can be used to model the beliefs of different agents, we reconsider Example 7.4, but now assume we also have a second database, talking about which students are minorities:

$$DB_2 = \left\{ \begin{array}{l} \neg Minority(Ann). \\ Minority(John) \end{array} \right\}.$$

Clearly, both databases have contradictory beliefs about which students are minorities. We can now build another layer T_+ on top of these two databases that expresses how such contradictory data should be handled; for example, T_+ could express that when the two databases do not agree, the student should be interviewed:

$$T_+ = \left\{ \begin{array}{l} \forall x\ Interview(x) \Leftrightarrow \\ \qquad K_{DB}(Minority(x)) \wedge K_{DB_2}(\neg Minority(x)) \\ \vee\ K_{DB}(\neg Minority(x)) \wedge K_{DB_2}(Minority(x)) \end{array} \right\}$$

The theory of interest here, T_+, has exactly one Herbrand model I, in which $Interview^I = \{Ann\}$.

7.2 Problem Solving

This section studies the inference tasks we can do for OEL and d-OEL. We start by showing that every d-OEL theory is equivalent to a d-OEL theory consisting only of first-order theories. A consequence of this, is that each OEL theory is equivalent to a single FO theory. In the case of finite Herbrand domains, these first-order theories can actually be constructed, and can then be used as the basis for implementing model checking, model generation and query answering methods. Next, we investigate data complexity of model checking, satisfiability checking and query answering. Finally, we propose to use the propagation method from Chapter 3 to define an approximative but more efficient reasoning method for OEL and d-OEL. This reasoning method is of course not complete, but unfortunately also not always sound. However, we show that for a number of practically useful OEL and d-OEL theories, we *can* guarantee soundness.

7.2.1 From OEL and d-OEL to FO

We start by showing that each d-OEL theory is equivalent to a d-OEL theory consisting of only pure FO theories. In order to define these theories precisely, we first need some additional vocabulary.

Definition 7.16. For a given set of vocabularies $\Sigma_{T_1}, \ldots, \Sigma_{T_n}$ and a d-OEL theory $\mathcal{T} = \{T_1, \ldots, T_n\}$, we define, for each i, $\Sigma_{T_i}^{FO}$ as the vocabulary consisting of Σ_{T_i} together with, for each modal expression $K_{T_j}(\varphi[\bar{x}])$ occurring in T_i, a new predicate symbol $P_{T_j,\varphi}$ with arity $\#(\bar{x})$.

Using this vocabulary, we can now define a new d-OEL theory, only consisting of FO theories. However, note that this d-OEL theory depends crucially on the given Herbrand domain, because for each formula $K_T\varphi(\bar{x})$ *and* each tuple of constants $\bar{c}$, we will explicitly add either a literal $P_{T,\varphi}(\bar{c})$ or a literal $\neg P_{T,\varphi}(\bar{c})$ to the theory (depending on whether T knows φ or not). As a consequence, the transformation will lead to an increase of the size of the theory that is polynomial in the size of the domain and, as we will show below, assuming a finite domain, computing it might involve solving an NP-complete problem.

Definition 7.17. For a given set of vocabularies $\Sigma_{T_1}, \ldots, \Sigma_{T_n}$ and a d-OEL theory $\mathcal{T} = \{T_1, \ldots, T_n\}$, we define $FO(T_i)$ as the first-order theory over $\Sigma_{T_i}^{FO}$ obtained from T_i by substituting $P_{T_j,\varphi}(\bar{x})$ for modal literals $K_{T_j}(\varphi[\bar{x}])$, and adding, for each such modal literal and tuple $\bar{t}$ of terms (i.e., elements of the Herbrand universe), either the literal $P_{T_j,\varphi}(\bar{t})$

if $T_j \models_{dist} \varphi[\bar{t}]$, or $\neg P_{T_j,\varphi}(\bar{t})$ otherwise. Now $FO(\mathcal{T})$ is defined as the d-OEL theory $\{FO(T_1), \ldots FO(T_n)\}$ (inheriting the order from $\mathcal{T}$).

The following proposition now states that $FO(\mathcal{T})$ is actually equivalent to $\mathcal{T}$ in the following sense.

Proposition 7.18. *For a given set of vocabularies $\Sigma_{T_1}, \ldots, \Sigma_{T_n}$ and a d-OEL theory $\mathcal{T} = \{T_1, \ldots, T_n\}$, there is an isomorphism between $mod(\mathcal{T})$ and $mod(FO(\mathcal{T}))$ as follows. Let $(\mathcal{M}_1, \ldots, \mathcal{M}_n)$ denote the unique belief system of $\mathcal{T}$ (i.e., $mod(\mathcal{T})$), and $(\mathcal{M}_1^{FO}, \ldots, \mathcal{M}_n^{FO})$ the unique belief system of $FO(\mathcal{T})$ (i.e., $mod(FO(\mathcal{T}))$). Then for all i, $\mathcal{M}_i^{FO}|_{\Sigma_{T_i}} = \mathcal{M}_i$.*

Proof. This follows immediately from an easy argument over induction on the order of the theories, from the semantics of d-OEL (Definition 7.10) and the way the literals $P_{T_i,\varphi}(\bar{t})$ are added to the theories. □

The way in which we define $FO(\mathcal{T})$ here, is similar to the RES procedure of (Levesque & Lakemeyer, 2000). This procedure transforms a modal query w.r.t. a first-order knowledge base into a standard first-order query. Our transformation here can be seen as a generalization of the RES procedure, in the sense that Levesque & Lakemeyer assume a particular fixed infinite domain (isomorphic to the natural numbers), and that they only consider two theories: an FO database, and on top of that a theory which only contains a single modal literal.

This now brings us to the question of *constructing* $FO(\mathcal{T})$. Assume we have a such a d-OEL-theory $\mathcal{T}$ consisting of only an FO theory T and a theory consisting of a single modal literal $K_T(\varphi)$. In order to construct $FO(\mathcal{T})$, we need to find out the value of $K_T(\varphi)$, and therefore whether $T \models_{dist} \varphi$. Since T is a pure FO theory, this is equivalent to finding out whether $T \models \varphi$ (in the context of a given domain). In general, this task is undecidable. This is, for example, the case for the infinite domain that Levesque & Lakemeyer considers. However, in the case of a *finite* Herbrand domain, we can actually construct $FO(\mathcal{T})$.

We assume w.l.o.g. that $\mathcal{T} = \{T_1, \ldots, T_n\}$ where $T_1 < T_2 < \ldots < T_n$. Now the theory $FO(\mathcal{T})$ can be constructed over the order < on the theories, starting from $FO(T_1) = T_1$ as a base case. Assume we have already constructed $FO(T_i)$. We can then construct $FO(T_{i+1})$, by first adding all formulas of T_{i+1} in which modal literals $K_{T_j}(\varphi\bar{x})$ have been replaced by $P_{T_j,\varphi}(\bar{x})$. Then all that is left to do is adding for every modal literal $K_{T_j}\varphi$ of T_{i+1} and tuple $\bar{t}$ of terms, either $P_{T_j,\varphi}(\bar{t})$ or $\neg P_{T_j,\varphi}(\bar{t})$. To decide which of these two to add, we need to know whether $T_j \models_{dist} \varphi(\bar{t})$. Now Proposition 7.18 implies that this is the case if $FO(T_i) \wedge \neg\varphi(\bar{t})$ does not have a Herbrand model. Any FO model generator, such as (Wittocx et al., 2008; Claessen & Sörensson, 2003; Torlak & Jackson, 2007) could be used to check this. Because we add one literal for each tuple $\bar{t}$ of terms, for a fixed number of theories and modal literals, we get that constructing $FO(\mathcal{T})$ can be done by a polynomial number (in the size of the Herbrand universe) of calls to an NP-oracle, and thus is at most Δ_2^P.

While an d-OEL-theory is equivalent to another d-OEL-theory only consisting of first-order theories, the following proposition states that an OEL-theory is actually equivalent to a single FO theory.

Proposition 7.19. *For a given vocabulary Σ (and thus also a given Herbrand universe) and an OEL theory $\mathcal{T}$, there exists an FO theory $FO(\mathcal{T})$ over an extended vocabulary $\Sigma' \supseteq \Sigma$, such that for every Σ'-Herbrand interpretation M it holds that $M \models FO(\mathcal{T})$ iff $M|_\Sigma \in Mod(\mathcal{T})$.*

Proof. This follows immediately from Proposition 7.14, which states that every OEL theory is equivalent with a d-OEL theory, in the sense that the models of the top-level theory T^{max} of the d-OEL theory correspond to the models of the OEL theory. Together with Proposition 7.18, this immediately proves the proposition. □

7.2.2 Complexity of model checking, satisfiability checking and query answering

We start this section by looking at the data complexity of model checking, satisfiability checking and query answering. To measure data complexity, we consider inference in a class of theories of the form $\mathcal{T} = \{DB, T_1, \ldots, T_n\}$ over finite vocabularies Σ without function symbols in which $T_1, \ldots, T_n$ are fixed and only DB and the set of constants in Σ vary. Moreover, we assume that DB is a database theory, i.e., a set of ground literals, and that it is minimal in the theory order $\leq$. We denote the class of such theories as $\mathcal{DB}$.

Proposition 7.20. *The decision problems of model checking, satisfiability checking and query answering in the class of d-OEL theories $\mathcal{DB}$ are in Δ_2^P. There are theories for which these tasks are both NP-hard and co-NP-hard. The same holds for the class of OEL theories $\mathcal{DB}$.*

Proof. Assume $\mathcal{T}$ is a d-OEL-theory. As we already showed, the theory $FO(\mathcal{T})$ can be constructed with a polynomial (in the size of the Herbrand universe) number of calls to an NP-oracle. Now, each of the three tasks can be easily implemented using the theory $FO(\mathcal{T})$. Because, in Herbrand interpretations, model checking for FO is polynomial in the size of the domain, satisfiability checking is NP and query answering is co-NP, we obtain that all three tasks are at most Δ_2^P for d-OEL. It follows from the proof of Proposition 7.14 that constructing a d-OEL theory equivalent with a given OEL theory $\mathcal{T}$ can be done in linear time (in the size of the Herbrand universe), therefore the same complexity results hold for OEL theories.

As for the second statement of the proposition, it suffices to focus on the least complex of the three problems, i.e., model checking. We prove the hardness properties by encoding the NP-complete graph kernel problem as a model checking problem of d-OEL. Consider the parameterized theory $\mathcal{T}$ consisting of $DB < T$ where DB encodes a graph G and consists

of, for each pair A, B of domain elements, either the atom $G(A, B)$ or the negative literal $\neg G(A, B)$. This is a complete description of the graph G on the Herbrand universe. Σ^P contains G and two other predicates $Kernel/1$ and P which do not appear in DB.

Next we define T. T specifies that the new propositional symbol P is true iff DB knows that the set represented by $Kernel$ is not a graph kernel of G. A graph kernel is a set of vertices such that no kernel elements are directly connected and each non-kernel element is connected to a kernel element. I.e., T is singleton containing the formula

$$P \Leftrightarrow K_{DB}\neg\varphi$$

where φ expresses that $Kernel$ is a kernel of G:

$$\forall x \forall y(Kernel(x) \wedge Kernel(y) \Rightarrow \neg G(x, y)) \wedge$$
$$\forall x(\neg Kernel(x) \Rightarrow \exists y(Kernel(y) \wedge G(y, x)))$$

It is easy to see that since DB contains no information about $Kernel$, the only way in which DB can know that $Kernel$ is not a kernel of G is if it knows that G has no kernels. Hence, deciding whether $K_{DB}\neg\varphi$ holds, boils down to solving a co-NP hard problem. Thus, checking whether an interpretation in which P is true is a model is a co-NP-complete problem. Checking whether an interpretation in which P is false is a model is an NP-hard problem. This concludes the proof. □

As shown by Niemelä 1992 and Gottlob 1992, the problem of deciding whether an autoepistemic theory has a stable expansion is Σ_2^P-complete. The result in Proposition 7.20 thus asserts our earlier claim that OEL is of a lower complexity then AEL (the inclusion $\Delta_2^P \subseteq \Sigma_2^P$ is believed to be strict).

All this now also shows that inference tasks in d-OEL can be implemented using FO tools. For instance, a d-OEL model generator could be implemented on top of any FO model generator, such as (Wittocx et al., 2008; Claessen & Sörensson, 2003; Torlak & Jackson, 2007), by using the layered approach described in the proof above: first we use a polynomial number of calls to such a model generator to construct $FO(\mathcal{T})$ and then we call the model generator again to find out whether FO($\mathcal{T}$) has a Herbrand model, from which we then can extract the model of $\mathcal{T}$. Such a model generator could be used to solve the following practical problem.

Example 7.21. Imagine that every week a hospital needs to distribute patients over its available rooms. For this, it uses a knowledge base that expresses what a valid assignment is. Possible constraints might be that assigned rooms have to be available, that the maximum capacity of the rooms cannot be exceeded, that patients with a specific treatment need

rooms with appropriate facilities, etc. In FO(·), this would become:

$$T_s = \left\{ \begin{array}{l} (\mathit{Used}(r,d) \Leftrightarrow \exists p\, \mathit{RoomOfPat}(r,p,d)). \\ \forall r\, d\, (\mathit{Used}(r,d) \Rightarrow \mathit{Avail}(r,d)). \\ \forall d\, r\, (\#\{p : \mathit{RoomOfPat}(r,p,d)\} \leq \mathit{Capac}(r)). \\ \forall d\, p\, r\, (\mathit{RoomOfPat}(r,p,d) \wedge \mathit{ReqFacil}(p,f,d) \\ \qquad\qquad \Rightarrow \mathit{HasFacil}(r,f)). \end{array} \right\}$$

The first sentence of this theory, for example, defines that a room is used on a particular day, if on that day, the room is assigned to a certain patient. The other sentences express constraints on how the rooms are used, e.g., the second sentence says that if a room is used on particular day, it has to be available. Assume that a room is unavailable if it is being cleaned, and the cleaning company provides a complete database T_{cl} about when which room is available. A model generator can then be used on $T_s \cup T_{cl}$, together with another complete data base DB about the capacity, etc., to generate a valid schedule.

Now, assume that the hospital is refurbishing its rooms over an extended period of time, and regularly some rooms might not be available. The company that executes the refurbishments has its own knowledge base T_{ref} expressing when rooms will be available. However, unlike the cleaning company, this company can no longer provide *complete* information about the availability. Instead, it might say:

$$T_{ref} = \left\{ \begin{array}{l} \mathit{Avail}(R1, Mo) \wedge \neg \mathit{Avail}(R3, Tue). \\ \mathit{Avail}(R2, Fri) \wedge \forall r\, \neg \mathit{Avail}(r, Thu). \end{array} \right\}$$

The knowledge base T_{ref} indeed contains no information about, e.g., the availability of R_1 on Tuesday. Again, the hospital would like to get a valid schedule. However, this time, we cannot simply give $T_s \cup T_{ref} \cup DB$ to a model generator, since that would try to fill in the missing information about the availability, leading to many schedules: one where R_1 is available on Tuesday, one where R_1 is not available on Tuesday, and so on. The problem here is that we want to use T_s and T_{ref} in different ways: we want to do model generation on T_s, but not on T_{ref}.

We can achieve this by means of OEL's epistemic operator $K_{T_{ref}}$. Let T_s' be T_s together with the sentence:

$$\forall r \forall d (\mathit{Avail}(r,d) \Leftrightarrow K_{T_{ref}}(\mathit{Avail}(r,d))).$$

The interpretation of predicate *RoomOfPat* in the models of the OEL theory $\mathcal{T} = \{T_s', T_{ref}\}$ now represents a *safe* schedule, that is, only rooms of which it is *certain* that they are available (according to T_{ref}) are used in the schedule. Alternatively, the hospital might decide that it is allowed to use a maximum of two 'uncertain' rooms. Such a schedule is represented by the interpretation of *RoomOfPat* in the models of $\mathcal{T}' = \{T_s'', T_{ref}\}$, where T_s'' is the theory

T_s with the rule $\forall r\, d\, (Used(r,d) \Rightarrow Avail(r,d))$ replaced by the following two rules:

$$\forall d\, (2 \geq \#\{r : Used(r,d) \wedge \neg K_{T_{ref}} \neg Avail(r,d) \wedge \neg K_{T_{ref}} Avail(r,d)\}).$$

$$\forall r\, d\, (Used(r,d) \Rightarrow \neg K_{T_{ref}} \neg Avail(r,d)).$$

Note that for the theories $\mathcal{T}$ and $\mathcal{T}$' the OEL and d-OEL semantics coincide (in the sense of Proposition 7.14). However, suppose that the hospital wants to use both the theory T_{cl} from the cleaning company and the theory T_{ref} from the refurbishing company. Since these two theories come from different companies that know nothing about each other, they are most likely inconsistent with each other. This means that any OEL theory that contains both of them will also be inconsistent. In d-OEL, this is not a problem and the hospital can just add a formula:

$$\forall r \forall d (Avail(r,d) \Leftrightarrow K_{T_{ref}}(Avail(r,d))) \wedge K_{T_{cl}}(Avail(r,d)).$$

to T_s (we denote this theory by T_s'''). The interpretation of *RoomOfPatient* in the models of T_s''' of the d-OEL theory $\mathcal{T}'' = \{T_s''', T_{cl}, T_{ref}\}$ again represents such a schedule.

7.2.3 Approximative reasoning for OEL and d-OEL

In the previous section we investigated the complexity of a number of inference tasks for OEL and d-OEL. We proved that model checking, satisfiability checking and query answering are all in Δ_2^P for both OEL and d-OEL. The proof for this also suggested how a model generator could be implemented. Assuming we have an OEL theory $\mathcal{T} = \{T_1, \ldots, T_n\}$, the crucial step and the main source of complexity in the proof of Proposition 7.20 and thus in the model generation algorithm, is determining, for each modal literal $K_{T_j}\varphi$ occurring in T_i and tuple $\bar{t}$, whether $T_j \models_{OEL} \varphi[\bar{t}]$. First, let us assume that $\mathcal{T}$ consists only of two theories $T_1 < T_2$. Now, modal literals $K_{T_1}\varphi[\bar{t}]$ hold if and only if $T_1 \models_{FO} \varphi[\bar{t}]$. Recall that finding out whether a formula holds in all finite Herbrand models of a first-order theory is Co-NP-complete. However, in Chapter 3 we saw a sound but incomplete propagation method, that allows to determine in polynomial time whether $T_1 \models_{FO} \varphi[\bar{t}]$. The obvious question is now whether we can, as we have already done a number of times in this thesis, again use this propagation method to approximate certain inference tasks for OEL (and d-OEL), that is, solve them faster, but incompletely.

To make notation easier, we will from now on and for the remaining of this chapter, use $Approx_\sigma^{ct}(T)$ to denote the definition $Approx_\sigma(T) \cup \{A_T^{ct} \leftarrow \mathbf{t}\}$, as was defined in Chapter 3. Recall that Theorem 3.13 tells us that if $Approx^{ct}(T_1) \models A_\varphi^{ct}(\bar{t})$, then $T_1 \models \varphi[\bar{t}]$. The idea now is to replace the modal literal $K_{T_1}(\varphi[\bar{t}])$ in T_1 by the atom $A_\varphi^{ct}(\bar{t})$ and its definition, thus

obtaining a regular FO(ID) theory, for which computing models is NP-complete. However, since $A^{ct}_{\varphi}(\bar{t})$ is *less* true (according to the truth order $\leq$) than $K_{T_1}(\varphi[\bar{t}])$, the effect (and soundness) of such a replacement will depend on whether the modal literal occurs positively or negatively in T_2. We will illustrate this by means of the following example.

Example 7.22. First, let us look at the OEL theory $\mathcal{T} = \{T_1, T_2\}$ such that $T_1 = \{P \vee \neg P \Rightarrow R.\}$ and $T_2 = \{K_{T_1}(R) \Rightarrow Q.\}$. For this theory, we have that $Mod(\mathcal{T}) = \{\{R, Q\}, \{P, R, Q\}\}$, that is, R and Q are true in all models. Now let $\mathcal{T}^{app}$ be the FO(ID) theory

$$\mathcal{T}^{app} = \begin{array}{l} (P \vee \neg P) \Rightarrow R. \\ \left\{ \begin{array}{l} R^{ct} \leftarrow P^{ct} \vee P^{cf}. \\ P^{cf} \leftarrow R^{cf}. \\ \ldots \end{array} \right\} , \\ R^{ct} \Rightarrow Q. \end{array}$$

that is, $\mathcal{T}^{app}$ is the union of theories T_1 and T_2, in which we replaced the modal literal $K_{T_1}(R)$ by R^{ct}, and added the definition $Approx^{ct}(T_1)$, that defines, amongst others, the predicate R^{ct}.

Even though $T_1 \models R$, we see that $Approx^{ct}(T_1) \not\models R^{ct}$[5]. In this case, the modal literal $K_{T_1}(R)$ (and correspondingly R^{ct}) occurs only negatively in T_2, therefore, we have that $Mod(\mathcal{T}^{app}) = \{\{R\}, \{R, Q\}, \{P, R, Q\}, \{P, R\}\}$[6], that is, $Mod(\mathcal{T}) \subset Mod(\mathcal{T}^{app})$. As a consequence, we cannot use $\mathcal{T}^{app}$ to soundly approximate the *model generation* task for $\mathcal{T}$, indeed, a model M of $\mathcal{T}^{app}$ might not be a model of $\mathcal{T}$. However, $\mathcal{T}^{app}$ does give us a sound approximation of the *query answering* task, that is, $\mathcal{T}^{app} \models \varphi[\bar{d}] \Rightarrow \mathcal{T} \models \varphi[\bar{d}]$.

Let us now change the theory $\mathcal{T}$ a bit, such that $T_2 = \{\neg K_{T_1}(R) \Rightarrow Q.\}$, and denote this new theory by $\mathcal{T}'$. Let $\mathcal{T}'^{app}$ be the same as $\mathcal{T}^{app}$ above, but with the formula $R^{ct} \Rightarrow Q$ replaced by $\neg R^{ct} \Rightarrow Q$. In this case, the modal literal $K_{T_1}(R)$ occurs only positively in T_2. Therefore, we get the opposite effect as before, that is, $Mod(\mathcal{T}'^{app}) = \{\{R, Q\}, \{R, P, Q\}\} \subseteq Mod(\mathcal{T}')$. As a consequence we can use $\mathcal{T}'^{app}$ to soundly approximate model generation for $\mathcal{T}$, but query answering might not be sound.

Proposition 7.23. *Let $\mathcal{T} = \{T_1, T_2\}$ be an OEL theory such that $T_1 < T_2$ and such that inductive definitions do not contain modal literals. Define $\mathcal{T}^{app}$ as the FO(ID) theory such that*

$$\mathcal{T}^{app} = T_1 \cup Approx^{ct}(T_1) \cup T_2[K_{T_1}(\varphi[\bar{d}])/\varphi^{ct}[\bar{d}]],$$

where $T_2[K_{T_1}(\varphi[\bar{d}])/\varphi^{ct}[\bar{d}]]$ denotes the theory T_2 in which every modal literal has been replaced by the ct/cf version of the body φ.

[5] This is an illustration of the incompleteness of the propagation method. To find out that R^{ct} holds, one has to reason by cases, which is something that our propagation method does not do.

[6] Recall that for an FO(ID) theory T, $Mod(T)$ simply denotes the set of all models of T.

If modal literals occur only negatively *in T_2, then $\mathcal{T}^{app}$ gives a sound approximation for the query answering problem with respect to $\mathcal{T}$.*

If model literals occur only positively *in T_1, then $\mathcal{T}^{app}$ gives a sound approximation for the model generation problem with respect to $\mathcal{T}$.*

Proof. This follows immediately from the soundness results in Theorem 3.13. □

For d-OEL we can formulate a completely analogous proposition. When modal literals occur both positively and negatively in T_2 it becomes hard to guarantee completeness, and in general $Mod(\mathcal{T})$ is not necessary related to $Mod(\mathcal{T}^{app})$ in an obvious way. Consider, for example the OEL theory $\mathcal{T} = \{T_1, T_2\}$, where $T_1 = \{(P \vee \neg P) \Rightarrow R.\}$ and $T_2 = \{K_{T_1}(R) \Rightarrow Q.\ \neg K_{T_1}(R) \Rightarrow S.\ P.\}$. It is easy to verify that $Mod(\mathcal{T}) = \{\{P, R, Q\}, \{P, R, Q, S\}\}$, while $Mod(\mathcal{T}^{app}) = \{\{P, R, S, Q\}, \{P, R, S\}\}$.

The requirement that modal literals occur only positively or negatively in theory T_2 might seem overly restrictive, reducing the value of the above property. However, we feel that many useful theories will be of this form, e.g., note that all theories in Example 7.21 are of this form, or can easily be rewritten into an equivalent theory of this form, for example, for theory T_s' we can take the equivalent theory T_s with literals $Avail(r, d)$ replaced by $K_{T_{ref}}(Avail(r, d))$, instead of adding the formula $\forall r \forall d(Avail(r, d) \Leftrightarrow K_{T_{ref}}(Avail(r, d)))$.

In general, even when replacing modal literals in an OEL or d-OEL by their approximated values is sound, it is of course not complete. However, when the theory and modal literals satisfy certain conditions, we do obtain an equivalent theory.

Proposition 7.24. *Let $\mathcal{T} = \{DB, T_2\}$ be an OEL theory such that DB is a database theory, that is, containing only ground literals, and T_2 contains only modal literals of the form $K_{DB}(\varphi[\bar{d}])$, where $\varphi[\bar{d}]$ is in the $\mathcal{NF}$ normal form (according to (Liu & Levesque, 1998)). Then the FO(ID) theory $\mathcal{T}^{app}$ is equivalent with $\mathcal{T}$, in the sense that for every interpretation M, it holds that $M \vDash \mathcal{T}^{app}$ iff $M \in Mod(\mathcal{T})$.*

Finally, let us see what happens if we use the propagation method to approximate modal literals in OEL or d-OEL theories that contain more than just two sub-theories. As we did above, we would like to replace modal literals in each theory by their ct/cf approximation. We define here how this can be done for OEL theories. For d-OEL theories, the idea is similar.

Definition 7.25. Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be an OEL theory. We define the theory $\mathcal{T}^{app}$ as $\cup_{i \in [0..n]} T_i^*$, where for each i, T_i^* is a shorthand for

$$T_i^* = T_i[K/ct] \cup Approx^{ct}_{\{Aux(T_1) \cup \ldots \cup Aux(T_{i-1})\}}(\cup_{j \leq i} T_j[K/ct]),$$

and $T_i[K/ct]$ for $T_i[K_{T_j}(\varphi[\bar{d}])/\varphi^{ct}_{T_j}[\bar{d}]]$.

This definition is a straightforward extension of the case where there were only two theories. One additional complexity here is that since we are putting several approximative definitions in one theory here, one has to make sure not to re-use the auxiliary predicates. Therefore, we tag every auxiliary predicate with the theory determining its value. Indeed, while an atom $P(\bar{d})$ might not be true in all models of T_0, it might be true in all models of T_3.

Unfortunately, it becomes even more difficult to give soundness results for this more general setting, since now soundness might not only depend on whether modal literals appear positively or negatively, but also on which sub-theories these modal literals refer to and how theories with only positive occurrences are alternated with theories with only negative occurrences of modal literals. One special case for which we can give a soundness result, is when all sub-theories only contain negative modal literals, that is, all modal literals occur in the scope of an uneven number of negations. One can easily verify that in this case, the set of models of such a theory is a subset of the set of models of the approximated theory.

Proposition 7.26. *Given an OEL theory $\mathcal{T} = \{T_1, \ldots, T_n\}$, such that all modal literals occur in the scope of an uneven number of negations. Then $Mod(\mathcal{T}) \subseteq Mod(\mathcal{T}^{app})$.*

In summary, in this section, we started by showing that model checking, model generation and query answering for both d-OEL and OEL are in Δ_2^P. We explained how a prototype model generator can be implemented on top of an FO model generator, and illustrated, by means of an example, that a class of interesting constraint problems (in particular, model generation problems that need to query a possibly incomplete knowledge base), can be formulated as model generation problems for d-OEL or OEL. We also explored how the propagation method from Chapter 3 can be used to approximate model generation and query answering. While in general it proved difficult to provide soundness results, we were still able to provide a number of practically useful classes of theories for which we have soundness results.

7.3 Related work

OEL was first proposed by Konolige (1988a) out of his observation that in non-monotonic reasoning one often wants to do inference from a specific body of knowledge, and that neither default logic nor AEL preserve this stratified structure. It therefore comes at no surprise that, while OEL is clearly different from these approaches in that it does preserve this structure, it also is strongly related to them. In Section 7.1.1 we already explored the relationship between OEL and AEL. In this section we also have a closer look at the relationship between OEL and ASP and default logic. We also compare OEL to other hierarchical approaches, such as stratified AEL and stratified belief bases. We conclude by comparing different formalisms for handling distributed knowledge to d-OEL.

7.3.1 Answer Set Programming

This section takes a closer look at the relation between OEL and Answer Set Programming, more specifically its core language called Extended Logic Programming (Gelfond & Lifschitz, 1991). We first briefly recall the basics. An answer set program P over vocabulary Σ is a collection of rules:

$$\mathtt{l}'_0\mathtt{v}\ldots\mathtt{v}\mathtt{l}'_k \leftarrow \mathtt{l}_1,\ldots,\mathtt{l}_m,\mathtt{not}\ \mathtt{l}_{m+1},\ldots,\mathtt{not}\ \mathtt{l}_n.$$

where each $\mathtt{l}'_j, \mathtt{l}_i$ is an atom $\mathtt{P}(\mathtt{t}_1,\ldots,\mathtt{t}_k)$ or a strong negation literal $\neg\mathtt{P}(\mathtt{t}_1,\ldots,\mathtt{t}_k)$. A program with variables represents the set of all ground rules obtained by substituting ground terms of Σ for all variables. We call an answer set program *disjunctive* if $k > 1$ for at least one rule.

An answer set A of P is a $\subseteq$-minimal element in the collection of all sets S of ground atoms and strong negation literals, such that, for each rule

$$\mathtt{l}'_1\ \mathtt{v}\ldots\mathtt{v}\ \mathtt{l}'_k \leftarrow \mathtt{l}_1,\ldots,\mathtt{l}_m,\mathtt{not}\ \mathtt{l}_{m+1},\ldots,\mathtt{not}\ \mathtt{l}_n.,$$

if $l_1,\ldots,l_m \in S$ and $l_{m+1},\ldots l_n \notin S$, then at least one $l'_i \in S$. Intuitively, an answer set program represents the knowledge of an introspective epistemic agent: a rule of the above form states that the agent must believe one of the $\mathtt{l}'_k$ if he also believes $\mathtt{l}_1,\ldots,\mathtt{l}_m$ and does not know any of the literals $\mathtt{l}_{m+1},\ldots,\mathtt{l}_n$. Moreover, the agent does not believe anything he is not forced to believe. Each answer set represents a possible state of belief of the agent.

Let us now return to Example 7.15. The incomplete database DB' can be represented in ASP as follows:

$$DB' = \left\{ \begin{array}{ll} \mathtt{FairGPA(Ann).} & \mathtt{Minority(Ann).} \\ \mathtt{HighGPA(John).} & \mathtt{FairGPA(Bob).} \end{array} \right\}$$

The partial completeness of this database can be expressed by two CWA rules:

$$CWA = \left\{ \begin{array}{l} \neg\mathtt{FairGPA(x)} \leftarrow \mathtt{not\ FairGPA(x).} \\ \neg\mathtt{HighGPA(x)} \leftarrow \mathtt{not\ HighGPA(x).} \end{array} \right\}$$

The next rule expresses who is to be interviewed:

$$IV = \{\mathtt{Interview(x)} \leftarrow \mathtt{FairGPA(x)}, \mathtt{not\ Minority(x)},$$
$$\mathtt{not}\ \neg\mathtt{Minority(x).}\}$$

The program $DB' \cup CWA \cup IV$ has one answer set, in which only *Bob* is to be interviewed. Both CWA and IV exploit the epistemic nature of negation-as-failure: the former to assert that unknown facts are false, and the latter to check whether the minority status of x is unknown. This solution is not entirely equivalent to ours in Example 7.4. Indeed, this answer

set does not contain literals ¬Interview(...). Hence, whereas in our solution, *Bob* and *John* are known not to be interviewed, in the belief state corresponding to this answer set, this is unknown. To derive these conclusions, rules need to be added that correspond to the only-if part of the interview rule:

$$\neg\texttt{Interview(x)} \leftarrow \neg\texttt{FairGPA(x)}.$$

$$\neg\texttt{Interview(x)} \leftarrow \texttt{Minority(x)}.$$

$$\neg\texttt{Interview(x)} \leftarrow \neg\texttt{Minority(x)}.$$

While ASP was originally introduced as a logic for the (strongly related) forms of default and autoepistemic reasoning, already in 1991 Gelfond showed that the capability of expressing autoepistemic reasoning in ASP is very limited. In fact, he observed that in ASP one cannot perform autoepistemic reasoning on knowledge bases with incomplete disjunctive information. Consider again the database DB' above. Here in this case, the rule of interview correctly works. However, now consider a database with the disjunctive information that either Ann or Bob belongs to a minority: This might be by adding a disjunctive rule:

$$DB_{\vee} = \left\{ \begin{array}{c} \texttt{FairGPA(Ann)}.\,\texttt{FairGPA(Bob)}. \\ \texttt{Minority(Ann)}\ \texttt{v}\ \texttt{Minority(Bob)}. \end{array} \right\},$$

or it might be done also in another fashion, not involving disjunction

$$DB'_{\vee} = \left\{ \begin{array}{c} \texttt{FairGPA(Ann)}.\,\texttt{FairGPA(Bob)}. \\ \texttt{Minority(Ann)} \leftarrow \texttt{not Minority(Bob)}. \\ \texttt{Minority(Bob)} \leftarrow \texttt{not Minority(Ann)}. \end{array} \right\}$$

In both cases, we would like to infer again that both Ann and Bob must be interviewed, but $DB_{\vee} \cup IV$ (and also $DB'_{\vee} \cup IV$) will give two answer sets: one with Interview(Ann) and one with Interview(Bob). If we add the only-if rules, the first answer set will contain ¬Interview(Bob) and the second ¬Interview(Ann).

To solve this problem, Gelfond proposed a solution to this problem. He proposed to add to the language a second modal operator K, the *strong introspection operator*, which should be used to reason about predicates with disjunctive information. It can correctly express the definition of Interview as:

$$\begin{array}{l} \texttt{Interview(x)} \leftarrow \texttt{FairGPA(x)}, \\ \quad \texttt{not K not Minority(x)}, \texttt{not K Minority(x)}. \end{array}$$

This rule gives the right result for both $DB_{\vee}$, $DB'_{\vee}$ and DB'. However, we have the following problems with this operator. First, it raises the question of what negation-as-failure means in the context of disjunctive databases. Apparently it is some introspection operator but not the strong introspection operator. The distinction is not clear. Second, it makes ASP *less*

elaboration tolerant. Suppose we start from a theory without disjunctive information, where we can use NAF as an autoepistemic operator. Adding one piece of disjunctive information forces us to rewrite the operator. In OEL this problem is completely solved. In OEL, the disjunctive database can be represented by:

$$DB_{\vee}^{OEL} = \left\{ \begin{array}{l} FairGPA(Ann).\ FairGPA(Bob). \\ Minority(Ann) \vee Minority(Bob). \end{array} \right\}$$

This $DB_{\vee}^{OEL}$ can be queried by the same theory T_I as in Example 7.4 and this will result in the correct conclusion that both Ann and Bob have to be interviewed.

Unlike OEL, theories in ASP^K may have more than one "world view" (~ belief system), and as recently shown by Gelfond (2011), some of these world views are unintended. While that paper also suggests a change to the semantics of ASP^K, it is currently not known whether this solves all issues.

7.3.2 Default Logic

We already showed that OEL is able to represent a certain kind of default information, namely the Closed World Assumption. Other defaults are of course also possible. For instance, the following theory $\mathcal{T} = T_{bird} < T_{def}$ models one of the standard examples:

$$T_{bird} = \left\{ \begin{array}{c} Bird\,(Tweety\,) \wedge Bird\,(Clyde\,). \\ Penguin\,(Clyde\,). \\ \forall x(Penguin\,(x) \Rightarrow \neg Flies\,(x)). \end{array} \right\}$$

$$T_{def} = \left\{ \begin{array}{l} \forall x(K_{T_{bird}}(Bird\,(x)) \wedge \neg K_{T_{bird}}(\neg Flies\,(x)) \\ \qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow Flies\,(x)). \end{array} \right\}$$

Note that the theory $\mathcal{T}$ satisfies the conditions of Proposition 7.7, and therefore this OEL theory is equivalent to $AEL(\mathcal{T})$. Moreover, the stable extension and Moore's expansion semantics coincide on $AEL(\mathcal{T})$. Therefore, the theory $\mathcal{T}$ is a correct and equivalent formalization of the standard default theory, expressing the above default theory.

Although it seems that in the theoretical default reasoning literature, most attention has always gone to difficult theories that have multiple extensions, and for which expansion and extension semantics do not coincide, to us it seems the case that in a considerable class of practical examples, defaults are of a simpler, non-problematic form: defaults are hierarchical, defaults are designed to not contradict each other so that expansion and extension semantics coincide and only one expansion/extension exists. It is this observation in the first place

that led Konolige to propose OEL, and that led Denecker et al. to reinvent this logic. It is also this observation that led Gelfond and Lifschitz a long time ago to propose ASP as a simpler but practically useful logic for representing defaults and autoepistemic reasoning. Thus, we believe that in practice OEL is a very suitable logic for expressing defaults in many if not in most applications.

It is however interesting to look at what happens in cases of default theories with multiple default extensions. For instance, let us apply the above method to model Nixon's diamond:

$$T_{base} = \left\{ \begin{array}{c} Republican\,(Nixon\,) \wedge Quaker\,(Nixon\,) \\ \\ \forall x(\neg Dove\,(x) \vee \neg Hawk\,(x)) \end{array} \right\}$$

$$T_{def} = \left\{ \begin{array}{c} \forall x(K_{T_{base}}(Republican\,(x)) \wedge \neg K_{T_{base}}(\neg Hawk\,(x)) \\ \\ \Rightarrow Hawk\,(x)) \\ \\ \forall x(K_{T_{base}}(Quaker\,(x)) \wedge \neg K_{T_{base}}(\neg Dove\,(x)) \\ \\ \Rightarrow Dove\,(x)) \end{array} \right\}$$

This OEL theory entails that Nixon is dove and hawk and hence it is inconsistent. In contrast, the solution in default logic has two extensions. In some cases the latter may be the desired behavior. However, as argued in (Denecker et al., 2011), multiple extensions of a default theory are caused by conflicting defaults that offer insufficient guidance how to solve the conflict, for example, what happens for republican quakers? Such conflicts may arise inadvertently for the person writing down the theory in which case it is often better to resolve these conflicts by refining the condition of one or both constraints, for example, by expressing that in case of republican quakers, being a dove is the default. In this case the inconsistency reported by the OEL representation is a useful sign that user intervention is required.

When multiple extension are really desired, it is here too the case, as for AEL, that OEL's hierarchical structure is unsuitable. For such cases, a solution might be sought in the

direction of a non-hierarchical (or, alternatively, a self-referential) theory:

$$T_{base} = \left\{ \begin{array}{l} Republican(Nixon) \wedge Quaker(Nixon) \\ \\ \quad \forall x(\neg Dove(x) \vee \neg Hawk(x)) \end{array} \right\}$$

$$T_{hawk} = \left\{ \begin{array}{l} \forall x(Republican(x) \\ \\ \quad \wedge \neg K_{T_{dove}}(\neg Hawk(x)) \\ \\ \quad \Rightarrow Hawk(x)) \end{array} \right\}$$

$$T_{dove} = \left\{ \begin{array}{l} \forall x(Quaker(x) \wedge \neg K_{T_{hawk}}(\neg Dove(x)) \\ \\ \qquad\qquad\qquad\qquad \Rightarrow Dove(x)) \end{array} \right\}$$

Another interesting approach and topic for future work is to define an equilibrium semantics for OEL that would allow cyclic dependencies of this kind (and produce the same two extensions as Default Logic does for this example).

7.3.3 Other Hierarchical Approaches

Deciding whether an autoepistemic theory has expansions is Σ_2^P-complete in general. However, for certain subclasses, reasoning can be done more efficiently. A popular family of such subclasses are *stratified* theories, i.e., theories that can be divided into a number of levels, such that the expansions of the entire theory can be computed by combining the expansions of each level. This approach was widely studied in both AEL and Default Logic (Gelfond, 1987; Marek & Truszczyński, 1991; Cholewinski, 1994; Vennekens et al., 2006). In many of these works, stratified theories are guaranteed to have a unique expansion. Though this might seem similar to our approach here, there are some important differences. In these stratified approaches the idea is to give *syntactical constraints*, such that theories that satisfy these constraints can be *automatically* split up in smaller theories that do not circularly depend on each other. In OEL on the other hand, the goal is not to automatically split up a theory, but it is the *person writing down the theory* himself who is responsible for dividing the theory into suitable sub-theories. Because of this, our theories do not need to satisfy restrictive syntactic conditions which allows us to do more than is possible in these stratified approaches, for example, theories that contain *closed world assumptions* will typically not be stratified while, as we have illustrated with some examples, closed world assumptions can be expressed in OEL.

Other somewhat related work is that of *stratified belief bases*[7] (Brewka, 1989), which actually is an approach to inconsistency handling. Information is represented by stratified belief bases, i.e., finite sets of (possibly inconsistent) formulas equipped with a total pre-order which represents the available preferences over the given beliefs. The idea is then to select a preferred consistent subbase (where theories lower in the order have preference over theories higher in the order) of the given stratified belief base. While this framework also uses hierarchical knowledge bases, its purpose is clearly very different. First of all, we cannot see a stratified belief base as an OEL theory as such, since clearly an inconsistent OEL theory has no models. However, we can simulate inference for *certain* stratified belief bases with OEL (and similarly with d-OEL), that is, for belief bases where the conflicts arise *between* sub-theories, and not within. This would work as follows. Given a stratified belief base $\{\Delta_1, \ldots, \Delta_n\}$, we construct a OEL theory $\mathcal{T} = \{T_1, \ldots, T_n\}$ as follows. Take $T_1 = \Delta_1$, and for each formula $\varphi \in \Delta_i$ with $i > 1$, we add the formula $\neg K_{T_{i-1}}(\neg\varphi) \Rightarrow \varphi$ to T_i. A formula φ is now *strongly provable* (Brewka, 1989) in the stratified belief base if it holds that $M \vDash \varphi$ for *all* models M of $\mathcal{T}$ and *weakly provable* if $M \vDash \varphi$ for some model M of $\mathcal{T}$.

7.3.4 More Related Work

This section considers some knowledge representation formalisms that are related to d-OEL, that is, formalisms for handling distributed (possibly contradictory) knowledge.

Polleres et al. (2006) attempt to incorporate *contextual queries* and *non-monotonic* reasoning techniques into current Semantic Web technologies, in particular, extensions of RDF or OWL with rule languages. Because of the open-world setting in the Semantic Web, only a limited form of negation-as-failure is possible, in which negation has an explicitly defined, finite scope. As the authors claim, clear definitions and formal semantics for such scoped negation are missing, and thus they propose a *logic programming* based framework for the combination of *rule bases* on the web. One of the key features in this work is that every fact or rule is associated to a *context* (the URI of the rule base). Literals can then be tagged with these URIs to form *scoped literals*. This allows queries such as "*Which movies are rated "Bad" by http://moviereviews.com?*" to be represented as:

$$\texttt{Answer(x)} \leftarrow \texttt{Movie(X)},\ \texttt{Rated(X, Bad)@“http://moviereviews.com”}.$$

Negation is only allowed to appear in these scoped literals.

Each scoped literal corresponds naturally to a modal literal $K_{<uri>}P(\bar{t})$ or $K_{<uri>}\neg P(\bar{t})$ of d-OEL. As shown in (Denecker et al., 2010b), the principle for extending FO to OEL can be easily applied also to FO(ID), the extension of FO with a rule-based representation for inductive definitions (Denecker & Ternovska, 2008). Applying this to d-OEL leads to a

[7] Sometimes also referred to as prioritized knowledge/belief bases.

logic that generalizes the scoped literal approach by allowing arbitrary formulas both in the bodies of rules and in the scope of the modal operators. In this way, d-OEL can also handle queries for, e.g., disjunctive knowledge.

Scoped negation was also introduced to FLORA-2 (Kifer, 2005), which is a query answering system that integrates F-Logic with a number of other formalisms. In FLORA-2, a module is a container or a concrete knowledge base. Negative queries can then be posed to a certain module. Another formalism that has a form of scoped negation is N3 (Berners-Lee et al., 2005), which is a language for representing RDF rules on the Semantic Web.

The need for a semantically well-defined construct to query other sources of knowledge is not limited to the Semantic Web context, but also exists in other knowledge representation domains. For example, Barker (2009) defines a meta-model for access control policies, using a logic programming based rule language. Literals occurring in the body of rules may be defined in the local program, but also in a remotely located program identified by v. Such a literal is then denoted by "$L \mathrel{\leftarrow\!\!\!\!\!\!\rho} v$". Typically, this construct is used for querying the knowledge of another access control manager. The semantics of the rule language is based on Clark's completion (Clark, 1978), where, for remotely defined literals $L \mathrel{\leftarrow\!\!\!\!\!\!\rho} v$, the union of v and the local program is considered. Again, OEL generalizes this language.

Our work is also related to attempts to combine Logic Programs with Descriptions Logics. A representative approach is the hybrid combination proposed by Pührer et al. (2010). Here, an ASP program is allowed to query an OWL knowledge base to find out what it does and does not know. Like OEL, this is also a layered approach, with the obvious difference that DL-programs allow only two components, where the top component must be written in ASP and the bottom component in OWL. An interesting feature of DL-programs is that they allow additional atoms to be temporarily added to the OWL knowledge base for the evaluation of a query. In OEL, this could be simulated by exploiting the ability to have more than two components: a single minimal theory would then consist of the OWL knowledge base; in the next layer, we would find a number of theories that each add different additional facts to the knowledge base; finally, a single maximal theory would then correspond to the ASP program, posing queries to the different theories in the middle layer.

In addition to such hybrid integrations, there also exist a number of formalisms that offer a full integration of Logic Programming and Description Logics. These typically abandon the idea of a layered structure altogether and resort to languages that are very similar to AEL. In particular, de Bruijn et al. (2007) present a formalism that is embedded into first-order AEL, while Motik & Rosati (2010) uses an embedding into the logic MKNF, which has been shown to be equivalent to AEL (Rosati, 1997). Our approach can be seen as claiming a middle ground between these approaches and the hybrid ones: we do retain a layered structure, but use a single language in the different components.

7.4 Conclusions

In this chapter we further developed *ordered epistemic logic*, originally introduced by Konolige (1988a) and recently independently by Denecker et al. (2010b) to handle a range of interesting epistemic examples, without the high complexity of autoepistemic logic. We proved this claim by showing that model generation for OEL is in Δ_2^P. Given that this is still a relatively high complexity, we also investigated how and when the approximation techniques for FO from Chapter 3 can be used to approximately solve OEL and d-OEL queries in polynomial time. We gave a number of examples that show that the closed world assumption, defaults, … can indeed be naturally represented in OEL. While OEL does not allow the self-references of AEL, we proved that in some cases, an OEL theory can be seen as an AEL theory. Next, we defined a generalization of OEL, which we called *distributed ordered epistemic logic*, in which a higher level theory in the hierarchy no longer includes the knowledge of lower level theories. This logic is well-suited to model distributed (and potentially inconsistent) knowledge of agents. Applications for such a logic exists in the Semantic Web, where ontologies are distributed over the web, and one ontology might need the knowledge of another. Several approaches have been proposed to do this, and we argued that d-OEL is a natural generalization of several of them.

A limitation of our formalism is that it assumes the same fixed domain for all theories. Allowing different theories to have different *finite* domains is not problematic, since this can already be simulated in our current logic. Indeed, if we want each theory $T \in \mathcal{T}$ to have its own specific domain D_T, we can simulate this by taking the union $\cup_{T \in \mathcal{T}} D_T$ as a fixed domain throughout $\mathcal{T}$ and defining in each theory T a predicate $Domain_T$ that explicitly enumerates all the elements of D_T: $\forall x(Domain_T(x) \Leftrightarrow \bigvee_{c \in D_T} x = c)$. Allowing each theory to have its own infinite domain on the other hand, would be a real extension of the logic, and it is not clear which of the results presented here would carry over. Another interesting topic for future work would be to implement a model generator for OEL and d-OEL, along the lines sketched earlier in this chapter.

8

Conclusions

The central topic of this dissertation has been to investigate the feasibility of first-order logic and its extensions such as FO(·) as a knowledge representation language. Traditionally, people have doubted the usefulness of FO for declarative problem solving. Indeed, whole areas such as description logics, logic programming and non-monotonic reasoning even grew out of this belief. However, recently there have been a number of trends that make us feel this old belief should be reconsidered. First of all, we see that researchers have looked further than deduction, and that other forms of inference, such as model checking, model generation, query answering,etc. have been proven to be very useful in practice. Second, while, for example, the description logics area originated from the idea that FO should be limited, we now see that slowly they are adding more and more expressivity back to the language.

We believe that first-order logic is a good knowledge representation language, and, more importantly a useful one. Of course, this claim could never be proven by one single person, but we hope that this dissertation adds further body to this claim. In this text, we shown in a number of different settings that it is possible to have feasible forms of inference while still having an expressive FO-based logic, in which the relevant domain knowledge can be naturally represented. While the methods in this thesis have not yet been tested on large scale problems, we do believe that our results are promising and encouraging.

Even though there is the trade-off between the expressiveness of a knowledge representation language and the efficiency of inference methods for this language, it seems that there are ways out of this apparent deadlock, for example, by looking at cheaper forms of inference. In

Chapter 3, we therefore looked at propagation. As we saw throughout this text, propagation often forms the basis for other inference methods. We studied this propagation method for first order logic, and extended it to the extension of first order logic with inductive definitions (FO(ID)). While a number of implementations of this propagation method exist (e.g., the one used in the grounder of the IDP system (idp), or the implementation we used for our approximation method for $\exists\forall SO(ID)$ in Chapter 5), it might be useful to have an optimized, robust, stand-alone implementation, given the importance of this propagation method for a variety of different applications.

In Chapter 4 we did a first experiment towards a *Knowledge Base System* based on FO(·), through a case study of configuration software. We catalogued a number of key reasoning tasks that arise naturally within the context of configuration software, and investigated how they can be handled efficiently. While the results in this chapter are very promising, they are certainly not an end point, and numerous future work topics remain open. Some of these are of a theoretical nature, for example, studying new language constructs or new forms of inference, while others are more of a practical nature, such as optimizing implementations of inference algorithms and providing practical user interfaces for reasoners, and trying out the approach on a larger scale.

When inference tasks become too hard to be of practical use, a common solution is to approximately solve that task. In Chapter 5, we proposed a framework to approximately solve finite domain $\exists\forall SO(ID)$ satisfiability problems. Solving such problems exactly is hard, so we proposed a sound but incomplete method that is more efficient. An interesting feature of our approach is that it can serve as a general framework for a number of approximative approaches in the literature, and can also be used for solving useful practical problems.

Another domain where reasoning is typically hard, and where one can find approximate reasoning methods, is that of reasoning about and representing sensing actions and the knowledge of an agent. Our approach here was to first extend FO(·) even further so that we can intuitively represent how the knowledge of an agent changes through sensing actions. This leads to a very general representation language, for which solving most inference tasks exactly will be too hard to be of practical use. Our solution is then to show how the propagation method for FO(ID) can be used to define an approximative inference method for solving the projection problem, which again nicely generalizes a number of methods in the literature. It might be an interesting topic for future work to build a conditional planner on top of this approximative inference method.

In the final chapter of this dissertation we looked at Ordered Epistemic Logic. Instead of further extending an existing language, here we did the opposite. We observed that many existing epistemic language do not maintain the stratified structure that is inherent in many practical examples, which complicates their semantics and reasoning procedures considerably. By defining a logic that keeps this inherent structure explicit, we have the benefit of still having a very expressive logic, in which most of these examples can be

naturally represented, while inference tasks have a lower complexity. Again, it would be really interesting to investigate the practical usefulness of OEL and d-OEL in more depth. Here, especially the distributed variant d-OEL seems to have a variety of promising applications.



Example 3.9

This appendix gives the complete definition $Approx_{\{Wipe\}}(T)$ from Example 3.9, on the next page.

$$
\begin{cases}
A^{ct}_{\psi_{act}} \leftarrow A^{ct}_{0} \wedge A^{ct}_{8}. \\
A^{cf}_{\psi_{act}} \leftarrow A^{cf}_{0}. \\
A^{cf}_{\psi_{act}} \leftarrow A^{cf}_{8}. \\
A^{ct}_{0} \leftarrow A^{ct}_{\psi_{act}}. \\
A^{ct}_{8} \leftarrow A^{ct}_{\psi_{act}}. \\
A^{cf}_{0} \leftarrow A^{cf}_{\psi_{act}} \wedge A^{ct}_{8}. \\
A^{cf}_{8} \leftarrow A^{cf}_{\psi_{act}} \wedge A^{ct}_{0}. \\
\\
A^{ct}_{0} \leftarrow (\forall t : A^{ct}_{1}(t)). \\
A^{cf}_{0} \leftarrow (\exists t : A^{cf}_{1}(t)). \\
A^{ct}_{1}(t) \leftarrow A^{ct}_{0}. \\
A^{cf}_{1}(t) \leftarrow (A^{cf}_{0} \wedge (\forall t_1 : (t_1 = t \vee A^{ct}_{1}(t_1)))). \\
A^{cf}_{1}(t) \leftarrow A^{cf}_{2}(t). \\
A^{cf}_{1}(t) \leftarrow A^{cf}_{5}(t). \\
A^{ct}_{1}(t) \leftarrow (A^{ct}_{2}(t) \wedge A^{ct}_{5}(t)). \\
A^{ct}_{2}(t) \leftarrow A^{ct}_{1}(t). \\
A^{cf}_{2}(t) \leftarrow (A^{cf}_{1}(t) \wedge A^{ct}_{5}(t)). \\
A^{ct}_{5}(t) \leftarrow A^{ct}_{1}(t). \\
A^{cf}_{5}(t) \leftarrow (A^{cf}_{1}(t) \wedge A^{ct}_{2}(t)). \\
A^{ct}_{2}(t) \leftarrow Clean^{\,cf}((t+1)). \\
A^{ct}_{2}(t) \leftarrow A^{ct}_{4}(t). \\
A^{cf}_{2}(t) \leftarrow (Clean^{\,ct}(t+1)) \wedge A^{cf}_{4}(t)). \\
Clean^{\,ct}(t+1) \leftarrow A^{cf}_{2}(t). \\
Clean^{\,cf}(t+1) \leftarrow (A^{ct}_{2}(t) \wedge A^{cf}_{4}(t)). \\
A^{cf}_{4}(t) \leftarrow A^{cf}_{2}(t). \\
A^{ct}_{4}(t) \leftarrow (A^{ct}_{2}(t) \wedge Clean^{\,ct}(t+1)). \\
A^{ct}_{4}(t) \leftarrow Clean^{\,ct}(t). \\
A^{ct}_{4}(t) \leftarrow Wipe(t). \\
A^{cf}_{4}(t) \leftarrow (Clean^{\,cf}(t) \wedge \neg Wipe(t)). \\
Clean^{\,cf}(t) \leftarrow A^{cf}_{4}(t). \\
Clean^{\,ct}(t) \leftarrow (A^{ct}_{4}(t) \wedge \neg Wipe(t)). \\
A^{ct}_{5}(t) \leftarrow A^{ct}_{6}(t). \\
A^{ct}_{5}(t) \leftarrow Clean^{\,ct}(t+1). \\
A^{cf}_{5}(t) \leftarrow (A^{cf}_{6}(t) \wedge Clean^{\,cf}(t+1)). \\
A^{cf}_{6}(t) \leftarrow A^{cf}_{5}(t). \\
A^{ct}_{6}(t) \leftarrow (A^{ct}_{5}(t) \wedge Clean^{\,cf}(t+1)). \\
Clean^{\,cf}((t+1)) \leftarrow A^{cf}_{5}(t). \\
Clean^{\,ct}((t+1)) \leftarrow (A^{ct}_{5}(t) \wedge A^{cf}_{6}(t)). \\
A^{cf}_{6}(t) \leftarrow Clean^{\,ct}(t). \\
A^{cf}_{6}(t) \leftarrow Wipe(t). \\
A^{ct}_{6}(t) \leftarrow (Clean^{\,cf}(t) \wedge \neg Wipe(t)). \\
Clean^{\,cf}(t) \leftarrow A^{ct}_{6}(t). \\
Clean^{\,ct}(t) \leftarrow (A^{cf}_{6}(t) \wedge \neg Wipe(t)).
\end{cases}
\qquad
\begin{cases}
A^{cf}_{8} \leftarrow A^{cf}_{9}. \\
A^{cf}_{8} \leftarrow A^{cf}_{11}. \\
A^{ct}_{8} \leftarrow (A^{ct}_{9} \wedge A^{ct}_{11}). \\
A^{ct}_{9} \leftarrow A^{ct}_{8}. \\
A^{cf}_{9} \leftarrow (A^{cf}_{8} \wedge A^{ct}_{11}). \\
A^{ct}_{11} \leftarrow A^{ct}_{8}. \\
A^{cf}_{11} \leftarrow (A^{cf}_{8} \wedge A^{ct}_{9}). \\
A^{ct}_{9} \leftarrow Clean^{\,cf}(0). \\
A^{ct}_{9} \leftarrow InitiallyClean^{\,ct}. \\
A^{cf}_{9} \leftarrow (Clean^{\,ct}(0) \wedge InitiallyClean^{\,cf}). \\
Clean^{\,ct}(0) \leftarrow A^{cf}_{9}. \\
Clean^{\,cf}(0) \leftarrow (A^{ct}_{9} \wedge InitiallyClean^{\,cf}). \\
InitiallyClean^{\,cf} \leftarrow A^{cf}_{9}. \\
InitiallyClean^{\,ct} \leftarrow (A^{ct}_{9} \wedge Clean^{\,ct}(0). \\
A^{ct}_{11} \leftarrow A^{ct}_{12}. \\
A^{ct}_{11} \leftarrow Clean^{\,ct}(0). \\
A^{cf}_{11} \leftarrow (A^{cf}_{12} \wedge Clean^{\,cf}(0)). \\
A^{cf}_{12} \leftarrow A^{cf}_{11}. \\
A^{ct}_{12} \leftarrow (A^{ct}_{11} \wedge Clean^{\,cf}(0)). \\
Clean^{\,cf}(0) \leftarrow A^{cf}_{11}. \\
Clean^{\,ct}(0) \leftarrow (A^{ct}_{11} \wedge A^{cf}_{12}). \\
A^{ct}_{12} \leftarrow InitiallyClean^{\,cf}. \\
A^{cf}_{12} \leftarrow InitiallyClean^{\,ct}. \\
InitiallyClean^{\,cf} \leftarrow A^{ct}_{12}. \\
InitiallyClean^{\,ct} \leftarrow A^{cf}_{12}.
\end{cases}
$$

B

Nested Fixpoint Approximations

In this appendix we give some of the proofs that were left out in Chapter 3 having to do with the nested least and greatest fixpoint approximation of an FO(ID) theory. We start with the proof of Theorem 3.25.

B.1 Proof of Theorem 3.25

In order to prove this theorem we need some extra machinery. First we will show that we can also characterize the models of $FP_\sigma(\Delta)$ by means of an operator $\mathcal{C}^{\tilde{I}}_{FP_\sigma(\Delta)}$, that differs from the usual operator $\Gamma^{\tilde{I}}_{FP_\sigma(\Delta)}$, in the following sense. The first part in applying the operators is the same for both operators, i.e., a $Def(\mathcal{R}^{cf})$-interpretation is computed as the greatest fixpoint of the $T_{\mathcal{R}^{cf}}$ operator. However, for the second part, $\Gamma^{\tilde{I}}_{FP_\sigma(\Delta)}$ only applies the $T_{\mathcal{R}^{ct}}$-operator once, while $\mathcal{C}^{\tilde{I}}_{FP_\sigma(\Delta)}$ computes the least fixpoint of the $T_{\mathcal{R}^{ct}}$-operator.

Definition B.1. Let Δ be an inductive definition and let $\mathcal{D} = FP_\sigma(\Delta)$. Let $\tilde{I}$ be an $Open(\mathcal{D})$ interpretation that encodes a three-valued $Open(\Delta)$ interpretation $\mathcal{I}$. We define an operator $\mathcal{C}^{\tilde{I}}_{\mathcal{D}}$ on the set of $Def(\mathcal{D})$-interpretations as follows. We define $\mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J)$ as the interpretation $K + K'$ where

- K is the $Def(\mathcal{R}^{cf})$-interpretation such that

$$K = \text{gfp}(T^{J'}_{\mathcal{R}^{cf}}), \qquad \text{where } J' = \tilde{I} + J|_{Def(\mathcal{R}^{ct})}.$$

- K' is the $Def(\mathcal{R}^{ct})$-interpretation such that

$$K' = \text{lfp}(T^{J''}_{\mathcal{R}^{ct}}), \qquad \text{where } J'' = \tilde{I} + K.$$

The following lemma now relates this operator to the operator $\Gamma^{\tilde{I}}_{\mathcal{D}}$ we used to define the semantics of nested fixpoint expressions.

Lemma B.2. $\text{lfp}(\mathcal{C}^{\tilde{I}}_{\mathcal{D}}) = \text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}})$.

Proof. We prove this via induction over the construction of $\text{lfp}(\mathcal{C}^{\tilde{I}}_{\mathcal{D}})$ and $\text{lfp}(\Gamma^{\tilde{I}_{\mathcal{D}}})$. Indeed, $\text{lfp}(\Gamma^{\tilde{I}_{\mathcal{D}}})$ resp. $\text{lfp}(\mathcal{C}^{\tilde{I}}_{\mathcal{D}})$ is the limit of the sequence of $Def(\mathcal{D})$-interpretations $(I^i)_{i\geq 0}$, resp. $(J^i)_{i\geq 0}$ where $I^0 = J^0 = \bot$, and $I^{i+1} = \Gamma^{\tilde{I}}_{\mathcal{D}}(I^i)$ and $J^{i+1} = \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J^i)$. We will prove that for every i it holds that $I^i \leq J^i \leq \text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}})$. If we prove this, it follows via induction that this also holds for the limit of this sequence, and thus that $\text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}}) \leq \text{lfp}(\mathcal{C}^{\tilde{I}}_{\mathcal{D}}) \leq \text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}})$, which will prove our lemma. We now need to prove the following:

1. $I^0 \leq J^0 \leq \text{lfp}(\Gamma^{\tilde{I}_{\mathcal{D}}})$,
2. Given that $I^i \leq J^i \leq \text{lfp}(\Gamma^{\tilde{I}_{\mathcal{D}}})$, it follows that $\Gamma^{\tilde{I}}_{\mathcal{D}}(I^i) \leq \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J^i) \leq \text{lfp}(\Gamma^{\tilde{I}_{\mathcal{D}}})$.

The first case is clearly trivial. To prove the second point, we will prove the following slightly more general property:

> Given two $Def(\mathcal{D})$-interpretations I and J. If $I \leq J$, $J \leq \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J)$ and $J \leq \text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}})$, then $\Gamma^{\tilde{I}}_{\mathcal{D}}(I) \leq \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J) \leq \text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}})$.

We begin with assuming $I \leq J$ and $J \leq \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J)$, and with proving the first part of the inequality. When applying $\Gamma^{\tilde{I}}_{\mathcal{D}}$ and $\mathcal{C}^{\tilde{I}}_{\mathcal{D}}$ to I resp. J, the first step for both operators consists in computing a $Def(\mathcal{R}^{cf})$-interpretation $K_I = \text{gfp}(\Gamma^{\tilde{I}+I|_{Def(\mathcal{R}^{ct})}}_{\mathcal{R}^{cf}}) = \text{gfp}(T^{\tilde{I}+I}_{\mathcal{R}^{cf}})$ resp. $K_J = \text{gfp}(T^{\tilde{I}+J}_{\mathcal{R}^{cf}})$. We know for all interpretations X, X', K, K' such that $X \leq X'$, $K \leq K'$ and $K, K' \subseteq Open(\mathcal{D})$, and any set of rules $\mathcal{R}$, that $T^{\tilde{I}+K}_{\mathcal{R}}(X) \leq T^{\tilde{I}+K'}_{\mathcal{R}}(X')$. Thus, since $I \leq J$ we have $K_I \leq K_J$. The second step in computing $\Gamma^{\tilde{I}}_{\mathcal{D}}(I)$ consists then of computing the $Def(\mathcal{R}^{ct})$-interpretation $T^{\tilde{I}+K_I}_{\mathcal{R}^{ct}}(I) = K'_I$. For the $\mathcal{C}^{\tilde{I}}_{\mathcal{D}}$ operator the second step is computing $\text{lfp}(T^{\tilde{I}+K_J}_{\mathcal{R}^{ct}}) = K'_J$. Now

we have

$$\begin{array}{rcll} K'_J & \geq & J|_{Def(R^{ct})} & \text{(this follows from the assumption} \\ & \Downarrow & & \text{that } J \leq \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J)) \\ T^{K_J+\tilde{I}}_{\mathcal{R}^{ct}}(K'_J) = K'_J & \geq & T^{K_J+\tilde{I}}_{\mathcal{R}^{ct}}(J|_{Def(R^{ct})}) & \\ & \Downarrow & & \\ K'_J & \geq & T^{K_I+\tilde{I}}_{\mathcal{R}^{ct}}(J|_{Def(R^{ct})}) & \text{(since we assumed } I \leq J) \\ & \Downarrow & & \\ K'_J & \geq & T^{K_I+\tilde{I}}_{\mathcal{R}^{ct}}(I|_{Def(R^{ct})}) = K'_I & \end{array}$$

Together with $K_I \leq K_J$ it follows that $K_I + K'_I = \Gamma^{\tilde{I}}_{\mathcal{D}}(I) \leq \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J) = K_J + K'_J$.
Now for the second part of the inequality. Let $J' = \text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}})$. Since we assumed that $J \leq \text{lfp}(\Gamma^{\tilde{I}}_{\mathcal{D}})$, we have that $J|_{Def(\mathcal{R}^{ct})} \leq J'|_{Def(\mathcal{R}^{ct})}$ and thus

$$\begin{array}{ccc} \text{gfp}(T^{J|_{Def(\mathcal{R}^{ct})}+\tilde{I}}_{\mathcal{R}^{cf}}) & \leq & \text{gfp}(T^{J'|_{Def(\mathcal{R}^{ct})}+\tilde{I}}_{\mathcal{R}^{cf}}) \\ \| & & \| \\ L & & J'|_{Def(\mathcal{R}^{cf})} \end{array}$$

We also have

$$\begin{array}{ccccc} \text{lfp}(T^{L+\tilde{I}}_{\mathcal{R}^{ct}}) & \leq & \text{lfp}(T^{J'|_{Def(\mathcal{R}^{cf})}+\tilde{I}}_{\mathcal{R}^{ct}}) & \leq & J'|_{Def(\mathcal{R}^{ct})} \\ \| & & & & \\ L' & & & & \end{array}$$

The last inequality follows from the fact that $J'|_{Def(\mathcal{R})^{ct}}$ is a fixpoint of $T^{J'|_{Def(\mathcal{R}^{cf})}+\tilde{I}}_{\mathcal{R}^{ct}}$. Thus we have that $\mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J) = L + L' \leq \text{lfp}(\Gamma^{\mathcal{D}}_{I})$, which proves our property.

Because of how the sequence $(J^i)_{n\geq 0}$ is constructed and the fact that $\mathcal{C}^{\tilde{I}}_{\mathcal{D}}$ is a monotone operator, we have that $J^i \leq \mathcal{C}^{\tilde{I}}_{\mathcal{D}}(J^i)$. Thus the property that we just proved above is applicable and actually proves point 2. and thus concludes this proof. □

We are now ready to prove Theorem 3.25.

Proof of Theorem 3.25. Let $\mathcal{D} = FP_\sigma(\Delta)$. Without loss of generality we assume that σ is empty. Denote by (I°, J°) the least precise pair of $Def(\Delta)$-interpretations, i.e., for all predicates $P/n \in Def(\Delta)$ and domain tuples $\vec{d} \in D^n$ we have that $(P(\vec{d}))^{I^\circ} = \mathbf{f}$, and

$(P(\bar{d}))^{J^\circ} = \mathbf{t}$. Let $(I^{n+1}, J^{n+1}) = \mathcal{ST}_\Delta^{(O_1,O_2)}(I^n, J^n)$ for $n \geq 0$, where (O_1, O_2) is the pair of interpretations representing the partial information about the open predicates of Δ as given by $\mathcal{I}$. The limit of $(I^n, J^n)_{n\geq 0}$, denoted by $\mathcal{M}_\Delta$, is the three-valued well-founded model of Δ.

Denote by $\mathcal{J}^\circ$ the $Def(\mathcal{D})$-interpretation in which every atom is false. Let $\mathcal{J}^{n+1}|_{Def(\mathcal{R}^{cf})} = \text{gfp}(T_{\mathcal{R}^{cf}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{ct})}})$ and $\mathcal{J}^{n+1}|_{Def(\mathcal{R}^{ct})} = \text{lfp}(T_{\mathcal{R}^{ct}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{cf})}})$ for $n \geq 0$ and finally denote the limit of $(\mathcal{J}^n)_{n\geq 0}$ by $\mathcal{M}_\mathcal{D}$. $\mathcal{M}_\mathcal{D}$ is the unique model of $\mathcal{D}$. Indeed, it is easy to show inductively that $\mathcal{J}^n \leq (\mathcal{C}_\mathcal{D}^{\tilde{I}})^n(\mathcal{J}^\circ)$ for every $n \geq 0$, and also that $\mathcal{C}_\mathcal{D}^{\tilde{I}}(\mathcal{M}_\mathcal{D}) = \mathcal{M}_\mathcal{D}$. This gives us that $\mathcal{M}_\mathcal{D}$ is the least fixpoint of $\mathcal{C}_\mathcal{D}^{\tilde{I}}$. Applying Lemma B.2 then gives us that $\mathcal{M}_\mathcal{D}$ is indeed the model of $\mathcal{D}$.

In the rest of the proof, we will show that for every $n \geq 0$ and $P \in Def(\Delta)$ it holds that $I^n(P) = \mathcal{J}^n(P^{ct})$ and $J^n(\neg P) = \mathcal{J}^n(P^{cf})$. We will show this by induction.

- $n = 0$, trivial,
- Assume that $I^n(P) = \mathcal{J}^n(P^{ct})$ and $J^n(\neg P) = \mathcal{J}^n(P^{cf})$. By this induction hypothesis and the way we have constructed $\mathcal{J}$ it follows that $I^{n+1}(P) = ST_\Delta^{(O_1,O_2)}(J^n)(P) = \text{lfp}(T_{\mathcal{R}^{ct}}^{\mathcal{J}^n+\tilde{I}})(P^{ct}) = \mathcal{J}^{n+1}(P^{ct})$ for every predicate $P \in Def(\Delta)$.

  Now for the second part, consider a $Def(\mathcal{D})$-interpretation $\mathcal{K}$ such that $\mathcal{K}|_{Def(\mathcal{R}^{cf})} = \text{lfp}(T_{\mathcal{R}^{pt}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{ct})}})$, where we define $\mathcal{R}^{pt}$ as the set of rules $\{\forall \bar{x}(P^{cf}(\bar{x}) \leftarrow ((\varphi_P[\bar{x}])^{ct}[Q^{cf}/\neg Q^{cf}])|Q, P \in Def(\Delta)\}$. Intuitively, the ruleset $\mathcal{R}^{pt}$ switches around the meaning of P^{cf} such that P^{cf} actually defines for every predicate when an atom is *possibly true*. I.e., we have to interpret P^{cf} being true here, as 'P is possibly true'.

  For an interpretation I, let us denote with $\bar{I}$ the interpretation such that $I(P) = \neg\tilde{I}(P)$. It is easy to verify that $\overline{T_{\mathcal{R}^{cf\neg}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{ct})}}(I)} = T_{\mathcal{R}^{cf}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{ct})}}(\bar{I})$, and thus that $\overline{\text{lfp}(T_{\mathcal{R}^{cf\neg}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{ct})}})} = \text{gfp}(T_{\mathcal{R}^{cf}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{ct})}})$. It then holds that $\mathcal{J}^{n+1}(P^{cf}) = \neg\mathcal{K}(P^{cf})$. Moreover, by the induction hypothesis and the way we constructed $\mathcal{R}^{cf\neg}$, we have that $\mathcal{K}(P^{cf}) = \text{lfp}(T_{\mathcal{R}^{cf\neg}}^{\tilde{I}+\mathcal{J}^n|_{Def(\mathcal{R}^{ct})}})(P^{cf}) = ST_\Delta^{(O_1,O_2)}(I^n)(P)$. Putting this together, we obtain that $\mathcal{J}^{n+1}(P^{cf}) = J^{n+1}(\neg P)$, which concludes the proof.

□

B.2 A sound FO(FD) approximation of $\Delta \wedge \psi$

In this section we show how, for an FO(ID) theory $\Delta \wedge \psi$, we again combine the approximation $Approx_\sigma(\psi)$ of ψ with the $FP_\sigma(\Delta)$ to produce a sound approximation of $\Delta \wedge \psi$ in the form of one big nested fixpoint definition, similar to what we did for the ct/pt approximation definitions. One thing to note here is that for a first-order formula ψ, the approximating definition $Approx_\sigma(\psi)$ is a positive definition. This means that we can equally see them as a least fixpoint definition of FO(FP).

Definition B.3. Let F be a Σ-FO(ID) formula of the form $\Delta \wedge \psi$, where the Δ is an inductive definition and ψ is an FO formula, and $\sigma \subseteq \Sigma$ such that $Open(\Delta) \subseteq \sigma$. We then define $\mathcal{D}^{\Delta\wedge\psi}_{\sigma,FP}$ as the FO(FP) definition obtained from $FP_\sigma(\Delta)$ and $Approx_\sigma(\psi)$ as follows. Let Δ' be a renamed version of Δ, i.e., with all defined predicates $P \in Def(\Delta)$ replaced by new symbols P'. Then $\mathcal{D}^{\Delta\wedge\psi}_{\sigma,FP}$ is the union of $Approx_\sigma(\psi) \cup \{A^{ct}_\psi \leftarrow \mathbf{t}\}$ with $FP_\sigma(\Delta')$ but with the rules $P'^{ct} \leftarrow P^{ct}$, $P^{ct} \leftarrow P'^{ct}$ and $P^{cf} \leftarrow P'^{cf}$ added to the least fixpoint definition and $P'^{cf} \leftarrow P^{cf}$ added to the greatest fixpoint expression, and finally for every open symbol of $Open(\Delta) \setminus \sigma$ that does not occur in ψ the rule $O'^{ct} \leftarrow \mathbf{f}.$ is added to the least fixpoint and the rule $O'^{cf} \leftarrow \mathbf{f}.$ is added to the greatest fixpoint:

$$\mathcal{D}^{\Delta\wedge\psi}_{\sigma,FP} = \left[\begin{array}{l} Approx_\sigma(\psi) \cup \{A^{ct}_\psi \leftarrow \mathbf{t}\}. \\ P'^{ct} \leftarrow P^{ct}. \\ O'^{ct} \leftarrow \mathbf{f}. \\ \cdots \\ \mathcal{R}'^{ct} \\ \left[\begin{array}{ll} P'^{cf} \leftarrow & P^{cf}. \\ O'^{cf} \leftarrow \mathbf{f}. & \\ \cdots & \\ \mathcal{R}'^{cf} & \end{array}\right] \end{array}\right],$$

Example B.4. Consider the following FO(ID) formula:

$$F = (\{P \leftarrow P \vee Q\} \wedge (A \Rightarrow Q),$$

and let $\sigma = \{A\}$ Then we have that

$$\mathcal{D}^F_{\sigma,FP} = \left[\begin{array}{l} Q^{ct} \leftarrow A \wedge A^{ct}_{\psi_1}. \\ Q^{cf} \leftarrow A^{cf}_{\psi_1}. \\ A^{cf}_{\psi_1} \leftarrow A \wedge Q^{cf}. \\ A^{ct}_{\psi_1} \leftarrow \neg A \vee Q^{ct}. \\ A^{ct}_{\psi_1} \leftarrow . \\ P'^{ct} \leftarrow P'^{ct} \vee Q^{ct}. \\ P'^{ct} \leftarrow P^{ct}. \\ P^{ct} \leftarrow P'^{ct}. \\ P^{cf} \leftarrow P'^{cf}. \\ \left[\begin{array}{l} P'^{cf} \leftarrow P'^{cf} \wedge Q^{cf}. \\ \\ P'^{cf} \leftarrow P^{cf}. \end{array}\right] \end{array}\right].$$

Note that if $A = \mathbf{t}$, the model of the nested fixpoint definition contains P^{ct}. If $A = \mathbf{f}$, then both P^{ct} and P^{cf} are false, that is, the truthvalue of P is not known.

Proposition B.5. *Given a vocabulary Σ, a subvocabulary σ and an FO(ID) formula $\Delta \wedge \psi$. Then, for every σ-interpretation I, if $\mathcal{D}^{\Delta\wedge\psi}_{\sigma,FP} \models P^{ct}(\bar{d})$ (resp. $P^{cf}(\bar{d})$), then it holds for every model M of $\Delta \wedge \psi$ extending I that $M \models P(\bar{d})$ (resp. $M \models \neg P(\bar{d})$).*

Proof. We assume again w.l.o.g. that $\sigma = \{\}$. The proof now is similar to the proof of Proposition 3.22.

The first step in this proof is showing that we can 'plug in' the information coming from $Approx(\psi)$ into $FP(\Delta)$ in a sound manner (similar to Lemma 3.23). Let $\mathcal{D}$ denote the following definition.

$$\mathcal{D} = \left[\begin{array}{l} P'^{ct} \leftarrow P^{ct}. \\ O'^{ct} \leftarrow \mathbf{f}. \\ \cdots \\ \mathcal{R}'^{ct} \\ \left[\begin{array}{lll} P'^{cf} & \leftarrow & P^{cf}. \\ O'^{cf} & \leftarrow \mathbf{f}. & \\ \cdots & & \\ \mathcal{R}'^{cf} & & \end{array}\right] \end{array}\right]$$

Let us denote the rules in the least fixpoint definition of $\mathcal{D}$ with $\mathcal{R}_{\lfloor\rfloor}$, and the rules in the greatest fixpoint definition with $\mathcal{R}_{\lceil\rceil}$. Assume now we have a three-valued Σ-interpretation $\mathcal{I}$, such that $\mathcal{I}$ is an approximation of the models of $\Delta \wedge \psi$. We will first prove that the model of $\mathcal{D}$ is indeed again an approximation of $\Delta \wedge \psi$.

Denote with $\mathcal{J}^{\circ}$ the $Def(\mathcal{D})$-interpretation that interprets everything false, and define $\forall n \geq 0 : \mathcal{J}^{n+1} = \mathcal{C}_{\mathcal{D}}^{\mathcal{O}^{ct/cf}}(\mathcal{J}^n)$. Let $\mathcal{J}$ be the limit of the sequence $(\mathcal{J}^n)_{n\geq 0} = \mathrm{lfp}(\mathcal{C}_{\mathcal{D}}^{\mathcal{O}^{ct/cf}})$, i.e., the unique model of $\mathcal{D}$ (Lemma B.2). We prove for each i that $\mathcal{J}^i$ is an approximation of all models of $\Delta \wedge \psi$.

For $\mathcal{J}^{\circ}$ this is trivially the case. Assume now that J^n is an approximation of all models of $\Delta \wedge \psi$, i.e., it holds that if $(P'^{ct}(\bar{d}))^{\mathcal{J}^n} = \mathbf{t}$ then $\models \Delta \wedge \psi \Rightarrow P(\bar{d})$, and if $(P'^{cf}(\bar{d}))^{\mathcal{J}^n} = \mathbf{t}$ then $\models \Delta \wedge \psi \Rightarrow \neg P(\bar{d})$. The first part in computing $\mathcal{C}_{\mathcal{D}}^{\mathcal{O}^{ct/cf}}(\mathcal{J}^n)$ consists in computing the nested greatest fixpoint. We prove by contradiction that this greatest fixpoint correctly approximates what is certainly false. Let us assume that this greatest fixpoint does *not* correctly approximate what is certainly false. Then there exists a predicate P^{cf} and a tuple $\bar{d}$ of domain elements such that $P^{cf}(\bar{d})$ is $\mathbf{t}$ in $\mathrm{gfp}(T_{\mathcal{R}_{\lceil\rceil}}^{\mathcal{O}^{ct/cf},\mathcal{J}^n})$, while at the same time there exists a model M of $\Delta \wedge \psi$ such that $P(\bar{d})$ is true in M. Let us denote $(\mathcal{R}_{\lceil\rceil})^{ct/pt}$ with $\mathcal{R}_{\lceil\rceil}^{pt}$. It is easy to prove that $P'^{cf}(\bar{d})$ is $\mathbf{t}$ in $\mathrm{gfp}(T_{\mathcal{R}_{\lceil\rceil}}^{\mathcal{O}^{ct/cf},\mathcal{J}^n})$ iff $P'^{pt}(\bar{d})$ is $\mathbf{f}$ in $\mathrm{lfp}(T_{\mathcal{R}_{\lceil\rceil}^{pt}}^{\mathcal{O}^{ct/pt},\mathcal{J}^n})$. We also have the following.

$$T_{\mathcal{R}_{\lceil\rceil}^{pt}[P'^{pt}/P]}^{\mathcal{O}^{ct/pt},\mathcal{J}^n}(I) \geq T_{\Delta}^{M|_{Open(\Delta)}}(I), \text{ for } I \leq M, \tag{B.1}$$

Indeed, if we look closer at $\mathcal{R}_{\lceil\rceil}^{pt}$, we see that this ruleset contains rules $P'^{pt} \leftarrow \varphi'^{pt} \wedge P^{pt}$ for every rule $P \leftarrow \varphi$ in Δ. When applying $T_{\mathcal{R}_{\lceil\rceil}^{pt}}^{\mathcal{O}^{ct/pt},\mathcal{J}^n}$ we interpret negative literals by $\mathcal{J}^n$, which we assumed as our induction hypothesis to be an approximation of M. Also we assumed that atoms P^{pt} are interpreted by $\mathcal{O}^{ct/pt}$.

So assume that $P'^{pt}(\bar{d})$ is not inferred by the application of operator $T_{\mathcal{R}_{\lceil\rceil}^{pt}}^{\mathcal{O}^{ct/pt},\mathcal{J}^n}$ on a certain interpretation $I \leq M$ because $P^{pt}(\bar{d})$ is false in $\mathcal{O}^{ct/pt}$. But then it follows from our assumptions about $\mathcal{O}$ that there cannot exists a model M of $\Delta \wedge \psi$ such that $P(\bar{d})$ is $\mathbf{t}$ in M, and thus $T_{\Delta}^{M|_{Open(\Delta)}}$ will not infer $P(\bar{d})$ neither, and thus (B.1) follows.

Continuing, now from (B.1) it follows that

$$\mathrm{lfp}(T_{\mathcal{R}_{\lceil\rceil}^{pt}[P'^{pt}/P]}^{\mathcal{O}^{ct/pt},\mathcal{J}^n}) \quad \geq \quad \mathrm{lfp}(T_{\Delta}^{M|_{Open(\Delta)}}) \ ,$$

which is clearly a contradiction with our assumptions that both $P'^{pt}(\bar{d})$ is $\mathbf{f}$ in $\mathrm{lfp}(T_{\mathcal{R}_{\lceil\rceil}^{pt}}^{\mathcal{O}^{ct/pt},\mathcal{J}^n})$ and $P(\bar{d})$ is $\mathbf{t}$ in $\mathcal{M}$. So computing the greatest fixpoint, gives us a correct approximation of what is certainly false.

The second step in computing $\mathcal{J}^{n+1}$ consists in computing the least fixpoint of $\mathcal{D}$. Again we prove by contradiction that the least fixpoint approximates correctly what is certainly true. So assume that there exists a predicate P and tuple $\bar{d}$ of domain elements for which $P^{ct}(\bar{d}) = \mathbf{t}$ in $\text{lfp}(T_{\mathcal{R}_{\lfloor\rfloor}}^{\mathcal{O}^{ct/cf},\mathcal{J}^{n+1}|_{cf}})$, while there exists a model M of $\Delta \wedge \psi$ such that $P(\bar{d})$ is not true in M. Here we have that

$$T_{\mathcal{R}_{\lfloor\rfloor}[P'^{ct}/P]}^{\mathcal{O}^{ct/cf},\mathcal{J}^{n+1}|_{cf}}(I) \leq T_{\Delta}^{M|_{Open(\Delta)}}(I), \qquad \forall I \tag{B.2}$$

since we know that $\mathcal{O}$ (assumption) and $\mathcal{J}^{n+1}|_{cf}$ (just proven above) are approximations of all models of $\Delta \wedge \psi$ extending $\mathcal{O}$, and thus we obtain

$$\text{lfp}(T_{\mathcal{R}_{\lfloor\rfloor}[P'^{ct}/P]}^{\mathcal{O}^{ct/cf},\mathcal{J}^{n+1}|_{cf}}) \leq \text{lfp}(T_{\Delta}^{M}|_{Open(\Delta)}),$$

which gives us a contradiction again.

This gives us that $\mathcal{J}^{n+1}$ is indeed a sound approximation of all models of $\Delta \wedge \psi$, and thus, through induction, it follows that $\mathcal{J}$ is a sound approximation of $\Delta \wedge \psi$.

Now consider the following nested fixpoint definition.

$$\lfloor Approx(\psi) \cup \{A_{\psi}^{ct} \leftarrow \mathbf{t}\} \cup \mathcal{D} \rfloor \tag{B.3}$$

Since two nested least-fixpoint definitions $\left\lfloor \begin{array}{c} \mathcal{R}_1 \\ \lfloor \mathcal{R}_2 \rfloor \end{array} \right\rfloor$ can be merged into an equivalent single least fixpoint definition $\left\lfloor \begin{array}{c} \mathcal{R}_1 \\ \mathcal{R}_2 \end{array} \right\rfloor$, we see that (B.3) is equivalent to $\mathcal{D}_{FP}^{\Delta\wedge\psi}$.

Since we know that $Approx(\psi) \cup \{A_{\psi}^{ct} \leftarrow \mathbf{t}\}$ computes a sound approximation of all models of ψ (and therefore also of $\Delta \wedge \psi$), it follows through an easy induction on the construction of the model of $\mathcal{D}_{FP}^{\Delta\wedge\psi}$ that the definition $\mathcal{D}_{FP}^{\Delta\wedge\psi}$ correctly approximates $\Delta \wedge \psi$.

□

Bibliography

Axling, T. & Haridi, S. (1994). A tool for developing interactive configuration applications. *Journal of Logic Programming*, **19**. 72

Balduccini, M., Gelfond, M., & Nogueira, M. (2006). Answer set based design of knowledge systems. *Ann. Math. Artif. Intell.*, **47**(1-2), pp. 183–219. 73

Baral, C. (2003). *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, New York, NY, USA. ISBN 0521818028. 72, 77, 123

Baral, C., Gelfond, M., & Kosheleva, O. (1998). Expanding queries to incomplete databases by interpolating general logic programs. *J. Log. Program.*, **35**(3), pp. 195–230. 80, 96

Baral, C., Greco, G., Leone, N., & Terracina, G. (eds.) (2005). *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, volume 3662 of *LNCS*. Springer. ISBN 3-540-28538-5. 169, 172

Baral, C., Kreinovich, V., & Trejo, R. (2000). Computational complexity of planning and approximate planning in the presence of incompleteness. *Artif. Intell.*, **122**(1-2), pp. 241–267. 91

Baral, C. & Subrahmanian, V.S. (1991). Duality between alternative semantics of logic programs and nonmonotonic formalisms. *Intl. Workshop on Logic Programming and*

Nonmonotonic Reasoning (A. Nerode, W. Marek, & V. Subrahmanian, eds.). MIT Press, Washington DC., pp. 69–86. 18

Barker, S. (2009). The next 700 access control models or a unifying meta-model? *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM, SACMAT '09, pp. 187–196. 125, 149

Barwise, J. (1977). *Handbook of Mathematical Logic*. North-Holland, Amsterdam. 16

Belnap, N.D. (1977). A useful four-valued logic. *Modern Uses of Multiple-Valued Logic* (J.M. Dunn & G. Epstein, eds.), Reidel, Dordrecht, pp. 8–37. Invited papers from the Fifth International Symposium on Multiple-Valued Logic, held at Indiana University, Bloomington, Indiana, May 13-16, 1975. 12

Berners-Lee, T., Dan Connolly, E.P., & Scharf, Y. (2005). Experience with N3 rules. *W3C Workshop on Rule Languages for Interoperability*. 125, 149

Bouquet, P., Giunchiglia, F., Harmelen, F.V., Serafini, L., & Stuckenschmidt, H. (2003). C-OWL: Contextualizing ontologies. *Journal Of Web Semantics*. Springer Verlag, pp. 164–179. 125

Brewka, G. (1989). Preferred subtheories: An extended logical framework for default reasoning. *IJCAI*. pp. 1043–1048. 148

Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. *CACM*, **54**(12), pp. 92–103. 123

Calimeri, F., Ianni, G., Ricca, F., Alviano, M., Bria, A., Catalano, G., Cozza, S., Faber, W., Febbraro, O., Leone, N., Manna, M., Martello, A., Panetta, C., Perri, S., Reale, K., Santoro, M.C., Sirianni, M., Terracina, G., & Veltri, P. (2011). The third answer set programming system competition: Preliminary report of the system competition track. *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. Springer, pp. 388–403. 23, 77

Calus, N. & Vennekens, J. (2011). A Java API for a knowledge base system. *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*. pp. 4f53–454. 74

Chellas, B. (1980). *Modal Logic: An Introduction*. Cambridge University Press. 31

Cholewinski, P. (1994). Stratified default theories. *CSL*. pp. 456–470. 147

Claessen, K. & Sörensson, N. (2003). New techniques that improve MACE-style model finding. *MODEL*. 135, 137

Clark, K.L. (1978). Negation as failure. *Logic and Data Bases*. Plenum Press, pp. 293–322. ISBN 0-306-40060-X. 44, 149

Cook, S.A. (1971). The complexity of theorem-proving procedures. *STOC*. ACM, pp. 151–158. 15

Davulcu, H., Kifer, M., Ramakrishnan, C.R., & Ramakrishnan, I.V. (1998). Logic based modeling and analysis of workflows. *PODS '98*. 73

de Bruijn, J., Pearce, D., Polleres, A., & AugustínValverde (2007). Quantified equilibrium logic and hybrid rules. *International Conference on Web Reasoning and Rule Systems (RR)*. pp. 58–72. 149

Demolombe, R. & del Pilar Pozos Parra, M. (2000). A simple and tractable extension of situation calculus to epistemic logic. *ISMIS*. pp. 515–524. 101, 102, 121

Denecker, M., Cortés Calabuig, Á., Bruynooghe, M., & Arieli, O. (2010a). Towards a logical reconstruction of a theory for locally closed databases. *ACM Transactions on Database Systems*, **35**(3), pp. 22:1–22:60. 42, 80, 96, 97, 98

Denecker, M., Marek, V.W., & Truszczyński, M. (2000). Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. *Logic-based Artificial Intelligence*, Kluwer Academic Publishers, Boston, The Kluwer International Series in Engineering and Computer Science, pp. 127–144. 18, 19

Denecker, M., Marek, V.W., & Truszczynski, M. (2003). Uniform semantic treatment of default and autoepistemic logics. *Artif. Intell.*, **143**(1), pp. 79–122. 32, 131

Denecker, M., Marek, V.W., & Truszczyński, M. (2004). Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation*, **192**(1), pp. 84–121. 17, 18, 19

Denecker, M., Marek, V.W., & Truszczynski, M. (2011). Reiter's default logic is a logic of autoepistemic reasoning and a good one, too. *CoRR*, **abs/1108.3278**. 123, 146

Denecker, M. & Ternovska, E. (2004). A logic of non-monotone inductive definitions and its modularity properties. *LPNMR* (V. Lifschitz & I. Niemelä, eds.). Springer, volume 2923 of *LNCS*, pp. 47–60. ISBN 3-540-20721-X. 41

Denecker, M. & Ternovska, E. (2007). Inductive situation calculus. *Artificial Intelligence*, **171**(5-6), pp. 332–360. 25, 26, 27

Denecker, M. & Ternovska, E. (2008). A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL)*, **9**(2), p. Article 14. 3, 20, 24, 44, 79, 148

Denecker, M. & Vennekens, J. (2007). Well-founded semantics and the algebraic theory of non-monotone inductive definitions. *LPNMR* (C. Baral, G. Brewka, & J.S. Schlipf, eds.). Springer, volume 4483 of *LNCS*, pp. 84–96. ISBN 978-3-540-72199-4. 51

Denecker, M., Vennekens, J., Bond, S., Gebser, M., & Truszczyński, M. (2009). The second Answer Set Programming competition. *LPNMR* (E. Erdem, F. Lin, & T. Schaub, eds.). Springer, volume 5753 of *LNCS*, pp. 637–654. ISBN 978-3-642-04237-9. 77, 94

Denecker, M., Vennekens, J., Vlaeminck, H., Wittocx, J., & Bruynooghe, M. (2010b). Answer set programming's contributions to classical logic. An analysis of ASP methodology. *MG-65: Symposium on Constructive Mathematics in Computer Science* (M. Balduccini & S. Tran, eds.), Lexington. 5, 123, 124, 125, 146, 148, 150

Ditmarsch, H.v., van der Hoek, W., & Kooi, B. (2007). *Dynamic Epistemic Logic*. Springer Publishing Company, Incorporated, 1st edition. ISBN 1402058381, 9781402058387. 101, 121

Doherty, P., Magnusson, M., & Szalas, A. (2006). Approximate databases: a support tool for approximate reasoning. *Journal of Applied Non-Classical Logics*, **16**(1-2), pp. 87–118. 80, 96, 98, 99

E. Giunchiglia, Y.L. & Maratea, M. (2011). Cmodels homepage. http://www.cs.utexas.edu/users/tag/cmodels.html. 94

Enderton, H.B. (2001). *A Mathematical Introduction To Logic*. Academic Press, second edition. 7

Fagin, R. (1974). Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation*, **7**, pp. 43–74. 16, 77

Feferman, S. (1984). Toward useful type-free theories. *Journal of Symbolic Logic*, **49**(1), pp. 75–111. 14, 41, 98

Fitting, M. (2002). Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, **278**(1-2), pp. 25–51. 18

Gelfond, M. (1987). On Stratified Autoepistemic Theories. *National Conference on Artificial Intelligence (AAAI'87)*. Morgan Kaufman, pp. 207–211. 147

Gelfond, M. (1991). Strong introspection. *AAAI*. pp. 386–391. 124, 144

Gelfond, M. (2011). New semantics for epistemic specifications. *LPNMR* (J.P. Delgrande & W. Faber, eds.). Springer, volume 6645 of *Lecture Notes in Computer Science*, pp. 260–265. ISBN 978-3-642-20894-2. 145

Gelfond, M. & Lifschitz, V. (1988). The stable model semantics for logic programming. *ICLP/SLP* (R.A. Kowalski & K.A. Bowen, eds.). MIT Press, pp. 1070–1080. ISBN 0-262-61056-6. 23, 130

Gelfond, M. & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9**(3/4), pp. 365–386. 72, 123, 127, 143

Gelfond, M. & Lifschitz, V. (1998). Action languages. *Electron. Trans. Artif. Intell.*, **2**, pp. 193–210. 25

Gottlob, G. (1992). Complexity results for nonmonotonic logics. *J. Log. Comput.*, **2**(3), pp. 397–425. 137

Herzig, A., Lang, J., & Marquis, P. (2003). Action representation and partially observable planning using epistemic logic. *Proceedings of the 18th international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'03, pp. 1067–1072. 101, 121

Hou, P., De Cat, B., & Denecker, M. (2010). FO(FD): Extending classical logic with rule-based fixpoint definitions. *Theory and Practice of Logic Programming*, **10**(4-6), pp. 581–596. 28, 30, 31

Hughes, G.E. & Cresswell, M.J. (1996). *A New Introduction To Modal Logic*. Routledge. 31

Hunter, A. & Delgrande, J.P. (2011). Iterated belief change due to actions and observations. *J. Artif. Intell. Res. (JAIR)*, **40**, pp. 269–304. 121

idp (2012). The idp system. http://dtai.cs.kuleuven.be/krr/software. 25, 55, 64, 65, 71, 152

Immerman, N. (1998). *Descriptive Complexity*. Springer Verlag. 78

Kifer, M. (2005). Nonmonotonic reasoning in FLORA-2. (Baral et al., 2005), pp. 1–12. 149

Kleene, S.C. (1952). *Introduction to Metamathematics*. Van Nostrand. 13

Konolige, K. (1988a). Hierarchic autoepistemic theories for nonmonotonic reasoning. *AAAI*. pp. 439–443. 5, 123, 124, 125, 142, 146, 150

Konolige, K. (1988b). On the relation between default and autoepistemic logic. *Artificial Intelligence*, **35**, pp. 343–382. 131

Konolige, K. (1989). On the relation between autoepistem ic logic and circumscription. *IJCAI*. pp. 1213–1218. 131

Lakemeyer, G. & Levesque, H.J. (1998). Aol: A logic of acting, sensing, knowing, and only knowing. *KR* (A.G. Cohn, L.K. Schubert, & S.C. Shapiro, eds.). Morgan Kaufmann, pp. 316–329. 101, 121

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, **7**(3), pp. 499–562. 74

Levesque, H.J. (1990). All I know: A study in autoepistemic logic. *Artif. Intell.*, **42**(2-3), pp. 263–309. 31

Levesque, H.J. & Lakemeyer, G. (2000). *The logic of knowledge bases*. MIT Press. ISBN 978-0-262-12232-0. 135

Libkin, L. (2004). *Elements of Finite Model Theory*. Springer. 20

Liu, Y. & Levesque, H.J. (1998). A completeness result for reasoning with incomplete first-order knowledge bases. *KR*. pp. 14–23. 42, 82, 96, 99, 141

Liu, Y. & Levesque, H.J. (2005). Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. *IJCAI* (L.P. Kaelbling & A. Saffiotti, eds.). Professional Book Center, pp. 522–527. ISBN 0-938-07593-4. 101, 102, 121

Marek, V.W. & Truszczynski, M. (1989). Relating autoepistemic and default logics. *KR*. pp. 276–288. 131

Marek, V.W. & Truszczyński, M. (1991). Autoepistemic logic. *Journal of the ACM*, **38**(3), pp. 588–619. 147

Mariën, M., Wittocx, J., & Denecker, M. (2006). The IDP framework for declarative problem solving. *Search and Logic: Answer Set Programming and SAT*. pp. 19–34. 63, 64, 72, 80, 90, 94

McDermott, D. (1987). A critique of pure reason. *Computational Intelligence*, **3**, pp. 151–160. 91

Mitchell, D.G. & Ternovska, E. (2005). A framework for representing and solving NP search problems. *AAAI* (M.M. Veloso & S. Kambhampati, eds.). AAAI Press / The MIT Press, pp. 430–435. ISBN 1-57735-236-X. 77

Moore, R.C. (1984). Possible-world semantics for autoepistemic logic. *Proceedings of the Workshop on Non-Monotonic Reasoning*. pp. 344–354. Reprinted in: M. Ginsberg, ed., *Readings on Nonmonotonic Reasoning*, pages 137–142, Morgan Kaufmann, 1990. 31, 123

Moore, R.C. (1985). A Formal Theory of Knowledge and Action. *Formal Theories of the Commonsense World* (J.R. Hobbs & R.C. Moore, eds.), pp. 319–358. 101, 120

Motik, B. & Rosati, R. (2010). Reconciling description logics and rules. *Journal of the ACM (JACM)*, **57**. 149

Niemelä, I. (1992). On the decidability and complexity of autoepistemic reasoning. *Fundam. Inform.*, **17**(1-2), pp. 117–155. 137

Niemelä, I., Simons, P., & Syrjänen, T. (2000). Smodels: A system for answer set programming. *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*. Breckenridge, Colorado, USA. CoRR, cs.AI/0003033. 94

Nikolopoulus, C. (1997). *Expert Systems: Introduction to First and Second Generation and Hybrid Knowledge Based Systems*. 72

Oberschelp, A. (1962). Untersuchungen zur mehrsortigen quantorenlogik. *Mathematische Annalen*, **145**(4), pp. 297–333. 14

Oberschelp, A. (1989). Order sorted predicate logic. *Sorts and Types in Artificial Intelligence* (K. Bläsius, U. Hedtstück, & C.R. Rollinger, eds.). Springer-Verlag, pp. 8–17. 3

Palacios, H. & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research (JAIR)*, **35**, pp. 623–675. 94

Park, D. (1969). Fixpoint induction and proofs of program properties. *Machine Intelligence*, **5**, pp. 59–78. 28

Pelov, N., Denecker, M., & Bruynooghe, M. (2007). Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming (TPLP)*, **7**(3), pp. 301–353. 3

Petrick, R.P.A. & Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. *ICAPS*. pp. 2–11. 101, 102, 121

Petrick, R.P.A. & Levesque, H.J. (2002). Knowledge equivalence in combined action theories. *KR*. pp. 303–314. 102, 121

Polleres, A., Feier, C., & Harth, A. (2006). Rules with contextually scoped negation. *Proc. 3 rd European Semantic Web Conf. (ESWC2006*. Springer, pp. 332–347. 125, 148

Pührer, J., Heymans, S., & Eiter, T. (2010). Dealing with inconsistency when combining ontologies and rules using dl-programs. *ESWC (1)* (L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, & T. Tudorache, eds.). Springer, volume 6088 of *Lecture Notes in Computer Science*, pp. 183–197. ISBN 978-3-642-13485-2. 149

Reiter, R. (1977). On closed world data bases. *Logic and Data Bases* (H. Gallaire & J. Minker, eds.). Plenum Press, pp. 55–76. ISBN 0-306-40060-X. 128

Reiter, R. (1980a). Equality and domain closure in first-order databases. *Journal of the ACM*, **27**(2), pp. 235–249. 15

Reiter, R. (1980b). A logic for default reasoning. *Artif. Intell.*, **13**(1-2), pp. 81–132. 123, 131

Reiter, R. (1982). Towards a logical reconstruction of relational database theory. *On Conceptual Modelling (Intervale)* (M.L. Brodie, J. Mylopoulos, & J.W. Schmidt, eds.). Springer, pp. 191–233. ISBN 3-540-90842-0. 15

Reiter, R. (1991). Artificial intelligence and mathematical theory of computation. Academic Press Professional, Inc., San Diego, CA, USA, chapter The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression, pp. 359–380. ISBN 0-12-450010-2. 25, 26

Rosati, R. (1997). Embedding minimal knowledge into autoepistemic logic. *Proceedings of the Fifth Conference of the Italian Association for Artificial Intelligence (AI*IA'97)*. Springer, Lecture Notes in Artificial Intelligence, pp. 231–241. 149

Scherl, R.B. & Levesque, H.J. (2003). Knowledge, action, and the frame problem. *Artificial Intelligence*, **144**, pp. 1 – 39. 101, 120, 121

Schlipf, J.S. (1995). Complexity and undecidability results for logic programming. *Annals of Mathematics and Artificial Intelligence*, **15**(3-4), pp. 257–288. 23

Shanahan, M. (1997). *Solving the frame problem - a mathematical investigation of the common sense law of inertia*. MIT Press. 25

Soininen, T. & Niemelä, I. (1999). Developing a declarative rule language for applications in product configuration. *PADL*. 72

Son, T.C. & Baral, C. (2001). Formalizing sensing actions a transition function based approach. *Artif. Intell.*, **125**(1-2), pp. 19–91. 101, 121

Son, T.C., Tu, P.H., Gelfond, M., & Morales, A.R. (2005). An approximation of action theories of and its application to conformant planning. (Baral et al., 2005), pp. 172–184. 80, 87, 89, 91, 94, 100

Stockmeyer, L.J. (1976). The polynomial-time hierarchy. *Theoretical Computer Science*, **3**(1), pp. 1 – 22. 16, 78

Subbarayan, S., Jensen, R.M., Hadzic, T., Andersen, H.R., Hulgaard, H., & Møller, J. (2004). Comparing two implementations of a complete and backtrack-free interactive configurator. *Proc. of Workshop on CSP Techniques with Immediate Application, CP04*. 72, 74

Tamaki, H. & Sato, T. (1984). Unfold/fold transformations of logic programs. *ICLP*. pp. 127–138. 85

Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. 17

Thielscher, M. (2000). Representing the knowledge of a robot. *KR* (A.G. Cohn, F. Giunchiglia, & B. Selman, eds.). Morgan Kaufmann, pp. 109–120. 101, 121

Torlak, E. & Jackson, D. (2007). Kodkod: A relational model finder. *TACAS* (O. Grumberg & M. Huth, eds.). Springer, volume 4424 of *LNCS*, pp. 632–647. ISBN 978-3-540-71208-4. 135, 137

Tseitin, G.S. (1968). On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic II* (A.O. Slisenko, ed.), Consultants Bureau, N.Y., pp. 115–125. 35

Tu, P.H., Son, T.C., & Baral, C. (2007). Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *TPLP*, **7**(4), pp. 377–450. 101, 102, 121

Turner, H. (2002). Polynomial-length planning spans the polynomial hierarchy. *JELIA*. pp. 111–124. 91

van der Aalst, W.M.P. & ter Hofstede, A.H.M. (2005). Yawl: yet another workflow language. *Inf. Syst.*, **30**(4), pp. 245–275. 73

van Fraassen, B. (1966). Singular terms, truth-value gaps and free logic. *Journal of Philosophy*, **63**(17), pp. 481–495. 13

Van Gelder, A. (1993). The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, **47**(1), pp. 185–221. 23

Van Gelder, A., Ross, K.A., & Schlipf, J.S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, **38**(3), pp. 620–650. 23

Vanden Bossche, M., Ross, P., MacLarty, I., Van Nuffelen, B., & Pelov, N. (2007). Ontology driven software engineering for real life applications. *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE)*. 72

Vennekens, J. (2007). *Algebraic and logical study of constructive processes in knowledge representation*. Ph.D. thesis, K.U.Leuven, Leuven, Belgium. 18

Vennekens, J., Gilis, D., & Denecker, M. (2006). Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic*, **7**(4), pp. 765–797. 129, 147

Vlaeminck, H., Vennekens, J., Bruynooghe, M., & Denecker, M. (2012a). Ordered epistemic logic. *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning*. 6

Vlaeminck, H., Vennekens, J., & Denecker, M. (2009). A logical framework for configuration software. *PPDP* (A. Porto & F.J. López-Fraguas, eds.). ACM, pp. 141–148. ISBN 978-1-60558-568-0. 6

Vlaeminck, H., Vennekens, J., & Denecker, M. (2012b). A general representation and approximate inference algorithm for sensing actions. *Proceedings of the AI 2012 Conference*, (M. Tielscher & D. Zhang, eds.). 6

Vlaeminck, H., Vennekens, J., Denecker, M., & Bruynooghe, M. (2012c). An approximative inference method for solving ∃∀SO satisfiability problems. *The Journal of Artificial Intelligence Research*, **45**. 6

Wittocx, J. (2010). *Finite Domain and Symbolic Inference Methods for Extensions of First-Order Logic*. Ph.D. thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium. 19

Wittocx, J., Denecker, M., & Bruynooghe, M. (2010). Constraint propagation for extended first-order logic. *CoRR*, **abs/1008.2121**. 4, 5, 6, 34, 35, 37, 43, 55

Wittocx, J., Mariën, M., & Denecker, M. (2008). The idp system: a model expansion system for an extension of classical logic. *LaSh* (M. Denecker, ed.). pp. 153–165. 135, 137

Wittocx, J., Vlaeminck, H., & Denecker, M. (2009). Debugging for model expansion. *ICLP* (P.M. Hill & D.S. Warren, eds.). Springer, volume 5649 of *LNCS*, pp. 296–311. ISBN 978-3-642-02845-8. 69, 70

Curriculum vitae

Hanne Vlaeminck studied mathematics (licentiaat wiskunde) at the University of Leuven (KU Leuven), Belgium. Her master thesis was titled 'Het tiende probleem van Hilbert' and was supervised by professor Jan Denef. She graduated *cum laude* in June 2007.

After her mathematics studies she continued her studies at the University of Leuven, and obtained an additional degree in informatics, also *cum laude*, in June 2008.

In August 2008, she joined the Knowledge Representation and Reasoning group at the University of Leuven to start her doctoral studies. In January 2009 she received a four year personal grant from the Institute for the promotion of innovation through science and technology in Flanders (IWT Vlaanderen) to work on her Ph.D., under the supervision of Prof. M. Denecker.

List of publications

Articles in internationally reviewed journals

- H. Vlaeminck, J. Vennekens, M. Denecker, M. Bruynooghe. *An approximative inference method for solving $\exists\forall SO$ satisfiability problems*, The Journal of Artificial Intelligence Research, volume 45, pages 79-124, 2012.
  http://dx.doi.org/10.1613/jair.3658

Papers at international conferences and symposia, published in full in proceedings

- H. Vlaeminck, J. Vennekens, M. Denecker. *A general representation and approximate inference algorithm for sensing actions*, M. Tielscher, D. Zhang (eds.), AI, Sydney, 4-7 December 2012, Proceedings of the AI 2012 Conference.
- H. Vlaeminck, J. Vennekens, M. Bruynooghe, M. Denecker. *Ordered Epistemic Logic: Semantics, complexity and applications*, G. Brewka, T. Eiter, S. McIlraith (eds.), Knowledge Representation and Reasoning, Rome, 10-14 July 2012, Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, pages 369-379, AAAI Press.

- M. Denecker, J. Vennekens, H. Vlaeminck, J. Wittocx, M. Bruynooghe. *Answer Set Programming's contributions to classical logic - An analysis of ASP methodology*, MG-65: Symposium on Constructive Mathematics in Computer Science, M. Balduccini, T. Son (eds.), Springer Verlag, 2011.
http://dx.doi.org/10.1007/978-3-642-20832-4

- H. Vlaeminck, J. Wittocx, J. Vennekens, M. Denecker, M. Bruynooghe.*An approximative inference method for solving $\exists\forall SO$ satisfiability problems*, T. Janhunen, I. Niemelä (eds.), European Conference on Logics in Artificial Intelligence (JELIA), Helsinki, 13-16 September 2010, Lecture Notes in Computer Science, volume 6341, pages 326-338, Springer.
http://dx.doi.org/10.1007/978-3-642-15675-5_28

- H. Vlaeminck, J. Vennekens, M. Denecker. *A logical framework for configuration software*, Principles and Practice of Declarative Programming, Coimbra, Portugal, 7-9 September, 2009, Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming PPDP'09, ACM.
http://dx.doi.org/10.1145/1599410.1599428

- J. Wittocx, H. Vlaeminck, M. Denecker. *Debugging for model expansion*, International Conference on Logic Programming, ICLP, Pasadena, California, US, 14-17 July 2009, Lecture Notes in Computer Science, volume 5649, pages 296-311, Springer.
http://dx.doi.org/10.1007/978-3-642-02846-5_26

Contributions at international conferences, not published or only as abstract

- H. Vlaeminck, J. Vennekens, M. Bruynooghe, M. Denecker. *Ordered Epistemic Logic: Semantics, Complexity and Applications*, 24th Benelux Conference on Artificial Intelligence, BNAIC 2012, Maastricht, October 25-26, 2012.

- H. Vlaeminck, K. Abhinav, J. Vennekens, M. Denecker, J. Wittocx. *ConfigNow: a knowledge based approach to configuration software*, 22nd Benelux Conference on Artificial Intelligence, BNAIC 2010, Luxemburg, October 25-26, 2010.