Default Reasoning about Actions

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet

Informatik

vorgelegt

von M.Sc. Hannes Straß

geboren am 12. Februar 1984 in Karl-Marx-Stadt (heute Chemnitz).

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Gerhard Brewka, Universität Leipzig
   Prof. Dr. Michael Thielscher, The University of New South Wales, Sydney
2. Prof. Dr. Torsten Schaub, Universität Potsdam

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am
21. Juni 2012 mit dem Gesamtprädikat *magna cum laude*.

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

.....
(Ort, Datum)

.....
(Unterschrift)

Abstract

Action Theories are versatile and well-studied knowledge representation formalisms for modelling dynamic domains. However, traditional action theories allow only the specification of definite world knowledge, that is, universal rules for which there are no exceptions. When modelling a complex domain for which no complete knowledge can be obtained, axiomatisers face an unpleasant choice: either they cautiously restrict themselves to the available definite knowledge and live with a limited usefulness of the axiomatisation, or they bravely model some general, defeasible rules as definite knowledge and risk inconsistency in the case of an exception for such a rule.

This thesis presents a framework for default reasoning in action theories that overcomes these problems and offers useful default assumptions while retaining a correct treatment of default violations. The framework allows to extend action theories with defeasible statements that express how the domain usually behaves. Normality of the world is then assumed by default and can be used to conclude what holds in the domain under normal circumstances. In the case of an exception, the default assumption is retracted, whereby consistency of the domain axiomatisation is preserved.

Acknowledgements

Many people have contributed in one way or another to the successful completion of this thesis. Whilst it is impossible to mention everyone by name, I would nonetheless like to express my gratitude towards the following.

I always believed that my supervisors Michael Thielscher and Gerhard Brewka did their job very well, but only through numerous conversations with other PhD students did I learn that they were *in fact* the best supervisors I could have possibly hoped for. Michael Thielscher gave me the opportunity, guided me along the way and continued to support me after switching universities. Gerhard Brewka and I had many fruitful and enlightening discussions, furthermore he made it possible for me to keep on working in the field.

Torsten Schaub wrote a review for this thesis and gave constructive feedback on it.

Uwe Petersohn kindly agreed to act as my "Fachreferent."

We had a very pleasant working environment in Dresden's Computational Logic group, which would not have been the same without my colleagues (and mostly co-PhD students) Conrad Drescher, Bärbel Gärtner, Sandra Großmann, Sebastian Haufe, Yves Martin, Daniel Michulke and Stephan Schiffel. (So long, and thanks for all the cake!)

Although my seven months in Sydney seemed to me more like seven weeks, my collaborators, colleagues and co-PhD students at the UNSW made me feel like a part of the team as if I had been there for years – in particular I want to mention Maurice Pagnucco, David Rajaratnam, Nina Narodytska, Nawid Jamali, Bhuman Soni and Bradford Heap.

My colleagues in Leipzig, Ringo Baumann and Frank Loebe, provided a very warm welcome – I am looking forward to working with you.

Over the years, I have benefited from the criticism and suggestions of numerous anonymous reviewers. In addition, several people gave feedback on drafts of various documents; in particular, Gerhard Brewka and Sebastian Haufe read and commented on a previous version of this thesis.

My friends in Dresden, Sydney, Leipzig and all over the place provided many welcome distractions and made everything worthwhile.

Finally, my family, especially my parents, offered the constant support and encouragement that made all of this possible.

While I believe that all of those mentioned have contributed to an improved final version, none is, of course, responsible for remaining shortcomings. Still, I hope that the rest of the material will be enough to stimulate new insights into default reasoning about actions.

Contents

1 Introduction **1**
- 1.1 Reasoning about Actions . . . 4
  - 1.1.1 Situation Calculus . . . 6
  - 1.1.2 Fluent Calculus . . . 6
  - 1.1.3 Event Calculus . . . 7
  - 1.1.4 Action Languages . . . 7
  - 1.1.5 Unifying Action Calculus . . . 7
- 1.2 Nonmonotonic Reasoning . . . 8
  - 1.2.1 Default Logic . . . 9
  - 1.2.2 Circumscription . . . 9
  - 1.2.3 Autoepistemic Logic . . . 9
- 1.3 Previous Approaches for Combinations . . . 10
- 1.4 Publications . . . 10

2 Preliminaries **11**
- 2.1 Unifying Action Calculus . . . 15
- 2.2 Default Logic . . . 18
- 2.3 Logic Programming . . . 20
- 2.4 Notational Conventions . . . 21

3 Action Default Theories **23**
- 3.1 An Action Description Language . . . 23
  - 3.1.1 Syntax . . . 24
  - 3.1.2 Semantics . . . 26
- 3.2 Basic Action Default Theories . . . 28
- 3.3 Atomic State Defaults . . . 34
- 3.4 Normal Defaults . . . 40
  - 3.4.1 The Prerequisite . . . 40
  - 3.4.2 The Effect Axiom . . . 42
  - 3.4.3 Reifying Default Conclusions . . . 44
  - 3.4.4 Comparison to the Previous Approaches . . . 48
- 3.5 Conditional Effects . . . 49
- 3.6 Global Effects . . . 52
- 3.7 Disjunctive Effects . . . 54
- 3.8 Concluding Remarks . . . 56

4 Implementation **59**
4.1 From Action Domain Specifications to Answer Set Programs 60
4.1.1 From Action Domain Specifications to CNF Default Theories 61
4.1.2 . . . to Propositional Default Theories 63
4.1.3 . . . to Propositional Definite Horn Default Theories 66
4.1.4 . . . to Propositional Answer Set Programs 76
4.2 From Action Domain Specifications to Open Answer Set Programs 78
4.3 The Implemented System draculasp 82
4.4 Related Work 86

5 Ramification and Loop Formulas **89**
5.1 Specifying Indirect Effects 90
5.2 An Axiomatic Solution to the Ramification Problem 93
5.2.1 Loops and Loop Formulas 95
5.2.2 From Trigger Formulas to Trigger Literals 98
5.3 Correctness of the Solution 99
5.4 Concluding Remarks 102
5.4.1 Combining Ramifications and Defaults 102
5.4.2 Related Work on Ramification 103

6 Conclusion **105**
6.1 Related Work on Default Reasoning about Actions 105
6.1.1 Defaults in the Discrete Event Calculus 105
6.1.2 Action Language $\mathcal{C}+$ 106
6.1.3 Modal Situation Calculus with Only-knowing 108
6.1.4 Argumentation-based Approaches to Knowledge Qualification 109
6.1.5 Further Approaches 110
6.2 Directions for Future Work 111
6.2.1 State Defaults and the Qualification Problem 111
6.2.2 Preferred Default Logic 113
6.2.3 Further Future Work 113

Bibliography **113**

Index **125**

Chapter 1

Introduction

Artificial Intelligence (AI) aims at creating intelligent machines. Being embedded in a complex environment, they should be capable of rational thought and purposeful action to effectively interact with their surroundings. The problem-solving tasks we expect these artificially intelligent systems to solve will require a considerable amount of background knowledge about the world. Knowledge Representation (KR) is the sub-field of AI that investigates how to represent knowledge in a principled way by using mathematical formalisms. Apart from the ability to express *explicit* knowledge, a good KR formalism must offer inference mechanisms to derive *implicit* knowledge from the information expressly stated. Additionally, the meaning of pieces of knowledge should have an objective basis, avoiding potential mistakes through ambiguity and misunderstanding.

These requirements make mathematical logic ideally suited as a knowledge representation formalism: first, it has a well-defined formal semantics that gives a meaning to language constructs. What is more, logic is the systematic study of valid reasoning, and therefore naturally concerned with procedures that correctly infer implicit knowledge. Philosophers, mathematicians and computer scientists have been studying formal logics for more than a century, so there is a huge amount of results we can straightforwardly use.

Indeed, there is already quite a number of logic-based knowledge representation formalisms. For example, terminological languages are used to express *ontologies*, that define the concepts of an application domain and relationships between these concepts. For example, the medical domain uses ontologies that define different parts of the human body and how they relate to each other, like "the hand is a part of the arm" and "the arm is a part of the body." Now even if it is not explicitly stated, a suitably formalised ontology will entail the implicit information that the hand is a part of the body as well.

Although such terminological KR languages are successfully used, they are limited in their ability to express *change*. When the domain of discourse changes, the representation has to be changed on the meta-level to realign model and reality. This however means that change itself is not part of the ontology and cannot be easily represented. While such formalisms are *static* – that is, time-independent –, different formalisms exist that model time and change explicitly: action theories. They are used to model *dynamic* domains – worlds that change over time.

If we want to represent that the world changes over time, we first have to represent time itself. Decades of research in action theories have provided us with established formalisations of time that we can use off the shelf. We indeed go one step further and take an abstract stance with regard to time. Our view will be to treat time as consisting of atomic elements – *time points* – and a partial order that indicates whether there is a relative past/future relationship

between two given time points.

Next, we have to think about how to represent the state of the world at one time point. In principle, it would be feasible to treat states (of the world) as atomic entities as well. Just like in automata theory, transitions between states could then be employed to model change. But such a representation immediately leads to combinatorial explosion: if we have to distinguish n binary world properties to model the domain of discourse, there are potentially 2^n states to consider. For example an online shopping agent in a world with m items to shop for and n different shops must discriminate and explicitly represent at least $2^{m \cdot n}$ states. In particular, if we add a single item to this world, the number of states explicit in the model increases by a factor of 2^n.

Hence our alternative here will be to use states with an internal structure. We decompose them into atomic world properties that may change over time, so-called *fluents*. These fluents are represented *symbolically* instead of being encoded implicitly in an atomic state. More specifically, we use terms such as $\mathit{Offers}(\mathit{shop}, \mathit{item}, \mathit{price})$ saying that a shop offers an item at a particular price. Now adding a shop or an item is as simple as adding a function symbol to the underlying mathematical language and increases the model complexity only by a small constant.

It remains to incorporate change, the concept we wanted to represent in the first place. We take the stance that change is initiated through *actions*, that alter the state of the world. Since the state of the world is represented by the status of fluents, the change to a world state induced by actions is expressed using the fluents that are affected by the action. Now, structured states pay off: atomic states would make the specification of action effects complicated and cumbersome. For example, even the action of putting an item in a virtual shopping cart would have to be expressed separately for each possible cart at each shop! With structured states, however, effect specification is easy: the term $\mathit{PutIntoCart}(\mathit{item}, \mathit{shop})$ will represent the action of putting an item in the virtual cart of an online shop. The meaning of this action function is then given by appropriate formulas stating that the item is added to the respective cart.

These three components form the basis of our ontology underlying dynamic domains: time itself, changing world properties – fluents – and actions that initiate the changes. These aspects are relevant in every domain, further aspects are domain-dependent and extend this fundamental ontology. In the shopping agent domain, for instance, additional important aspects would be shops, items and prices. We will use *sorts* and sorted logic to represent such different ontological classes.

Reasoning about actions offers several established logic-based formalisms for modelling dynamic domains. But all of these traditional representations of dynamic domains model the world using universal rules for which there are no exceptions. For instance, they allow to say that specific shops offer free delivery for all book purchases. If ever an exception to such a rule is observed – a book that is imported from another country and for which a postal fee *is* waived –, the domain model becomes inconsistent and anything can be concluded from it. Although some formalisms do allow the expression of incomplete knowledge via non-deterministic statements like "delivery may or may not be free of charge," a rational reasoning agent always would have to consider the worst case, which effectively renders the statement close to useless.

What is needed is a way to express what is *normally* the case in the domain, like some shops delivering books free of charge *unless stated otherwise*. The following anecdote shows that natural intelligences natively and successfully use such provisional reasoning:[1]

An engineer, a mathematician and a philosopher are travelling by train in the South

[1] More of this type can be found at Norman Foo's web site at `http://www.cse.unsw.edu.au/~norman/JOKES.html`

> Island of New Zealand, home of much more sheep than people. On passing a mountainside full of sheep, they spy one lone black sheep amidst the sea of white sheep.
>
> ENGINEER: Oh, look, there are black sheep in New Zealand!
>
> MATHEMATICIAN: You surely mean there exists at least one black sheep in New Zealand!
>
> PHILOSOPHER: Hmmm ... both of you have drawn an unwarranted conclusion. Surely the only thing we now know is that in New Zealand there is at least one sheep that is black on one side!

The joke works because human intuition agrees mostly with the engineer, while mathematician and philosopher seem overly cautious. But strictly speaking, the philosopher goes out on a limb as well, for it might be that the train's windows are tinted such that the sheep's colour which is perceived as black is actually grey!

So all of them employ assumption-based reasoning of some sort. They all have only incomplete knowledge about the domain of discourse: none knows the colour of all sheep in New Zealand; and they have limited access to new information: it is practically impossible to observe all of New Zealand's sheep at the same time. To cope with their limitations, they jump to conclusions using rules of thumb like "if there is one black sheep, it is typically not the only one" (engineer), "sheep normally have the same colour on both sides" (mathematician, engineer) and "if I perceive an object as black, it usually *is* black" (philosopher, mathematician, engineer).

Discussing the colour of sheep seems to be out of touch with reality, but the human capability of making and withdrawing assumptions is also important in far more mundane situations. Consider the task of cooking a cup of tea. The plan seems simple enough: boil the water, pour it over the tea, done. But in addition to the obvious prerequisites of having access to water, a way to boil it, and tea itself, there are numerous implicit assumptions involved in this plan: For one, there must be water coming out of the tap when turning it on. This is only an assumption since no matter when it was last verified, it could have changed in between. In the same way, electricity is needed to boil the water. How can the cook tell there is or will be no power outage during the process? The list could be continued, but the important point is this: acting like the philosopher, all of these prerequisites would have to be proven to be sure that the plan for cooking a cup of tea will succeed. I suspect however that not even the aloof among philosophers carry out any formal derivations before going about such straightforward tasks.

Now there is of course a caveat in making useful assumptions about the world: it should not go as far as assuming there is a nice, warm cup of tea already on the desk! Although this technically "verifies" that the plan of doing nothing achieves the goal of having a cup of tea, it is of no real use for obvious reasons. Indeed, such an assumption must be considered irrational and unsound. Hence, a good trade-off must be found between soundness and practicality; in terms of the anecdote above, our goal here is to reason as practical as the engineer and still as sound as the philosopher.

Needless to say, making and withdrawing assumptions about dynamic domains is not only of avail for reasoning about sheep or cooking tea, a potential application lies for instance in expert systems for the medical domain: Doctors will at no point have complete knowledge about their patients, since relevant information may be too costly or simply impossible to obtain. Still the doctor can make a diagnosis and prescribe a therapy based on their experience and (implicit) normality assumptions. If a patient later turns out to be among the very few people with, say, hypersensitivity against a prescribed drug, the doctor revises their assumptions about the patient and changes the therapy accordingly.

Such is the reasoning we will investigate in this thesis: we want to make defeasible assumptions about how dynamic worlds usually behave, where defeasible means that we withdraw the assumptions if they are contradicted against. To achieve this goal, we need to combine two mechanisms: one for reasoning about dynamic worlds and how they change on occurrence of actions, and one for making assumptions that can be withdrawn in the light of contrary information. In this thesis, the role of the first mechanism will be played by logical action theories of a particular, yet general form. For the second mechanism, we will use a logic for default reasoning. The main technical challenge is then to combine these two mechanisms such that default assumptions and changing world properties (as well as those that do not change) interact in a well-defined way.

* * *

In the remainder of this chapter, we give a broad overview of the two fields we combine here. (Technical introductions to notions that we use and extend can be found in the next chapter.) In the first section, we review existing logical theories for reasoning about actions and change, since they represent the broadest field we are concerned with. The section thereafter is concerned with logical formalisations of defeasible reasoning. In the last section, we present some historical attempts to combine the two fields.

1.1 Reasoning about Actions

Reasoning about actions and change investigates the logical axiomatisation of dynamic domains. This area of research is among the oldest ones in artificial intelligence: as early as 1959, pioneer John McCarthy envisioned a system he called the "advice taker" [McCarthy, 1959] that would use predicate logic to maintain an internal model of the world and plan ahead its course of actions. While this general idea made intuitive sense, it was soon found out that it entails a multitude of subtle and not-so-subtle problems. Firstly and most importantly, [McCarthy and Hayes, 1969] discovered a fundamental problem that would haunt the reasoning about actions research community for decades to come: the *frame problem*.

Suppose we model an action domain with m actions and n fluents. Typically, each action A_i will only affect a small number $e_i < n$ of fluents, and each fluent F_j will in turn only be affected by a small number $c_j < m$ of actions. Furthermore, when m and n increase, the average e and c will remain about the same. If we represented action effects in a naïve way – by simple statements of the form "fluent F_i is a positive/negative effect of action A_j if formula γ_{ij} holds," "fluent F_i is *no* positive/negative effect of action A_j if formula γ_{ij} holds" – in total we would need $2 \cdot m \cdot n$ such statements of constant size, resulting in an axiomatisation size in $O(m \cdot n)$. However, most of these statements will be superfluous non-effect statements of the second kind. The frame problem now consists of finding a representation of action effects that explicitly expresses only statements of the first kind and implicitly assumes non-effect statements for fluents and actions not mentioned there. Such a representation would only have a size which grows linearly in the order of *n or m*, a considerable improvement from the quadratic $O(m \cdot n)$ for sufficiently large m, n. Additionally, a solution to the frame problem should allow a straightforward integration of new aspects of the domain – McCarthy [McCarthy, 2003] calls this *elaboration tolerance*. For example, the naïve representation with $m \cdot n$ statements is not elaboration tolerant, since for adding an effect we have to find and delete the respective non-effect statement.

From its discovery in 1969, it took more than 20 years until the frame problem was solved in a satisfactory way. [Reiter, 1991] presented a compilation of effect statements to *successor*

state axioms – a fluent-centred representation of action effects – where for each fluent there is a formula that expresses (1) which actions change it under what conditions and how, and (2) that the execution of all other actions will not affect the fluent. This results in n statements of size $O(c)$ and thus a total size in $O(c \cdot n)$. Remarkably, an axiomatiser is only concerned with providing the effect statements, the compilation takes care of representing the non-effects. Thus elaboration tolerance is achieved, since new effects are easily integrated by adding respective effect statements (and recompiling).

But actions and their effects are not the only components of action domains. In the early days of reasoning about actions, *state constraints* were used to rule out certain configurations of the world that the axiomatiser deemed physically impossible. Alas, the interaction of state constraints with actions gives rise to some more representational and inferential problems: Imagine you operate a database for some real-world domain. In reasoning about actions terminology, it represents the domain knowledge at the current time point. Integrity constraints are enforced in the database to assure data coherence, just like state constraints do for each time point in dynamic domains. Now in a perfectly coherent and consistent database state, you perform an update and find that after the update an integrity constraint is violated. This corresponds to executing an action and observing a violated state constraint in the resulting state. Since a database with a violated integrity constraint by definition does not correspond to a real-world state, there must be an error in the database. There are essentially two views of what caused the error (the constraint violation) and what steps can be taken to resolve it.

The first view concludes that the update was incomplete and additional changes have to be made to the database until the integrity constraints are satisfied. In reasoning about actions, this view regards the state constraint in question as inducing additional, indirect action effects, so-called ramifications [Ginsberg and Smith, 1987]. The associated *ramification problem* consists of concisely representing and efficiently inferring such indirect action effects.

The second view of the above problem of a violated integrity constraint concludes that the update itself was erroneous. The tenor there is that an update which leads to the violation of an integrity constraint should never have been performed. In reasoning about actions, this view regards the state constraint in question as inducing additional, implicit action preconditions, so-called qualifications [McCarthy, 1977]. The associated *endogenous qualification problem* consists of figuring out all implicit action preconditions and explicitly representing them, such that constraint violations due to unjustified action application are ruled out.

Observe that a formalism cannot find out syntactically whether a given state constraint will lead to a ramification or qualification [Lin and Reiter, 1994]; also, letting the user classify state constraints into ramification and qualification constraints will not work, since one and the same constraint can act differently for different actions [Thielscher, 1997, Example 10]. Apart from its endogenous aspects caused by state constraints, the qualification problem in its original form [McCarthy, 1977] remains hard even if there are no state constraints in the theory: this *exogenous qualification problem* stems from the fact that it is generally impossible for an axiomatiser to exhaustively enumerate all conditions that might potentially prevent the successful execution of an action. So although the theory predicts applicability of the action, it might fail due to unforeseen (and unforeseeable) circumstances. This is part of the general AI problem and so it is not surprising that – unlike for the frame problem – there are no universally accepted solutions to the ramification and qualification problems. We present our own contributions to this ongoing work in Chapter 5 and Section 6.2.1, respectively.

Our proposed solutions rest on an explicit representation of causality and the time-independent default reasoning framework we develop earlier, both comparably recent developments. Indeed, action formalisms did not have explicit notions of causation and time from the beginning: some early approaches used an implicit notion of time and formalised action effects

by updating sets of formulas or models [Ginsberg and Smith, 1987; Winslett, 1988]. Minimisation of change was enforced regardless of whether change was *caused*. Although [Kautz, 1986; Pearl, 1987] already recognised the importance of causation, it took some years until the concept took hold in action theories: using ideas from model-based diagnosis [Konolige, 1994], researchers started to integrate causality into reasoning about actions formalisms [Brewka and Hertzberg, 1993]. Causation gained momentum in 1995, when three researchers independently figured out its vital importance for solving the ramification problem [Lin, 1995; McCain and Turner, 1995; Thielscher, 1995]. More recently, [Giunchiglia et al., 2004] lifted causation to a semantical principle, which inspired the effect axioms developed in our work.

1.1.1 Situation Calculus

The Situation Calculus originated in [McCarthy, 1963] and [McCarthy and Hayes, 1969], that developed and formalised some of the ideas around the advice taker. The papers introduced the notions of fluents, actions and situations, that would give the name to the formalism.[2] The Situation Calculus differed from alternative approaches for representing change by its explicit notion of time, where situations are first-order objects that can be quantified over. Although this is a standard feature on the present day, various attempts to solve the frame problem did not employ explicit time. In another seminal paper, [McCarthy, 1977] extended this *reification* of time points to fluents and introduced the *Holds* predicate. Essentially, world properties that are usually expressed by predicates are lifted to objects of the universe, which allows an axiomatiser to quantify over them.

Remarkably, the first solution to the frame problem [Reiter, 1991] was formulated in the language of the Situation Calculus. It also introduced unique-names axioms for actions and started to axiomatise situations as sequences of actions, which was later completed by [Pirri and Reiter, 1999]. Their axiomatisation allows to view situations as hypothetical future time points or as histories encoding the actions that led to a time point.

There are several solutions to the ramification problem for the Situation Calculus (among them [Lin, 1995; McIlraith, 2000]). The qualification problem has received somewhat less attention [Lin and Reiter, 1994], partly because its endogenous version can be delegated to the user ("Make sure your action precondition specifications are complete!") and partly because it requires nonmonotonic reasoning.

To date, [Reiter, 2001] remains the most important book about the Situation Calculus. Therein, Reiter's successor state axioms, formalisation of situations and many more of his contributions culminate in the concept of *Basic Action Theories*.

1.1.2 Fluent Calculus

The Fluent Calculus is a close relative of the Situation Calculus and shares with its precursor the branching time structure of situations. It takes reification one step further and not only reifies fluents, but *states*. They are extensionally axiomatised as sets of fluents where intuitively, a state is identified with the fluents that hold in it. The Fluent Calculus originated in the logic-programming based planning formalism of [Hölldobler and Schneeberger, 1990] that first combined the notions state and situation. Its solution to the frame problem is the action-centred dual of the Situation Calculus solution: instead of successor state axioms for each fluent, the Fluent Calculus has state update axioms for each action [Thielscher, 1999]. So for each of m actions, a state update axiom specifies which e_i fluents are affected by the action

[2] Although in McCarthy's original reading, a situation was more like what was later to be called a state, "the complete state of the universe at an instance of time" [McCarthy and Hayes, 1969].

A_i and also encodes that the remaining fluents stay unchanged. This axiomatisation technique of effects consequently leads to a size in $O(m \cdot e)$. Moreover, it does not only solve the representational aspect of the frame problem, but also the inferential aspect: concluding that a fluent did not change during action execution requires no extra amount of work. In fact, all major problems of reasoning about actions that we mentioned above are solved for the Fluent Calculus [Thielscher, 2005b].

1.1.3 Event Calculus

The Event Calculus originated in [Kowalski and Sergot, 1986] as a logic programming-based formalism for reasoning about the dynamics of knowledge bases. It was further developed and logically reformulated in [Shanahan, 1997] and most recently by [Mueller, 2006].

In contrast to the Situation and Fluent Calculus – that use the branching time of situations –, the underlying ontology is based on an explicit notion of time that is independent of action occurrence. Instead of defining a time point as something which is reached by executing an action, time points are stand-alone ontological elements which exist per se without any additional assumptions.

Interdependence of time points, fluents, and actions – which is the crux of any action theory – is catered for by *narratives*. A narrative talks about the domain during a specific time span, about what actions actually happened and what actually holds at various time points.

1.1.4 Action Languages

Action languages are simple declarative languages for describing actions and their effects on the world. They are close to natural languages and thus allow for an intuitive yet concise specification of action domains. As a precursor of action languages, STRIPS [Fikes and Nilsson, 1971] offered simple statements expressing unconditional positive and negative effects. This was later extended to conditional effects in ADL [Pednault, 1989]. The propositional fragment of ADL gave rise to the first action language $\mathcal{A}$ [Gelfond and Lifschitz, 1993]. It was introduced to enhance the traditionally example-oriented mode of operation of reasoning about actions research. This original language $\mathcal{A}$ allowed only for very basic action descriptions, and so was soon extended to $\mathcal{B}$ [Gelfond and Lifschitz, 1998], that handles indirect effects through static laws and $\mathcal{AR}$ [Giunchiglia et al., 1997] that also offers (among other things) nondeterministic effects. A further extension came with $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998] enabling convenient description of concurrency and non-determinism. $\mathcal{C}+$ [Giunchiglia et al., 2004] then provided expressions to talk about causal dependencies between concurrently executed actions. The language $\mathcal{E}$ [Kakas and Miller, 1997] introduced an explicit notion of (linear) time into action languages, that had hitherto only possessed an implicit, branching time structure. The semantics of all those traditional action languages is defined in terms of transition systems – graphs whose nodes represent states (time points) and whose edges represent transitions between states due to actions. An exception for this is the language $\mathcal{K}$ [Eiter et al., 2000], which allows to represent transitions between (possibly incomplete) states of knowledge as opposed to fully specified states of the world.

1.1.5 Unifying Action Calculus

With the plethora of formalisms for reasoning about actions that we have seen so far, we inevitably face the question which of these we should use for combining them with default reasoning in order to achieve our objectives. In particular, we have to think about which

time structure we should use. These are important questions, since the underlying ontological assumptions of different formalisms are likely to affect the generality of their extensions. We do not commit to a particular time structure here, but use a general, abstract calculus that is parametric in its notion of time, the Unifying Action Calculus (UAC). We provide an in-depth presentation of it in the next chapter.

1.2 Nonmonotonic Reasoning

Nonmonotonic reasoning formalises jumping to conclusions in the absence of information to the contrary. The name "nonmonotonic" itself – obviously the negation of "monotonic" – pertains to the absence of the monotonicity property of entailment. In classical logic, a conclusion that can be drawn from a set S of premises can also be drawn from all supersets of S. This is inherent in the definition of classical semantical entailment. It also has profound implications for logic-based knowledge representation: by adding information to a knowledge base, previous conclusions can never be invalidated.

As discussed earlier, human reasoning crucially depends on making and withdrawing assumptions due to the constant addition of information to (constantly) incomplete knowledge bases. This observation was also made by artificial intelligence researchers: [Sandewall, 1972] was one of the first papers that dealt explicitly with nonmonotonic reasoning. Sandewall proposed the *Unless* operator to make statements about what *cannot* be inferred. In his language, a rule $A, Unless\ B \vdash C$ would mean "C can be inferred if A can be inferred and B cannot be inferred."

An approach that was more directly inspired by human reasoning [Minsky, 1974] introduced so-called frames. A *frame* in the social sciences is a mental stance that provides a shortcut to evaluate a situation or an event. In Minsky's formalisation, a frame is a data structure that has a number of *slots*, that represent world properties; each slot has a value that indicates the status of the world property. The slots may be initialised with default values that can later be replaced by new values. For example, the slot *flies* of the frame *bird* may be initialised with the value *default:true* to indicate that birds fly by default.

In terminological languages for representing the relations between classes of objects, [Fahlman, 1979; Fahlman et al., 1981] proposed nonmonotonic inheritance networks. These are graphs whose nodes are classes and objects and where edges express relations between them such as subclass, membership or non-membership. However, nonmonotonic inheritance networks did not have a clearly defined semantics. Their meaning was given only operationally by the software systems operating on them, where different systems would yield different conclusions on the same graph.

Since these early days, nonmonotonic reasoning has undergone rigorous logical formalisation and has produced a great number of frameworks. At the present day, three major formalisms stand out: default logic, circumscription and autoepistemic logic. Default logic can be considered as the most influential nonmonotonic formalism of all. At the time of printing (June 2012), [Reiter, 1980] has amassed 2023 citations,[3] while the respective original papers introducing circumscription [McCarthy, 1980] and autoepistemic logic [Moore, 1985] have been cited 989 and 611 times, respectively.[4]

[3] According to Microsoft Academic Research at `http://academic.research.microsoft.com`. Alternative numbers come from the Thomson Reuters Web of Science at `http://apps.webofknowledge.com` (1227 citations), and Google Scholar at `http://scholar.google.com` (3952) whose figures are however grossly inflated [Jacso, 2009; Beel and Gipp, 2010].

[4] These are Microsoft Academic Research numbers again. Thomson Reuters Web of Science gives 666 and 333, Google Scholar 2137 and 1285.

1.2.1 Default Logic

Reiter's seminal formalism for default reasoning will be used in this thesis to make and withdraw assumptions about dynamic domains. A technical in-depth presentation can be found in the next chapter. Here, we only look at some criticisms of default logic that have emerged over the years.

[Lifschitz, 1990] finds fault with Reiter's treatment of variables in defaults via grounding, and provides an alternative semantics which accesses elements of the universe directly instead of referring to them by ground terms. He concurs that with domain closure there are no problems with open defaults [Lifschitz, 1990, Proposition 4] but calls domain closure "sometimes unacceptable." As our intended usage is concerned, however, the domain closure assumption is mostly acceptable. For practical implementations, assuming a finite domain is often necessary purely for feasibility. But that does not make it overly limiting: in [Pagnucco et al., 2011], we show how to use a designated set of object names to refer to as-of-yet unidentified objects. In any case, the worst that can happen in an axiomatisation where the domain closure assumption was not feasible is that the user gets less default conclusions than they expected.

[Halpern, 1997] attacks the very definition of extensions. Reiter states natural-language properties (D1–D3) that extensions should possess and then goes on to formalise these properties as in Definition 2.8. Halpern now argues that Reiter's extensions are not the only objects that satisfy the properties. He provides alternative characterisations that lead to autoepistemic logic and different notions of "only knowing." Today, the relationships between the different nonmonotonic formalisms are much better understood [Denecker et al., 2003] and Reiter's initial choice has stood the test of time. In particular the groundedness property of default logic extensions is useful as we shall see later on.

1.2.2 Circumscription

Circumscription [McCarthy, 1980] can be used to express that certain objects with a specified property are the *only* objects with this property. It can be equivalently cast as syntactical manipulation in higher-order logic or entailment with respect to a certain class of preferred models. This is where circumscription differs from default logic and autoepistemic logic, that define sets of acceptable beliefs and define conclusions with respect to these sets. [Lifschitz, 1985; Lifschitz, 1986] further extended circumscription to a general and powerful method. Today, it is also used for first-order logic programming in an answer set semantics that does not refer to grounding [Ferraris et al., 2011].

1.2.3 Autoepistemic Logic

[Moore, 1985] strives to formalise an ideally rational agent reasoning about its own beliefs. He uses a belief modality $\mathbf{L}$ to explicitly refer to the agent's belief within the language. Given a set A of formulas (the initial beliefs), a set T is an *expansion* of A if it coincides with the deductive closure of the set $A \cup \{\mathbf{L}P \mid P \in T\} \cup \{\neg\mathbf{L}P \mid P \notin T\}$. In words, T is an expansion if it equals what can be derived using the initial beliefs A and positive and negative introspection with respect to T itself. It was later discovered that this definition of expansions allows unfounded, self-justifying beliefs. Such beliefs are however not always desirable when representing the knowledge of reasoning agents. Nonetheless, autoepistemic logic was a major influence for nonmonotonic logics of belief. For example, it formed the basis of the logic of "only knowing" [Levesque, 1990], which in turn gave rise to the language $\mathcal{ES}_{\mathcal{O}}$ [Lakemeyer and Levesque, 2009].

1.3 Previous Approaches for Combinations

This thesis is by no means the first work to combine reasoning about actions with nonmonotonic reasoning. Most remarkably, the first explicit mention of nonmonotonic reasoning proposed to use it to solve the frame problem [Sandewall, 1972]. In the same spirit, [Reiter, 1980] presented defaults that concluded persistence of world properties to model the commonsense assumption of inertia. Solving the qualification problem of reasoning about actions was among the stated objectives for creating circumscription [McCarthy, 1980] and was later worked out in more detail [McCarthy, 1986]. We stated earlier that the first satisfactory solution to the frame problem was monotonic, so the reader can already guess that most nonmonotonic action theories did not catch on. What is more, [Hanks and McDermott, 1987] even claimed that nonmonotonic reasoning is *generally* unsuited to solve temporal projection problems in reasoning about actions. Although their criticisms were refuted [Baker, 1991], soon [Kartha, 1994] pointed out new problems with nondeterministic effects. When alternative monotonic solutions to the frame problem were found [Thielscher, 1999], the role of default reasoning in action theories was – if at all present – mostly restricted to the qualification problem.

* * *

In the remainder of the thesis, we will show how action theories and default logic can be combined in a novel way. Nonmonotonic reasoning will not be used to solve the frame problem: our solution is monotonic, drawing inspirations from [Reiter, 1991; Thielscher, 1999; Giunchiglia et al., 2004]. Defaults are used to make assumptions about what is normally the case in the domain in question, and to withdraw such assumptions in case of conflicting information.

1.4 Publications

Most of the results in this thesis have been published in refereed conferences and workshops. For reference, we provide pointers here.

- The results of Section 3.2 have been published in [Strass and Thielscher, 2009a] which received an Outstanding Student Paper Award at Commonsense 2009, and in [Strass and Thielscher, 2009b];
- Section 3.3 has appeared in [Strass and Thielscher, 2009c];
- a previous version of Section 3.4 was published in [Baumann et al., 2010];
- Sections 3.5–3.7 have appeared as [Strass, 2011];
- parts of Chapter 4 are contained in [Strass and Thielscher, 2010a];
- a condensed version of Chapter 4 has been published in [Strass, 2012];
- in [Pagnucco et al., 2011] we demonstrated how the system of Section 4.3 can be used to plan with erroneous information;
- a preliminary version of Chapter 5 has been published in [Strass and Thielscher, 2010b];
- first steps towards solving the qualification problem as outlined in Chapter 6 has appeared in [Strass and Thielscher, 2012].

Chapter 2

Preliminaries

This chapter introduces the notation and the fundamental technical concepts we will use throughout the thesis. All defined notions of this thesis are typeset in *italics*, and an index-like list of page references to the definitions can be found at the end of the document.

Mathematical Notation

- $\emptyset$ is the empty set;
- $\{x \mid P(x)\}$ is the set of all x for which $P(x)$ holds;
- $\in, \cup, \cap$ denote set membership, union and intersection;
- $\subseteq, \supseteq$ denote sub- and superset relations (allowing set equality);
- $|S|$ denotes the cardinality of the set S;
- $A \times B$ is the Cartesian product of sets A and B – the set of pairs $\{(a, b) \mid a \in A, b \in B\}$;
- for a set S, the powerset of S (the set of all of its subsets) is denoted by 2^S;
- $f : D_1 \times \cdots \times D_n \to R$ denotes an n-ary function f with domains $D_1, \ldots, D_n$ and range R;
- $\vec{x}$ stands for a sequence $x_1, \ldots, x_n$ of syntactical objects (such as variables or terms);
- $\mathbb{N}_0$ denotes the set of natural numbers including zero;
- the symbol $\stackrel{\text{def}}{=}$ defines a syntactical macro, where the left-hand side is shorthand for the right-hand side.

Many-sorted First-order Logic with Equality

Signatures A *signature* Ξ is a quadruple $(\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ where

- $\mathfrak{S}$ is a non-empty set of *sorts*. It has an associated reflexive, transitive and anti-symmetric relation $\sqsubseteq \subseteq \mathfrak{S} \times \mathfrak{S}$ where for $\mathfrak{s}_1, \mathfrak{s}_2 \in \mathfrak{S}$, $\mathfrak{s}_1 \sqsubseteq \mathfrak{s}_2$ means that $\mathfrak{s}_1$ is a subsort of $\mathfrak{s}_2$.
- $\mathfrak{P}$ is a set of predicate symbols with an associated *arity*, a non-negative integer. To express that an n-ary predicate symbol P has argument sorts $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ we write $P : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n$.

- $\mathfrak{F}$ is a set of function symbols with an associated arity. To indicate that an n-ary function symbol F has argument sorts $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ and range sort $\mathfrak{s}$ we write $F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \to \mathfrak{s}$. In particular, sorted constant symbols will be referred to as $C : \mathfrak{s}$.

- $\mathfrak{X}$ is a family of non-empty sets $\mathfrak{X}_\mathfrak{s}$ of variables, one for each sort $\mathfrak{s} \in \mathfrak{S}$.

When it is clear what we mean from the context, we abbreviate $C : \mathfrak{s} \in \mathfrak{F}$ by $C : \mathfrak{s} \in \Xi$ and do the same for sorts, predicates and variables. A signature is *relational* iff it contains no function symbols of positive arity. An *unsorted signature* contains only a single sort. A signature is *propositional* iff $\mathfrak{P}$ contains only zero-ary predicate symbols. In this case, the choice of $\mathfrak{S}, \mathfrak{F}, \mathfrak{X}$ is immaterial and we will denote the signature by $\mathfrak{P}$.

Terms For a fixed signature, the set of *terms* is the smallest set such that:

- each sorted variable x of sort $\mathfrak{s}$ is a term of sort $\mathfrak{s}$ and

- if $F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \to \mathfrak{s} \in \Xi$ is a function symbol and $\mathfrak{t}_1, \ldots, \mathfrak{t}_n$ are terms of respective sorts $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$, then $F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)$ is a term of sort $\mathfrak{s}$.

So in particular, constant symbols (the case where $n = 0$ above) are terms. The set of *variables of a term* is defined by

$$\begin{aligned} \mathit{Vars}(x) &\stackrel{\text{def}}{=} \{x\} \\ \mathit{Vars}(F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)) &\stackrel{\text{def}}{=} \bigcup_{1 \le i \le n} \mathit{Vars}(\mathfrak{t}_i) \end{aligned}$$

A term $\mathfrak{t}$ is called *ground* iff $\mathit{Vars}(\mathfrak{t}) = \emptyset$. To express that a term $\mathfrak{t}$ over a signature is of sort $\mathfrak{s}$, we write $\mathfrak{t} : \mathfrak{s}$. The *term depth* indicates the maximal nesting of function symbols in a given term. It is defined recursively by

$$\begin{aligned} \mathit{Depth}(\mathfrak{t}) &\stackrel{\text{def}}{=} 0 \text{ if } \mathfrak{t} \text{ is a variable or constant} \\ \mathit{Depth}(F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)) &\stackrel{\text{def}}{=} 1 + \max\{\mathit{Depth}(\mathfrak{t}_1), \ldots, \mathit{Depth}(\mathfrak{t}_n)\} \end{aligned}$$

Formulas For a fixed signature, the set of *formulas* is the smallest set such that

- $\top$ (true) and $\bot$ (false) are formulas,

- if $P : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \in \Xi$ is a predicate symbol and $\mathfrak{t}_1 : \mathfrak{s}_1, \ldots, \mathfrak{t}_n : \mathfrak{s}_n$ are terms then $P(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)$ is a formula,

- if $\mathfrak{t}_1 : \mathfrak{s}$ and $\mathfrak{t}_2 : \mathfrak{s}$ are terms then $\mathfrak{t}_1 = \mathfrak{t}_2$ is a formula,

- if Φ is a formula then $\neg\Phi$ is a formula,

- if Φ_1, Φ_2 are formulas then $\Phi_1 \wedge \Phi_2$ (a conjunction) and $\Phi_1 \vee \Phi_2$ (a disjunction) are formulas,

- if x is a variable of sort $\mathfrak{s}$ and Φ is a formula then $(\forall x : \mathfrak{s})\Phi$ (a universal quantification) and $(\exists x : \mathfrak{s})\Phi$ (an existential quantification) are formulas.

A formula of the form $P(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)$ or $\mathfrak{t}_1 = \mathfrak{t}_2$ is an *atomic formula* or *atom*. A *literal* is an atom (a *positive literal*) or its negation (a *negative literal*). To reverse the polarity of a literal, we use meta-level negation $\overline{\cdot}$, which for an atom P is defined by $\overline{P} \stackrel{\text{def}}{=} \neg P$ and $\overline{\neg P} \stackrel{\text{def}}{=} P$. The notation $|\cdot|$ will be used to extract the affirmative component of a literal, that is, $|\neg P| = |P| = P$. For a set S of literals, let $\overline{S} \stackrel{\text{def}}{=} \left\{\overline{L} \;\middle|\; L \in S\right\}$. A *clause* is a literal (a *unit clause*) or a disjunction of clauses. A clause containing exactly one positive literal is a *definite Horn clause*. A formula is in *conjunctive normal form (CNF)* iff it is a clause or a conjunction of CNFs; Conversely, a conjunctive clause is a unit clause or a conjunction of conjunctive clauses; a formula is in *disjunctive normal form (DNF)* iff it is a conjunctive clause or a disjunction of DNFs. A formula is in *negation normal form (NNF)* iff negation only precedes atoms. Some logical connectives we use are only shorthands, among them

$$\begin{aligned}
\mathfrak{t}_1 \neq \mathfrak{t}_2 &\stackrel{\text{def}}{=} \neg(\mathfrak{t}_1 = \mathfrak{t}_2) \\
\Phi_1 \supset \Phi_2 &\stackrel{\text{def}}{=} \neg\Phi_1 \vee \Phi_2 \\
\Phi_1 \equiv \Phi_2 &\stackrel{\text{def}}{=} (\Phi_1 \supset \Phi_2) \wedge (\Phi_2 \supset \Phi_1) \\
\Phi_1 \not\equiv \Phi_2 &\stackrel{\text{def}}{=} \neg(\Phi_1 \equiv \Phi_2)
\end{aligned}$$

We will not introduce an explicit precedence ordering over the logical connectives, but use parentheses and indentation to clarify the structure of formulas.

The *size of a term* is given by

$$\begin{aligned}
\|x : \mathfrak{s}\| &\stackrel{\text{def}}{=} 1 \\
\|F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)\| &\stackrel{\text{def}}{=} 1 + \sum_{i=1}^{n} \|\mathfrak{t}_i\|
\end{aligned}$$

For a formula Φ, its *size* $\|\Phi\|$ is defined as follows.

$$\begin{aligned}
\|\top\| &\stackrel{\text{def}}{=} 1 \\
\|\bot\| &\stackrel{\text{def}}{=} 1 \\
\|P(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)\| &\stackrel{\text{def}}{=} 1 + \sum_{i=1}^{n} \|\mathfrak{t}_i\| \\
\|\mathfrak{t}_1 = \mathfrak{t}_2\| &\stackrel{\text{def}}{=} 1 + \|\mathfrak{t}_1\| + \|\mathfrak{t}_2\| \\
\|\neg\Phi\| &\stackrel{\text{def}}{=} 1 + \|\Phi\| \\
\|\Phi_1 \circ \Phi_2\| &\stackrel{\text{def}}{=} 1 + \|\Phi_1\| + \|\Phi_2\| \text{ for } \circ \in \{\wedge, \vee\} \\
\|(Qx : \mathfrak{s})\Phi\| &\stackrel{\text{def}}{=} 1 + \|\Phi\| \text{ for } Q \in \{\forall, \exists\}
\end{aligned}$$

The size of a finite set T of formulas is $\|T\| \stackrel{\text{def}}{=} \sum_{\Phi \in T} \|\Phi\|$.

Free variables, sentences The set of *free variables of a formula* is defined as follows.

$$\begin{aligned}
\mathit{FreeVars}(\top) &\stackrel{\text{def}}{=} \emptyset \\
\mathit{FreeVars}(\bot) &\stackrel{\text{def}}{=} \emptyset \\
\mathit{FreeVars}(P(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)) &\stackrel{\text{def}}{=} \bigcup_{1 \leq i \leq n} \mathit{Vars}(\mathfrak{t}_i) \\
\mathit{FreeVars}(\mathfrak{t}_1 = \mathfrak{t}_2) &\stackrel{\text{def}}{=} \mathit{Vars}(\mathfrak{t}_1) \cup \mathit{Vars}(\mathfrak{t}_2) \\
\mathit{FreeVars}(\neg\Phi) &\stackrel{\text{def}}{=} \mathit{FreeVars}(\Phi) \\
\mathit{FreeVars}(\Phi_1 \circ \Phi_2) &\stackrel{\text{def}}{=} \mathit{FreeVars}(\Phi_1) \cup \mathit{FreeVars}(\Phi_2) \text{ for } \circ \in \{\wedge, \vee\} \\
\mathit{FreeVars}((Qx : \mathfrak{s})\Phi) &\stackrel{\text{def}}{=} \mathit{FreeVars}(\Phi) \setminus \{x\} \text{ for } Q \in \{\forall, \exists\}
\end{aligned}$$

A formula Φ is *closed* (or a *sentence*) if $\mathit{FreeVars}(\Phi) = \emptyset$; otherwise, it is *open*. To indicate that Φ is a formula with free variables among $\vec{x}$, we write it as $\Phi(\vec{x})$.

Interpretations A *structure* $\mathfrak{M}$ for a signature Ξ consists of the following.

- For each sort $\mathfrak{s}$, a non-empty set $\mathfrak{D}_\mathfrak{s}$ (the domain of sort $\mathfrak{s}$) such that $\mathfrak{s}_1 \sqsubseteq \mathfrak{s}_2$ implies $\mathfrak{D}_{\mathfrak{s}_1} \subseteq \mathfrak{D}_{\mathfrak{s}_2}$;
- for each function symbol $F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \to \mathfrak{s} \in \Xi$, a function $F^\mathfrak{M} : \mathfrak{D}_{\mathfrak{s}_1} \times \cdots \times \mathfrak{D}_{\mathfrak{s}_n} \to \mathfrak{D}_\mathfrak{s}$;
- for each predicate symbol $P : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \in \Xi$, a function $P^\mathfrak{M} : \mathfrak{D}_{\mathfrak{s}_1} \times \cdots \times \mathfrak{D}_{\mathfrak{s}_n} \to \{\mathsf{t}, \mathsf{f}\}$ that assigns truth values to tuples of domain elements of the right sorts. We will at times specify them by enumerating all the tuples $\vec{\mathfrak{t}}$ such that $P^\mathfrak{M}(\vec{\mathfrak{t}}) = \mathsf{t}$.

A *variable assignment* $\mathfrak{V}$ on a structure is a function from sorted variables $x : \mathfrak{s}$ to their sort's domain $\mathfrak{D}_\mathfrak{s}$. An *interpretation* $\mathfrak{I}$ is a pair $(\mathfrak{M}, \mathfrak{V})$ consisting of a structure and a variable assignment. We denote by $\mathfrak{D}_\mathfrak{s}^\mathfrak{I}$ the domain of sort $\mathfrak{s}$ of the structure of $\mathfrak{I}$. For terms, define

- $x^\mathfrak{I} \stackrel{\text{def}}{=} \mathfrak{V}(x)$ for variables $x : \mathfrak{s}$ and
- $F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n))^\mathfrak{I} \stackrel{\text{def}}{=} F^\mathfrak{M}(\mathfrak{t}_1^\mathfrak{I}, \ldots, \mathfrak{t}_n^\mathfrak{I})$ for compound terms.

For an interpretation $\mathfrak{I} = (\mathfrak{M}, \mathfrak{V})$, a variable $x : \mathfrak{s}$ and an $\mathfrak{x} \in \mathfrak{D}_\mathfrak{s}$, we denote by $\mathfrak{I}, x \mapsto \mathfrak{x}$ the interpretation that is as $\mathfrak{I}$ except that $\mathfrak{V}$ maps x to $\mathfrak{x}$.

Models For a signature Ξ and an interpretation $\mathfrak{I}$ for Ξ, the *satisfaction relation* $\models$ between interpretations and formulas is defined as follows.

- $\mathfrak{I} \models \top$ and $\mathfrak{I} \not\models \bot$
- $\mathfrak{I} \models P(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)$ iff $P^\mathfrak{M}(\mathfrak{t}_1^\mathfrak{I}, \ldots, \mathfrak{t}_n^\mathfrak{I}) = \mathsf{t}$
- $\mathfrak{I} \models \mathfrak{t}_1 = \mathfrak{t}_2$ iff $\mathfrak{t}_1^\mathfrak{I}$ and $\mathfrak{t}_2^\mathfrak{I}$ are identical
- $\mathfrak{I} \models \neg\Phi$ iff $\mathfrak{I} \not\models \Phi$
- $\mathfrak{I} \models \Phi_1 \wedge \Phi_2$ iff both $\mathfrak{I} \models \Phi_1$ and $\mathfrak{I} \models \Phi_2$
- $\mathfrak{I} \models \Phi_1 \vee \Phi_2$ iff one of $\mathfrak{I} \models \Phi_1$ or $\mathfrak{I} \models \Phi_2$
- $\mathfrak{I} \models (\forall x : \mathfrak{s})\Phi$ iff $\mathfrak{I}, x \mapsto \mathfrak{x} \models \Phi$ for each $\mathfrak{x} \in \mathfrak{D}_\mathfrak{s}$
- $\mathfrak{I} \models (\exists x : \mathfrak{s})\Phi$ iff $\mathfrak{I}, x \mapsto \mathfrak{x} \models \Phi$ for some $\mathfrak{x} \in \mathfrak{D}_\mathfrak{s}$

An interpretation $\mathfrak{I}$ is a *model* for a formula Φ if and only if $\mathfrak{I} \models \Phi$. From these definitions, it follows that conjunction $\wedge$ and disjunction $\vee$ are associative and idempotent. We will use these properties to ease reading, for example by using notation such as $\Phi_1 \circ \Phi_2 \circ \ldots \circ \Phi_{n-1} \circ \Phi_n$ for $\circ \in \{\wedge, \vee\}$ with the understanding that placement of parentheses is immaterial.

Entailment A formula Φ is *satisfiable* if there is an interpretation $\mathfrak{I}$ with $\mathfrak{I} \models \Phi$. If $\mathfrak{I} \models \Phi$ for all interpretations $\mathfrak{I}$, then Φ is *valid*. For a set S of formulas, we write $\mathfrak{I} \models S$ iff $\mathfrak{I} \models \Phi$ for all $\Phi \in S$. Then, we say that the set S *entails* the formula Φ and write $S \models \Phi$ iff $\mathfrak{I} \models S$ implies $\mathfrak{I} \models \Phi$ for each interpretation $\mathfrak{I}$. We abbreviate $\emptyset \models \Phi$ by $\models \Phi$. The *deductive closure* (or *theory*) of a set of formulas S is $\mathit{Th}(S) \stackrel{\text{def}}{=} \{\Phi \mid S \models \Phi\}$.

Unification A *substitution* is a mapping from variables to terms. Applying a substitution θ to a term $\mathfrak{t}$ is written in postfix notation $\mathfrak{t}\theta$ and defined by the structural recursion

$$x\theta \stackrel{\text{def}}{=} \theta(x) \text{ for } x \in \mathfrak{X}$$
$$F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)\theta \stackrel{\text{def}}{=} F(\mathfrak{t}_1\theta, \ldots, \mathfrak{t}_n\theta)$$

We will sometimes use anonymous substitutions and write them as sets of pairs $\{x_1 \mapsto \mathfrak{t}_1, \ldots\}$. Two terms $\mathfrak{t}_1, \mathfrak{t}_2$ are *unifiable* if there exists a substitution θ such that $\mathfrak{t}_1\theta = \mathfrak{t}_2\theta$. In this case, θ is a *unifier* of $\mathfrak{t}_1$ and $\mathfrak{t}_2$. For two unifiers θ_1 and θ_2 we say that θ_1 is *more general* than θ_2 if there is a substitution θ_3 such that $\theta_1\theta_3 = \theta_2$. Substitution θ is the *most general unifier* of $\mathfrak{t}_1$ and $\mathfrak{t}_2$ if it is more general than all unifiers of $\mathfrak{t}_1$ and $\mathfrak{t}_2$.

Resolution Let $c_1 = \{k_1, \ldots, k_m\}$ and $c_2 = \{l_1, \ldots, l_n\}$ be literal sets that represent ground clauses. c_1 and c_2 are *resolvable* iff there are $r \in \{1, \ldots, m\}$ and $s \in \{1, \ldots, n\}$ such that for some atom P we have $\{k_r, l_s\} = \{P, \neg P\}$. In this case, the *resolvent of c_1 and c_2* is $Res(c_1, c_2)$, the disjunction of the literals in $\{k_1, \ldots, k_{r-1}, k_{r+1}, \ldots, k_m\} \cup \{l_1, \ldots, l_{s-1}, l_{s+1}, \ldots, l_n\}$. For a set C of literal sets $c_1, \ldots, c_m$ representing a conjunction of ground clauses, C and c' are resolvable iff c_i and c' are resolvable for some $1 \leq i \leq m$; in this case $Res(C, c') \stackrel{\text{def}}{=} \bigwedge_{c_i \text{ and } c' \text{ resolvable}} Res(c_i, c')$.

Equivalences Some equivalences that we will use are

$$\models (\forall x)(\forall y)\Phi(x, y) \equiv (\forall y)(\forall x)\Phi(x, y)$$
$$\models (\forall x)(\Phi_1(x) \wedge \Phi_2(x)) \equiv ((\forall x)\Phi_1(x) \wedge (\forall x)\Phi_2(x))$$
$$\models (\Phi \supset (\Psi_1 \equiv \Psi_2)) \equiv ((\Phi \wedge \Psi_1) \equiv (\Phi \wedge \Psi_2))$$

Unique-names axioms Let $F_1, \ldots, F_n$ be distinct function symbols. The *unique-names axiom for $F_1, \ldots, F_n$* is

$$\bigwedge_{1 \leq i < j \leq n} F_i(\vec{x}) \neq F_j(\vec{y}) \wedge \bigwedge_{1 \leq k \leq n} F_k(\vec{x}) = F_k(\vec{y}) \supset \vec{x} = \vec{y} \tag{2.1}$$

where $\vec{x} = \vec{y}$ for $\vec{x} = x_1, \ldots, x_m$ and $\vec{y} = y_1, \ldots, y_m$ stands for $x_1 = y_1 \wedge \ldots \wedge x_m = y_m$. For a sort $\mathfrak{s}$, the *unique-names axioms for sort $\mathfrak{s}$* are the unique-names axioms for the function symbols of sort $\mathfrak{s}$.

Second-order logic For the specification of the foundational axioms for situations in the next section, we will need second-order logic. Since this is the only time we use second-order logic in this thesis, we do not formally introduce it. We just remark that second-order logic extends the afore introduced first-order logic by object-level variables **P** that range not only over domain elements, but functions and relations over domain elements.

2.1 Unifying Action Calculus

[Thielscher, 2011] proposed a Unifying Action Calculus (UAC) with the objective of bundling research efforts in action formalisms. It does not confine to a particular time structure and can thus be instantiated with situation-based action calculi, like the Situation Calculus or the Fluent Calculus, as well as with formalisms using a linear time structure, like the Event Calculus.

The UAC is based on a finite sorted signature with predefined sorts for the basic building blocks of virtually all action formalisms: time itself, world properties that change over time (fluents), and actions, that initiate these changes.

Definition 2.1. A *domain signature* is a signature which includes the sorts TIME, FLUENT and ACTION along with the predicates

- $<$: TIME $\times$ TIME denoting an ordering of time points,
- *Holds* : FLUENT $\times$ TIME stating whether a fluent evaluates to true at a given time point,
- *Poss* : ACTION $\times$ TIME $\times$ TIME indicating whether an action is applicable for particular starting and ending time points.

The following definition introduces the most important types of formulas of the Unifying Action Calculus: they allow to express properties of states and applicability conditions and effects of actions.

Definition 2.2. Let $\vec{s}$ be a sequence of variables of sort TIME.

- A *state formula* $\Phi[\vec{s}]$ *in* $\vec{s}$ is a first-order formula with free variables $\vec{s}$ where
  - for each occurrence of $Holds(f, s)$ in $\Phi[\vec{s}]$ we have $s \in \vec{s}$ and
  - predicate *Poss* does not occur.

Let s, t be variables of sort TIME and A be a function into sort ACTION.

- A *UAC precondition axiom* is of the form

$$Poss(A(\vec{x}), s, t) \equiv \pi_A[s] \tag{2.2}$$

  where $\pi_A[s]$ is a state formula in s with free variables among $s, t, \vec{x}$.

- An *UAC effect axiom* is of the form

$$Poss(A(\vec{x}), s, t) \supset \Upsilon_1[s, t] \vee \ldots \vee \Upsilon_k[s, t] \tag{2.3}$$

  where $k \geq 1$ and each Υ_i $(1 \leq i \leq k)$ is a formula of the form

$$\begin{aligned}(\exists \vec{y}_i)(\Phi_i[s] \wedge (\forall f)(\Gamma_i^+ \supset Holds(f, t)) \\ \wedge (\forall f)(\Gamma_i^- \supset \neg Holds(f, t)))\end{aligned} \tag{2.4}$$

  in which $\Phi_i[s]$ is a state formula in s with free variables among $s, \vec{x}, \vec{y}$, and both $\Gamma^+[s, t]$ and $\Gamma^-[s, t]$ are state formulas in s, t with free variables among $f, s, t, \vec{x}, \vec{y}$.

Next, we formalise the concept of an (action) domain axiomatisation with its notion of time and action laws.

Definition 2.3. Consider a fixed domain signature. A *UAC domain axiomatisation* is a set of axioms $\Sigma = \Omega \cup \Pi \cup \Upsilon \cup \Sigma_{aux}$, where

- Ω is a finite set of foundational axioms that define the underlying time structure,
- Π is a set of precondition axioms (2.2),
- Υ is a set of effect axioms (2.3),

- Π and Υ contain exactly one axiom for each function into sort ACTION and
- Σ_{aux} is a set of auxiliary axioms containing unique-names axioms for sorts FLUENT and ACTION.

The auxiliary axioms will later be extended for various purposes like action occurrences, default closure axioms and formulas for inferring direct effects.

In this thesis, we will only be concerned with two specific time structures, situations (known from Situation and Fluent Calculus) and the natural numbers (as used by the Event Calculus). The foundational axioms for situations from [Pirri and Reiter, 1999] are below. The first axiom (2.5) says that S_0 has no predecessor and is thus indeed the initial situation. The next axiom (2.6) defines the ordering $<$ on situations, and (2.7) states that Do is injective. Finally, (2.8) allows to prove properties of situations with the help of induction. It says that any property $\mathbf{P}$ (expressed by a second-order predicate variable) that holds for the initial situation S_0 and is preserved through action application (from s to $Do(a,s)$) holds for all situations s'.

$$\neg(s < S_0) \tag{2.5}$$
$$s < Do(a,s') \equiv (s < s' \vee s = s') \tag{2.6}$$
$$Do(a,s) = Do(a',s') \equiv (a = a' \wedge s = s') \tag{2.7}$$
$$(\forall \mathbf{P})((\mathbf{P}(S_0) \wedge (\mathbf{P}(s) \supset \mathbf{P}(Do(a,s)))) \supset \mathbf{P}(s')) \tag{2.8}$$

We abbreviate these axioms by $\Omega_{sit} \stackrel{\text{def}}{=} \{(2.5),(2.6),(2.7),(2.8)\}$. The notation $Do([a_1,\ldots,a_n],s)$ will be used as abbreviation for $Do(a_n,Do(\ldots,Do(a_1,s)\ldots))$.

As is common practice in the Event Calculus literature [Shanahan, 1997; Mueller, 2006], we do not provide an axiomatisation of the natural numbers here, but just assume a standard model of the natural numbers including zero for sort TIME in every interpretation and indicate this by writing Ω_{nat} in place of the foundational axioms. In particular, in the domain signatures of linear time domains, we assume the constants of sort TIME given by $\mathbb{N}_0$.

By abstracting from a specific time structure, the UAC introduces some general notions concerning useful properties of time structures in domain axiomatisations.

Definition 2.4. A domain axiomatisation is *progressing*, if

- $\Omega \models (\exists s : \text{TIME})(\forall t : \text{TIME}) s \leq t$ and
- $\Omega \cup \Pi \models Poss(a,s,t) \supset s < t$.

A domain axiomatisation is *sequential*, if it is progressing and

$$\Omega \cup \Pi \models (Poss(a,s,t) \wedge Poss(a',s',t')) \supset ((t < t' \supset t \leq s') \wedge (t = t' \supset (a = a' \wedge s = s')))$$

That is, a domain axiomatisation is progressing if there exists a least time point and time always increases when applying an action. A sequential domain axiomatisation furthermore requires that no two actions overlap. For progressing domain axiomatisations, we define $Init(t) \stackrel{\text{def}}{=} \neg(\exists s) s < t$. For the two time structures we use in this thesis, it readily follows that they each have a unique initial time point:

$$\Omega_{sit} \models Init(t) \equiv t = S_0 \qquad \text{and} \qquad \Omega_{nat} \models Init(t) \equiv t = 0$$

The definition of *Init* above also works for more general time structures with multiple initial (minimal) time points. We are however only concerned with progressing domain axiomatisations in this thesis.

We next define what it means for a time point to be reachable in an action domain. Intuitively, it means that there is a sequence of actions that leads to the time point when applied in sequence starting in the initial time point. In particular, the initial time point itself is reachable.

Definition 2.5. Let Σ be a domain axiomatisation. A time point τ is called *reachable in* Σ if there exist ground actions $\alpha_1, \ldots, \alpha_n$ and time points $\tau_0, \ldots, \tau_n$ such that

- $\Sigma \models \mathit{Init}(\tau_0)$
- $\Sigma \models \mathit{Poss}(\alpha_i, \tau_{i-1}, \tau_i)$ for all $1 \leq i \leq n$,
- $\tau_n = \tau$.

This notion of reachability coincides with the one from [Reiter, 1993] for the Situation Calculus. Reachability allows us to do induction on reachable time points – to show that a given property P holds for all reachable time points, it suffices to prove the following: (1) P holds for the initial time point; and (2) whenever P holds for a time point s, and an action a is possible from s to t, then P holds for t. We will use this technique in several proofs.

2.2 Default Logic

Default logic [Reiter, 1980] is for closing gaps in incomplete knowledge bases. This is done by *defaults*, that allow to express rules of thumb such as "birds usually fly" and "tools usually work."

Definition 2.6. For a given logical signature, a *default* is any expression of the form

$$\frac{\beta : \kappa_1, \ldots, \kappa_n}{\omega} \tag{2.9}$$

where $\beta, \kappa_1, \ldots, \kappa_n, \omega$ are formulas of the underlying logical signature. β is called the *prerequisite*, the κ_i are the *justifications* and ω is called the *consequent*. A default is

- *justification-free* if $n = 0$,
- *prerequisite-free* if $\beta = \top$,
- *normal* if $n = 1$ and $\kappa_1 = \omega$,
- *supernormal* if it is normal and prerequisite-free,
- *disjunction-free* if β is a conjunction of literals and $\kappa_1, \ldots, \kappa_n, \omega$ are literals,
- *definite Horn* if β is a conjunction of atoms, all the κ_i are negative literals and ω is an atom,
- *open* if it contains free variables, otherwise it is *closed*.

For supernormal defaults, we will usually drop the prerequisite $\top$ and simply write $\frac{:\omega}{\omega}$. In plain text, we will write the default (2.9) as $\beta : \kappa_1, \ldots, \kappa_n / \omega$. In this thesis, we will only be concerned with defaults with at most one justification, but usually give definitions for the general case. To access the consequent of a default, we use $\mathit{Consequent}(\beta : \kappa_1, \ldots, \kappa_n / \omega) \stackrel{\text{def}}{=} \omega$. This generalises to sets of defaults via $\mathit{Consequents}(D) \stackrel{\text{def}}{=} \{\mathit{Consequent}(\delta) \mid \delta \in D\}$.

Definition 2.7. A *default theory* is a pair (W, D), where W is a set of closed formulas and D is a set of defaults. (W, D) is called *closed* if all defaults in D are closed, otherwise it is *open*. All other properties of defaults (e.g. normal) generalise similarly to default theories.

The meaning of default theories is given through the notion of *extensions*. An extension of a default theory (W, D) is "interpreted as an acceptable set of beliefs that one may hold about the incompletely specified world W" [Reiter, 1980]. The following definition is for closed default theories only. When speaking about a set D of open defaults, we take it to mean the set of all ground instances of defaults in D.

Definition 2.8. Let (W, D) be a closed default theory. For any set S of closed formulas let $\Gamma(S)$ be the smallest set satisfying

1. $W \subseteq \Gamma(S)$
2. $Th(\Gamma(S)) = \Gamma(S)$
3. If $\dfrac{\beta : \kappa_1, \ldots, \kappa_n}{\omega} \in D$, $\beta \in \Gamma(S)$ and $\neg\kappa_1, \ldots, \neg\kappa_n \notin S$, then $\omega \in \Gamma(S)$.

A set E of closed formulas is called an *extension for* (W, D) iff $\Gamma(E) = E$.

An alternative characterisation of extensions is given by the following result.

Theorem 2.9 (Theorem 2.1, [Reiter, 1980])**.** *Let (W, D) be a closed default theory and E be a set of closed formulas. Define $E_0 \stackrel{\text{def}}{=} W$ and $E_{i+1} \stackrel{\text{def}}{=} Th(E_i) \cup Consequents(D_i)$ for $i \geq 0$, where*

$$D_i \stackrel{\text{def}}{=} \left\{ \frac{\beta : \kappa_1, \ldots, \kappa_n}{\omega} \;\middle|\; \frac{\beta : \kappa_1, \ldots, \kappa_n}{\omega} \in D \text{ with } \beta \in E_i \text{ and } \neg\kappa_1, \ldots, \neg\kappa_n \notin E \right\}$$

Then E is an extension for (W, D) iff $E = \bigcup_{i=0}^{\infty} E_i$.

The defaults in the D_i are E's *generating defaults* $GD(E) \stackrel{\text{def}}{=} \bigcup_{i=0}^{\infty} D_i$. Throughout the thesis, whenever we refer to extensions E or sets D of defaults with subscripts, we mean the E_i and D_i from the theorem above.

Based on extensions, one can define sceptical and credulous conclusions for default theories: sceptical conclusions are formulas that are contained in every extension, credulous conclusions are those that are contained in at least one extension.

Definition 2.10. Let (W, D) be a normal default theory and Ψ be a formula. We say that Ψ is a *sceptical consequence of* (W, D) and write $(W, D) \mid\!\approx \Psi$ iff Ψ is contained in every extension of (W, D). Similarly, we say that Ψ is a *credulous consequence of* (W, D) and write $(W, D) \mid\!\approx_{cred} \Psi$ iff Ψ is contained in some extension of (W, D).

In this thesis, we will mostly be concerned with sceptical consequences.

At several points we apply a useful result for default theories of a special form.

Theorem 2.11 (Theorem 3.1, [Reiter, 1980])**.** *Every closed normal default theory has an extension.*

2.3 Logic Programming

Definite logic programs Robert Kowalski and others [Kowalski and Kuehner, 1971; Kowalski, 1974; van Emden and Kowalski, 1976] shaped this important formalism for declarative programming and knowledge representation. For a propositional or first-order signature, a *definite logic program rule* is of the form

$$H \leftarrow B_1, \ldots, B_m \tag{2.10}$$

where $H, B_1, \ldots, B_m$ are atoms; H is the *head* and the B_i are the *body* atoms. A definite logic program rule represents the definite Horn clause $H \vee \neg B_1 \vee \ldots \vee \neg B_m$, which could itself be written as the implication $(B_1 \wedge \ldots \wedge B_m) \supset H$. A *definite logic program* Λ is a set of definite logic program rules.

Herbrand interpretations and models For an unsorted first-order signature Ξ, the *Herbrand universe* is the set of all ground terms over Ξ. The *Herbrand base* is the set of all ground atoms over Ξ. An *Herbrand interpretation* is an interpretation whose domain is the Herbrand universe. All the function symbols are interpreted by themselves, hence each Herbrand interpretation can be written as a subset of the Herbrand base. For a definite logic program Λ, we define Herbrand entailment $\Lambda \models_H P$ iff P is true in all Herbrand models of Λ.

Normal logic programs To treat negation in rule bodies in a declarative way, [Gelfond and Lifschitz, 1988] introduced the stable model semantics. For a given signature, a *normal logic program rule* is of the form

$$H \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n$$

where $H, B_1, \ldots, B_m, C_1, \ldots, C_n$ are atoms – $B_1, \ldots, B_m$ are the *positive body atoms* and $C_1, \ldots, C_n$ are the *negative body atoms*. A variable occurs *positively* in a rule if it occurs in a positive body atom. A variable occurs *negatively* in a rule if it occurs in the head or in a negative body atom. A rule is called *safe* if any variable that occurs negatively in the rule also occurs positively. A variable is called *local* if it occurs in the body but not in the head.

A rule is called *ground* if it does not contain variables. A *normal logic program* is a set of normal logic program rules. A normal logic program is *ground* if all its rules are ground.

For a ground normal logic program Λ and a set M of ground atoms, the *Gelfond-Lifschitz reduct* Λ^M is the definite logic program obtained from Λ by

- eliminating each rule whose body contains a literal *not* C with $C \in M$, and
- deleting all literals of the form *not* C from the bodies of the remaining rules.

M is called a *stable model* (or *answer set*) for Λ iff M is the subset-minimal model of Λ^M.

Loops and Loop Formulas [Lin and Zhao, 2004] discovered an interesting relationship between Clark's completion for definite logic programs [Clark, 1978] and the answer set semantics. The relationship focuses on dependencies among program atoms, where *P depends on Q* if there is a program rule with head P such that Q is among the rule's body atoms. Answer set semantics implicitly "checks" for positive cyclic dependencies among program atoms and disallows answer sets that contain unjustified self-referential loops. As Clark completion does not check for cycles, it may allow models that do not correspond to answer sets. [Lin and Zhao,

2004] showed that adding so-called loop formulas for each loop of Λ to the Clark completion, there is a one-to-one correspondence between the models of the resulting propositional theory and the answer sets of Λ.

[Chen et al., 2006] later generalised the notion of a loop for programs with variables to avoid computing essentially the same loops on different ground instantiations of a program. They provided a theoretical result that guarantees the existence of a finite, complete set of loops for programs over relational signatures. In their work, open program clauses are viewed as representative of their ground instances.

Let Λ be a logic program over a relational signature Ξ. The *dependency graph* G_Λ for Λ is the (possibly infinite) graph (V, E) where V is the set of atoms over Ξ and for $\mu, \nu \in V$, we have $(\mu, \nu) \in E$ iff there is a program rule $H \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n \in \Lambda$ and a substitution θ such that $H\theta = \mu$ and $B_i\theta = \nu$ for some $1 \leq i \leq m$. A finite, non-empty subset L of V is called a *first-order loop of* Λ iff for all $\mu, \nu \in L$, there is a directed, non-zero length path from μ to ν in the subgraph of G_Λ induced by L. For two sets of literals L_1, L_2 we say that L_1 *subsumes* L_2 if there is a substitution θ with $L_1\theta = L_2$. A set $\mathfrak{L}$ of loops of a program Λ is *complete*, if for each loop L of Λ there is an $L' \in \mathfrak{L}$ that subsumes L.

Lemma 2.12 (Proposition 9, [Chen et al., 2006])**.** *Let Λ be a normal logic program over a relational signature. Λ has a finite, complete set of loops if no rule in Λ contains local variables.*

Stable models and default theory extensions Soon after the discovery of the stable model semantics, researchers began to investigate the relationship between this new logic programming semantics and the already known nonmonotonic formalisms [Bidoit and Froidevaux, 1987]. It turned out that there exist translations from normal logic programs to default theories such that there is a one-to-one correspondence between stable models and default theory extensions [Marek and Truszczyński, 1989; Bidoit and Froidevaux, 1991]. This means that default logic can be used to specify a meaning for logic programs that coincides with the stable model semantics. Marek and Truszcziński's translation tr_3 and its respective correspondence result below is the basis of the implementation presented in Chapter 4.

Theorem 2.13 (Theorem 3.7, [Marek and Truszczyński, 1989])**.** *Let Λ be a normal logic program containing clauses $r = H \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n$ over a propositional signature $\mathfrak{P}$. Define the* default theory (W_Λ, D_Λ) of the logic program Λ *by*

$$W_\Lambda \stackrel{\text{def}}{=} \{(B_1 \wedge \ldots \wedge B_m) \supset H \mid r \in \Lambda, n = 0\}$$
$$D_\Lambda \stackrel{\text{def}}{=} \left\{ \frac{B_1 \wedge \ldots \wedge B_m : \neg C_1, \ldots, \neg C_n}{H} \;\middle|\; r \in \Lambda, n \neq 0 \right\}$$

A set M of atoms is stable for Λ iff there is an extension E for (W_Λ, D_Λ) such that $M = E \cap \mathfrak{P}$.

The proof of this theorem is quite involved, but mainly based on the fact that both semantics can be recast in a guess-and-check form: for the stable model semantics, that means guessing a model, computing the Gelfond-Lifschitz reduct and checking whether the minimal model of the reduct coincides with the initial guess; for default logic, it means guessing an extension E, constructing the E_i of Theorem 2.9 and checking whether the resulting theory is the initial guess.

2.4 Notational Conventions

A few words on notation and naming conventions:

- SORTS are typeset in small caps
- lower-case letters will denote object-level variables, we use (unless indicated otherwise)
  - $f, f_1, f_2, \ldots$ for sort FLUENT,
  - $a, a_1, a_2, \ldots$ for sort ACTION,
  - $s, t, t_1, t_2, \ldots$ for sort TIME,
  - x, y, z, possibly with indices, for all other sorts
- *Predicates* start with a capital letter and are typeset in italics;
- Fluents, Actions and other function symbols of all sorts start with a capital letter and are typeset in sans serif;
- lower case Greek letters will denote meta-level variables, we use
  - $\varphi, \varphi_1, \varphi_2, \ldots$ for sort FLUENT,
  - $\alpha, \alpha_1, \alpha_2, \ldots$ for sort ACTION,
  - $\rho, \sigma, \tau, \tau_1, \tau_2, \ldots$ for sort TIME,
  - μ, χ, ψ, possibly with indices, for fluent literals,
  - $\theta, \theta_1, \theta_2, \ldots$ for substitutions;
- capital Greek letters are used for a variety of objects:
  - Φ, Ψ are used for formulas,
  - Ξ for signatures,
  - Θ for action domain specifications,
  - Σ for domain axiomatisations with constituents $\Omega, \Pi, \Upsilon, \Sigma_{aux}$ and
  - Λ for logic programs
- capital Fraktur letters are used for components of signatures
  - $\mathfrak{S}$ ... sorts
  - $\mathfrak{P}$ ... predicates
  - $\mathfrak{F}$ ... functions
  - $\mathfrak{X}$ ... variables
- lower case Fraktur letters are used as meta-level variables:
  - $\mathfrak{t}, \mathfrak{t}_1, \mathfrak{t}_2, \ldots$ for (possibly non-ground) terms
  - $\mathfrak{s}, \mathfrak{s}_1, \mathfrak{s}_2, \ldots$ for sorts
  - $\mathfrak{o}, \mathfrak{o}_1, \mathfrak{o}_2, \ldots$ for domain objects
- $\mathfrak{t}_1 \neq \mathfrak{t}_2 \stackrel{\text{def}}{=} \neg(\mathfrak{t}_1 = \mathfrak{t}_2)$ and $\mathfrak{t}_1 \leq \mathfrak{t}_2 \stackrel{\text{def}}{=} \mathfrak{t}_1 < \mathfrak{t}_2 \vee \mathfrak{t}_1 = \mathfrak{t}_2$
- formulas with occurrences of free variables are assumed universally prenex-quantified;
- We collapse consecutive quantifiers, e.g. $(\forall x, y)\Phi$ abbreviates $(\forall x)((\forall y)\Phi)$
- for predicates with an obvious polarity (like *DirT*, *DirF*), we use a neutral version (like *Dir*) with fluent literals L, where $Dir(L, a, s, t)$ denotes $DirF(F, a, s, t)$ if $L = \neg F$ for some fluent F and $DirT(L, a, s, t)$ otherwise.

Chapter 3

Action Default Theories

Action default theories are for default reasoning about actions. They combine the research areas of reasoning about actions – concerned with action theories –, and non-monotonic reasoning – concerned with default theories. Traditional action theories allow to model dynamic domains and make definite predictions about how the domain will behave. Extending this, action default theories allow to model dynamic domains with an additional focus on what usually holds true in the domain. Techniques from nonmonotonic reasoning are employed to make and withdraw normality assumptions in a principled manner.

To allow an end-user to specify a dynamic world, we first define a language for specifying action domains, $\mathcal{D}$.[1] The description language only determines the time-independent aspects of a domain, the time-dependent aspects are added separately. This provides a clean distinction between domain – the general properties of the world –, and instance – a specific starting configuration or a specific narrative. From action domain descriptions in this language, we automatically create action default theories for this domain. We start out with basic defaults that express static normal world properties and later extend this step by step to defaults for dynamic normal world properties. It turns out that it is useful to reify normality into the logical language and that it is necessary to pay special attention to *abnormal* states – states where default assumptions are violated.

3.1 An Action Description Language

In some fields of KR, researchers deal with great numbers of interrelated formal languages and thus began to introduce a nomenclature for naming these languages. For example, in the area of description logics [Baader et al., 2003], the basic language $\mathcal{ALC}$ (for Attributive Language with Complements) extended with transitive roles is abbreviated by $\mathcal{S}$, which in turn gives rise to language names as $\mathcal{SHOIQ}$ via further extensions. In knowledge compilation [Darwiche and Marquis, 2002], researchers investigate properties of propositional negation normal forms, so-called `NNF`s. A formula in this language that is also smooth, deterministic and decomposable is then a member of the language `sd-DNNF`.

In reasoning about actions, there have not been any attempts of introducing a similar nomenclature for description languages. Although there exists a multitude of action languages (see also Section 1.1.4), their names rarely make an indication about language features or

[1] To be precise, we define a whole family of action description languages, but this is only an aside from a user's point of view.

expressiveness. For the languages we are concerned with in this thesis, we use a naming methodology that clarifies language features with respect to defaults and action effects. Constructions we define from these $\mathcal{D}$ languages will always be with respect to a fixed domain signature, but not depending on a specific one – they are parametric in the domain signature.

3.1.1 Syntax

Since our treatment of time should be as general as possible, we first introduce an important syntactical device. It allows us to make statements about the world without making any reference to a time point or time at all.[2] These fluent formulas are just like FOL formulas with atomic formulas replaced by fluent terms. If we want to express that the world property expressed by a fluent formula Φ holds at a time point τ, it is straightforward to create a state formula $\Phi[\tau]$ for precisely that purpose.

Definition 3.1. For a fixed domain signature Ξ, an *atomic fluent formula* is of the form $F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)$, where $F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \to \textsc{fluent} \in \Xi$ is a function symbol and $\mathfrak{t}_1 : \mathfrak{s}_1, \ldots, \mathfrak{t}_n : \mathfrak{s}_n$ are terms. Accordingly, *fluent formulas* are defined inductively like formulas of sorted first-order logic without equality, with the only difference that atomic fluent formulas play the role of atomic formulas. The notion of a *fluent literal* and all literal-related notation carries over.

For a given fluent formula Φ and a term $\tau : \textsc{time}$, the notation $\Phi[\tau]$ is defined as follows.

$$\begin{aligned}
\top[\tau] &\stackrel{\text{def}}{=} \top \\
\bot[\tau] &\stackrel{\text{def}}{=} \bot \\
F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)[\tau] &\stackrel{\text{def}}{=} \mathit{Holds}(F(\mathfrak{t}_1, \ldots, \mathfrak{t}_n), \tau) \\
(\neg\Phi)[\tau] &\stackrel{\text{def}}{=} \neg(\Phi[\tau]) \\
(\Phi_1 \circ \Phi_2)[\tau] &\stackrel{\text{def}}{=} \Phi_1[\tau] \circ \Phi_2[\tau] \text{ for } \circ \in \{\wedge, \vee\} \\
((Qx : \mathfrak{s})\Phi)[\tau] &\stackrel{\text{def}}{=} (Qx : \mathfrak{s})(\Phi[\tau]) \text{ for } Q \in \{\forall, \exists\}
\end{aligned}$$

These fluent formulas are used to make general, time-independent statements about action domains. In particular, the following definition formalises how action preconditions and effects and normality statements about the world can be expressed.

Definition 3.2. Consider a fixed domain signature. Let A be a function into sort ACTION and $\vec{x}$ be a sequence of variables matching A's arity.

- A *precondition law for* $A(\vec{x})$ is of the form

$$\underline{\texttt{possible}}\ A(\vec{x})\ \underline{\texttt{iff}}\ \Phi_A \tag{3.1}$$

  where Φ_A is a fluent formula with free variables in $\vec{x}$. A precondition law is *disjunction-free* iff Φ_A is a conjunction of fluent literals.

- A *direct effect law for* $A(\vec{x})$ is of the form

$$\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \psi_1\ \underline{\texttt{or}}\ \ldots\ \underline{\texttt{or}}\ \psi_n\ \underline{\texttt{if}}\ \Phi \tag{3.2}$$

  where $n \geq 1$, the ψ_i for $1 \leq i \leq n$ (the *effects*) are fluent literals and Φ (the *condition*) is a fluent formula. An effect law is:

  - *deterministic* if $n = 1$, otherwise it is *disjunctive*;

[2] In general logic, this is trivial. In action theories however, one is usually concerned with timed domains.

- an *occlusion for* $A(\vec{x})$ if $n = 2$ and $\psi_1 = \varphi$ and $\psi_2 = \neg\varphi$ for some fluent term φ;
- *local-effect* if all free variables in Φ and the ψ_i are among $\vec{x}$, otherwise it is *global-effect*;
- *unconditional* if $\Phi = \top$, otherwise it is *conditional*. Unconditional direct effect laws will be written as $\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \psi_1\ \underline{\texttt{or}}\ \ldots\ \underline{\texttt{or}}\ \psi_n$.

A *law* is either a precondition law or a direct effect law for some action.

- A *state default* is of the form

$$\underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi \tag{3.3}$$

where ψ (the *consequent*) is a fluent literal and Φ (the *prerequisite*) is a fluent formula.[3] A state default is:

- *disjunction-free* if Φ is a conjunction of fluent literals;
- *atomic* if Φ is a single fluent literal;
- *prerequisite-free* if $\Phi = \top$ and will be written as $\underline{\texttt{normally}}\ \psi$.

We will define additional laws in later chapters; when speaking about laws, we always mean the latest definition. To describe an action domain, all a user now has to do is write laws and state defaults. The laws state how the domain *always* behaves, while the state defaults indicate how the domain *normally* behaves. Note that such a domain description does not explicitly mention time.

Definition 3.3. Consider a fixed domain signature. An *action domain specification (ADS)* Θ – for short, domain – is a finite set of laws and state defaults.

We assume without loss of generality that different effect laws for the same action $A(\vec{x})$ share only variables among $\vec{x}$, and that these constitute the only pairs of elements of action domain specifications that share variables.

Whenever all state defaults of an action domain specification Θ have certain properties, the ADS inherits this property. For example, if all state defaults in Θ are prerequisite-free, we will also call the action domain specification Θ prerequisite-free. The same holds for properties of effect laws, so an unconditional ADS contains only unconditional direct effect laws. An exception will be made for occlusions: in terms of nomenclature, "ADSs with occlusions" refers to ADSs where all direct effect laws are occlusions or deterministic effects. Naturally, we can combine these properties and speak about, say, an action domain specification that is prerequisite-free, unconditional and local-effect. We will use structured prefixes to refer to specific sublanguages of $\mathcal{D}$, where the first part of the prefix classifies the state defaults, the second part classifies the action effects and the prefix parts are separated by a hyphen "-". For example, *p-*$\mathcal{D}$ denotes the set of all prerequisite-free action domain specifications, *-l*$\mathcal{D}$ the set of all local-effect ADSs and *p-l*$\mathcal{D}$ is the intersection of the two. Table 3.1 shows all possible endorsements that create the $\mathcal{D}$ family of action specification languages. So strictly, Definition 3.3 speaks about ADSs for *n-cgs*$\mathcal{D}$.

[3] The notions *prerequisite* and *consequent* have already been defined for defaults. It will however be clear from the context which of the two we refer to.

State defaults	prerequisite-free: p	atomic: a	disjunction-free: d	normal: n
Direct effects	unconditional: u	conditional: c		
	local: l	global: g		
	deterministic: e	with occlusions: o	disjunctive: s	

Table 3.1: The $\mathcal{D}$ family of action description languages. Properties of state defaults will be referenced via the part before the "-" and the part afterwards refers to properties of laws. Expressiveness increases from left to right along the dimension specified by the first column: for example, according to the second row u-$\mathcal{D}$ $\subseteq$ c-$\mathcal{D}$. Although several of the properties are binary, we always use endorsements to avoid confusion.

3.1.2 Semantics

In contrast to transition-system based semantics of existing action languages, the semantics of the $\mathcal{D}$ family will be defined in terms of a translation to default logic. Roughly, the laws of a domain specification will be translated into a UAC domain axiomatisation, and the state defaults will be translated into Reiter defaults. Alas, the domain specification makes no mention of a time structure or world properties at any time point. Indeed, laws and state defaults are wholly independent of a specific notion of time. However, for the translation into a domain axiomatisation to be fully defined this information needs to be added. At the point of translation the user opts for a time structure and according to their choice completes the domain specification: for branching time, they specify the state of the world at the initial situation; for linear time, they specify a narrative consisting of action occurrences and states of the world at various points in time. As specification completeness is concerned, we generally take a user-friendly stance: if an ADS contains no precondition law for some action A, we take it to be $\Phi_A = \top$.

Branching Time: Situations

To specify a branching-time domain axiomatisation, we need the following for each function A into sort ACTION and matching sequence of variables $\vec{x}$:

- a fluent formula Φ_A with free variables among $\vec{x}$, that denotes the necessary and sufficient preconditions for executing $A(\vec{x})$,
- an effect axiom (2.3) and
- a logical axiomatisation Σ_0 of the state of the world in the initial situation S_0.

Since we already have the foundational axioms for situations Ω_{sit} at our perusal, it is straightforward to create a UAC axiomatisation for this domain. Note that the syntactic form of the precondition axioms together with Ω_{sit} entail that these branching-time axiomatisations are always sequential.

Definition 3.4. Let Θ be a $\mathcal{D}$ action domain specification. A *branching-time domain axiomatisation for* Θ is a set of axioms $\Sigma = \Omega_{sit} \cup \Pi \cup \Upsilon \cup \Sigma_0 \cup \Sigma_{aux}$, where

- Ω_{sit} are the foundational axioms for situations,
- Π contains a precondition axiom of the form

$$Poss(A(\vec{x}), s, t) \equiv (\Phi_A[s] \wedge t = Do(A(\vec{x}), s)) \tag{3.4}$$

for each possible $A(\vec{x})$ iff $\Phi_A \in \Theta$,

- Υ contains an effect axiom (2.3) for each function into sort ACTION,
- Σ_0 is a set of state formulas in S_0 characterising the initial situation and
- Σ_{aux} is a set of auxiliary axioms containing unique-names axioms for sorts FLUENT and ACTION.

As we define more and more general forms of domain axiomatisations, we will (ab)use Σ_{aux} to accommodate a variety of additional auxiliary axioms. For brevity, we will only ever state the latest addition, hence unique-names axioms are always part of our domain axiomatisations.

Linear Time

For linear-time domain axiomatisations, the user also needs to provide action preconditions and effect axioms, just as for branching time. Additionally, they specify a narrative – a finite set of action occurrences and world properties at specific time points. To this end, we introduce a new fluent $\mathsf{Happens}(a, s, t)$ expressing occurrence of an action a from time points s to t. The actions of a narrative are now easily written by the user as a sequence of ground fluent atoms

$$\mathsf{Happens}(\alpha_1, \sigma_1, \tau_1), \ldots, \mathsf{Happens}(\alpha_n, \sigma_n, \tau_n) \tag{3.5}$$

which will be translated into an action occurrence axiom in the resulting domain axiomatisation, similar to minimisation of event occurrence via circumscription of the predicate *Happens* in the Event Calculus [Mueller, 2006]. Properties of the world at certain time points in the narrative are specified by a set Σ_N of state formulas. The linear-time precondition axiom as defined below is restricted to time-consuming actions, thus our linear-time domain axiomatisations will always be progressing.

Definition 3.5. Let Θ be a $\mathcal{D}$ action domain specification and extend its domain signature by the (w.l.o.g.) fresh function symbol $\mathsf{Happens} : \text{ACTION} \times \text{TIME} \times \text{TIME} \to \text{FLUENT}$. A *linear-time domain axiomatisation* is a set of axioms $\Sigma = \Omega_{nat} \cup \Pi \cup \Upsilon \cup \Sigma_N \cup \Sigma_{aux}$, where

- Ω_{nat} contains an axiomatisation of the natural numbers,
- Π contains a precondition axiom of the form

$$Poss(A(\vec{x}), s, t) \equiv (\Phi_A[s] \wedge Holds(\mathsf{Happens}(A(\vec{x}), s, t), s) \wedge s < t) \tag{3.6}$$

  for each `possible` $A(\vec{x})$ `iff` $\Phi_A \in \Theta$,
- Υ contains an effect axiom (2.3) for each function into sort ACTION,
- Σ_N contains
  - for a sequence (3.5) an *action occurrence axiom* of the form

$$(\forall a, s, t)(Holds(\mathsf{Happens}(a, s, t), s) \equiv ((a = \alpha_1 \wedge s = \sigma_1 \wedge t = \tau_1) \vee \ldots \vee (a = \alpha_n \wedge s = \sigma_n \wedge t = \tau_n)) \tag{3.7}$$

  - closed state formulas in single time points.
- Σ_{aux} is a set of auxiliary axioms containing unique-names axioms for sorts FLUENT and ACTION.

This specification of narratives makes no implicit persistence assumption for time spans without action occurrence: for example, if the narrative only contains a state formula about 0 and action occurrences from 0 to 1 and 3 to 5, then state properties do not carry over from 1 to 3. If this is desired, the user can explicitly add the occurrence of an action without effects like $\mathsf{Happens}(\mathsf{Wait}, 1, 3)$. Even more interestingly, they can use occlusions to exclude only specific fluents from the frame assumption and let all others persist.

We can see that this compilation/representation of narratives does not allow incomplete knowledge about actions. As we have in mind agents that reason about their own past, incomplete knowledge about their own actions seems however far-fetched. The specification of linear-time domains in the UAC shown here is indeed only one possibility among several, alternative formulations can be found in [Thielscher, 2011].

* * *

We will see examples for domain axiomatisations of both of these specific forms in the remainder of this chapter. We will work our way up from basic action domain descriptions to fairly elaborate and expressive ones. The next section starts out with the most basic language *p-ule*$\mathcal{D}$, where all state defaults are prerequisite-free and the actions have only unconditional, local, deterministic effects. Section 3.3 then introduces state defaults with atomic prerequisites and actions with occlusions, hence deals with the language *a-ulo*$\mathcal{D}$. More general state default prerequisites along with a novel axiomatisation technique are presented in Section 3.4. This axiomatisation technique is then further extended to conditional effects (Section 3.5), global effects (Section 3.6) and disjunctive effects (Section 3.7).

3.2 Basic Action Default Theories

This section is concerned with *p-ule*$\mathcal{D}$ action domain specifications, where all state defaults are prerequisite-free and all actions have only unconditional, local and deterministic effects. As we shall see, the action default theories for these domains have certain nice properties. Most notably, it suffices to instantiate the defaults with the least time point to exhaustively draw all possible intuitive conclusions. Here, intuitive means the conclusions are independent of future time points. We begin with illustrating the language *p-ule*$\mathcal{D}$ using a variant of the well-known Yale Shooting scenario [Hanks and McDermott, 1987].

Example 3.6 (Fred Meets Destiny)**.** Bestowed with an almost Promethean fate, turkey Fred lives only to get shot at, die, and then be resurrected again (since 1987). Let the signature of this domain consist of the actions Load, Wait and Shoot with obvious meanings, fluents Loaded, Broken and Alive saying that the gun is loaded, broken and Fred is alive, respectively. It is possible to shoot the gun if it is loaded and not broken, loading and waiting are always possible. Loading the gun causes it to be loaded, and after shooting it Fred ceases to be alive. Normally, the gun works fine. In $\mathcal{D}$, this domain is specified as follows.

$$\begin{aligned}\Theta_{Yale} = \{&\underline{\texttt{possible}}\ \mathsf{Shoot}\ \underline{\texttt{iff}}\ \mathsf{Loaded} \wedge \neg\mathsf{Broken},\\ &\underline{\texttt{action}}\ \mathsf{Shoot}\ \underline{\texttt{causes}}\ \neg\mathsf{Alive},\\ &\underline{\texttt{action}}\ \mathsf{Load}\ \underline{\texttt{causes}}\ \mathsf{Loaded},\\ &\underline{\texttt{normally}}\ \neg\mathsf{Broken}\}\end{aligned}$$

To reason about this domain, we will translate it into a default theory of a specific syntactic form. The semantics of default logic then specifies the meaning of the domain. At first, we define the effect axioms expressing unconditional, local and deterministic effects.

Definition 3.7. Let Θ be a *p-ule*$\mathcal{D}$ action domain specification. Let A be a function into sort ACTION and $\vec{x}$ be a sequence of variables matching A's arity. An *effect axiom with unconditional effects* is of the form

$$Poss(A(\vec{x}), s, t) \supset (\forall f)(Holds(f, t) \equiv (\gamma_A^+ \vee (Holds(f, s) \wedge \neg\gamma_A^-))) \tag{3.8}$$

where

$$\gamma_A^+ = \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \varphi \in \Theta} f = \varphi \quad \text{and} \quad \gamma_A^- = \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \neg\varphi \in \Theta} f = \varphi$$

This fully automatic way of creating effect axioms from direct effect laws together with the translation of precondition laws seen in the previous section yields a translation of action domain specifications into UAC domain axiomatisations. The following definition extends such a domain axiomatisation with a set of defaults created from state defaults.

Definition 3.8. Let Θ be a *p-ule*$\mathcal{D}$ action domain specification. A *p-ule*$\mathcal{D}$ *action default theory* is a pair $(\Sigma, \Delta[s])$ for some variable s : TIME, where

- Σ is a UAC domain axiomatisation with effect axioms (3.8) and
- $\Delta[s]$ is a set of supernormal defaults, containing
  - a default $\dfrac{: Holds(\varphi, s)}{Holds(\varphi, s)}$ for each state default $\underline{\texttt{normally}}\ \varphi \in \Theta$ and
  - a default $\dfrac{: \neg Holds(\varphi, s)}{\neg Holds(\varphi, s)}$ for each state default $\underline{\texttt{normally}}\ \neg\varphi \in \Theta$, where s : TIME is a variable.

By $\Delta[\sigma]$ we denote the set of defaults in $\Delta[s]$ where s has been instantiated by the term σ.

Example 3.6 (Continued)**.** Let us compile Θ_{Yale} into the branching-time domain axiomatisation $\Sigma_{Yale} = \Omega_{sit} \cup \Pi \cup \Upsilon \cup \Sigma_{aux}$. The precondition law $\underline{\texttt{possible}}$ Shoot $\underline{\texttt{iff}}$ Loaded $\wedge \neg$Broken and the omitted preconditions of Load and Wait determine

$$\begin{aligned}\Pi = \{&Poss(\mathsf{Shoot}, s, t) \equiv (Holds(\mathsf{Loaded}, s) \wedge \neg Holds(\mathsf{Broken}, s) \wedge t = Do(\mathsf{Shoot}, s)),\\ &Poss(\mathsf{Load}, s, t) \equiv t = Do(\mathsf{Load}, s),\\ &Poss(\mathsf{Wait}, s, t) \equiv t = Do(\mathsf{Wait}, s)\}\end{aligned}$$

Translating the direct effect laws is equally easy: all effect axioms in Υ are of the form (3.8), we state only the $\gamma^{\pm}$ different from the empty disjunction:

$$\gamma^-_{\mathsf{Shoot}} = (f = \mathsf{Alive}) \quad \text{and} \quad \gamma^+_{\mathsf{Load}} = (f = \mathsf{Loaded})$$

Waiting has no effects, thus the effect axiom of Wait is equivalent to

$$Poss(\mathsf{Wait}, s, t) \supset (\forall f)(Holds(f, t) \equiv Holds(f, s))$$

Finally, we state that the turkey is alive in the initial situation:

$$\Sigma_0 = \{Holds(\mathsf{Alive}, S_0)\}$$

The state default normally ¬Broken saying the gun works properly yields the set of defaults

$$\Delta[s] = \left\{ \frac{: \neg \mathit{Holds}(\mathsf{Broken}, s)}{\neg \mathit{Holds}(\mathsf{Broken}, s)} \right\}$$

We can now employ sceptical entailment to answer the question whether Fred is still alive after applying the actions Load, Wait, and Shoot, respectively. Observe that without the default assumption, it cannot be concluded that the action Shoot is possible after performing Load and Wait since it cannot be inferred that the gun is not broken. Using the abbreviations $S_1 = \mathit{Do}(\mathsf{Load}, S_0)$, $S_2 = \mathit{Do}(\mathsf{Wait}, S_1)$, and $S_3 = \mathit{Do}(\mathsf{Shoot}, S_2)$, we illustrate how the nonmonotonic entailment relation defined earlier enables us to use the default to draw the desired conclusion:

$$(\Sigma \cup \Sigma_0, \Delta[S_0]) \mid\!\approx \neg \mathit{Holds}(\mathsf{Broken}, S_2) \wedge \mathit{Poss}(\mathsf{Shoot}, S_2, S_3) \wedge \neg \mathit{Holds}(\mathsf{Alive}, S_3)$$

The default conclusion that the gun works correctly, drawn in S_0, carries over to S_2 and allows to conclude applicability of Shoot in S_2 and its effects on S_3.

In the example just seen, default reasoning could be restricted to the initial situation. As it turns out, this is sufficient for the type of action domains considered in this section: effect axiom (3.8) never "removes" information about fluents and thus never makes more defaults active after executing an action. This observation is formalised by the following lemma. It essentially says that to reason about a time point in which an action ends, it makes no difference whether we apply the defaults to the resulting time point or to the time point when the action starts. This holds of course only due to the restricted nature of effect axiom (3.8). For the lemma and all theoretical results in the remainder of this section, by action default theory we mean *p-ule*$\mathcal{D}$ action default theory.

Lemma 3.9. *Let* $(\Sigma, \Delta[s])$ *be a p-ule*$\mathcal{D}$ *action default theory,* α *be a ground action such that* $\Sigma \models \mathit{Poss}(\alpha, \sigma, \tau)$ *for some* σ, τ : TIME*, and let* $\Psi[\tau]$ *be a state formula in* τ*. Then*

$$(\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\tau] \textit{ iff } (\Sigma, \Delta[\tau]) \mid\!\approx \Psi[\tau]$$

Proof. By induction on the structure of state formulas; we restrict our attention to the case where $\Psi[\tau]$ is a fluent literal $\psi[\tau]$. If Σ is inconsistent or $\Sigma \models \psi[\tau]$ the claim is immediate, so for the following we assume $\Sigma \cup \{\neg\psi\}$ is consistent. This immediately rules out a direct effect ψ of α and leaves a default as only option:

$$\begin{aligned}
&(\Sigma, \Delta[\sigma]) \mid\!\approx \psi[\tau] \\
&\text{iff } \psi[\tau] \in E \text{ for each extension } E \text{ for } (\Sigma, \Delta[\sigma]) \\
&\text{iff there is a } \top : \psi[\sigma]/\psi[\sigma] \in \Delta[\sigma] \text{ and no } \top : \overline{\psi}[\sigma]/\overline{\psi}[\sigma] \in \Delta[\sigma] \\
&\text{iff there is a } \top : \psi[s]/\psi[s] \in \Delta[s] \text{ and no } \top : \overline{\psi}[s]/\overline{\psi}[s] \in \Delta[s] \\
&\text{iff there is a } \top : \psi[\tau]/\psi[\tau] \in \Delta[\tau] \text{ and no } \top : \overline{\psi}[\tau]/\overline{\psi}[\tau] \in \Delta[\tau] \\
&\text{iff } \psi[\tau] \in E' \text{ for each extension } E' \text{ for } (\Sigma, \Delta[\tau])
\end{aligned}$$

□

We next introduce a helpful regression operator which is inspired by the one from [Reiter, 1991]. It uses the structure of the effect axioms to reduce reasoning about a time point that is the result of applying an action to reasoning about the time point in which the action started.

Definition 3.10. The operator $\mathcal{R}_\alpha^{\sigma,\tau}$ maps, for a given action α, a state formula in τ into a state formula in σ as follows.

$$\begin{aligned}
\mathcal{R}_\alpha^{\sigma,\tau}(\mathit{Holds}(\varphi,\tau)) &\stackrel{\text{def}}{\equiv} (\gamma_\alpha^+\{f \mapsto \varphi\} \vee (\mathit{Holds}(\varphi,\sigma) \wedge \neg\gamma_\alpha^-\{f \mapsto \varphi\})) \\
\mathcal{R}_\alpha^{\sigma,\tau}(P(\vec{\mathfrak{t}})) &\stackrel{\text{def}}{\equiv} P(\vec{\mathfrak{t}}) \text{ (where } P \text{ is not } \mathit{Holds}) \\
\mathcal{R}_\alpha^{\sigma,\tau}(\neg\Psi) &\stackrel{\text{def}}{\equiv} \neg\mathcal{R}_\alpha^{\sigma,\tau}(\Psi) \\
\mathcal{R}_\alpha^{\sigma,\tau}(\Psi_1 \circ \Psi_2) &\stackrel{\text{def}}{\equiv} \mathcal{R}_\alpha^{\sigma,\tau}(\Psi_1) \circ \mathcal{R}_\alpha^{\sigma,\tau}(\Psi_2) \text{ (where } \circ \in \{\wedge,\vee\}\text{)} \\
\mathcal{R}_\alpha^{\sigma,\tau}((Qx:\mathfrak{s})\Psi) &\stackrel{\text{def}}{\equiv} (Qx:\mathfrak{s})\mathcal{R}_\alpha^{\sigma,\tau}(\Psi) \text{ (where } Q \in \{\forall,\exists\}\text{)}
\end{aligned}$$

Now whenever an action α is possible and its effect axiom is available, a state formula in the resulting time point and its regression are indeed equivalent.

Proposition 3.11. *Let α be a ground term of sort* ACTION *and S be a consistent set of closed formulas that contains an effect axiom (3.8) for action α and where $S \models \mathit{Poss}(\alpha,\sigma,\tau)$ for some σ,τ :* TIME *and let $\Psi[s]$ be a state formula. Then*

$$S \models \Psi[\tau] \equiv \mathcal{R}_\alpha^{\sigma,\tau}(\Psi)[\sigma]$$

Proof. By structural induction on Ψ. The only interesting case is $\Psi = \mathit{Holds}(\varphi,\tau)$ for some fluent φ. Let $\mathfrak{I}$ be a model for S.

$$\begin{aligned}
&\mathfrak{I} \models \mathit{Holds}(\varphi,\tau) \\
\text{iff } &\mathfrak{I} \models \gamma_\alpha^+\{f \mapsto \varphi\} \vee (\mathit{Holds}(\varphi,\sigma) \wedge \neg\gamma_\alpha^-\{f \mapsto \varphi\}) \\
&\text{(since } \mathfrak{I} \models \mathit{Poss}(\alpha,\sigma,\tau) \text{ and } \mathfrak{I} \text{ is a model for } \alpha\text{'s effect axiom)} \\
\text{iff } &\mathfrak{I} \models \mathcal{R}_\alpha^{\sigma,\tau}(\mathit{Holds}(\varphi,\tau)) \text{ (by definition)}
\end{aligned}$$

□

We next define what it means for a time point to be reachable by default in an action default theory. This is an enhancement of the reachability notion from Definition 2.5: where monotonic reachability only considers provably applicable actions, default reachability also takes into account actions that are only executable by default.

Definition 3.12. Let $(\Sigma,\Delta[s])$ be an action default theory. A time point τ is

- *weakly sceptically reachable in* $(\Sigma,\Delta[\sigma])$, if σ is finitely reachable in Σ and for all extensions E for $(\Sigma,\Delta[\sigma])$, τ is reachable in E;
- *strongly sceptically reachable in* $(\Sigma,\Delta[\sigma])$, if σ is finitely reachable in Σ and there exist ground actions $\alpha_1,\ldots,\alpha_n$ and time points $\tau_0,\ldots,\tau_n$ such that
  - $\sigma = \tau_0$,
  - $(\Sigma,\Delta[\sigma]) \mid\!\approx \mathit{Poss}(\alpha_i,\tau_{i-1},\tau_i)$ for all $1 \le i \le n$
  - $\tau_n = \tau$.

For action default theories (Σ,Δ) without explicitly mentioned TIME variable, a time point τ is

- *weakly sceptically reachable in* (Σ,Δ), if τ is reachable in all extensions E for (Σ,Δ);
- *strongly sceptically reachable in* (Σ,Δ), there exist ground actions $\alpha_1,\ldots,\alpha_n$ and time points $\tau_0,\ldots,\tau_n$ such that
  - $\Sigma \models \mathit{Init}(\tau_0)$,

- $(\Sigma, \Delta) \mid\!\approx Poss(\alpha_i, \tau_{i-1}, \tau_i)$ for all $1 \leq i \leq n$
- $\tau_n = \tau$.

While weak sceptical reachability only requires reachability in all extensions, the strong version also needs agreement on the *path* by which the time point is reached. With situations as underlying time structure, weak and strong sceptical reachability coincide. This is because the foundational axioms for situations Ω_{sit} entail that situations have unique predecessors.

For instance, in Θ_{Yale} from Example 3.6, we have the following:

- S_0 is reachable in Σ simply because it is initial,
- $Do(\mathsf{Load}, S_0)$ is reachable in Σ because S_0 is reachable and Load has a trivial precondition law $\underline{\texttt{possible}}$ Load $\underline{\texttt{iff}}$ $\top$,
- $Do(\mathsf{Shoot}, Do(\mathsf{Load}, S_0))$ is not reachable in Σ because, although $Do(\mathsf{Load}, S_0)$ is reachable we have $\underline{\texttt{possible}}$ Shoot $\underline{\texttt{iff}}$ Loaded $\wedge \neg$Broken $\in \Theta_{Yale}$ but $\Sigma \not\models \neg Holds(\mathsf{Broken}, Do(\mathsf{Load}, S_0))$,
- $Do(\mathsf{Shoot}, Do(\mathsf{Load}, S_0))$ is strongly sceptically reachable in $(\Sigma, \Delta[S_0])$ since $Do(\mathsf{Load}, S_0)$ is reachable in Σ and $(\Sigma, \Delta[S_0]) \mid\!\approx Poss(\mathsf{Shoot}, Do(\mathsf{Load}, S_0), Do(\mathsf{Shoot}, Do(\mathsf{Load}, S_0)))$.

The next theorem says that all *local* conclusions about a finitely reachable time point σ (that is, all conclusions about σ using defaults from $\Delta[\sigma]$) are exactly the conclusions about σ that we can draw by instantiating the defaults only with the least time point.

Theorem 3.13. *Let $(\Sigma, \Delta[s])$ be a p-ule$\mathcal{D}$ action default theory, λ its least time point, σ : TIME be finitely reachable in Σ, and $\Psi[\sigma]$ be a state formula. Then*

$$(\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\sigma] \text{ iff } (\Sigma, \Delta[\lambda]) \mid\!\approx \Psi[\sigma]$$

Proof. By induction on σ. The base case is trivial. For the induction step, assume that $\Sigma \models Poss(\alpha, \sigma, \tau)$.

$$\begin{aligned}
&(\Sigma, \Delta[\tau]) \mid\!\approx \Psi[\tau] && \\
\text{iff } &(\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\tau] && \text{(Lemma 3.9)} \\
\text{iff } &(\Sigma, \Delta[\sigma]) \mid\!\approx \mathcal{R}_\alpha^{\sigma,\tau}(\Psi)[\sigma] && \text{(Proposition 3.11)} \\
\text{iff } &(\Sigma, \Delta[\lambda]) \mid\!\approx \mathcal{R}_\alpha^{\sigma,\tau}(\Psi)[\sigma] && \text{(induction hypothesis)} \\
\text{iff } &(\Sigma, \Delta[\lambda]) \mid\!\approx \Psi[\tau] && \text{(Proposition 3.11)}
\end{aligned}$$

□

It thus remains to show that local defaults are indeed exhaustive with respect to local conclusions. The next lemma takes a step into this direction: it states that action application does not increase default knowledge about past time points.

Lemma 3.14. *Let $(\Sigma, \Delta[s])$ be a p-ule$\mathcal{D}$ action default theory, α be a ground action such that $\Sigma \models Poss(\alpha, \sigma, \tau)$ for some σ, τ : TIME, and let $\Psi[\rho]$ be a state formula in ρ : TIME where $\rho \leq \sigma$. Then*

$$(\Sigma, \Delta[\tau]) \mid\!\approx \Psi[\rho] \text{ implies } (\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\rho]$$

Proof. We presume that σ is reachable from ρ in Σ for otherwise the claim is immediate. To show the contrapositive, let $(\Sigma, \Delta[\sigma]) \not\mid\!\approx \Psi[\rho]$. Then there exists an extension E for $(\Sigma, \Delta[\sigma])$ such that $\Psi[\rho] \notin E$. We construct an extension F for $(\Sigma, \Delta[\tau])$ with $\Psi[\rho] \notin F$ as follows. Set $F_0 \stackrel{\text{def}}{=}$

Σ and for $i \geq 0$ let $F_{i+1} \stackrel{\text{def}}{=} Th(F_i) \cup \{\omega[\tau] \mid \top : \omega[\tau]/\omega[\tau] \in \Delta[\tau], \omega[\tau] \in E\}$. (Observe that $\Psi[\rho] \notin E$ and $E = Th(E)$ means that E is consistent and hence $\omega[\tau] \in E$ implies $\neg\omega[\tau] \notin E$.) Now $Consequents(GD(F)) \subseteq E$ and $F = Th(\Sigma \cup Consequent(GD(F)))$ imply that every model for E is a model for F. Hence $\Psi[\rho] \notin F$ and $(\Sigma, \Delta[\tau]) \not\models\!\!\sim \Psi[\rho]$. □

The converse of the lemma does not hold, since an action effect might preclude a default conclusion about the past. Using the above lemma and simple induction on the length of action sequences, one can establish the following.

Theorem 3.15. *Let $(\Sigma, \Delta[s])$ be a p-ule$\mathcal{D}$ action default theory, let $\Psi[s]$ be a state formula, $\sigma \leq \tau$ be time points, and σ be reachable in Σ. Then*

$$(\Sigma, \Delta[\tau]) \models\!\!\sim \Psi[\sigma] \textit{ implies } (\Sigma, \Delta[\sigma]) \models\!\!\sim \Psi[\sigma]$$

Proof. If τ is not reachable from σ in Σ, the result is immediate, so let τ be reachable from σ in Σ. We do induction on the length of the action sequence connecting σ and τ. The base case is obvious. For the induction step assume that $(\Sigma, \Delta[\tau']) \models\!\!\sim \Psi[\sigma]$ implies $(\Sigma, \Delta[\sigma]) \models\!\!\sim \Psi[\sigma]$, $\Sigma \models Poss(\alpha, \tau', \tau)$ and $(\Sigma, \Delta[\tau]) \models\!\!\sim \Psi[\sigma]$. By Lemma 3.14, we get $(\Sigma, \Delta[\tau']) \models\!\!\sim \Psi[\sigma]$; the claim now follows from the induction hypothesis. □

The next theorem, the first main result of this section, now combines Theorems 3.13 and 3.15 and tells us that default instantiation to the least time point subsumes default instantiation in any time point in the future of the time point we want to reason about.

Theorem 3.16. *Let $(\Sigma, \Delta[s])$ be a p-ule$\mathcal{D}$ action default theory, λ be its least time point, $\Psi[s]$ be a state formula, and $\sigma < \tau$ be terms of sort* TIME *where σ is finitely reachable in Σ. Then*

$$(\Sigma, \Delta[\tau]) \models\!\!\sim \Psi[\sigma] \textit{ implies } (\Sigma, \Delta[\lambda]) \models\!\!\sim \Psi[\sigma]$$

Proof. $(\Sigma, \Delta[\tau]) \models\!\!\sim \Psi[\sigma]$ implies $(\Sigma, \Delta[\sigma]) \models\!\!\sim \Psi[\sigma]$ by Theorem 3.15. By Theorem 3.13, this is the case iff $(\Sigma, \Delta[\lambda]) \models\!\!\sim \Psi[\sigma]$. □

What this theorem misses out, however, are time points that are not finitely reachable in Σ only, but where some action application along the way depends crucially on a default conclusion. To illustrate this, recall Example 3.6: the situation $Do([\mathsf{Load}, \mathsf{Wait}, \mathsf{Shoot}], S_0)$ is not reachable in Σ, because the necessary precondition that the gun is not broken cannot be inferred without the respective default.

The following theorem, the second main result of this section, now assures sufficiency of instantiation with the least time point also for time points that are only reachable by default.

Theorem 3.17. *Let $(\Sigma, \Delta[s])$ be a p-ule$\mathcal{D}$ action default theory, λ its least time point, σ be a time point that is reachable in Σ, $\Psi[s]$ be a state formula, and τ be a time point that is strongly sceptically reachable in $(\Sigma, \Delta[\sigma])$.*

1. *τ is strongly sceptically reachable in $(\Sigma, \Delta[\lambda])$.*
2. *$(\Sigma, \Delta[\sigma]) \models\!\!\sim \Psi[\tau]$ iff $(\Sigma, \Delta[\lambda]) \models\!\!\sim \Psi[\tau]$.*

Proof. 1. Obvious: σ is reachable in Σ and τ is strongly sceptically reachable in $(\Sigma, \Delta[\sigma])$.

2. By assumption, there exist ground actions $\alpha_1, \ldots, \alpha_n$ and time points $\tau_0, \ldots, \tau_n$ such that $(\Sigma, \Delta[\sigma]) \models\!\!\sim Poss(\alpha_i, \tau_{i-1}, \tau_i)$ for all $1 \leq i \leq n$, $\tau_0 = \sigma$, and $\tau_n = \tau$. We will now prove by induction (on natural numbers) that $(\Sigma, \Delta[\sigma]) \models\!\!\sim \Psi[\tau_i]$ iff $(\Sigma, \Delta[\lambda]) \models\!\!\sim \Psi[\tau_i]$ for all i; the claim then follows from $\tau_n = \tau$.

$i = 0$: By Theorem 3.13 and $\tau_0 = \sigma$, we have $(\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\tau_0]$ iff $(\Sigma, \Delta[\lambda]) \mid\!\approx \Psi[\tau_0]$.

$i \Rightarrow i+1$: Let $(\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\tau_i]$ iff $(\Sigma, \Delta[\lambda]) \mid\!\approx \Psi[\tau_i]$ (IH) and $\Phi^{\alpha}_{i+1}[s]$ be the right-hand side of α_{i+1}'s precondition axiom. We have

$$\begin{aligned}
&(\Sigma, \Delta[\sigma]) \mid\!\approx \mathit{Poss}(\alpha_{i+1}, \tau_i, \tau_{i+1}) && \text{(presumption)}\\
\text{iff } &(\Sigma, \Delta[\sigma]) \mid\!\approx \Phi^{\alpha}_{i+1}[\tau_i] && \text{(precondition axiom)}\\
\text{iff } &(\Sigma, \Delta[\lambda]) \mid\!\approx \Phi^{\alpha}_{i+1}[\tau_i] && \text{(IH)}\\
\text{iff } &(\Sigma, \Delta[\lambda]) \mid\!\approx \mathit{Poss}(\alpha_{i+1}, \tau_i, \tau_{i+1}) && \text{(precondition axiom)}
\end{aligned}$$

Hence, by effect axiom (3.8), we have $(\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\tau_{i+1}]$ iff $(\Sigma, \Delta[\lambda]) \mid\!\approx \Psi[\tau_{i+1}]$. □

An immediate consequence of this result is that instantiation in the least time point also provides a "maximal" number of reachable time points: default instantiation with a later time point might potentially prevent actions in the least time point, which in turn might render yet another time point unreachable. Furthermore, it is even enough to make default assumptions only about a single time point – the initial state – without losing any of the conclusions.

3.3 Atomic State Defaults

In the previous section, we have used atomic, normal defaults without prerequisites to express static world properties. These properties are assumed once if consistent and then persist over time until supported or refuted by a definite action effect.

But concluding atomic propositions about the world is not always enough. Sometimes we wish to express defaults of the form "in general, x are y" – for example, "in general, paper airplanes fly."[4] Surely, we could instantiate the state default $\underline{\texttt{normally}}\ \mathsf{Flies}(x)$ by all objects x which are known to be paper airplanes. But this is by no means elaboration tolerant [McCarthy, 2003] and furthermore does not account for previously unknown paper airplanes. We would much rather have a state default

$$\underline{\texttt{normally}}\ \mathsf{Flies}(x)\ \underline{\texttt{if}}\ \mathsf{PA}(x)$$

which will let us draw the desired conclusion whenever there is no contradicting information.

In this section, we extend the mechanism of the previous section to state defaults *with* prerequisites. They allow us to specify dynamic defaults, that is, default properties that arise and elapse with changing world features.

First, we show two straightforward generalisations of the approach presented in the previous section. As an example shows, these naïve treatments of default prerequisites allow for unintended default conclusions. To overcome this, we introduce a general, automatic method that is proven to render such conclusions impossible. Finally, we show how the idea behind this method can also be used to specify default effects of simple non-deterministic actions.

The language we are dealing with in this section is *a-uleD*, where the actions are as before and the state defaults are atomic, that is, of the form

$$\underline{\texttt{normally}}\ (\neg)\varphi_2\ \underline{\texttt{if}}\ (\neg)\varphi_1 \tag{3.9}$$

for fluents φ_1, φ_2.

[4] Yes, paper airplanes. Birds are not the only objects that should fly by default.

Naïve Generalisations

In this section, we present two methods of representing state default prerequisites in action default theories and show how these representations clash with our intuitive notion of relevance. The first method represents a state default (3.9) as a supernormal default

$$\frac{: (\neg)\mathit{Holds}(\varphi_1, s) \supset (\neg)\mathit{Holds}(\varphi_2, s)}{(\neg)\mathit{Holds}(\varphi_1, s) \supset (\neg)\mathit{Holds}(\varphi_2, s)}$$

and the second method expresses the same state default as a normal default

$$\frac{(\neg)\mathit{Holds}(\varphi_1, s) : (\neg)\mathit{Holds}(\varphi_2, s)}{(\neg)\mathit{Holds}(\varphi_2, s)}$$

While the second method is arguably more intuitive in terms of causality (it does not let us conclude the contrapositive of the state default), both methods enable default conclusions to be "carried back in time." This clearly disqualifies for solving projection problems, since we would have to take an infinite number of future time points into account.

Example 3.18 (I Fought the Law of Persistence and the Law Won)**.** In the paper airplane domain, the objective is to fold a sheet of paper (fluent SOP) into a paper airplane (fluent PA) and conclude by default that it flies (fluent Flies). The logical formalisation should be able to consistently incorporate the subsequent observation that the paper airplane does not fly. The $\mathcal{D}$ action domain specification is:

$$\begin{aligned}\Theta_{PA} = \{&\underline{\texttt{possible}}\ \mathsf{Fold}(x)\ \underline{\texttt{iff}}\ \mathsf{SOP}(x),\\ &\underline{\texttt{action}}\ \mathsf{Fold}(x)\ \underline{\texttt{causes}}\ \mathsf{PA}(x),\\ &\underline{\texttt{action}}\ \mathsf{Fold}(x)\ \underline{\texttt{causes}}\ \neg\mathsf{SOP}(x),\\ &\underline{\texttt{normally}}\ \mathsf{Flies}(y)\ \underline{\texttt{if}}\ \mathsf{PA}(y)\}\end{aligned}$$

Let the corresponding domain axiomatisation be $\Sigma = \Omega_{sit} \cup \Pi \cup \Upsilon \cup \Sigma_0 \cup \Sigma_{aux}$ according to Definition 3.8 where the initial situation is characterised by $\Sigma_0 = \{\mathit{Holds}(\mathsf{SOP}(\mathsf{P}), S_0)\}$. The rest of the example is the same regardless of which of the two straightforward translations of state defaults into Reiter defaults we use, that is, regardless of whether we employ default

$$\frac{: \mathit{Holds}(\mathsf{PA}(x), s) \supset \mathit{Holds}(\mathsf{Flies}(x), s)}{\mathit{Holds}(\mathsf{PA}(x), s) \supset \mathit{Holds}(\mathsf{Flies}(x), s)} \quad \text{or} \quad \frac{\mathit{Holds}(\mathsf{PA}(x), s) : \mathit{Holds}(\mathsf{Flies}(x), s)}{\mathit{Holds}(\mathsf{Flies}(x), s)}$$

After folding P into a paper airplane (using the abbreviation $S_1 = \mathit{Do}(\mathsf{Fold}(\mathsf{P}), S_0)$), we can indeed make the desired conclusion that it flies:

$$(\Sigma, \Delta[S_1]) \mid\!\approx \mathit{Holds}(\mathsf{Flies}(\mathsf{P}), S_1)$$

So far, so good. But there is another conclusion we can draw in S_1 and that refers to the past:

$$(\Sigma, \Delta[S_1]) \mid\!\approx \mathit{Holds}(\mathsf{Flies}(\mathsf{P}), S_0)$$

Spelled out, the sheet of paper *already flew before it was folded!* Moreover, this conclusion about S_0 could not be drawn in the initial situation itself without utilising a future situation:

$$(\Sigma, \Delta[S_0]) \not\mid\!\approx \mathit{Holds}(\mathsf{Flies}(\mathsf{P}), S_0)$$

This line of argument could be read as: "If I folded the sheet of paper into a paper airplane, it would fly. Therefore, it flies." This is counterfactual reasoning gone awry. So what happened?

The problem stems from effect axiom (3.8) and its incorporated solution to the frame problem: since $\mathsf{Flies}(\mathsf{P})$ holds after $\mathsf{Fold}(\mathsf{P})$ but was not a positive effect of the action, according to the effect axiom it must have held beforehand.

This example shows that state defaults with prerequisites (when expressed in the above way) can have unintended effects in the presence of actions: they are, locally instantiated, not exhaustive with respect to local conclusions. A proposition similar to Theorem 3.15 can thus not be made when using defaults with disjunctive consequents or non-tautological prerequisites. In the following subsections, we will extend our definition of effect axioms to incorporate the more general form of state defaults considered here.

Relaxing the Frame Assumption

We next extend the way actions can influence the truth values of fluents. Up to now we only had positive and negative effects and persistence – the action could make fluents true or false, respectively, or not change them at all. In this subsection, we extend the compilation procedure to *occlusions* (the term first occurred in [Sandewall, 1994]; our usage of occlusions is inspired by this work). They do not fix a truth value for the respective fluents in the resulting time point of the action and thus allow them to fluctuate freely. In particular, it is then impossible to determine an occluded fluent's truth value at the starting time point employing only information about the ending time point. The resulting action description language *a-ulo*$\mathcal{D}$ allows atomic normal defaults and unconditional, local action effects with occlusions.

Definition 3.19. Let Θ be a *a-ulo*$\mathcal{D}$ action domain specification and A be a function into sort ACTION with matching sequence of variables $\vec{x}$. An *effect axiom with unconditional effects and occlusions* is of the form

$$Poss(A(\vec{x}), s, t) \supset (\forall f)(\gamma_A^? \vee (Holds(f, t) \equiv (\gamma_A^+ \vee (Holds(f, s) \wedge \neg\gamma_A^-)))) \tag{3.10}$$

where

$$\gamma_A^+ = \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \varphi \in \Theta} f = \varphi$$

$$\gamma_A^- = \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \neg\varphi \in \Theta} f = \varphi$$

$$\gamma_A^? = \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \varphi\ \underline{\texttt{or}}\ \neg\varphi \in \Theta} f = \varphi$$

For any *p-ule*$\mathcal{D}$ action domain specification Θ, *p-ulo*$\mathcal{D}$ effect axiom (3.10) is logically equivalent to its *p-ule*$\mathcal{D}$ counterpart (3.8). This is easy to see, since Θ contains no occlusions and thus $\gamma_A^? = \bot$.

. . . to Prevent Default Reasoning Backwards in Time

With this new tool of being able to "forget about" fluents, we can now treat the paper airplane domain right.

Example 3.20 (I Fought the Law of Persistence and I Won)**.** We extend Θ_{PA} by the occlusion

$$\underline{\texttt{action}}\ \mathsf{Fold}(x)\ \underline{\texttt{causes}}\ \mathsf{Flies}(x)\ \underline{\texttt{or}}\ \neg\mathsf{Flies}(x)$$

saying that x may or may not fly after folding. The obtained axiomatisation is $\Sigma' = \Omega_{sit} \cup \Pi \cup \Upsilon' \cup \Sigma_0$, where Υ' contains effect axiom (3.10) for the action $\mathsf{Fold}(x)$. We see that the desired conclusion is preserved, and the undesired one is now disabled:

$$(\Sigma, \Delta[S_1]) \mid\!\approx \mathit{Holds}(\mathsf{Flies}(\mathsf{P}), S_1) \quad \text{and} \quad (\Sigma, \Delta[S_1]) \not\mid\!\approx \mathit{Holds}(\mathsf{Flies}(\mathsf{P}), S_0)$$

Specifying the occlusions for the action in the example was easy – there was only one default, and we had a precise understanding of the desired and undesired inferences. In general, however, defaults might interact and it might become less obvious which of them to exclude from the frame assumption.

Definition 3.21 below shows a general method of identifying the fluents that are to be occluded, taking into account given defaults. It takes as input positive and negative effects of an action A and the state defaults of a domain Θ and computes the set of default occlusions of Θ for A. The intuition behind it is simple: it iterates over a set S of fluents potentially influenced by A. This set is initialised with the definite action effects and then extended according to defaults until a fixed point is obtained.

Definition 3.21. Let Θ be an *a-uloD* action domain specification and A be a function into sort ACTION with matching sequence of variables $\vec{x}$. The *occlusion completion of* Θ *for* $A(\vec{x})$ is computed by the following procedure:

1. Set $i := 0$ and $S^0 := \{\psi \mid \underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \psi \in \Theta\}$.
2. Set $S^{i+1} := S^i \cup \{\psi\theta \mid \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \chi \in \Theta \text{ and } \chi\theta \in S^i \text{ for some } \theta\}$ and increment i.
3. Repeat step 2 until a fixed point S^i is obtained.
4. For each fluent $\varphi \in \{|\psi| \mid \psi \in S^i\} \setminus \{|\psi| \mid \psi \in S^0\}$, add to Θ the direct effect law

$$\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \varphi\ \underline{\texttt{or}}\ \neg\varphi$$

The *occlusion completion of* Θ is then the union of the occlusion completions for all functions into sort ACTION in Θ's signature.

Note that prerequisite-free state defaults do not contribute to the computation of occlusions. This is semantically perfectly alright: the intended reading of prerequisite-free defaults is that of static world properties that are once assumed (if consistent) and then persist over time until an action effect either refutes or confirms them.

Example 3.20 (Continued)**.** Applying Definition 3.21 to our running example initialises $S^0 = \{\mathsf{PA}(x), \neg\mathsf{SOP}(x)\}$. The first iteration sets $S^1 = S^0 \cup \{\mathsf{Flies}(x)\}$ due to the state default $\underline{\texttt{normally}}\ \mathsf{Flies}(y)\ \underline{\texttt{if}}\ \mathsf{PA}(y)$ and substitution $\{y \mapsto x\}$. The procedure then computes

$$|\{\mathsf{PA}(x), \neg\mathsf{SOP}(x), \mathsf{Flies}(x)\}| \setminus |\{\mathsf{PA}(x), \neg\mathsf{SOP}(x)\}| = \{\mathsf{Flies}(x)\}$$

in its last step and returns the occlusion $\underline{\texttt{action}}\ \mathsf{Fold}(x)\ \underline{\texttt{causes}}\ \mathsf{Flies}(x)\ \underline{\texttt{or}}\ \neg\mathsf{Flies}(x)$ that we already figured out earlier "by hand."

The following theoretical results of this section apply to *a-ule*$\mathcal{D}$ action domain specifications and their action default theories $(\Sigma, \Delta[s])$ with effect axioms of the form (3.10) where the default occlusions are constructed for each action of Σ as in Definition 3.21. We start out with an observation that follows immediately from the syntactic structure of the effect axioms: any fluent not explicitly mentioned by neither effects nor occlusions is captured by the frame assumption.

Proposition 3.22. *Let σ, τ be time points and α be a ground action such that $\Sigma \models Poss(\alpha, \sigma, \tau)$. For all fluents φ that do not occur as effects in direct effect laws for α, we have*

$$\Sigma \models Holds(\varphi, \sigma) \textit{ iff } \Sigma \models Holds(\varphi, \tau)$$

Proof. By assumption, we have

$$\Sigma \models \gamma_\alpha^? \{f \mapsto \varphi\} \vee (Holds(\varphi, \tau) \equiv (\gamma_\alpha^+ \{f \mapsto \varphi\} \vee (Holds(\varphi, \sigma) \wedge \neg\gamma_\alpha^- \{f \mapsto \varphi\})))$$

and $\gamma_\alpha^? \{f \mapsto \varphi\} \equiv \gamma_\alpha^+ \{f \mapsto \varphi\} \equiv \gamma_\alpha^- \{f \mapsto \varphi\} \equiv \bot$. Combining the two assumptions yields

$$\Sigma \models Holds(\varphi, \tau) \equiv Holds(\varphi, \sigma)$$

which makes the claim obvious. □

Similar to Lemma 3.14, the following lemma says that action application does not increase default knowledge about the past, even in the presence of occlusions. This is intuitively straightforward since occlusions by definition only affect knowledge about the future.

Lemma 3.23. *Let α be a ground action and ρ, σ, τ be terms of sort* TIME *such that $\Sigma \models Poss(\alpha, \sigma, \tau)$ and $\rho \leq \sigma$, and let $\Psi[s]$ be a state formula.*

$$(\Sigma, \Delta[\tau]) \mathrel{|\!\approx} \Psi[\rho] \textit{ implies } (\Sigma, \Delta[\sigma]) \mathrel{|\!\approx} \Psi[\rho]$$

Proof. We first assume that Σ is consistent and ρ is reachable in Σ for otherwise the claim is immediate. We do structural induction on $\Psi[\rho]$ (with the only interesting case $\Psi[\rho] = Holds(\varphi, \rho)$ for a fluent φ) and prove the contrapositive, hence let $(\Sigma, \Delta[\sigma]) \not\mathrel{|\!\approx} Holds(\varphi, \rho)$. Then there exists an extension E for $(\Sigma, \Delta[\sigma])$ with $Holds(\varphi, \rho) \notin E$. We construct an extension F for $(\Sigma, \Delta[\tau])$ as follows. Select a maximal

$$GD(F) \subseteq \{\Phi[\tau] : \psi[\tau] / \psi[\tau] \in \Delta[\tau] \mid E \models \Phi[\tau], E \not\models \neg\psi[\tau]\}$$

such that $\Sigma \cup Consequents(GD(F))$ is consistent. Now assume to the contrary that $Holds(\varphi, \rho) \in F$. We make a case distinction on the reason for that.

1. $\Sigma \models Holds(\varphi, \rho)$. Contradiction to the presumption.
2. There is a supernormal default $\top : Holds(\varphi, \tau) / Holds(\varphi, \tau) \in GD(F)$. Then there is a default $\top : Holds(\varphi, \sigma) / Holds(\varphi, \sigma) \in GD(E)$ in contradiction to $Holds(\varphi, \rho) \notin E$.
3. There is a default with consequent $Holds(\varphi, \tau)$ in $GD(F)$ that is not supernormal. Then by the construction in Definition 3.21 φ is also an occlusion of α. By effect axiom (3.10) we have $Holds(\varphi, \sigma) \notin F$ and hence $Holds(\varphi, \rho) \notin F$, contradiction. □

The absence of unintended inferences about time points connected via a single action then immediately generalises to time points connected via a sequence of actions and trivially generalises to unconnected time points. This is the main result of this section stating the impossibility of undesired default conclusions about the past.

Theorem 3.24. *Let σ, τ be time points such that σ is reachable and $\sigma \leq \tau$.*

$$(\Sigma, \Delta[\tau]) \mid\!\approx \Psi[\sigma] \textit{ implies } (\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\sigma]$$

Proof. If τ is not reachable from σ, then $\Sigma \models \Psi[\sigma]$ and the claim is obvious; so let τ be reachable from σ. We use induction on σ, thus let $(\Sigma, \Delta[\tau]) \mid\!\approx \Psi[\sigma]$. The base case, $\sigma = \tau$, is trivial. For the induction step, assume $\Sigma \models \mathit{Poss}(\alpha, \tau', \tau)$ for some α and τ', and $(\Sigma, \Delta[\tau']) \mid\!\approx \Psi[\sigma]$ implies $(\Sigma, \Delta[\sigma]) \mid\!\approx \Psi[\sigma]$ (IH). By assumption $(\Sigma, \Delta[\tau]) \mid\!\approx \Psi[\sigma]$ and Lemma 3.23, we have $(\Sigma, \Delta[\tau']) \mid\!\approx \psi[\sigma]$. This enables the induction hypothesis to conclude the proof of the claim. □

Another noteworthy property of the presented default reasoning mechanism is the preservation of default conclusions: even if the prerequisite of a default is invalidated due to a contradicting action effect, the associated consequent (if not also contradicted) stays intact. This means the definition does not occlude unnecessarily many fluents. It would be fairly easy to modify Definition 3.21 such that the resulting effect axioms also "forget" default conclusions whose generating rules have become inapplicable – we would just have to replace all occurrences of literals by their respective affirmative component.

. . . to Model Default Effects of Actions

The usage of occlusions as advocated up to this point is of course not the only way to make use of this concept. When we allow the user to specify them along with action effects in *a-uloD* domains instead of computing them automatically, occlusions are an excellent means of modelling default effects of simple non-deterministic actions:

Example 3.25 (Tales of Heads and Tails)**.** We model the action of tossing a coin via excluding the fluent Heads (whose intention is to denote whether heads is showing upwards) from the action Toss's frame axiom. However, the coin of this example is unbalanced and has a strong tendency towards landing with heads facing upwards. This is modelled by having a state default that states the result Heads as usual outcome. There is another action, Wait, that is always possible and does not change the truth value of any fluent.

$$\begin{aligned}\Theta_{\mathit{Coin}} = \{&\underline{\texttt{action}}\ \mathsf{Toss}\ \underline{\texttt{causes}}\ \mathsf{Heads}\ \underline{\texttt{or}}\ \neg\mathsf{Heads},\\ &\underline{\texttt{normally}}\ \mathsf{Heads}\}\end{aligned}$$

Using the branching-time domain axiomatisation Σ containing effect axioms (3.10) for Toss and Wait and the observation $\Sigma_O = \{\neg \mathit{Holds}(\mathsf{Heads}, \mathit{Do}(\mathsf{Toss}, S_0))\}$ we can draw the conclusion

$$\Sigma \cup \Sigma_O \models \neg \mathit{Holds}(\mathsf{Heads}, \mathit{Do}(\mathsf{Wait}, \mathit{Do}(\mathsf{Toss}, S_0))) \tag{3.11}$$

which shows that the observation "the outcome of tossing was tails" persists during Wait, that is, the fluent Heads does not change its truth value during an "irrelevant" action. Tossing the coin again (which results in situation $S_3 = \mathit{Do}(\mathsf{Toss}, \mathit{Do}(\mathsf{Wait}, \mathit{Do}(\mathsf{Toss}, S_0)))$), this time without an observation about the outcome, the state default can be applied and yields the normal result regardless of previous observations:

$$(\Sigma \cup \Sigma_O, \Delta[S_3]) \mid\!\approx \mathit{Holds}(\mathsf{Heads}, S_3)$$

Hence, Definition 3.21 can also be used to complete a user-specified set of occlusions regarding potential default effects of actions. When trying to achieve the above behaviour without specifying the occlusions manually, that is, using a procedure in the spirit of Definition 3.21 that takes as input only definite effects and defaults, one is unlikely to succeed:

automatically creating occlusions for all prerequisite-free defaults will cause all these defaults to apply after every action. In the example above, the coin would then magically flip its side (into the default state Heads) after Wait in $Do(\mathsf{Toss}, S_0)$. We could not infer (3.11) contrary to our intuition that Wait has no effects.

3.4 Normal Defaults

In the previous section, defaults had only atomic prerequisites. We used a syntax-based algorithm to exclude all fluents from the frame assumption that might become a default effect of an action. This simple algorithm however is not easily adapted to more general default prerequisites. For example, even for the simple case of a state default $\underline{\texttt{normally}}\ H\ \underline{\texttt{if}}\ F \wedge G$ and an action that makes F true, the question whether to exclude H from the frame assumption cannot be answered statically, that is, without knowing the status of G.[5]

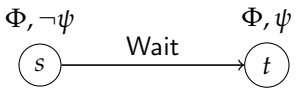
In this section, we deal with the problem on a semantical level. Becoming true by default will be reified and treated by a special predicate. To incorporate the predicate into the effect axiom, we will use a specific axiomatisation technique that implements the principle of universal causation. These new effect axioms easily allow the addition of new causes and will form the basis not only for the action default theories in the rest of this chapter, but also for our solution to the ramification problem in Chapter 5.

The action description language we deal with in this section is *n-ule*$\mathcal{D}$ – normal state defaults and unconditional, local and deterministic effects. In the remainder, we first motivate our choice of default prerequisites; we present the effect axiom and how we integrate default effects into it; finally, we compare the new axiomatisation technique to the ones from the previous sections.

3.4.1 The Prerequisite

Devising a suitable Reiter-default prerequisite from a user-given state default is a delicate task. Often, an appeal to human intuition is the only sensible justification we can give for writing a default one way or the other. In the following, we consider several possible default prerequisites for the state default $\delta = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi$ and its application during an action happening from s to t.

Straightforward The first prerequisite that comes to mind is the straightforward $\Phi[t]$, that just checks whether the state default's prerequisite holds at the resulting time point t. But with this prerequisite, the defaults are too eagerly applied, which is unintended in particular during dummy actions:

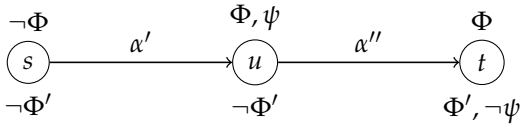


Assume that the default is violated at the starting time point, that is, we have $\Phi[s] \wedge \neg\psi[s]$. Now we apply an action that – from a designer's perspective – does not interfere with fluents mentioned in Φ and ψ. It is precisely this non-interference that lets $\Phi[s]$ persist and causes $\Phi[t]$. This in turn makes the default applicable and concludes $\psi[t]$ in contrast to $\neg\psi[s]$ and the action not interfering with ψ!

[5]Of course, we could be cautious and occlude H anyway, but this generally leads to a loss of knowledge rather than a gain.

Requiring change Apparently, part of the problem with the straightforward prerequisite was caused by the fact that Φ did not change from s to t, yet ψ did. So an easy fix might be to require that Φ change during the action by writing the prerequisite $\neg\Phi[s] \wedge \Phi[t]$. The one obvious flaw of the straightforward prerequisite is gone, but another one awaits. Assume there is a second state default $\delta' = \underline{\texttt{normally}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi'$ with an opposite conclusion. If both prerequisites become true at the same time – $\neg\Phi[s] \wedge \neg\Phi'[s]$ and $\Phi[t] \wedge \Phi'[t]$ – the defaults are in conflict and we get two extensions:

This is perfectly acceptable, in this case we cannot settle for one default conclusion or the other. But now imagine that the change from $\neg\Phi'$ to Φ' is delayed by one (irrelevant) action, and we get the following picture.



At first, $\neg\Phi[s] \wedge \neg\Phi'[s]$. Then an action occurs from s to u that affects only Φ, hence $\Phi[u] \wedge \psi[u] \wedge \neg\Phi'[u]$. Then another action happens that affects only Φ', and we have $\Phi[t] \wedge \Phi'[t] \wedge \neg\psi[t]$. Intuitively, both state defaults are applicable in this state, yet only one of them is applied. It is preferred simply because it happens to have been made applicable later in time. Swapping the order in which the two actions occurred, we would draw the opposite conclusion. This is clearly unintended – we would rather preserve the intuitive conflict between the state defaults irrespective of whether they have been made applicable at the same time or in some arbitrary order.

There may be cases where different orders of action application *should* influence the default conclusions. This behaviour can be emulated in the approach we chose. The converse is not possible: prerequisites that only require change cannot emulate irrelevance of action order.

Checking for violation If we require a change in the truth of the state default's prerequisite to be able to make the default conclusion, the consequent is essentially inferred once and then persists until there is evidence to the contrary. This evidence might as well be a conflicting default conclusion, thus leading to undesired temporal minimisation as seen above.

In this thesis, we have settled for a prerequisite that checks (1) that the state default's prerequisite holds at the resulting time point and (2) that the state default was not violated at the starting time point of the action. By default violation, we mean that the prerequisite of a state default is known to be met, yet the opposite of the consequent prevails. We use the macro $\mathit{Violated}_\delta(s) = \Phi[s] \wedge \neg\psi[s]$ to indicate this for state default δ. The prerequisite is now $\Phi[t] \wedge \neg\mathit{Violated}_\delta(s)$, and the default conclusion is re-drawn as long as it was previously not violated. In the example of the previous paragraph, we get the desired two extensions and thereby eliminate the temporal minimisation-like behaviour:

In both extensions, the state default $\delta = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi$ is applied from s to u, while $\delta' = \underline{\texttt{normally}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi'$ stays inapplicable. At this time point, neither state default is violated. Between u and t, δ' becomes applicable, leading to two conflicting defaults being applicable at the same time and thus to two extensions. Later, we will extend this prerequisite to cope with conflicts arising from conditional and indirect effects.

* * *

Getting the default prerequisite right is necessary, but not sufficient to get the overall behaviour right. Indeed, the above reflections are of course dependent on the rest of the logical theory. The next part of our axiomatisation that needs careful attention is the effect axiom.

3.4.2 The Effect Axiom

An axiom of the form we use here was first presented in a particular example scenario of [Thielscher, 2011] and is, as mentioned there, inspired by the work of [Giunchiglia et al., 2004]. It formalises the idea of truth by causation: everything that is true must be caused, and vice versa. In the most simple form of the effect axiom, we allow two causes to determine a fluent's truth value: persistence and direct effects. Before introducing the axiom itself, we formalise the individual causes. For the first cause – persistence – we introduce a pair of macros expressing that a fluent f persists from s to t.

$$FrameT(f,s,t) \stackrel{\text{def}}{\equiv} Holds(f,s) \wedge Holds(f,t) \tag{3.12}$$

$$FrameF(f,s,t) \stackrel{\text{def}}{\equiv} \neg Holds(f,s) \wedge \neg Holds(f,t) \tag{3.13}$$

For the second cause – a fluent being a direct effect of an action – we introduce two new predicates $DirT(f,a,s,t)$ and $DirF(f,a,s,t)$ expressing that a fluent f is a direct positive/negative effect of an action a from s to t. When the direct (positive and negative) effects of an action are given by unconditional direct effect laws, they can easily be translated into formulas that determine the truth values of all relevant *DirT* and *DirF* atoms.

Definition 3.26. Let Θ be a *n-ule*$\mathcal{D}$ action domain specification and A be a function into sort ACTION with matching sequence of variables $\vec{x}$. The *direct positive and negative effect formulas for* $A(\vec{x})$ are

$$DirT(f, A(\vec{x}), s, t) \equiv \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \varphi \in \Theta} f = \varphi \tag{3.14}$$

$$DirF(f, A(\vec{x}), s, t) \equiv \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \neg\varphi \in \Theta} f = \varphi \tag{3.15}$$

In fact, the truth values of *DirT* and *DirF* atoms will always be fully determined for unconditional effects. We should remark that we did not introduce these predicates in [Baumann et al., 2010], but used macros with the same name instead. The two variants are logically equivalent (modulo truth values for the newly introduced predicates). When we introduce conditional effects in the next section, it will become apparent why we have opted for the predicate instead of the macro variant. The direct effect formulas that determine *DirT* and *DirF*

will later be redefined twice; we will understand the above definition to be retrofitted with their latest version.

Now taking the two causes "persistence" and "direct effect" and putting them together yields the basic version of this section's effect axiom.

Definition 3.27. Let Θ be an *n-ule$\mathcal{D}$* action domain specification and A be a function into sort ACTION. An *effect axiom with unconditional effects and the frame assumption* is of the form

$$\begin{aligned} Poss(A(\vec{x}),s,t) \supset\ & (\forall f)(Holds(f,t) \equiv CausedT(f,A(\vec{x}),s,t)) \wedge \\ & (\forall f)(\neg Holds(f,t) \equiv CausedF(f,A(\vec{x}),s,t)) \end{aligned} \tag{3.16}$$

where

$$CausedT(f,A(\vec{x}),s,t) \stackrel{\text{def}}{\equiv} FrameT(f,s,t) \vee DirT(f,A(\vec{x}),s,t) \tag{3.17}$$

$$CausedF(f,A(\vec{x}),s,t) \stackrel{\text{def}}{\equiv} FrameF(f,s,t) \vee DirF(f,A(\vec{x}),s,t) \tag{3.18}$$

The macros $CausedT, CausedF$ will be re-defined several times throughout the rest of this chapter. When speaking about effect axiom (3.16), we will understand it retrofitted with their latest version.[6]

The design principle underlying our axiomatisation technique is that of causation: a fluent holds at a time point that is the end point of an action if and only if there is a cause for that; similarly, a fluent does not hold if and only if there is a cause for that, too.

Definition 3.28. Let Θ be a *n-ule$\mathcal{D}$* action domain specification. The corresponding *n-ule$\mathcal{D}$ domain axiomatisation* Σ has the following properties:

- all effect axioms in Σ are of the form (3.16) and
- Σ_{aux} contains the direct positive and negative effect formulas (3.14, 3.15) for each function into sort ACTION of Θ's domain signature.

As usual, an example will demonstrate how the definition works.

Example 3.29 (Rowboat Robot)**.** The missionaries and cannibals puzzle [McCarthy, 2003] involves crossing a river in a rowboat, which we model here in isolation. The fluent $\mathsf{Usable}(x)$ shall say that a tool x is usable; fluent $\mathsf{At}(x)$ shall express that the agent is at a location x, which is either Left or Right – the left and right banks of the river.

$$\begin{aligned} \Theta_{Boat} = \{ & \underline{\texttt{possible}}\ \mathsf{CrossRL}\ \underline{\texttt{iff}}\ \mathsf{At}(\mathsf{Right}) \wedge \mathsf{Usable}(\mathsf{Boat}), \\ & \underline{\texttt{action}}\ \mathsf{CrossRL}\ \underline{\texttt{causes}}\ \mathsf{At}(\mathsf{Left}), \\ & \underline{\texttt{action}}\ \mathsf{CrossRL}\ \underline{\texttt{causes}}\ \neg\mathsf{At}(\mathsf{Right})\} \end{aligned}$$

For the action CrossRL of crossing from the right to the left bank, the precondition law says that this is possible whenever the agent is at the right bank and the boat is usable. The effects express that after crossing, the agent is at the left and not at the right bank any more. Applying Definition 3.26 above creates the direct effect formulas

$$DirT(f,\mathsf{CrossRL},s,t) \equiv f = \mathsf{At}(\mathsf{Left})$$
$$DirF(f,\mathsf{CrossRL},s,t) \equiv f = \mathsf{At}(\mathsf{Right})$$

[6] The attentive reader will have noticed that the syntax of axiom (3.16) does not quite correspond to (2.3). Simple syntactical manipulations can however be conducted to transform the effect axiom into a form that matches the structure of (2.3).

Irrespective of the underlying time structure, these direct effect formulas along with effect axiom (3.16) entail that after crossing, the agent is at the right bank and not at the left any more.

$$\Sigma_{Boat} \models Poss(\mathsf{CrossRL}, t_1, t_2) \supset (Holds(\mathsf{At}(\mathsf{Right}), t_2) \land \neg Holds(\mathsf{At}(\mathsf{Left}), t_2))$$

The first formal result about our effect axiom shows that it correctly establishes action effects while still providing a solution to the frame problem.

Proposition 3.30. *Let Θ be an n-ule$\mathcal{D}$ action domain specification and Σ its corresponding domain axiomatisation. Assume $\Sigma \models Poss(\alpha, \sigma, \tau)$ for some ground action $\alpha = A(\vec{o})$ and time points σ, τ.*

1. *Direct effects override persistence:* `action` $A(\vec{x})$ `causes` $\psi \in \Theta$ *implies* $\Sigma \models \psi[\tau]\{\vec{x} \mapsto \vec{o}\}$.
2. *The frame assumption is correctly implemented: if the fluent function of a ground fluent literal ψ is not mentioned as an effect in any direct effect law for $A(\vec{x})$, we have $\Sigma \models \psi[\sigma] \equiv \psi[\tau]$.*

Proof.

1. By Definition 3.26, $Dir(\psi, \alpha, \sigma, \tau) \equiv |\psi| = |\psi| \lor \ldots$, hence $Caused(\psi, \alpha, \sigma, \tau) \equiv |\psi| = |\psi| \lor \ldots$. By axiom (3.16) and the assumption $\Sigma \models Poss(\alpha, \sigma, \tau)$, we have $\Sigma \models \psi[\tau]$.
2. $DirT(|\psi|, \alpha, \sigma, \tau) \equiv \bot$ and $DirF(|\psi|, \alpha, \sigma, \tau) \equiv \bot$ due to the assumption. Hence, by expanding macros (3.17) and (3.18), we get $CausedT(|\psi|, \alpha, \sigma, \tau) \equiv FrameT(\varphi, \sigma, \tau)$ and $CausedF(|\psi|, \alpha, \sigma, \tau) \equiv FrameF(\varphi, \sigma, \tau)$. Together with assumption $\Sigma \models Poss(\alpha, \sigma, \tau)$, this yields $\Sigma \models (\psi[\tau] \equiv (\psi[\sigma] \land \psi[\tau])) \land (\neg\psi[\tau] \equiv (\neg\psi[\sigma] \land \neg\psi[\tau]))$ and, in consequence, $\Sigma \models (\psi[\tau] \supset \psi[\sigma]) \land (\neg\psi[\tau] \supset \neg\psi[\sigma])$. □

3.4.3 Reifying Default Conclusions

Assuming a user has given their impression of how the world normally behaves by including state defaults into the action domain specification, we can now turn to translating these into the logical language we employ in this work. The special predicate symbol $DefT(f, s, t)$ will be used to express that a fluent f is normally true at time point t. Likewise, $DefF(f, s, t)$ means that f is normally false at t. Note that this is not the same as $\neg DefT(f, s, t)$, which only means that f is not normally true at time point t. The additional TIME argument s is used to keep track of the starting time point of the action that led to t. With this in mind, we can now define how to (automatically) create Reiter defaults from user-specified state defaults. (Observe that all defaults thus created are normal.)

Definition 3.31. Let $\delta =$ `normally` ψ `if` Φ be a state default.

$$\delta_{Init} \stackrel{\text{def}}{=} \frac{Init(t) \land \Phi[t] : \psi[t]}{\psi[t]} \tag{3.19}$$

$$\delta_{Poss} \stackrel{\text{def}}{=} \frac{\Phi[t] \land \neg Violated_\delta(s) : Def(\psi, s, t)}{Def(\psi, s, t)} \tag{3.20}$$

$$Violated_\delta(s) \stackrel{\text{def}}{=} \Phi[s] \land \neg\psi[s]$$

$$Def(\psi, s, t) \stackrel{\text{def}}{=} \begin{cases} DefT(\psi, s, t) & \text{if } \psi = |\psi| \\ DefF(|\psi|, s, t) & \text{otherwise} \end{cases}$$

For an action domain specification Θ, the corresponding set of defaults is defined as

$$\Delta \stackrel{\text{def}}{=} \{\delta_{Init}, \delta_{Poss} \mid \delta = \texttt{normally}\ \psi\ \texttt{if}\ \Phi \in \Theta\} \tag{3.21}$$

The intuition behind the *Init* defaults for the initial time point should be clear: whenever, initially, the prerequisite is fulfilled and there is no reason to believe otherwise, we can safely assume the consequent. Note that we bypass the *Def* predicate in this case – this is not a problem since there is no action leading to the initial time point and thus no effect axiom interfering with the default conclusion. For the *Poss* defaults concerning two time points s, t connected via action application, we require that (1) the state default's prerequisite hold at the resulting time point t, and (2) the state default not be violated at the starting time point s.

Example 3.29 (Continued)**.** A further modification to the rowboat domain could take the boat to be usable if the agent has at least one oar:[7]

$$\delta^{\mathsf{Oar}} = \underline{\texttt{normally}}\ \mathsf{Usable}(\mathsf{Boat})\ \underline{\texttt{if}}\ \mathsf{Has}(\mathsf{Oar}_1) \vee \mathsf{Has}(\mathsf{Oar}_2)$$

The rowboat state default from above is translated to $\delta^{\mathsf{Oar}}_{Init}$ for the initial time point, where $\Phi^{\mathsf{Oar}}(t) \stackrel{\text{def}}{=} \mathit{Holds}(\mathsf{Has}(\mathsf{Oar}_1), t) \vee \mathit{Holds}(\mathsf{Has}(\mathsf{Oar}_2), t)$:

$$\frac{\mathit{Init}(t) \wedge \Phi^{\mathsf{Oar}}(t) : \mathit{Holds}(\mathsf{Usable}(\mathsf{Boat}), t)}{\mathit{Holds}(\mathsf{Usable}(\mathsf{Boat}), t)}$$

The respective default $\delta^{\mathsf{Oar}}_{Poss}$ for action application is

$$\frac{\Phi^{\mathsf{Oar}}(t) \wedge \neg \mathit{Violated}_{\delta^{\mathsf{Oar}}}(s) : \mathit{DefT}(\mathsf{Usable}(\mathsf{Boat}), s, t)}{\mathit{DefT}(\mathsf{Usable}(\mathsf{Boat}), s, t)}$$

with $\mathit{Violated}_{\delta^{\mathsf{Oar}}}(s) = \Phi^{\mathsf{Oar}}(s) \wedge \neg \mathit{Holds}(\mathsf{Usable}(\mathsf{Boat}), s)$.

Up to here, default conclusions and "hard facts" live in completely different, disconnected worlds (*Init* defaults aside). It is the first modification to our effect axiom that brings them together: if a fluent is *normally* true (false) after applying an action we accept this as a cause for its being *actually* true (false).

Definition 3.32. Let A be a function into sort ACTION. An *effect axiom with unconditional effects, the frame assumption and normal state defaults* is of the form (3.16), where

$$\mathit{CausedT}(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} \mathit{FrameT}(f, s, t) \vee \mathit{DirT}(f, A(\vec{x}), s, t) \vee \mathit{DefT}(f, s, t) \tag{3.22}$$

$$\mathit{CausedF}(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} \mathit{FrameF}(f, s, t) \vee \mathit{DirF}(f, A(\vec{x}), s, t) \vee \mathit{DefF}(f, s, t) \tag{3.23}$$

Whenever it is definitely known that $\mathit{Holds}(f, t)$ after $\mathit{Poss}(a, s, t)$, it follows from the effect axiom that $\neg \mathit{DefF}(f, s, t)$; a symmetrical argument applies if $\neg \mathit{Holds}(f, t)$. This means that definite knowledge about a fluent inhibits the opposite default conclusion.

But now imagine the following scenario: we know that a fluent f holds at a time point, $\mathit{Holds}(f, s)$. Nothing further is known about f – in particular no default information. Then an action a occurs and leads to time point t, that is, $\mathit{Poss}(a, s, t)$, where the effects of action a do not involve f. Intuitively, by persistence, we should be able to conclude that f still holds at t. But instead of $\mathit{Holds}(f, t)$, we only get the weaker conclusion $\mathit{Holds}(f, t) \vee (\neg \mathit{Holds}(f, t) \wedge \mathit{DefF}(f, s, t))$, which means that f either stays true or becomes false due to a default conclusion. Since we know the latter is impossible, we would like to incorporate this information somewhere, rather than let the solution of one problem (the state

[7] Observe that this is stronger than the two state defaults $\underline{\texttt{normally}}\ \mathsf{Usable}(\mathsf{Boat})\ \underline{\texttt{if}}\ \mathsf{Has}(\mathsf{Oar}_1)$ and $\underline{\texttt{normally}}\ \mathsf{Usable}(\mathsf{Boat})\ \underline{\texttt{if}}\ \mathsf{Has}(\mathsf{Oar}_2)$.

default problem) disrupt the solution of another (the frame problem). The following addition ensures that knowledge about inapplicability of defaults is adequately represented in our automatic translations. Obviously, a state default is inapplicable if its prerequisite is false or it was previously violated.

Definition 3.33. Let $\delta(\vec{y}) = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi$ be a state default with free variables among $\vec{y}$.

$$\mathit{Inapplicable}_{\delta(\vec{y})}(s,t) \stackrel{\text{def}}{\equiv} \neg\Phi[t] \vee \mathit{Violated}_{\delta(\vec{y})}(s)$$

Let Θ be a $\mathcal{D}$ action domain specification, F be a function into sort fluent with matching sequence of variables $\vec{x}$ and s, t be variables of sort TIME. The *default closure axioms for* $F(\vec{x})$ *with respect to the state defaults in* Θ are

$$\left(\bigwedge_{\delta(\vec{y})=\underline{\texttt{normally}}\ F(\vec{y})\ \underline{\texttt{if}}\ \Phi(\vec{y})\in\Theta} \Big(F(\vec{x}) = F(\vec{y}) \wedge \mathit{Inapplicable}_{\delta(\vec{y})}(s,t)\Big)\right) \supset \neg \mathit{DefT}(F(\vec{x}),s,t) \tag{3.24}$$

$$\left(\bigwedge_{\delta(\vec{y})=\underline{\texttt{normally}}\ \neg F(\vec{y})\ \underline{\texttt{if}}\ \Phi(\vec{y})\in\Theta} \Big(F(\vec{x}) = F(\vec{y}) \wedge \mathit{Inapplicable}_{\delta(\vec{y})}(s,t)\Big)\right) \supset \neg \mathit{DefF}(F(\vec{x}),s,t) \tag{3.25}$$

The *default closure axioms for* Θ are all default closure axioms for functions into sort FLUENT of Θ's signature.

E.g., the default closure axioms for $\mathsf{Flies}(x)$ with respect to the state defaults in Θ_{PA} are

$$\begin{aligned}(\mathsf{Flies}(x) = \mathsf{Flies}(y) \wedge \mathit{Inapplicable}_{\delta(y)}(s,t)) &\supset \neg \mathit{DefT}(\mathsf{Flies}(x),s,t)\\ \top &\supset \neg \mathit{DefF}(\mathsf{Flies}(x),s,t)\end{aligned}$$

where $\mathit{Inapplicable}_{\delta(y)}(s,t) = \neg\mathit{Holds}(\mathsf{PA}(y),t) \vee (\mathit{Holds}(\mathsf{PA}(y),s) \wedge \neg\mathit{Holds}(\mathsf{Flies}(y),s))$. If Θ_{PA} contained another state default $\delta'(z) = \underline{\texttt{normally}}\ \mathsf{Flies}(z)\ \underline{\texttt{if}}\ \mathsf{Bird}(z)$, we would get the positive default closure axiom

$$\begin{aligned}&((\mathsf{Flies}(x) = \mathsf{Flies}(y) \wedge \mathit{Inapplicable}_{\delta(y)}(s,t)) \wedge\\ &(\mathsf{Flies}(x) = \mathsf{Flies}(z) \wedge \mathit{Inapplicable}_{\delta'(z)}(s,t))) \supset \neg \mathit{DefT}(\mathsf{Flies}(x),s,t)\end{aligned}$$

In general, for a fluent function F not mentioned as a consequent of a state default in Θ the default closure axiom is equivalent to $\top \supset \neg\mathit{DefT}(F(\vec{x}),s,t)$, which correctly says that a default conclusion about F can never be made. This definition of default closure axioms is slightly different from the one given in [Baumann et al., 2010]. There, the default closure axioms are defined for general fluent literals and the resulting set of axioms need not necessarily be finite.

We are now ready to define the fundamental notion of our solution to the state default problem: a default theory where the incompletely specified world consists of a UAC domain axiomatisation augmented by suitable default closure axioms, and the defaults are the automatic translations of user-specified, domain-dependent state defaults.

Definition 3.34. Let Θ be a *n-ule*$\mathcal{D}$ action domain specification. The corresponding *n-ule*$\mathcal{D}$ *action default theory* is the pair (Σ, Δ), where

- all effect axioms in Σ are of the form (3.16),

- Σ_{aux} contains
  - unique-names axioms for sorts FLUENT and ACTION,
  - the direct positive and negative effect formulas (3.14, 3.15) for each function into sort ACTION of Θ's domain signature and
  - the default closure axioms for Θ, and
- Δ is defined from Θ as in Definition 3.31.

The workings of all of the preceding definitions are best understood with the help of the example domain.

Example 3.29 (Continued)**.** Choosing situations as underlying time structure and assuming the agent is initially at the right bank – $\Sigma_0 = \{Holds(\mathsf{At}(\mathsf{Right}), S_0)\}$ –, Definitions 3.4 and 3.34 create from Θ_{Boat} the action default theory (Σ, Δ), where $\Sigma = \Omega_{sit} \cup \Pi \cup \Upsilon \cup \Sigma_0 \cup \Sigma_{aux}$. As it is, crossing to the left is initially not necessarily possible, since the boat may be not usable, $(\Sigma, \Delta) \not\models\!\!\approx Poss(\mathsf{CrossRL}, S_0, Do(\mathsf{CrossRL}, S_0))$. But once we learn the agent has at least one of the oars, crossing becomes possible:

$$(\Sigma \cup \{Holds(\mathsf{Has}(\mathsf{Oar}_1), S_0) \vee Holds(\mathsf{Has}(\mathsf{Oar}_2), S_0)\}, \Delta) \models\!\!\approx Poss(\mathsf{CrossRL}, S_0, Do(\mathsf{CrossRL}, S_0))$$

Much like it was the case for the simple form of our effect axiom, there is also a formal result which shows that axiom (3.16), apart from solving the frame problem, implements a particular preference ordering among potential reasons for a fluent to hold or not to hold. Taking state defaults into account, the priorities become (from most to least preferred)

$$\text{direct effects} < \text{default conclusions} < \text{persistence}.$$

Theorem 3.35. *Let Θ be an n-ule$\mathcal{D}$ action domain specification and (Σ, Δ) be its action default theory, E be an extension for (Σ, Δ) with $E \models Poss(\alpha, \sigma, \tau)$ for some ground action $\alpha = A(\vec{o})$ and time points σ, τ.*

1. *Effects override everything:* $\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \psi \in \Theta$ *implies* $E \models \psi[\tau]\,\{\vec{x} \mapsto \vec{o}\}$.
2. *Defaults override persistence: let* $\delta = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi \in \Theta$ *be a state default,*
   (A) ψ *not be mentioned as an effect of* α *in* Θ,
   (B) for each $\delta' = \underline{\texttt{normally}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi' \in \Theta$, *let* $E \not\models \Phi'[\tau]$; *and*
   (C) $E \models \Phi[\tau] \wedge \neg Violated_\delta(\sigma)$.
   Then $E \models \psi[\tau]$.
3. *The frame assumption is correctly implemented:*
   Let the fluent of ψ *not be mentioned as an effect of* $A(\vec{x})$ *in* Θ *and for all state defaults* $\delta_1 = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi_1, \delta_2 = \underline{\texttt{normally}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi_2 \in \Theta$, *let* $E \not\models \Phi_i[\tau]$ *or* $E \models Violated_{\delta_i}(\sigma)$. *Then*

$$E \models \psi[\sigma] \equiv \psi[\tau]$$

Proof. If E is inconsistent, the claims are immediate, so in what follows assume that E is consistent.

1. By Definition 3.32, $Dir(\psi, \alpha, \sigma, \tau) \equiv |\psi| = |\psi| \vee \ldots$ and consequently $Caused(\psi, \alpha, \sigma, \tau) \equiv |\psi| = |\psi| \vee \ldots$, thus by effect axiom (3.16) and assumption $E \models Poss(\alpha, \sigma, \tau)$, we get $E \models \psi[\tau]$.

2. Assume $E \not\models \neg Def(\psi, \sigma, \tau)$. Together with assumption (C) this means that default δ_{Poss} is applicable to E. Since E is an extension, we have $E \models Def(\psi, \sigma, \tau)$. Invoking effect axiom (3.16) yields the claim. It remains to establish $E \not\models \neg Def(\psi, \sigma, \tau)$. By Theorem 2.9, there exist E_i, $i \geq 0$, with $E_0 = \Sigma$ and for $i \geq 0$, $E_{i+1} = Th(E_i) \cup \{\beta \mid \alpha : \beta/\beta \in \Delta, \alpha \in E_i, \neg\beta \notin E\}$ such that $E = \bigcup_{i=0}^{\infty} E_i$. We first show $E_0 \not\models \neg Def(\psi, \sigma, \tau)$. Due to (A) and effect axiom (3.16), we know that $E_0 \not\models \psi[\tau]$ and $E_0 \not\models \neg\psi[\tau]$. By assumption (C), consistency of E, and $E_0 \subseteq E$, we get $E_0 \not\models \neg\Phi[\tau] \vee Violated_{\delta}(\sigma)$. In combination with $E_0 \not\models \neg\psi[\tau]$ this yields $E_0 \not\models \neg Def(\psi, \sigma, \tau)$, since default closure axioms and α's effect axiom are the only ways to conclude $\neg Def(\psi, \sigma, \tau)$. Now assume to the contrary that $E \models \neg Def(\psi, \sigma, \tau)$. Then there is a minimal integer $i \geq 0$ such that $E_i \not\models \neg Def(\psi, \sigma, \tau)$ and $E_{i+1} \models \neg Def(\psi, \sigma, \tau)$. This must be due to a default $\delta' = \underline{\texttt{normally}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi' \in \Theta$ with $E_i \models \Phi'[\tau] \wedge \neg Violated_{\delta'}(\sigma)$. But then $E_i \subseteq E$ implies $E \models \Phi'[\tau]$, which is a contradiction to assumption (B).

3. By the assumption that ψ and $\neg\psi$ are not amongst α's direct effects, $\Sigma \models \neg Dir(\psi, \sigma, \tau) \wedge \neg Dir(\neg\psi, \sigma, \tau)$. The second assumption ensures that the left-hand sides of all relevant default closure axioms become true, hence $E \models \neg Def(\psi, \sigma, \tau) \wedge \neg Def(\neg\psi, \sigma, \tau)$. In consequence, $Caused(\psi, \alpha, \sigma, \tau)$ reduces to $Frame(\psi, \sigma, \tau)$, which proves the claim. □

As important and nice as these properties are, a default theory would be useless if it did not admit any extension at all. But the existence of extensions for our default theories follows immediately from a result by [Reiter, 1980], since the defaults automatically created by Definition 3.31 are all normal. If the involved domain axiomatisation is consistent, we can even guarantee all its extensions are consistent, too.

Theorem 3.36. *Let Θ be an n-ule$\mathcal{D}$ action domain specification. Then its corresponding action default theory (Σ, Δ) has an extension. If furthermore Σ without the default closure axioms is consistent, then so are all extensions for (Σ, Δ).*

Proof. Existence of an extension is a corollary of [Reiter, 1980, Theorem 3.1] since all defaults in Δ are normal. Let Σ' be Σ without default closure axioms. The default closure axioms cannot make Σ inconsistent: for any model $\mathfrak{I}$ for Σ' which violates a default closure axiom we have (w.l.o.g.) $\mathfrak{I} \models (\exists f, s, t) DefT(f, s, t)$. We can build a model $\mathfrak{I}'$ for Σ with $\left|DefT^{\mathfrak{I}'}\right| < \left|DefT^{\mathfrak{I}}\right|$ by removing an element from $DefT^{\mathfrak{I}}$ and adjusting $Holds^{\mathfrak{I}'}$ and $Poss^{\mathfrak{I}'}$ accordingly. The same can be done with respect to $DefF$, hence there exists a model for Σ' which is by the syntactic structure of (3.24,3.25) also a model for Σ. Consistency of all extensions now follows from [Reiter, 1980, Corollary 2.2]. □

3.4.4 Comparison to the Previous Approaches

For the action description language *a-ule$\mathcal{D}$* we provided two different translations to action default theories. Since it is not obvious from the definitions how they formally relate to each other, we provide an example that illuminates the most significant difference between the two.

Example 3.37. Consider a simplified version of the paper airplane domain given by $\underline{\texttt{action}}$ Fold $\underline{\texttt{causes}}$ PA and $\underline{\texttt{normally}}$ Flies $\underline{\texttt{if}}$ PA. Let us first compile the statements into the action default theory (Σ^1, Δ^1) according to Definition 3.19. After Definition 3.21 adds the default occlusion $\underline{\texttt{action}}$ Fold $\underline{\texttt{causes}}$ Flies $\underline{\texttt{or}}$ ¬Flies, we obtain the effect axiom

$$Poss(\mathsf{Fold}, s, t) \supset (\forall f)(f = \mathsf{Flies} \vee (Holds(f, t) \equiv (f = \mathsf{PA} \vee Holds(f, s))))$$

and the single default

$$\frac{Holds(\mathsf{PA}, s) : Holds(\mathsf{Flies}, s)}{Holds(\mathsf{Flies}, s)} \in \Delta^1$$

Using Definition 3.34 to compile the specification above into the action default theory (Σ^2, Δ^2) yields the direct effect formulas $DirT(f, \mathsf{Fold}, s, t) \equiv f = \mathsf{PA}$ and $DirF(f, \mathsf{Fold}, s, t) \equiv \bot$, and the effect axiom

$$\begin{aligned} Poss(\mathsf{Fold}, s, t) \supset\ & (\forall f)(Holds(f, t) \equiv (FrameT(f, s, t) \vee DirT(f, \mathsf{Fold}, s, t) \vee DefT(f, s, t)) \wedge \\ & (\forall f)(\neg Holds(f, t) \equiv (FrameF(f, s, t) \vee Dir(f, \mathsf{Fold}, s, t) \vee DefF(f, s, t)) \end{aligned}$$

The defaults created by Definition 3.31 are

$$\Delta^2 = \left\{ \frac{Init(t) \wedge Holds(\mathsf{PA}, t) : Holds(\mathsf{Flies}, t)}{Holds(\mathsf{Flies}, t)}, \right. \\ \left. \frac{Holds(\mathsf{PA}, t) \wedge \neg(Holds(\mathsf{PA}, s) \wedge \neg Holds(\mathsf{Flies}, s)) : DefT(\mathsf{Flies}, s, t)}{DefT(\mathsf{Flies}, s, t)} \right\}$$

We assume that $\Sigma_0 = \emptyset$ is the same for both axiomatisations, and look at what we can conclude about the situation $Do(\mathsf{Fold}, S_0)$. In (Σ^1, Δ^1), the action effect PA and the occlusion of Flies make the default applicable and we get $(\Sigma^1, \Delta^1) \mid\!\approx Holds(\mathsf{Flies}, Do(\mathsf{Fold}, S_0))$.

In (Σ^2, Δ^2) on the other hand, due to the underspecified initial situation, it does not follow that the state default `normally` Flies `if` PA was not violated at S_0. Its corresponding *Poss* default is thus inapplicable, whence we get $(\Sigma^2, \Delta^2) \not\mid\!\approx Holds(\mathsf{Flies}, Do(\mathsf{Fold}, S_0))$.

According to this example, it may seem that the simpler approach allows an intuitive conclusion while the more complicated one is too cautious. But this has to be qualified: the conclusions shown above would be the same for $\Sigma_0 = \{Holds(\mathsf{PA}, S_0) \wedge \neg Holds(\mathsf{Flies}, S_0)\}$! That is because the simple approach does not check for default violations at all. While this allows more conclusions in general, these conclusions are not always sensible.

3.5 Conditional Effects

We now investigate how the default reasoning framework of the previous sections can be extended to conditional effect actions. As we will show, there is subtle interdependence between conditional effects and default conclusions, which requires a revision of the defaults constructed in Definition 3.31.

Allowing preconditions of effects means stepping from *n-ule*$\mathcal{D}$ to *n-cle*$\mathcal{D}$ by allowing effect laws of the form

$$\texttt{action}\ A(\vec{x})\ \texttt{causes}\ \psi\ \texttt{if}\ \Phi$$

with a possibly non-trivial fluent formula Φ that specifies the conditions under which the effect materialises. With this specification of action effects, it is easy to express the implication "effect precondition implies effect" via suitable formulas. For this purpose, we modify the direct effect formulas (3.14) and (3.15) to account for effect preconditions. In a slight abuse of notation, we set the sign of a direct effect law to be the sign of its effect, that is, $sign(\texttt{action}\ A(\vec{x})\ \texttt{causes}\ \psi\ \texttt{if}\ \Phi) \stackrel{\text{def}}{=} sign(\psi)$.

Definition 3.38. Let Θ be a *n-cle*$\mathcal{D}$ action domain specification, A be a function into sort ACTION with matching sequence of variables $\vec{x}$, $\varepsilon = \underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi$ be a direct effect law and f : FLUENT and s, t : TIME be variables. The following macro expresses that ε *has been activated for* f *from* s *to* t:[8]

$$Activated_\varepsilon(f, s, t) \stackrel{\text{def}}{=} (f = |\psi| \wedge \Phi[s])$$

The *direct positive and negative effect formulas for* $A(\vec{x})$ are

$$DirT(f, A(\vec{x}), s, t) \equiv \bigvee_{\varepsilon \in \Theta, sign(\varepsilon) = +} Activated_\varepsilon(f, s, t) \tag{3.26}$$

$$DirF(f, A(\vec{x}), s, t) \equiv \bigvee_{\varepsilon \in \Theta, sign(\varepsilon) = -} Activated_\varepsilon(f, s, t) \tag{3.27}$$

An *effect axiom with conditional effects, the frame assumption and normal state defaults* is of the form (3.16).

While this extended definition of action effects is straightforward, it severely affects the correctness of default reasoning in the action theory: as the following example shows, one cannot naïvely take this updated version of the effect axioms and use the Reiter defaults as before.

Example 3.39 (Handle with Care)**.** Imagine a robot that can move around and carry objects, among them a vase. When the robot drops an object x, it does not carry x any more and additionally x is broken if it was fragile. Usually, however, objects are not broken unless there is information to the contrary. The fluents that we use to describe this domain are $\mathsf{Carries}(x)$ (the robot carries x), $\mathsf{Fragile}(x)$ (x is fragile) and $\mathsf{Broken}(x)$ (x is broken); the only function of sort ACTION is $\mathsf{Drop}(x)$. Dropping an object is possible if and only if the robot carries the object; the state defaults of this domain say that objects are normally not broken. In $\mathcal{D}$, this natural-language specification is written as

$$\begin{aligned}\Theta_{Break} = \{&\underline{\texttt{possible}}\ \mathsf{Drop}(x)\ \underline{\texttt{iff}}\ \mathsf{Carries}(x),\\ &\underline{\texttt{action}}\ \mathsf{Drop}(x)\ \underline{\texttt{causes}}\ \neg\mathsf{Carries}(x),\\ &\underline{\texttt{action}}\ \mathsf{Drop}(x)\ \underline{\texttt{causes}}\ \mathsf{Broken}(x)\ \underline{\texttt{if}}\ \mathsf{Fragile}(x),\\ &\underline{\texttt{normally}}\ \neg\mathsf{Broken}(x)\}\end{aligned}$$

Applying the definitions from above to this specification results in the domain axiomatisation with defaults $(\Sigma^{Break}, \Delta^{Break})$, where Σ^{Break} contains effect axiom (3.16), the above precondition axiom for Drop and the default closure axioms. For the defaults, we have

$$\Delta^{Break} = \left\{ \frac{Init(t) : \neg Holds(\mathsf{Broken}(x), t)}{\neg Holds(\mathsf{Broken}(x), t)}, \frac{\neg Holds(\mathsf{Broken}(x), s) : DefF(\mathsf{Broken}(x), s, t)}{DefF(\mathsf{Broken}(x), s, t)} \right\}$$

Assume now that all we know is that the robot initially carries the vase, $Holds(\mathsf{Carries}(\mathsf{Vase}), S_0)$. The effect axiom tells us that the robot does not carry the vase any more at $S_1 \stackrel{\text{def}}{=} Do(\mathsf{Drop}(\mathsf{Vase}), S_0)$. Additionally, since we do not know whether the vase was fragile at S_0, there is no reason to believe that it is broken after dropping it, hence $\neg\mathsf{Broken}(\mathsf{Vase})$ still holds by default at S_1. But now, due

[8] The second time argument t of macro $Activated_\varepsilon(f, s, t)$ will only be needed later when we introduce non-deterministic effects.

to the presence of conditional effects, the effect axiom for $\mathsf{Drop}(\mathsf{Vase})$ clearly entails $\neg \mathit{Holds}(\mathsf{Broken}(\mathsf{Vase}), S_1) \supset \neg \mathit{Holds}(\mathsf{Fragile}(\mathsf{Vase}), S_0)$,[9] and thus we can draw the conclusion

$$(\Sigma^{\mathit{Break}}, \Delta^{\mathit{Break}}) \mid\!\approx \neg \mathit{Holds}(\mathsf{Fragile}(\mathsf{Vase}), S_0)$$

This is undesired as it lets us conclude something about the present (S_0) using knowledge about the future (S_1) which we could not conclude using only knowledge and default knowledge about the present – there is no default that could conclude $\neg\mathsf{Fragile}(\mathsf{Vase})$.

The flaw with this inference is that it makes a default conclusion about a fluent whose truth value is affected by an action at the same time. This somewhat contradicts our intended usage of defaults about states: we originally wanted to express reasonable assumptions about fluents whose values are unknown.

Generalising the example, the undesired behaviour occurs whenever there exists a state default $\underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi_D$ with conclusion ψ whose negation $\neg\psi$ might be brought about by a conditional effect $\underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi_C$. The faulty inference then goes like this:

$$\Phi_D[t] \supset \mathit{Def}(\psi, s, t) \supset \psi[t] \supset \neg \mathit{Dir}(\neg\psi, s, t) \supset \neg\Phi_C[s]$$

From the default's prerequisite, we conclude the default's consequent normally holds; by the effect axiom, we reason from this that it indeed holds; hence its opposite effect cannot have occurred and the precondition of the effect cannot have been true at the starting time point. Obviously, this inference is only undesired if there is no information about the effect's precondition at the starting time point of the action. This motivates our formal definition of the conditions under which a so-called *conflict* between an action effect and a default conclusion arises.

Definition 3.40. Let (Σ, Δ) be an action default theory, E be an extension for (Σ, Δ), α be a ground action and $\delta = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi$ be a ground state default. We say that there is a *conflict between α and δ in E* iff there exist ground time points σ and τ such that for some $i \geq 0$ we have

1. (a) $E_i \not\models \mathit{Poss}(\alpha, \sigma, \tau) \supset \neg \mathit{Dir}(\neg\psi, \alpha, \sigma, \tau)$
   (b) $E_i \not\models \mathit{Def}(\psi, \alpha, \sigma, \tau)$
2. (a) $E_{i+1} \models \mathit{Poss}(\alpha, \sigma, \tau) \supset \neg \mathit{Dir}(\neg\psi, \alpha, \sigma, \tau)$
   (b) $E_{i+1} \models \mathit{Def}(\psi, \sigma, \tau)$

In words, a conflict arises in an extension if up to some stage i, before we make the default conclusion ψ, we cannot conclude the effect $\neg\psi$ will not occur (1); after concluding ψ by default, we infer that $\neg\psi$ cannot occur as direct effect (2). We can now go back to the example seen earlier and verify that the counter-intuitive conclusion drawn there was indeed due to a conflict in the sense of the above definition.

Example 3.39 (Continued)**.** Consider the only extension E^{Break} for $(\Sigma^{\mathit{Break}}, \Delta^{\mathit{Break}})$. Before applying any defaults whatsoever, we know that dropping the vase is possible: $E_0^{\mathit{Break}} \models \mathit{Poss}(\mathsf{Drop}(\mathsf{Vase}), S_0, S_1)$; but we do not know if the vase is fragile and hence $E_0^{\mathit{Break}} \not\models \neg \mathit{DirT}(\mathsf{Broken}(\mathsf{Vase}), \mathsf{Drop}(\mathsf{Vase}), S_0, S_1)$ (item 1). After applying all the defaults, we know that the vase is not broken at S_1: $E_1^{\mathit{Break}} \models \mathit{DefF}(\mathsf{Broken}(\mathsf{Vase}), S_0, S_1)$. Hence, it cannot have been broken by dropping it in S_0, that is, $E_1^{\mathit{Break}} \models \neg \mathit{DirT}(\mathsf{Broken}(\mathsf{Vase}), \mathsf{Drop}(\mathsf{Vase}), S_0, S_1)$ (item 2), thus cannot have been fragile in the initial situation.

[9] This is just the contrapositive of the implication expressed by the effect axiom.

In the following, we will modify the definition of Reiter defaults to eliminate the possibility of such conflicts. The underlying idea is to apply a default only if it is known that a conflict cannot arise, that is, if it is known that the contradictory direct effect cannot materialise. To this end, we extend the original default prerequisite $\Phi[t] \wedge \neg \mathit{Violated}_\delta(s)$ that only requires the state default's prerequisite to hold at the resulting time point and the state default not to be violated at the starting time point: we will additionally stipulate that any action a happening at the same time cannot create a conflict.

Definition 3.41. Let $\delta = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi$ be a state default and s, t : TIME be variables.

$$\begin{aligned} \mathit{Safe}_\delta(s,t) &\stackrel{\text{def}}{\equiv} (\forall a)(\mathit{Poss}(a,s,t) \supset \neg \mathit{Dir}(\neg\psi, a, s, t)) \\ \delta_{\mathit{Poss}} &\stackrel{\text{def}}{\equiv} \frac{\Phi[t] \wedge \neg \mathit{Violated}_\delta(s) \wedge \mathit{Safe}_\delta(s,t) : \mathit{Def}(\psi, s, t)}{\mathit{Def}(\psi, s, t)} \end{aligned} \tag{3.28}$$

In the example domain, applying the above definition yields the following.

Example 3.39 (Continued)**.** For the state default δ^{Break} saying that objects are usually not broken, we have

$$\mathit{Safe}_{\delta^{\mathit{Break}}}(s,t) = (\forall a)(\mathit{Poss}(a,s,t) \supset \neg \mathit{DirT}(\mathsf{Broken}(x), a, s, t))$$

This expresses that the state default can be safely applied from s to t whenever for any action a happening at the same time, it is known that a does not cause the opposite conclusion of this default at the ending time point t. The resulting default $\delta^{\mathit{Break}}_{\mathit{Poss}}$ is

$$\frac{\neg \mathit{Holds}(\mathsf{Broken}(x), s) \wedge \mathit{Safe}_{\delta^{\mathit{Break}}}(s,t) : \mathit{DefF}(\mathsf{Broken}(x), s, t)}{\mathit{DefF}(\mathsf{Broken}(x), s, t)}$$

As we will see later (Theorem 3.35), the default closure axioms for preserving the commonsense principle of inertia in the presence of inapplicable defaults need not be modified. The extension to conditional effects is a proper generalisation of the approach of the previous section for the special case of unconditional effect actions: this is immediate from the respective definitions of the direct effect formulas (3.14,3.15) and (3.26, 3.27).

3.6 Global Effects

Up to here, we only looked at *local* effect laws ε for an action $A(\vec{x})$, where the variables in ε were restricted to variables among $\vec{x}$. Considering a ground instance $A(\vec{\zeta})$ of an action, this means that the set of objects that can possibly be affected by this action is already fixed to $\vec{\zeta}$. This is a restriction because it can make the specification of certain actions at least cumbersome or utterly impossible, for example actions that affect a vast number of (or all of the) domain elements at once.

The gain in expressiveness when allowing non-local action effects comes at a relatively low cost: it suffices to allow additional free variables $\vec{y}$ in the effect laws. They represent the objects that may be affected by the action without being among the action arguments $\vec{x}$. Thus we transition from action description language *n-cle*$\mathcal{D}$ to language *n-cge*$\mathcal{D}$.

Definition 3.42. Let Θ be an *n-cge*$\mathcal{D}$ action domain specification, A be a function into sort ACTION with matching sequence of variables $\vec{x}$, ε be a direct effect law of the form

$\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ F(\vec{x}',\vec{y})\ \underline{\texttt{if}}\ \Phi$ or $\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \neg F(\vec{x}',\vec{y})\ \underline{\texttt{if}}\ \Phi$ with free variables $\vec{x}',\vec{y}$, where $\vec{x}' \subseteq \vec{x}$ and $\vec{y}$ is disjoint from $\vec{x}$. For variables f : FLUENT and s,t : TIME, the following macro expresses that ε *has been activated for* f *from* s *to* t:

$$Activated_\varepsilon(f,s,t) \stackrel{\text{def}}{\equiv} (\exists\vec{y})(f = F(\vec{x}',\vec{y}) \wedge \Phi[s])$$

The *direct positive and negative effect formulas* are of the form (3.26) and (3.27).

Note that according to this definition, free variables $\vec{y}$ are quantified existentially when they occur in the condition Φ and universally when they occur in the effect ψ. In addition to non-local effects they thus also express non-local conditions.

Example 3.43 (Exploding Bomb [Reiter, 1991])**.** In this domain, objects might get broken not by being dropped, but because a bomb in their proximity explodes:

$$\underline{\texttt{action}}\ \mathsf{Detonate}(b)\ \underline{\texttt{causes}}\ \mathsf{Broken}(x)\ \underline{\texttt{if}}\ \mathsf{Near}(b,x)$$

Definition 3.42 yields the direct effect formulas $DirF(f,\mathsf{Detonate}(b),s,t) \equiv \bot$ and $DirT(f,\mathsf{Detonate}(b),s,t) \equiv (\exists x)(f = \mathsf{Broken}(x) \wedge Holds(\mathsf{Near}(x,b),s))$.

For this example, the defaults from Definition 3.41 also prevent conflicts possibly arising from non-local effects. We will later see that this is the case for all domains with local and non-local effect actions.

Like the compilation for *n-ule*$\mathcal{D}$ domains, the extension of this section implements a particular preference ordering between causes that determine a fluent's truth value. This means that whenever two causes are in conflict – for example, a state default says an object is not broken, and an action effect says it *is* – the preferred cause takes precedence. The preferences are (still)

$$\text{direct effects} < \text{default conclusions} < \text{persistence},$$

The theorem below proves that this preference ordering is indeed established.

Theorem 3.44. *Let* Θ *be a n-cge*$\mathcal{D}$ *action domain specification with action default theory* (Σ,Δ)*,* $\delta = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi \in \Theta$ *be a state default,* E *be an extension for the action default theory* (Σ,Δ) *with* $E \models Poss(\alpha,\sigma,\tau)$ *for a ground action* $\alpha = A(\vec{o})$ *and time points* σ,τ*.*

1. *Effects override everything:*

   $$\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \psi'\ \underline{\texttt{if}}\ \Phi' \in \Theta \text{ implies } E \models (\Phi'[\sigma] \supset \psi'[\tau])\,\{\vec{x} \mapsto \vec{o}\}$$

2. *Defaults override persistence:*

   (A) Let $\underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi'', \underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi'' \notin \Theta$ *for all* Φ''*;*

   (B) for each $\delta' = \underline{\texttt{normally}}\ \neg\psi\ \underline{\texttt{if}}\ \Phi' \in \Delta$*, let* δ' *not be applicable to* E*; and*

   (C) $E \models \Phi[\tau] \wedge \neg Violated_\delta(\sigma) \wedge Safe_\delta(\sigma,\tau)$*.*

   Then $E \models \psi[\tau]$*.*

3. *The frame assumption is correctly implemented:*
   For all fluent formulas Φ''*, let* $\underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi'', \underline{\texttt{action}}\ \Phi''\ \underline{\texttt{causes}} \notin \underline{\texttt{if}}\ \neg\psi\Theta$ *and for all state defaults* δ' *with consequent* ψ *or* $\neg\psi$*, let* $E \models \neg(\Phi[\tau] \wedge \neg Violated_\delta(\sigma))$*. Then* $E \models \psi[\sigma] \equiv \psi[\tau]$*.*

Proof. If E is inconsistent, the claims are immediate, so in what follows assume that E is consistent. The proofs of 2 and 3 carry over from Theorem 3.35, so we only show 1.

1. By Definition 3.26, we get $E \models \Phi'[\sigma] \supset Dir(\psi',\alpha,\sigma,\tau)\,\{\vec{x} \mapsto \vec{o}\}$ and consequently $E \models \Phi'[\sigma] \supset CausedT(\psi',\alpha,\sigma,\tau)\,\{\vec{x} \mapsto \vec{o}\}$ Thus by effect axiom (3.16) and assumption $E \models Poss(\alpha,\sigma,\tau)$, we get $E \models \Phi'[\sigma] \supset \psi'[\tau]\,\{\vec{x} \mapsto \vec{o}\}$ □

3.7 Disjunctive Effects

The next and final addition to effect axiom (3.16) is the step of generalising the purely deterministic action effects of *n-cle*$\mathcal{D}$ to the disjunctive ones of *n-cls*$\mathcal{D}$. Disjunctive action effects have been studied in the past [Kartha, 1994; Shanahan, 1997; Giunchiglia et al., 1997; Thielscher, 2000]. Our contribution here is two-fold. First, we express disjunctive effects directly, by building them into the effect axiom. This works without introducing additional function symbols – called *determining fluents* [Shanahan, 1997] – for which persistence is not assumed and that are used to derive indeterminate effects via conditional effects. The second and more important contribution is the combination of non-deterministic effects with state defaults. We claim that it brings a significant representational advantage: Disjunctive effects can explicitly represent potentially different outcomes of an action of which none is necessarily predictable. At the same time, state defaults can be used to model the action effect that *normally* obtains. For example, dropping an object might not always completely break it, but most of the time only damage it. This can be modelled in our framework by specifying "broken or damaged" as disjunctive effect of the drop action, and then including the default "normally, dropped objects are damaged" to express the usual outcome.

Next, we define how disjunctive effects are accommodated into the theory. The basic idea is to allow disjunctions of fluent literals ψ_1 $\underline{\texttt{or}}$ $\ldots$ $\underline{\texttt{or}}$ ψ_n in the effect part of an effect law. The intended meaning of these disjunctions is that after action execution, some of the ψ_i hold. To achieve this, we firstly want to guarantee that at least one effect out of ψ_1 $\underline{\texttt{or}}$ $\ldots$ $\underline{\texttt{or}}$ ψ_n occurs. Hence, we say for each ψ_i that non-occurrence of all the other effects ψ_j with $j \neq i$ is a sufficient cause for ψ_i to occur. We build into the effect axiom (in the same way as before) the n implications

$$\begin{gathered}\Phi[s] \wedge \neg\psi_2[t] \wedge \ldots \wedge \neg\psi_n[t] \supset \mathit{Caused}(\psi_1, a, s, t) \\ \vdots \\ \Phi[s] \wedge \neg\psi_1[t] \wedge \ldots \wedge \neg\psi_{n-1}[t] \supset \mathit{Caused}(\psi_n, a, s, t)\end{gathered}$$

This, together with the persistence assumption, effectively axiomatises an exclusive disjunction where only exactly one effect ψ_i occurs (given that no other effects occur simultaneously). Thus we add, for each literal, its truth as sufficient cause for itself being true:

$$\begin{gathered}\Phi[s] \wedge \psi_1[t] \supset \mathit{Caused}(\psi_1, a, s, t) \\ \vdots \\ \Phi[s] \wedge \psi_n[t] \supset \mathit{Caused}(\psi_n, a, s, t)\end{gathered}$$

This makes every interpretation where at least one of the mentioned literals became true a model of the effect axiom. For the next definition, we identify a $\mathcal{D}$-disjunction of literals $\Psi = \psi_1$ $\underline{\texttt{or}}$ $\ldots$ $\underline{\texttt{or}}$ ψ_n with the set of literals $\{\psi_1, \ldots, \psi_n\}$.

Definition 3.45. Let Θ be an *n-cgs*$\mathcal{D}$ action domain specification, A be a function into sort ACTION with matching sequence of variables $\vec{x}$, $\varepsilon = \underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \Psi\ \underline{\texttt{if}}\ \Phi$ be a direct effect law, $\psi \in \Psi$ and f : FLUENT and s, t : TIME be variables. The following macro expresses that *effect ψ of direct effect law ε has been activated for f from s to t*:

$$\mathit{Activated}_{\varepsilon,\psi}(f, s, t) \stackrel{\text{def}}{=} f = |\psi| \wedge \Phi[s] \wedge \left(\left(\bigwedge_{\psi' \in \Psi \setminus \{\psi\}} \neg\psi'[t] \right) \vee \psi[t] \right) \tag{3.29}$$

The *direct positive and negative effect formulas* are

$$DirT(f, A(\vec{x}), s, t) \equiv \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \Psi\ \underline{\texttt{if}}\ \Phi \in \Theta,\ \psi \in \Psi,\ sign(\psi)=+} Activated_{\varepsilon,\psi}(f, s, t) \tag{3.30}$$

$$DirF(f, A(\vec{x}), s, t) \equiv \bigvee_{\underline{\texttt{action}}\ A(\vec{x})\ \underline{\texttt{causes}}\ \Psi\ \underline{\texttt{if}}\ \Phi \in \Theta,\ \psi \in \Psi,\ sign(\psi)=-} Activated_{\varepsilon,\psi}(f, s, t) \tag{3.31}$$

The implementation of the example sketched above illustrates the definition.

Example 3.46 (Definitely Maybe)**.** We once again modify the action $\mathsf{Drop}(x)$ from Example 3.39. Now a fragile object that is dropped becomes not necessarily completely broken, but might only get damaged. To this end, we record in the new fluent $\mathsf{Dropped}(x)$ that the object has been dropped and write the state default δ below saying that dropped objects are usually damaged. Together, these two express the *normal* outcome of the drop action. Still, it is definite knowledge that an object may be broken or may be damaged after dropping, even if this hints at non-determinism. Formally, the domain is given by

$$\begin{aligned}\Theta = \{&\underline{\texttt{normally}}\ \mathsf{Damaged}(x)\ \underline{\texttt{if}}\ \mathsf{Dropped}(x),\\ &\underline{\texttt{action}}\ \mathsf{Drop}(x)\ \underline{\texttt{causes}}\ \neg\mathsf{Carries}(x),\\ &\underline{\texttt{action}}\ \mathsf{Drop}(x)\ \underline{\texttt{causes}}\ \mathsf{Dropped}(x),\\ &\underline{\texttt{action}}\ \mathsf{Drop}(x)\ \underline{\texttt{causes}}\ \mathsf{Broken}(x)\ \underline{\texttt{or}}\ \mathsf{Damaged}(x)\ \underline{\texttt{if}}\ \mathsf{Fragile}(x)\}\end{aligned}$$

Constructing the direct effect formulas as per Definition 3.45 yields

$$\begin{aligned}DirT(f, \mathsf{Drop}(x), s, t) \equiv\ & f = \mathsf{Dropped}(x)\\ \vee\ & (f = \mathsf{Broken}(x) \wedge Holds(\mathsf{Fragile}(x), s) \wedge\\ & \quad (\neg Holds(\mathsf{Damaged}(x), t) \vee Holds(\mathsf{Broken}(x), t)))\\ \vee\ & (f = \mathsf{Damaged}(x) \wedge Holds(\mathsf{Fragile}(x), s) \wedge\\ & \quad (\neg Holds(\mathsf{Broken}(x), t) \vee Holds(\mathsf{Damaged}(x), t)))\end{aligned}$$

Since the effect axiom of $\mathsf{Drop}(x)$ is itself not determined about the status of $\mathsf{Broken}(x)$ and $\mathsf{Damaged}(x)$ (but is determined about $\mathsf{Damaged}(x)$ not being among its negative effects), the default δ_{Poss} is applicable and we conclude

$$(\Sigma^{Break}, \Delta^{Break}) \mid\!\approx Holds(\mathsf{Carries}(\mathsf{Vase}), S_0) \wedge Holds(\mathsf{Damaged}(\mathsf{Vase}), S_1)$$

If we now observe that the vase is broken after all – $Holds(\mathsf{Broken}(\mathsf{Vase}), S_1)$ – and add this information to the knowledge base, we will learn that this was an action effect:

$$(\Sigma^{Break}, \Delta^{Break}) \mid\!\approx Holds(\mathsf{Broken}(\mathsf{Vase}), S_1) \supset DirT(\mathsf{Broken}(\mathsf{Vase}), \mathsf{Drop}(\mathsf{Vase}), S_0, S_1)$$

Furthermore, the observation allows us to rightly infer that the vase was fragile at S_0.

It is worth noting that for an effect law $\underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \Psi\ \underline{\texttt{if}}\ \Phi$ with deterministic effect $\Psi = \{\psi\}$, the macro $Activated_{\underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \Psi\ \underline{\texttt{if}}\ \Phi, \psi}(f, s, t)$ expressing activation of this effect is equivalent to $Activated_{\underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi}(f, s, t)$ from Definition 3.26 for activation of the deterministic effect; hence the direct effect formulas (3.30) for disjunctive effects are a generalisation of (3.26), the ones for deterministic effects. We have considered here only

local non-deterministic effects to keep the presentation simple. Of course, the notion can be extended to non-local effects without harm.

We finally prove that conflicts between conditional effects and default conclusions in the sense of Definition 3.40 cannot occur.

Theorem 3.47. *Let (Σ, Δ) be an action default theory, E be an extension for (Σ, Δ) and $\delta = \underline{\texttt{normally}}\ \psi\ \underline{\texttt{if}}\ \Phi$ be a state default. Furthermore, let $i \geq 0$ be such that $Def(\psi, \sigma, \tau) \notin E_i$ and $Def(\psi, \sigma, \tau) \in E_{i+1}$. Then for all ground actions α, $Poss(\alpha, \sigma, \tau) \supset \neg Dir(\neg\psi, \alpha, \sigma, \tau) \in E_i$.*

Proof. According to Theorem 2.9, we have $E_{i+1} = Th(E_i) \cup \Delta_i$; hence, $Def(\psi, \sigma, \tau) \in E_{i+1}$ can have two possible reasons:

1. $Def(\psi, \sigma, \tau) \in Th(E_i)$. By construction, this can only be due to effect axiom (3.16), more specifically, we have (1) $E_i \models Caused(\psi, \alpha, \sigma, \tau) \wedge \neg Frame(\psi, \sigma, \tau) \wedge \neg Dir(\psi, \sigma, \tau)$ and (2) $E_i \models \neg Caused(\neg\psi, \alpha, \sigma, \tau)$, whence $E_i \models \neg Dir(\neg\psi, \alpha, \sigma, \tau)$ proving the claim.
2. $Def(\psi, \sigma, \tau) \in \Delta_i$. By definition of δ_{Poss} in Def. 3.41, $Pre_\delta(\sigma, \tau) \wedge Safe_\delta(\sigma, \tau) \in E_i$, whereby we can conclude $Poss(\alpha, \sigma, \tau) \supset \neg Dir(\neg\psi, \alpha, \sigma, \tau) \in E_i$. □

Note that conflicts already arise with conditional, local effects; the framework however makes sure there are no conflicts even for conditional, non-local, disjunctive effects.

The existence of extensions for domain axiomatisations with state defaults can still be guaranteed for the extended framework. Additionally, it is easy to see that the domain specifications provided by the user are still modular: different parts of the specifications, such as conditional effect expressions and state defaults, are completely independent of each other from a user's point of view. Yet, the intricate semantic interactions between them are correctly dealt with.

3.8 Concluding Remarks

Relation to Reiter's Successor State Axioms

By a result due to Vadim Zaslawski [Baumann et al., 2010, Theorem 2], the effect axioms (3.16) with direct effects and a solution to the frame problem instantiated by a fluent F are in principle Reiter-style successor state axioms [Reiter, 1991] enhanced by the consistency criterion

$$\neg(\gamma_F^+(a, s) \wedge \gamma_F^-(a, s)) \tag{3.32}$$

which requires that an action a must not make F both true and false at the same time. In the Situation Calculus, it is left to the axiomatiser to make sure the consistency condition is satisfied for all actions. In particular, the violation of (3.32) in a Situation Calculus action theory may keep consistency but still give nonsensical results.

Example 3.48 (Dead or Alive)**.** Ann and Bob want to axiomatise an action domain that involves shooting turkeys.[10] They agree to use the Situation Calculus with reified fluents and Reiter-style successor state axioms

$$Holds(F, Do(a, s)) \equiv (\gamma_F^+(a, s) \vee (Holds(F, s) \wedge \neg\gamma_F^-(a, s)))$$

A single fluent shall describe whether the turkey is still living, the only action Shoot is intended to end the turkey's poor existence. The two come up with the following axiomatisations:

[10] It remains unclear why in the world they (or anybody else, for that matter) would want to do that.

Ann specifies a fluent Alive that denotes whether the turkey is still alive and characterises the action Shoot by the negative effect of making Alive false, that is, setting $\gamma^{-}_{\mathsf{Alive}}(a,s) \stackrel{\text{def}}{\equiv} (a = \mathsf{Shoot})$. The resulting successor state axiom is, quite correctly,

$$\mathit{Holds}(\mathsf{Alive}, \mathit{Do}(a,s)) \equiv (\mathit{Holds}(\mathsf{Alive}, s) \wedge a \neq \mathsf{Shoot})$$

Bob, on the other hand, takes a different approach. He uses the fluent Dead to indicate that the turkey is not alive and axiomatises the action Shoot by the positive effect of making Dead true, i.e. $\gamma^{+}_{\mathsf{Dead}}(a,s) \stackrel{\text{def}}{\equiv} (a = \mathsf{Shoot})$. His (equally correct) successor state axiom looks like this:

$$\mathit{Holds}(\mathsf{Dead}, \mathit{Do}(a,s)) \equiv (a = \mathsf{Shoot} \vee (\mathit{Holds}(\mathsf{Dead}, s)))$$

Both of them "verify" their axiomatisations by observing that the resulting formulas entail the desired result that an initially vivid turkey drops dead after shooting it.

But now imagine they had wanted to axiomatise a more challenging example, and both of them fell prey to a complicated subtlety of a sophisticated domain, which expresses itself in that they violated assumption (3.32) for some action. For the purpose of illustration, transfer this error scenario back to our example domain concerning the shooting of turkeys. Here, in Ann's axiomatisation Σ_A, the (erroneous) successor state axiom looks thus:

$$\mathit{Holds}(\mathsf{Alive}, \mathit{Do}(a,s)) \equiv (a = \mathsf{Shoot} \vee (\mathit{Holds}(\mathsf{Alive}, s) \wedge a \neq \mathsf{Shoot}))$$

In Bob's version of the domain, Σ_B, the same error has resulted in

$$\mathit{Holds}(\mathsf{Dead}, \mathit{Do}(a,s)) \equiv (a = \mathsf{Shoot} \vee (\mathit{Holds}(\mathsf{Dead}, s) \wedge a \neq \mathsf{Shoot}))$$

During reasoning, Ann finds that $\Sigma_A \models \mathit{Holds}(\mathsf{Alive}, \mathit{Do}(\mathsf{Shoot}, S_0))$ contrary to her expectations, and concludes she made a mistake. Bob observes that $\Sigma_B \models \mathit{Holds}(\mathsf{Dead}, \mathit{Do}(\mathsf{Shoot}, S_0))$ as he desired, and assumes his axiomatisation is correct.

The advertent reader however knows that neither of their axiomatisations is free of errors: they are structurally identical, only the meaning of fluents and effects of actions are swapped. Had the two initially agreed to use the dual form of the successor state axioms

$$\neg\mathit{Holds}(F, \mathit{Do}(a,s)) \equiv (\gamma^{-}_{F}(a,s) \vee (\neg\mathit{Holds}(F,s) \wedge \neg\gamma^{+}_{F}(a,s)))$$

they would draw the opposite conclusions

$$\Sigma'_A \models \neg\mathit{Holds}(\mathsf{Alive}, \mathit{Do}(\mathsf{Shoot}, S_0)) \text{ and } \Sigma'_B \models \neg\mathit{Holds}(\mathsf{Dead}, \mathit{Do}(\mathsf{Shoot}, S_0))$$

The cause of this problem is the asymmetry of the successor state axiom, which is not equivalent to its dual and prefers the positive effect in case of conflict. Using such an asymmetric effect axiom complicates the debugging process since axiomatisation errors like the above go unnoticed easily. Although the error in this example was quite obvious and involved only direct effects, this need not always be the case. The same error would manifest itself if a direct effect contradicted the final effect of an arbitrarily long chain of indirect effects.

Our effect axioms (3.16) on the other hand, directly entail (3.32)'s equivalent

$$\neg(\mathit{CausedT}(f,a,s,t) \wedge \mathit{CausedF}(f,a,s,t))$$

which means that a violation of this consistency principle immediately makes the effect axioms and thus the domain axiomatisation inconsistent.

Further Generalisations

While the action part of our action default theories is quite general, the state defaults are still restricted to literal consequents. To gain expressivity, it might be worthwhile to generalise state defaults to consequents that are arbitrary formulas. Alas, this would also remarkably increase the axiomatisation's complexity: Consider a "general" state default $\delta = \underline{\texttt{normally}}\ \Psi\ \underline{\texttt{if}}\ \Phi$ where Φ and Ψ are formulas. Translating the state default to a (normal) Reiter default seems fairly easy – replace all negative fluent literals $\neg\varphi$ occurring in Ψ by $\mathit{DefF}(\varphi, s, t)$ and symmetrically all positive fluent literals φ in Ψ by $\mathit{DefT}(\varphi, s, t)$. But now the trouble begins: how should we treat interactions of direct/indirect and default effects as in Example 3.39? It is not easy to identify conditional action effects that might possibly lead to conflicts. And even if we restrict ourselves to unconditional effects, how should default closure axioms for such general Reiter defaults look like? For both cases, trying to generalise our current approaches seems to be prone to be too restrictive towards possible default conclusions.

A user might sometimes even desire the expressivity of general, non-normal defaults. In addition to the problems mentioned above, this might deprive the default theories of extension existence and brings along the risk of introducing errors that are hard to spot.

Example 3.49 (Meta-Level Domain Constraint)**.** Consider the default $\mathit{Holds}(\mathsf{F}, s) : \neg P / P \in \Delta$ of an action default theory (Σ, Δ) with general defaults. Whenever Σ entails $\mathit{Holds}(\mathsf{F}, \sigma)$ for some time point σ, the default theory has no extension. In a way, the default can thus be seen as a state constraint saying "F may never be true." However, this is not reflected in that the default theory entails $\neg\mathit{Holds}(F, s)$ for all time points, but it is only discovered in case of violation.

In some cases, non-normal features of defaults are only used to encode specificity as in

$$\frac{\mathit{Bird}(x) : \mathit{Flies}(x), \neg\mathit{Penguin}(x)}{\mathit{Flies}(x)} \text{ and } \frac{\mathit{Penguin}(x) : \neg\mathit{Flies}(x)}{\neg\mathit{Flies}(x)}$$

saying that birds normally fly unless they are penguins, that normally do not fly. For this purpose the user should rather use explicit qualitative preferences and preferred default logic [Brewka, 1994; Brewka and Eiter, 1999; Delgrande and Schaub, 2000] such as $\delta_{\mathit{Penguin}} \prec \delta_{\mathit{Bird}}$. (See also Section 6.2.2.)

Chapter 4

Implementation

This thesis has so far stayed entirely on the theoretical level. In this chapter, we present an actual implementation of default reasoning about actions. The implemented system takes as input a $\mathcal{D}$ action domain specification and answers queries about the domain. To the best of our knowledge, this system is the first implementation of default reasoning in action theories.

Of course, we cannot expect to be able to implement full-fledged reasoning in Reiter's default logic: after all, extension existence for closed normal first-order default theories is not even semi-decidable [Reiter, 1980]. A high computational complexity is retained even through restriction to propositional logic: sceptical reasoning is Π_2^{P}-complete for propositional normal default theories [Gottlob, 1992]. Thus, we will have to make some restricting assumptions on our input domains that make an implementation feasible in principle (that is, make the relevant reasoning problems decidable) and in practice (that is, allow for efficient implementations of reasoning procedures).

But even for a sufficiently expressive yet simple enough input language, implementing a reasoner from scratch is complicated and error-prone. Although it could be adapted and tuned for efficiency, its range of applicability would be rather limited. Fortunately, there exist efficient general-purpose reasoners for default theories of a particular form: answer set programming solvers.

In Theorem 2.13 we have seen that there is a close connection between default logic and answer set programming. This connection will form the basis of our implementation. For finite domains and a finite time horizon, current ASP technology can be used off the shelf to efficiently reason about action domains. This efficiency is due to significant advancements the area of answer set programming has made in the last decade. Current solvers can treat programs with millions of variables despite the theoretical worst-case NP-hardness. Still, if the polynomial hierarchy does not collapse there remains an exponential gap between $\Pi_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$ (sceptical reasoning in propositional normal default theories) and NP (answer set existence for normal logic programs).

When aiming for an approach that compiles action domain specifications into logic programs and reduces reasoning about the domains to reasoning in these programs, we therefore have to expect a worst-case exponential blowup or at least a worst-case exponential runtime for these translations. This is acceptable if we compile only once to ask multiple queries afterwards. But in the case of an agent situated in a changing domain that constantly executes actions and makes new observations, it may be too costly to recompile its whole world knowledge after adding to it the observations associated to each action. Consequently, we are looking for a translation where the domain description is compiled once and additional in-

formation about action executions and observations can be added in a *modular* way. Formally, for two languages L_1, L_2, a translation function $f : 2^{L_1} \to 2^{L_2}$ is *modular* iff for all $A \subseteq L_1$, the function satisfies $f(A) = \bigcup_{a \in A} f(\{a\})$, that is, sets of language elements can be translated element-wise.

Also, since we are dealing with a non-monotonic formalism, another issue is getting into the way of a straightforward implementation: for monotonic semantics, there exists the possibility of an efficient implementation that is sound, but not complete. (Examples of such systems are [Gelfond and Lifschitz, 1993; Thielscher, 2005a].) With non-monotonic semantics, this possibility does not exist in general: for any conclusion that follows semantically but is not made by the system due to incompleteness, there may exist a default that relies on absence of this conclusion and thus makes an unsound default conclusion. (In the case of default logic this works even with only supernormal defaults.)

The system we present in this chapter is motivated by these reflections and rests on a fragment of $\mathcal{D}$ for which a sound and complete modular translation from the resulting action default theories into answer set programs exists. Figure 4.1 on the right details the architecture of the translation and sets out a roadmap for the rest of the chapter.

* * *

We begin the rest of the chapter by presenting the translation in detail and identifying a class of domains for which a sound and complete modular translation to logic programs exists. As we will see afterwards (Section 4.2), the efficiency of the overall translation can be greatly improved by simply swapping two of the constituent steps. We show that this change does not affect the semantics of the translation. In Section 4.3 we sketch our implementation prototype. Section 4.4 discusses related work and concludes.

4.1 From Action Domain Specifications to Answer Set Programs

Marek and Truszczyński's translation from normal logic programs yields propositional definite Horn default theories. In order to reverse the translation, we need to create default theories of this precise form. This will be done in two steps: first, we transform the first-order default theories into propositional default theories; afterwards, the propositional theories are transformed into definite Horn default theories. To illustrate the various translation steps, we use the following running example domain.

Example 4.1 (Swipe Card Domain)**.** The objective of this domain is to open an electronically Locked door using a swipe card. If the agent has a card (HasCard), it can Swipe the card to unlock the door; if the door is unlocked and not Jammed, it can be Pushed Open. Normally, the door is not jammed. The $\mathcal{D}$ action domain specification is

$$\begin{aligned}\Theta_{SwipeCard} = \{&\texttt{possible}\ \mathsf{Swipe}\ \texttt{iff}\ \mathsf{HasCard},\\ &\texttt{possible}\ \mathsf{Push}\ \texttt{iff}\ \neg\mathsf{Locked} \wedge \neg\mathsf{Jammed},\\ &\texttt{action}\ \mathsf{Swipe}\ \texttt{causes}\ \neg\mathsf{Locked},\\ &\texttt{action}\ \mathsf{Push}\ \texttt{causes}\ \mathsf{Open},\\ &\texttt{normally}\ \neg\mathsf{Jammed}\}\end{aligned}$$

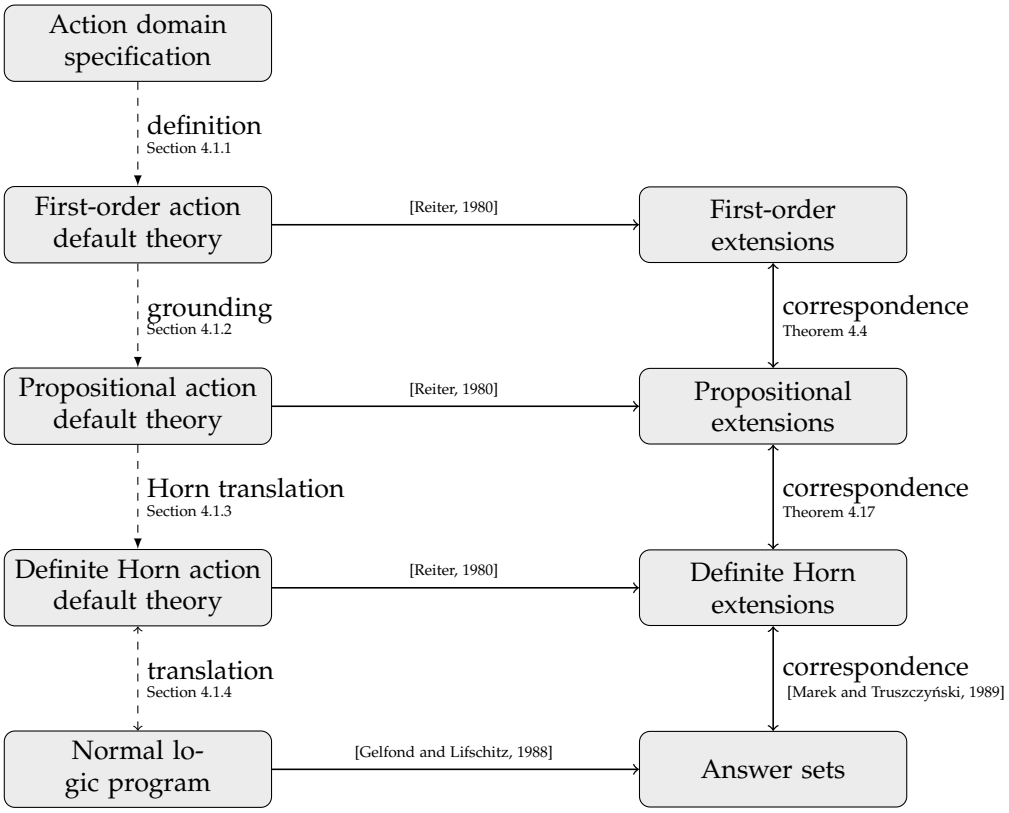


Figure 4.1: Idea behind the implementation. The previous chapter defined how action domain specifications (top layer) are translated into first-order default theories. Consequently, the meaning of these specifications is defined via the extensions of action default theories (second layer). For finite domains, first-order default theories can be replaced by propositional default theories (layer three). A certain class of domains can be faithfully transformed in definite Horn action default theories. By Theorem 2.13 their extensions correspond one-to-one to the answer sets of the respective logic programs. Existing answer set programming systems can efficiently compute answer sets of normal logic programs (bottom layer).

4.1.1 From Action Domain Specifications to CNF Default Theories

In the previous chapter, we have defined how to turn action domain specifications of various $\mathcal{D}$ dialects into default theories. For the purpose of this chapter, we modify the definitions from the previous chapter to create open default theories where all formulas are quantifier-free. This is possible because we will be dealing with a $\mathcal{D}$ fragment whose translations do not use existential quantification. Although we could equally well keep the universal quantifiers, our reasons for dropping them will become clear in the next section. We also assume that the formulas in domain axiomatisations are in conjunctive normal form to simplify the definitions in the sequel.

Additionally, we make some more syntactical manipulations. First, we turn the macros *FrameT* and *FrameF* into predicates and add the respective defining axioms. The same is done for *Violated* and *Inapplicable*, except that for supernormal state defaults $\delta = \underline{\texttt{normally}}\ \psi$ we have $\mathit{Inapplicable}_\delta(s,t) \equiv \mathit{Violated}_\delta(s) \equiv \neg\psi[s]$ and do not need these additional predicates. For syntactic uniformity, we introduce a new predicate $\mathit{Occurs} : \textsc{action} \times \textsc{time} \times \textsc{time}$ that

speaks about hypothetical or actual action occurrence: for situations, $Occurs(a, s, t)$ means hypothetical occurrence $t = Do(a, s)$ (t is the hypothetical time point that results from executing a in s); for linear time, $Occurs(a, s, t)$ means actual occurrence $Holds(\mathsf{Happens}(a, s, t), s)$ (action a happened from s to t). We do not translate axiomatisations of time points into answer set programs, but instead explicitly enumerate all the time points we will need and distinguish the initial one by the predicate $Init : \textsc{time}$ (that was used as a macro before). For linear time, this only turns the domain of sort TIME from infinite to finite. For situations, the foundational axioms for situations are adequately represented in our translation: axiom (2.5) saying that S_0 is the initial situation is expressed by the atom $Init(S_0)$; the next axiom (2.6) defines the ordering $<$ on situations which we do not use; (2.7) is a unique-names axiom for Do, which is implicitly encoded in our target language; finally, the second-order induction axiom (2.8) is not needed since we do not intend to prove properties about *all* situations.

The creation of Reiter defaults Δ remains unchanged, that is, as in Definition 3.41. At this point in the translation, we have a quantifier-free default theory (Σ, Δ) that speaks about the domain in general.

Example 4.1 (Continued)**.** The quantifier-free action default theory of the swipe card domain is $(\Sigma^{SwipeCard}, \Delta^{SwipeCard})$ below. We have omitted the conversion to CNF for readability. In $\Sigma^{SwipeCard}$, we find the precondition axioms

$$Poss(\mathsf{Swipe}, t_2, t_3) \equiv (Holds(\mathsf{HasCard}, t_2) \wedge Occurs(\mathsf{Swipe}, t_2, t_3))$$
$$Poss(\mathsf{Push}, t_2, t_3) \equiv ((\neg Holds(\mathsf{Locked}, t_2) \wedge \neg Holds(\mathsf{Jammed}, t_2)) \wedge Occurs(\mathsf{Push}, t_2, t_3))$$

the effect axiom (UAC-compliant effect axioms can be obtained by instantiating a_1)

$$Poss(a_1, t_2, t_3) \supset Holds(f_4, t_3) \equiv (FrameT(f_4, t_2, t_3) \vee (DirT(f_4, a_1, t_2, t_3) \vee DefT(f_4, t_2, t_3))) \wedge$$
$$\neg Holds(f_4, t_3) \equiv (FrameF(f_4, t_2, t_3) \vee (DirF(f_4, a_1, t_2, t_3) \vee DefF(f_4, t_2, t_3)))$$

the direct effect formulas

$$\neg DirT(\mathsf{Locked}, \mathsf{Push}, t_2, t_3) \qquad \neg DirF(\mathsf{Locked}, \mathsf{Push}, t_2, t_3)$$
$$DirT(\mathsf{Open}, \mathsf{Push}, t_2, t_3) \qquad \neg DirF(\mathsf{Open}, \mathsf{Push}, t_2, t_3)$$
$$\neg DirT(\mathsf{HasCard}, \mathsf{Push}, t_2, t_3) \qquad \neg DirF(\mathsf{HasCard}, \mathsf{Push}, t_2, t_3)$$
$$\neg DirT(\mathsf{Jammed}, \mathsf{Push}, t_2, t_3) \qquad \neg DirF(\mathsf{Jammed}, \mathsf{Push}, t_2, t_3)$$
$$\neg DirT(\mathsf{Locked}, \mathsf{Swipe}, t_2, t_3) \qquad DirF(\mathsf{Locked}, \mathsf{Swipe}, t_2, t_3)$$
$$\neg DirT(\mathsf{Open}, \mathsf{Swipe}, t_2, t_3) \qquad \neg DirF(\mathsf{Open}, \mathsf{Swipe}, t_2, t_3)$$
$$\neg DirT(\mathsf{HasCard}, \mathsf{Swipe}, t_2, t_3) \qquad \neg DirF(\mathsf{HasCard}, \mathsf{Swipe}, t_2, t_3)$$
$$\neg DirT(\mathsf{Jammed}, \mathsf{Swipe}, t_2, t_3) \qquad \neg DirF(\mathsf{Jammed}, \mathsf{Swipe}, t_2, t_3)$$

the default closure axioms

$$\neg DefT(\mathsf{Locked}, t_2, t_3) \qquad \neg DefF(\mathsf{Locked}, t_2, t_3)$$
$$\neg DefT(\mathsf{HasCard}, t_2, t_3) \qquad \neg DefF(\mathsf{HasCard}, t_2, t_3)$$
$$\neg DefT(\mathsf{Open}, t_2, t_3) \qquad \neg DefF(\mathsf{Open}, t_2, t_3)$$
$$\neg DefT(\mathsf{Jammed}, t_2, t_3) \qquad Holds(\mathsf{Jammed}, t_2) \supset \neg DefF(\mathsf{Jammed}, t_2, t_3)$$

the macro definition axioms

$$FrameT(f_4, t_2, t_3) \equiv (Holds(f_4, t_2) \wedge Holds(f_4, t_3))$$
$$FrameF(f_4, t_2, t_3) \equiv (\neg Holds(f_4, t_2) \wedge \neg Holds(f_4, t_3))$$

The Reiter defaults are

$$\Delta^{SwipeCard} = \left\{ \frac{Init(t_5) : \neg Holds(\mathsf{Jammed}, t_5)}{\neg Holds(\mathsf{Jammed}, t_5)}, \frac{\neg Holds(\mathsf{Jammed}, t_2) : DefF(\mathsf{Jammed}, t_2, t_3)}{DefF(\mathsf{Jammed}, t_2, t_3)} \right\}$$

In addition to this general domain information, the user can specify information about certain instances of the domain. An instance is first characterised by a time structure, situations or linear time. Technically, this extends the signature of the domain by the appropriate function symbols into sort TIME. Second, the instance information contains additional world knowledge whose form depends on the chosen notion of time. For situations, the user can provide a characterisation of the initial situation via a conjunction of literals $(\neg)Holds(\varphi, S_0)$. For linear time, they can specify a narrative consisting of action occurrence statements and *Holds* literals.

This user-specified information is easily transformed into a set of formulas Σ_{inst}:

- in the case of situations,
  - the axioms $Init(S_0)$ and $Occurs(a, s, t) \equiv t = Do(a, s)$
  - ground literals $(\neg)Holds(\varphi_1, S_0), \ldots, (\neg)Holds(\varphi_m, S_0)$;
- in the case of linear time, a narrative consisting of
  - ground atoms $Occurs(\alpha_1, \sigma_1, \tau_1), \ldots, Occurs(\alpha_n, \sigma_n, \tau_n)$ for actual action occurrences and
  - the axiom $Init(\tau_0)$ for some ground τ_0, and ground literals $(\neg)Holds(\varphi_1, \tau_1), \ldots, (\neg)Holds(\varphi_k, \tau_k)$ for observations.

The instance formulas Σ_{inst} are added to the default theory about the domain, resulting in $(\Sigma \cup \Sigma_{inst}, \Delta)$.

Example 4.1 (Continued)**.** For situations as time structure and an initial time point where the agent has a card, we get

$$\Sigma_{inst}^{SwipeCard} = \{Init(S_0), Occurs(a_1, t_2, t_3) \equiv t_3 = Do(a_1, t_2), Holds(\mathsf{HasCard}, S_0)\}$$

4.1.2 ... to Propositional Default Theories

Although the semantics defined in the previous chapter can in principle deal with infinite domains, the same cannot be expected from a general implementation. In particular, answer set programming systems are based on propositional logic, which precludes the use of function symbols of positive arity in any way that leads to infinite domains. So for the implementation, we restrict our attention to domains with a finite number of objects. Furthermore, we assume for the time being that we are dealing with relational domain signatures, that contain no function symbols of positive arity. We will later see that these restrictions can be lifted in order to ease domain specification for the user.

To model a finite and fully known domain logically, we use so-called domain closure axioms. Generally, the domain closure axiom for a sort $\mathfrak{s}$ says that the set $\mathfrak{D}_\mathfrak{s}$ of constant symbols contains the *only* objects of this sort. For a sort $\mathfrak{s}$ of a relational signature with $\mathfrak{D} = \{C \mid C : \mathfrak{s} \in \mathfrak{F}\}$, the *domain closure axiom* for $\mathfrak{s}$ is

$$(\forall x : \mathfrak{s}) \bigvee_{C \in \mathfrak{D}} x = C$$

In the implementation, we express the same circumstance with a different device: when the domain of a sort is finite and fully known, we can syntactically replace formulas with universally quantified variables by ground conjunctions. There, each ground conjunct represents the proposition of the formula for a specific ground valuation of the variables. Universal quantification is expressed by having a ground conjunct for each possible valuation. Formally, this well-sorted grounding is defined as follows.

Definition 4.2. Let Ξ be a relational domain signature with a finite set $\mathfrak{D}_\mathfrak{s} = \{C \mid C : \mathfrak{s} \in \Xi\}$ for each of its sorts $\mathfrak{s}$. For a quantifier-free formula $\Phi(x_1, \ldots, x_n)$ with free variables $x_1 : \mathfrak{s}_1, \ldots, x_n : \mathfrak{s}_n$, its *well-sorted grounding* $\lfloor \Phi(x_1, \ldots, x_n) \rfloor$ is defined inductively by

$$\lfloor \Phi(\mathfrak{o}_1, \ldots, \mathfrak{o}_n) \rfloor \stackrel{\text{def}}{=} \Phi(\mathfrak{o}_1, \ldots, \mathfrak{o}_n) \text{ where } (\mathfrak{o}_1, \ldots, \mathfrak{o}_n) \in \mathfrak{D}_1 \times \cdots \times \mathfrak{D}_n$$
$$\lfloor \Phi(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, x_i, x_{i+1} \ldots, x_n) \rfloor \stackrel{\text{def}}{=} \bigwedge_{\mathfrak{o}_i \in \mathfrak{D}_i} \lfloor \Phi(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, \mathfrak{o}_i, x_{i+1}, \ldots, x_n) \rfloor$$

For a set W of formulas, $\lfloor W \rfloor \stackrel{\text{def}}{=} \{\lfloor \Phi \rfloor \mid \Phi \in W\}$.

For example, grounding the formula $P(x) \equiv x = A$ where $P : \mathfrak{s}$ and over the domain $\mathfrak{D}_\mathfrak{s} = \{A, B\}$ yields the formula $(P(A) \equiv A = A) \wedge (P(B) \equiv B = A)$ which is equivalent to $P(A) \wedge \neg P(B)$ under the unique-names assumption. For a more complicated example with another free variable, consider the clause $C(x, y) = P(x) \vee Q(x, y)$ with P as above and $Q : \mathfrak{s} \times \mathfrak{s}'$ where $\mathfrak{D}_{\mathfrak{s}'} = \{1, 2\}$. We get

$$\begin{aligned}
\lfloor C(x,y) \rfloor &= \lfloor P(x) \vee Q(x,y) \rfloor \\
&= \lfloor P(A) \vee Q(A,y) \rfloor \wedge \lfloor P(B) \vee Q(B,y) \rfloor \\
&= \lfloor P(A) \vee Q(A,1) \rfloor \wedge \lfloor P(A) \vee Q(A,2) \rfloor \wedge \lfloor P(B) \vee Q(B,1) \rfloor \wedge \lfloor P(B) \vee Q(B,2) \rfloor \\
&= (P(A) \vee Q(A,1)) \wedge (P(A) \vee Q(A,2)) \wedge (P(B) \vee Q(B,1)) \wedge (P(B) \vee Q(B,2))
\end{aligned}$$

In general, grounding involves a blowup in formula size that is exponential in the number of free variables and polynomial in the domain sizes: for a set W of quantifier-free formulas with n free variables and domains $\mathfrak{D}$, we have $\|\lfloor W \rfloor\| \leq \|W\| \cdot |\mathfrak{D}|^n$.

According to the following lemma, domain closure axioms ensure that the well-sorted grounding of formulas indeed captures the semantics of universal quantification over a finite and fully known domain. Notice that unique-names axioms are also needed here to establish a one-to-one correspondence between a sort's domain and its constant symbols.

Lemma 4.3. *Let $\Xi = (\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ be a relational domain signature. For each sort $\mathfrak{s}_i \in \mathfrak{S} = \{\mathfrak{s}_1, \ldots, \mathfrak{s}_m\}$, denote by $\mathfrak{D}_i \stackrel{\text{def}}{=} \{C \mid C : \mathfrak{s}_i \in \mathfrak{F}\}$ the set of constant symbols of this sort. Further, let $\Sigma_{dc} \cup \Sigma_{una}$ be the set of domain closure (with respect to $\mathfrak{D}_i$) and unique-names axioms for Ξ's sorts. For a quantifier-free formula $\Phi(x_1, \ldots, x_n)$ we have*

$$\Sigma_{dc} \cup \Sigma_{una} \models (\forall x_1, \ldots, x_n) \Phi(x_1, \ldots, x_n) \equiv \lfloor \Phi(x_1, \ldots, x_n) \rfloor$$

Proof. Let $\mathfrak{I}$ be an interpretation for Ξ with $\mathfrak{I} \models \Sigma_{dc} \cup \Sigma_{una}$. We first observe that this implies the existence of a bijection $f : \mathfrak{D}_{\mathfrak{s}_i}^\mathfrak{I} \to \mathfrak{D}_i$ for each sort $\mathfrak{s}_i$ with $1 \leq i \leq m$. (Since $\mathfrak{I}$ is an interpretation, the constant symbols of each sort must be interpreted by some domain elements; since $\mathfrak{I} \models \Sigma_{una}$, there must be at least $|\mathfrak{D}_i|$ distinct elements in each sort domain $\mathfrak{D}_{\mathfrak{s}_i}^\mathfrak{I}$; since $\mathfrak{I} \models \Sigma_{dc}$, there can be at most $|\mathfrak{D}_i|$ elements in each sort domain $\mathfrak{D}_{\mathfrak{s}_i}^\mathfrak{I}$.) In the proof, we will therefore without loss of generality assume that in each structure $\mathfrak{M}$ for Ξ, the sort domains $\mathfrak{D}_{\mathfrak{s}_i}^\mathfrak{I}$ are given by the

$\mathfrak{D}_i$. In the remainder, we proceed by induction on the number n of free variables. The base case is trivial. For the induction step, let

$$\Sigma_{dc} \cup \Sigma_{una} \models (\forall x_1, \ldots, x_n)\Phi(x_1, \ldots, x_n) \equiv \lfloor \Phi(x_1, \ldots, x_n) \rfloor \tag{IH}$$

for all Φ and consider the formula $\Psi(x_0, x_1, \ldots, x_n)$ where $x_0 : \mathfrak{s}_0$ and let $\mathfrak{I}$ be a model for $\Sigma_{dc} \cup \Sigma_{una}$.

$$\begin{aligned}
&\mathfrak{I} \models (\forall x_0, x_1, \ldots, x_n)\Psi(x_0, x_1, \ldots, x_n) \\
&\text{iff } \mathfrak{I}, x_0 \mapsto \mathfrak{o} \models (\forall x_1, \ldots, x_n)\Psi(x_0, x_1, \ldots, x_n) \text{ for all } \mathfrak{o} \in \mathfrak{D}_0 && \text{(Definition of } \models\text{)} \\
&\text{iff } \mathfrak{I} \models (\forall x_1, \ldots, x_n)\Psi(\mathfrak{o}, x_1, \ldots, x_n) \text{ for all } \mathfrak{o} \in \mathfrak{D}_0 \\
&\text{iff } \mathfrak{I} \models \lfloor \Psi(\mathfrak{o}, x_1, \ldots, x_n) \rfloor \text{ for all } \mathfrak{o} \in \mathfrak{D}_0 && \text{(IH)} \\
&\text{iff } \mathfrak{I} \models \bigwedge_{\mathfrak{o} \in \mathfrak{D}_0} \lfloor \Psi(\mathfrak{o}, x_1, \ldots, x_n) \rfloor && (\mathfrak{D}_0 \text{ is finite}) \\
&\text{iff } \mathfrak{I} \models \lfloor \Psi(x_0, x_1, \ldots, x_n) \rfloor && \text{(Definition 4.2)}
\end{aligned}$$

□

Hence considering our quantifier-free action default theories for finite domains, we can equivalently replace them by essentially propositional action default theories. The expression "essentially propositional" captures the intuition that the syntax of first-order logic is used to ease specification but not semantically necessary.

Theorem 4.4. *Let Ξ be a relational domain signature, (Σ, Δ) be a quantifier-free action default theory over Ξ and let Σ_{dc} be the domain closure axioms with respect to the constants of all of Ξ's sorts. The default theories $(\Sigma_{dc} \cup \Sigma_{una} \cup \Sigma, \Delta)$ and $(\Sigma_{dc} \cup \Sigma_{una} \cup \lfloor \Sigma \rfloor, \Delta)$ are equivalent.*

Proof. The result is straightforward from Lemma 4.3, since we can replace each $\Phi \in \Sigma$ by its equivalent $\lfloor \Phi \rfloor$. □

For any such essentially propositional default theory, it is straightforward how to truly propositionalise it: for each ground atom $P(\vec{t})$ mentioned in the theory, we introduce a new null-ary predicate and replace all occurrences of $P(\vec{t})$ by the new predicate. Note that defaults with free variables are only notational representatives of their (well-sorted) ground instances, so any Δ is already essentially propositional. It is also finite due to the finiteness of the involved sort domains.

Definition 4.5. Let $\Xi = (\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ be a relational domain signature. For a ground atomic formula $P(\vec{t})$, its propositional counterpart $Prop(P(\vec{t}))$ is the new predicate $Q_{P(\vec{t})}$. For a composite formula Φ, its propositional counterpart $Prop(\Phi)$ is defined by structural induction:

$$\begin{aligned}
Prop(\top) &\stackrel{\text{def}}{=} \top \\
Prop(\bot) &\stackrel{\text{def}}{=} \bot \\
Prop(\mathfrak{t}_1 = \mathfrak{t}_2) &\stackrel{\text{def}}{=} \begin{cases} \top & \text{if } \mathfrak{t}_1 \text{ and } \mathfrak{t}_2 \text{ are identical} \\ \bot & \text{otherwise} \end{cases} \\
Prop(\neg\Phi) &\stackrel{\text{def}}{=} \neg Prop(\Phi) \\
Prop(\Phi_1 \circ \Phi_2) &\stackrel{\text{def}}{=} Prop(\Phi_1) \circ Prop(\Phi_2) \text{ for } \circ \in \{\wedge, \vee\}
\end{aligned}$$

For a closed, quantifier-free (i.e., ground) default, we have

$$Prop\left(\frac{\beta : \kappa_1, \ldots, \kappa_n}{\omega}\right) \stackrel{\text{def}}{=} \frac{Prop(\beta) : Prop(\kappa_1), \ldots, Prop(\kappa_n)}{Prop(\omega)}$$

Let (Σ, Δ) be a quantifier-free action default theory over Ξ. Its propositional counterpart is $Prop(\Sigma, \Delta) \stackrel{\text{def}}{=} (\{Prop(\Phi) \mid \Phi \in \Sigma\}, \{Prop(\delta) \mid \delta \in \Delta\})$ over the propositional signature that contains all $Q_{P(\vec{\mathfrak{t}})}$ for atoms $P(\vec{\mathfrak{t}})$ that occur in (Σ, Δ).

Note that ground atoms $P(\vec{\mathfrak{t}_1})$ and $P(\vec{\mathfrak{t}_2})$ where $\vec{\mathfrak{t}_1}$ and $\vec{\mathfrak{t}_2}$ differ will be mapped to different propositional variables. That way, the once explicit unique-names assumption is implicitly encoded into the language. For the sake of readability we drop the $Q_{P(\vec{\mathfrak{t}})}$ notation in the sequel and use ground first-order atoms and their propositionalisations interchangeably. In particular, we will speak about time points, fluents and actions as if using the grounded first-order axiomatisation.

4.1.3 . . . to Propositional Definite Horn Default Theories

Now we know how to translate action domain specifications into propositional action default theories in conjunctive normal form. The next step will turn a set of general clauses into a set of definite Horn clauses, that is, clauses of the form $P \vee \neg P_1 \vee \ldots \vee \neg P_m$ for $m \geq 0$. The obvious difference between such a definite Horn clause and a general clause is that the latter can contain any number of positive literals, while the former must contain exactly one. To conform to this restriction, we extend the signature to allow us to express negative literals of the original signature through atoms of the extended signature. The structure of the original clauses is conveyed by making explicit all implications contained in them. For example, for the clause $P \vee Q$ we express that in any model of the clause, (1) "P must be true if Q is false" and (2) "Q must be true if P is false." What goes missing, however, is the more general information that "at least one of P or Q must be true."

We start the presentation of our translation with (in a sense) removing classical negation from the underlying language. The basic idea of "reifying negation" is to duplicate the predicates of the language. So for each (possibly non-propositional, sorted) predicate symbol P, we add a placeholder $\text{-}P$ (of the same sort) for its negation. This goes back to [Gelfond and Lifschitz, 1991] and has become a standard method in ASP-based encodings of logical languages. Although it would suffice at this point to give the definition for propositional signatures only, we already introduce the general translation that we will need later.

Definition 4.6. Let $\Xi = (\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ be a signature. Define a new signature $\Xi^{\pm} = (\mathfrak{S}, \mathfrak{P}^{\pm}, \mathfrak{F}, \mathfrak{X})$ as follows. Set $\mathfrak{P}^{\pm} \stackrel{\text{def}}{=} \mathfrak{P} \cup \{\text{-}P : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \mid P : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \in \mathfrak{P}\}$ and furthermore add the new binary predicate $\neq$ for inequality.[1] For $P \in \mathfrak{P}$, the *positive name* $\langle\cdot\rangle^+$ and *negative name* $\langle\cdot\rangle^-$ of a $\mathfrak{P}^{\pm}$ literal are defined by

$$\begin{aligned}
\langle P(\vec{\mathfrak{t}})\rangle^+ &\stackrel{\text{def}}{=} P(\vec{\mathfrak{t}}) & \langle P(\vec{\mathfrak{t}})\rangle^- &\stackrel{\text{def}}{=} \text{-}P(\vec{\mathfrak{t}}) \\
\langle \neg P(\vec{\mathfrak{t}})\rangle^+ &\stackrel{\text{def}}{=} \text{-}P(\vec{\mathfrak{t}}) & \langle \neg P(\vec{\mathfrak{t}})\rangle^- &\stackrel{\text{def}}{=} P(\vec{\mathfrak{t}}) \\
\langle \text{-}P(\vec{\mathfrak{t}})\rangle^+ &\stackrel{\text{def}}{=} \text{-}P(\vec{\mathfrak{t}}) & \langle \text{-}P(\vec{\mathfrak{t}})\rangle^- &\stackrel{\text{def}}{=} P(\vec{\mathfrak{t}}) \\
\langle \neg\text{-}P(\vec{\mathfrak{t}})\rangle^+ &\stackrel{\text{def}}{=} P(\vec{\mathfrak{t}}) & \langle \neg\text{-}P(\vec{\mathfrak{t}})\rangle^- &\stackrel{\text{def}}{=} \text{-}P(\vec{\mathfrak{t}})
\end{aligned}$$

[1] Outside of this chapter, however, $\neq$ is a macro where $\mathfrak{t}_1 \neq \mathfrak{t}_2$ abbreviates $\neg(\mathfrak{t}_1 = \mathfrak{t}_2)$.

For equality literals, the positive and negative names are given by[2]

$$\langle t_1 = t_2\rangle^+ \stackrel{\text{def}}{=} t_1 = t_2 \qquad \langle t_1 = t_2\rangle^- \stackrel{\text{def}}{=} t_1 \neq t_2$$
$$\langle \neg(t_1 = t_2)\rangle^+ \stackrel{\text{def}}{=} t_1 \neq t_2 \qquad \langle \neg(t_1 = t_2)\rangle^- \stackrel{\text{def}}{=} t_1 = t_2$$

We first point out that for all literals ψ over $\mathfrak{P}^{\pm}$, the name functions satisfy $\langle\neg\psi\rangle^+ = \langle\psi\rangle^-$ and $\langle\neg\psi\rangle^- = \langle\psi\rangle^+$. The crucial thing to note for this definition is however that $\langle\cdot\rangle^+$ and $\langle\cdot\rangle^-$ always return positive atoms over $\mathfrak{P}^{\pm}$. The next definition now uses these name functions to encode the implications of a given clause into definite Horn clauses of the extended signature.

Definition 4.7. Let $c = l_1 \vee \ldots \vee l_m$ with $m \geq 1$ be a clause over a signature Ξ. The *definite Horn translation of clause* c is the conjunction of definite Horn clauses $[\![c]\!] \stackrel{\text{def}}{=} [\![c]\!]_1 \wedge \ldots \wedge [\![c]\!]_m$ over $\Xi^{\pm}$ where for $1 \leq i \leq m$

$$[\![c]\!]_i \stackrel{\text{def}}{=} \langle l_i\rangle^+ \vee \bigvee_{\substack{1\leq j\leq m,\\ j\neq i}} \neg\langle l_j\rangle^-$$

Accordingly, for a conjunction $C = c_1 \wedge \ldots \wedge c_n$ of clauses we define $[\![C]\!] \stackrel{\text{def}}{=} [\![c_1]\!] \wedge \ldots \wedge [\![c_n]\!]$; for a set Φ of CNFs, $[\![\Phi]\!] \stackrel{\text{def}}{=} \{[\![C]\!] \mid C \in \Phi\}$. The empty clause remains unchanged by the translation, $[\![\bot]\!] \stackrel{\text{def}}{=} \bot$.

For a clause $c = l_1 \vee \ldots \vee l_m$, the construction for $[\![c]\!]$ above expands to

$$\begin{aligned}
[\![c]\!]_1 &= \langle l_1\rangle^+ \vee \neg\langle l_2\rangle^- \vee \neg\langle l_3\rangle^- \vee \ldots \vee \neg\langle l_m\rangle^-\\
[\![c]\!]_2 &= \langle l_2\rangle^+ \vee \neg\langle l_1\rangle^- \vee \neg\langle l_3\rangle^- \vee \ldots \vee \neg\langle l_m\rangle^-\\
&\vdots\\
[\![c]\!]_m &= \langle l_m\rangle^+ \vee \neg\langle l_1\rangle^- \vee \neg\langle l_2\rangle^- \vee \ldots \vee \neg\langle l_{m-1}\rangle^-
\end{aligned}$$

Each of the individual clauses encodes an implication contained in the original clause; for example, $[\![c]\!]_1$ expresses $(\langle l_2\rangle^- \wedge \langle l_3\rangle^- \wedge \ldots \wedge \langle l_m\rangle^-) \supset \langle l_1\rangle^+$ that says "l_1 is true if all other literals are false." There, falsity of original atoms is modelled through truth of auxiliary atoms. Note that a unit clause $c = l_1$ becomes an atom $[\![c]\!] = \langle l_1\rangle^+$.

The first and most important property of the translation $[\![\cdot]\!]$ is that it is sound with respect to consequences in propositional logic. To show this, we will first define how to extend interpretations for a propositional signature $\mathfrak{P}$ to interpretations for the extended signature $\mathfrak{P}^{\pm}$.

Definition 4.8. Let $\mathfrak{P}$ be a propositional signature and I be an interpretation for $\mathfrak{P}$. The interpretation's *Horn extension* $[\![I]\!]$ is an interpretation for $\mathfrak{P}^{\pm}$ that coincides with I on $\mathfrak{P}$, and for each predicate $\text{-}P \in \mathfrak{P}^{\pm} \setminus \mathfrak{P}$, we have

$$[\![I]\!](\text{-}P) \stackrel{\text{def}}{=} \begin{cases} \mathsf{f} & \text{if } I(P) = \mathsf{t}\\ \mathsf{t} & \text{otherwise}\end{cases}$$

It immediately follows from this definition that for an interpretation I and an atom $P \in \mathfrak{P}$, we have $I \models P$ iff $[\![I]\!] \models P$ iff $[\![I]\!] \not\models \text{-}P$. Conversely, for $\text{-}P \in \mathfrak{P}^{\pm} \setminus \mathfrak{P}$, it holds that $I \models \neg P$ iff $[\![I]\!] \models \text{-}P$ iff $[\![I]\!] \not\models P$. This leads to the following lemma.

[2]Strictly technically speaking, introducing inequality as a separate predicate requires the existence of a universal supersort, say TERM, which allows us to set $\neq$: TERM $\times$ TERM. This is however straightforward to achieve and we do not present it for the sake of readability.

Lemma 4.9. *Let $C = l_1 \vee \ldots \vee l_m$ be a propositional clause and I be an interpretation for C's signature.*

$$I \models C \textit{ iff } [\![I]\!] \models [\![C]\!]$$

Proof.

$$\begin{aligned}
& I \models C \\
\text{iff } & I \models l_1 \vee \ldots \vee l_m \\
\text{iff } & I \models l_1 \text{ or } \ldots \text{ or } I \models l_m \\
\text{iff } & [\![I]\!] \models \langle l_1 \rangle^+ \text{ or } \ldots \text{ or } [\![I]\!] \models \langle l_m \rangle^+ \\
\text{iff } & [\![I]\!] \models \langle l_i \rangle^+ \text{ or } [\![I]\!] \models \langle l_1 \rangle^+ \text{ or } \ldots \text{ or } [\![I]\!] \models \langle l_{i-1} \rangle^+ \text{ or } [\![I]\!] \models \langle l_{i+1} \rangle^+ \text{ or } \ldots \\
& \quad \text{or } [\![I]\!] \models \langle l_m \rangle^+ \text{ for all } i \\
\text{iff } & [\![I]\!] \models \langle l_i \rangle^+ \text{ or } [\![I]\!] \not\models \langle l_1 \rangle^- \text{ or } \ldots \text{ or } [\![I]\!] \not\models \langle l_{i-1} \rangle^- \text{ or } [\![I]\!] \not\models \langle l_{i+1} \rangle^- \text{ or } \ldots \\
& \quad \text{or } [\![I]\!] \not\models \langle l_m \rangle^- \text{ for all } i \\
\text{iff } & [\![I]\!] \models \langle l_i \rangle^+ \text{ or } [\![I]\!] \models \neg\langle l_1 \rangle^- \text{ or } \ldots \text{ or } [\![I]\!] \models \neg\langle l_{i-1} \rangle^- \text{ or } [\![I]\!] \models \neg\langle l_{i+1} \rangle^- \text{ or } \ldots \\
& \quad \text{or } [\![I]\!] \models \neg\langle l_m \rangle^- \text{ for all } i \\
\text{iff } & [\![I]\!] \models \langle l_i \rangle^+ \text{ or } [\![I]\!] \models \neg\langle l_1 \rangle^- \vee \ldots \vee \neg\langle l_{i-1} \rangle^- \vee \neg\langle l_{i+1} \rangle^- \vee \ldots \vee \neg\langle l_m \rangle^- \text{ for all } i \\
\text{iff } & [\![I]\!] \models \langle l_i \rangle^+ \vee \neg\langle l_1 \rangle^- \vee \ldots \vee \neg\langle l_{i-1} \rangle^- \vee \neg\langle l_{i+1} \rangle^- \vee \ldots \vee \neg\langle l_m \rangle^- \text{ for all } i \\
\text{iff } & [\![I]\!] \models [\![C]\!]
\end{aligned}$$

□

This model correspondence holds of course only for "meaningful" interpretations $[\![I]\!]$ that correspond to interpretations I of the original signature. Using the lemma, soundness of $[\![\cdot]\!]$ is easy to prove.

Theorem 4.10. *Let $W \cup \{C\}$ be a set of propositional clauses.*

$$[\![W]\!] \models [\![C]\!] \textit{ implies } W \models C$$

Proof. We show the contrapositive. Let $W \not\models C$. Then there is an interpretation I with $I \models W$ and $I \not\models C$. By Lemma 4.9, we have $[\![I]\!] \models [\![W]\!]$ and $[\![I]\!] \not\models [\![C]\!]$, thus $[\![W]\!] \not\models [\![C]\!]$. □

Another useful property of the translation is that it preserves a clause's potential for unit resolution. This will later be used to show that our approach allows a modular translation while still being sound and complete. Since the proof of this lemma below is quite symbolic, we will first illustrate the property with an example.

Example 4.11. Consider the clause $C_1 = P \vee Q \vee \neg R$ and the unit clause $C_2 = \neg Q$. Obviously, C_1 and C_2 are resolvable with $Res(C_1, C_2) = P \vee \neg R$. The Horn translation of the resolvent is $[\![Res(C_1, C_2)]\!] = (P \vee \neg R) \wedge (\text{-}R \vee \neg\text{-}P)$. Applying the Horn translation to C_1 and C_2 individually yields

$$[\![C_1]\!] = (P \vee \neg\text{-}Q \vee \neg R) \wedge (\neg\text{-}P \vee Q \vee \neg R) \wedge (\neg\text{-}P \vee \neg\text{-}Q \vee \text{-}R) \qquad [\![C_2]\!] = \text{-}Q$$

We can see that $[\![C_2]\!]$ is resolvable with $[\![C_1]\!]_1$ and $[\![C_1]\!]_3$. The respective resolvents are easily verified to be identical to $[\![Res(C_1, C_2)]\!]$.

As mentioned above, this preservation of unit-resolvability always holds. The proof can be visualised as follows: For a clause of length m, the Horn translation creates an $m \times m$-array of literals. From this array, we remove one row (the conjunct that is not resolvable) and one column (the literals complementary to the unit clause we resolve with), which results in an $m-1 \times m-1$-array. Removing literal i from the original clause and then creating the Horn translation from the clause of length $m-1$ results in the same array.

Lemma 4.12. *Let $C = l_1 \vee \ldots \vee l_m$ with $m \geq 2$ be a non-unit clause and C' be a unit clause. If C and C' are resolvable, then $[\![C]\!]$ and $[\![C']\!]$ are resolvable and $[\![Res(C, C')]\!]$ is subsumed by $Res([\![C]\!], [\![C']\!])$.*

Proof. Applying Definition 4.7 to C yields

$$[\![C]\!] = \bigwedge_{1 \leq j \leq m} \left(\langle l_j \rangle^+ \vee \bigvee_{\substack{1 \leq k \leq m, \\ k \neq j}} \neg \langle l_k \rangle^- \right)$$

Now assume w.l.o.g. that $l_i = \neg C'$. This implies $[\![C']\!] = \langle C' \rangle^+ = \langle l_i \rangle^-$. Since $m \geq 2$, there exists a j with $1 \leq j \leq m$ and $j \neq i$. Furthermore, for all such j, the clause $[\![C]\!]_j$ contains the literal $\neg \langle l_i \rangle^-$ and is therefore resolvable with $[\![C']\!] = \langle l_i \rangle^-$. For each j, this leads to the resolvent

$$Res\Big([\![C]\!]_j, [\![C']\!]\Big) = \langle l_j \rangle^+ \vee \bigvee_{\substack{1 \leq k \leq m \\ k \neq j, k \neq i}} \neg \langle l_k \rangle^-$$

On the other hand, $Res(C, C') = l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_m = \bigvee_{\substack{1 \leq j \leq m, \\ j \neq i}} l_j$ and

$$[\![Res(C, C')]\!] = \bigwedge_{\substack{1 \leq j \leq m, \\ j \neq i}} \left(\langle l_j \rangle^+ \vee \bigvee_{\substack{1 \leq k \leq m \\ k \neq i, k \neq j}} \neg \langle l_k \rangle^- \right) = \bigwedge_{\substack{1 \leq j \leq m, \\ j \neq i}} Res\Big([\![C]\!]_j, [\![C']\!]\Big)$$

□

In a strictly syntactical sense, the idea of the lemma even holds for arbitrary pairs of non-unit clauses. However, due to the syntactical replication of negation, there is a catch.

Example 4.13. Consider the clauses $C_1 = P \vee Q$ and $C_2 = \neg P \vee Q$. They are obviously resolvable with resolvent Q. Their translations $[\![C_1]\!] = (P \vee \neg\text{-}Q) \wedge (\neg\text{-}P \vee Q)$ and $[\![C_2]\!] = (\text{-}P \vee \neg\text{-}Q) \wedge (\neg P \vee Q)$ are also resolvable. Their resolution results in the clauses $Q \vee \neg\text{-}Q$ and $\neg\text{-}Q \vee Q$. Not incidentally, these clauses are the Horn translation of $Q \vee Q$ and thus illustrate the general incompleteness of $[\![\cdot]\!]$ with respect to clausal entailment: the translation does not perform the (implicit) reduction step from $Q \vee Q$ to Q – in the resulting signature, Q and $\text{-}Q$ are just like *any* two atoms. This issue cannot be fixed in a way that is modular with respect to clauses, since it arises only from the interaction of different clauses.

Observe that Lemma 4.12 also does not hold in general if both clauses are unit: $[\![Res(P, \neg P)]\!] = [\![\bot]\!] = \bot$, but quite to the contrary the pair of clauses $([\![P]\!], [\![\neg P]\!]) = (P, \text{-}P)$ cannot be resolved. However, the unsatisfiability of $P \wedge \neg P$ which was removed by replacing $\neg P$ by $\text{-}P$ will still be detected thanks to the second part of the following definition. The first part specifies how to transform defaults to be able to reverse the translation of Theorem 2.13.

Definition 4.14. Let $\delta = \beta : \kappa/\omega$ be a quantifier-free disjunction-free default over a signature Ξ. The *corresponding definite Horn default* of δ is

$$[\![\delta]\!] \stackrel{\text{def}}{=} \frac{[\![\beta]\!] : \neg\langle\kappa\rangle^-}{\langle\omega\rangle^+} \tag{4.1}$$

Let (W, D) be a quantifier-free default theory such that all formulas in W are in conjunctive normal form and all defaults in D are disjunction-free. Its *corresponding definite Horn default theory* is $[\![(W, D)]\!] \stackrel{\text{def}}{=} ([\![W]\!], [\![D]\!] \cup \lightning_\Xi)$ with

$$\begin{aligned} [\![D]\!] &\stackrel{\text{def}}{=} \{[\![\delta]\!] \mid \delta \in \Delta\} \\ \lightning_\Xi &\stackrel{\text{def}}{=} \left\{ \frac{P \wedge \text{-}P : \neg Q}{Q} \;\middle|\; P \in \mathfrak{P} \right\} \end{aligned} \tag{4.2}$$

where Q is a fresh null-ary predicate symbol not occurring in Ξ.

Note that for a disjunction-free default δ above, β is a conjunction of literals. Accordingly, $[\![\beta]\!]$ is a conjunction of atoms and the resulting default $[\![\delta]\!]$ is indeed definite Horn.

Example 4.1 (Continued)**.** The defaults of the swipe card domain are translated as follows. For any ground valuation for t_2, t_3, t_5,

$$\frac{\mathit{Init}(t_5) : \neg\mathit{Holds}(\mathsf{Jammed}, t_5)}{\neg\mathit{Holds}(\mathsf{Jammed}, t_5)} \text{ is translated to } \frac{\mathit{Init}(t_5) : \neg\mathit{Holds}(\mathsf{Jammed}, t_5)}{\text{-}\mathit{Holds}(\mathsf{Jammed}, t_5)}, \text{ and}$$

$$\frac{\neg\mathit{Holds}(\mathsf{Jammed}, t_2) : \mathit{DefF}(\mathsf{Jammed}, t_2, t_3)}{\mathit{DefF}(\mathsf{Jammed}, t_2, t_3)} \text{ to } \frac{\text{-}\mathit{Holds}(\mathsf{Jammed}, t_2) : \neg\text{-}\mathit{DefF}(\mathsf{Jammed}, t_2, t_3)}{\mathit{DefF}(\mathsf{Jammed}, t_2, t_3)}$$

Now we have at our disposal a translation from default theories in CNF to definite Horn default theories. For a single clause, the blowup in formula size due to the definite Horn translation is obviously quadratic: a clause of length m becomes m clauses of length m. For our action default theories (Σ, Δ) the general increase in size is given by $\|[\![\Sigma]\!]\| \leq \|\Sigma\|^2$ and $\|[\![\Delta]\!]\| \leq \|\Delta\| + |\mathfrak{P}|$. The defaults do not increase in size, but only in number due to the integrity constraints for the predicates of the language.

Although the translation from default theories into definite Horn default theories presented thus far is defined for general disjunction-free default theories, its soundness and completeness cannot be guaranteed. Below, we introduce a class of action domain descriptions and corresponding action default theories that allow for a modular translation to definite Horn action default theories that is sound and complete with respect to knowledge about reachable time points.

Definition 4.15. Let Θ be a *p-ule*$\mathcal{D}$ action domain specification and Σ_{inst} be instance information for the domain. The domain Θ is *admissible* if all precondition laws in Θ are disjunction-free. The instance information Σ_{inst} is *admissible* if

- there is a unique ground atom $\mathit{Init}(\tau_0) \in \Sigma_{inst}$ and
- all state formulas in Σ_{inst} are ground literals in τ_0.

The swipe card domain from Example 4.1 is admissible. In general, for a given admissible domain Θ, the corresponding action default theory contains the following axioms.

1. Frame axioms

$$FrameT(f,s,t) \equiv (Holds(f,s) \wedge Holds(f,t))$$
$$FrameF(f,s,t) \equiv (\neg Holds(f,s) \wedge \neg Holds(f,t))$$

whose conjunctive normal forms are

$$(\neg Holds(f,s) \vee \neg Holds(f,t) \vee FrameT(f,s,t)) \wedge$$
$$(\neg FrameT(f,s,t) \vee Holds(f,s)) \wedge$$
$$(\neg FrameT(f,s,t) \vee Holds(f,t))$$

and

$$(Holds(f,s) \vee Holds(f,t) \vee FrameF(f,s,t)) \wedge$$
$$(\neg FrameF(f,s,t) \vee \neg Holds(f,s)) \wedge$$
$$(\neg FrameF(f,s,t) \vee \neg Holds(f,t))$$

2. for each function A into sort ACTION,

   (a) direct effect formulas

$$DirT(f,A(\vec{x}),s,t) \equiv (f = F_1^+(\vec{x}) \vee \ldots \vee f = F_m^+(\vec{x}))$$
$$DirF(f,A(\vec{x}),s,t) \equiv (f = F_1^-(\vec{x}) \vee \ldots \vee f = F_n^-(\vec{x}))$$

   where `action` $A(\vec{x})$ `causes` $F_i^+(\vec{x})$ for $1 \leq i \leq m$ are all positive direct effect laws for $A(\vec{x})$ in Θ; and `action` $A(\vec{x})$ `causes` $\neg F_j^-(\vec{x})$ for $1 \leq j \leq n$ are all negative direct effect laws for $A(\vec{x})$ in Θ; the CNF of a direct effect formula $Dir(f,A(\vec{x}),s,t) \equiv (f = F_1(\vec{x}) \vee \ldots \vee f = F_m(\vec{x}))$ is

$$(\neg Dir(f,A(\vec{x}),s,t) \vee f = F_1(\vec{x}) \vee \ldots \vee f = F_m(\vec{x})) \wedge$$
$$(\neg f = F_1(\vec{x}) \vee Dir(f,A(\vec{x}),s,t)) \wedge$$
$$\vdots$$
$$(\neg f = F_m(\vec{x}) \vee Dir(f,A(\vec{x}),s,t))$$

   (b) a precondition axiom

$$Poss(A(\vec{x}),s,t) \equiv (\Phi_A[s] \wedge Occurs(A(\vec{x}),s,t))$$

   for `possible` $A(\vec{x})$ `iff` $\Phi_A \in \Theta$, where $\Phi_A = \psi_1(\vec{x})[s] \wedge \ldots \wedge \psi_k(\vec{x})[s]$; its CNF is

$$(Poss(A(\vec{x}),s,t) \vee \neg\psi_1(\vec{x})[s] \vee \ldots \vee \neg\psi_k(\vec{x})[s] \vee \neg Occurs(A(\vec{x}),s,t)) \wedge$$
$$(\neg Poss(A(\vec{x}),s,t) \vee \psi_1(\vec{x})[s]) \wedge$$
$$\vdots$$
$$(\neg Poss(A(\vec{x}),s,t) \vee \psi_n(\vec{x})[s]) \wedge$$
$$(\neg Poss(A(\vec{x}),s,t) \vee Occurs(A(\vec{x}),s,t))$$

(c) an effect axiom

$$\begin{aligned}&Poss(A(\vec{x}),s,t) \supset \\ &\quad (Holds(f,t) \equiv (FrameT(f,s,t) \vee DirT(f,A(\vec{x}),s,t) \vee DefT(f,s,t))) \wedge \\ &\quad (\neg Holds(f,t) \equiv (FrameF(f,s,t) \vee DirF(f,A(\vec{x}),s,t) \vee DefF(f,s,t)))\end{aligned}$$

which gives rise to the clauses $\neg Poss(A(\vec{x}),s,t) \vee$

$$\begin{aligned}&(\neg FrameT(f,s,t) \vee Holds(f,t)) \wedge \\ &(\neg DirT(f,s,t) \vee Holds(f,t)) \wedge \\ &(\neg DefT(f,s,t) \vee Holds(f,t)) \wedge \\ &(\neg Holds(f,t) \vee FrameT(f,s,t) \vee DirT(f,A(\vec{x}),s,t) \vee DefT(f,s,t)) \wedge \\ &(\neg FrameF(f,s,t) \vee \neg Holds(f,t)) \wedge \\ &(\neg DirF(f,s,t) \vee \neg Holds(f,t)) \wedge \\ &(\neg DefF(f,s,t) \vee \neg Holds(f,t)) \wedge \\ &(Holds(f,t) \vee FrameF(f,s,t) \vee DirF(f,A(\vec{x}),s,t) \vee DefF(f,s,t))\end{aligned}$$

3. for each function F into sort FLUENT, the default closure axioms

$$\left(\bigwedge_{\underline{\texttt{normally}}\ F(\vec{y}) \in \Theta} (F(\vec{x}) = F(\vec{y}) \wedge \neg Holds(F(\vec{y}),s))\right) \supset \neg DefT(F(\vec{x}),s,t)$$

$$\left(\bigwedge_{\underline{\texttt{normally}}\ \neg F(\vec{y}) \in \Theta} (F(\vec{x}) = F(\vec{y}) \wedge Holds(F(\vec{y}),s))\right) \supset \neg DefF(F(\vec{x}),s,t)$$

with conjunctive normal form

$$\left(\bigvee_{\underline{\texttt{normally}}\ F(\vec{y}) \in \Theta} (F(\vec{x}) \neq F(\vec{y}) \vee Holds(F(\vec{y}),s))\right) \vee \neg DefT(F(\vec{x}),s,t)$$

$$\left(\bigvee_{\underline{\texttt{normally}}\ \neg F(\vec{y}) \in \Theta} (F(\vec{x}) \neq F(\vec{y}) \vee \neg Holds(F(\vec{y}),s))\right) \vee \neg DefF(F(\vec{x}),s,t)$$

When the clauses (1)–(3) and the Reiter defaults are grounded and propositionalised, the only structural changes are made when equality atoms are replaced by truth or falsity. For instance, for a ground fluent literal ψ, the default closure axiom is simply $\neg\psi[\sigma] \vee \neg Def(\psi,\sigma,\tau)$. The following theorem uses the structure of the ground clauses to argue that the Horn translation preserves their meaning with respect to entailment of *Holds* literals.

Theorem 4.16. *Let Θ be an admissible action domain specification with a consistent domain axiomatisation Σ with admissible instance information, and denote by $\Sigma' \stackrel{\text{def}}{=} [\![\Sigma]\!]$ its corresponding propositional definite Horn domain axiomatisation. A time point τ is reachable in Σ iff it is reachable in Σ', furthermore for every time point τ and ground fluent literal ψ, we have $\Sigma \models \psi[\tau]$ iff $\Sigma' \models \langle\psi[\tau]\rangle^+$.*

Proof. It follows from Theorem 4.10 that reachability of τ in Σ' implies reachability of τ in Σ; for the same reason, $\Sigma' \models \langle\psi[\tau]\rangle^+$ implies $\Sigma \models \psi[\tau]$. It remains to prove the converse. Let τ

be a time point and ψ be a ground fluent literal. If τ is not reachable in Σ, we have $\Sigma \not\models \psi[\tau]$ and the claim is immediate. In the following, let τ be reachable and assume $\Sigma \models \psi[\tau]$. We use induction on τ.

$\Sigma \models \mathit{Init}(\tau)$: Σ and Σ' coincide on the *Init* atom by construction, so $\Sigma' \models \mathit{Init}(\tau)$. World properties at the initial time point are given by *Holds* literals stated in the theory, hence $\psi[\tau] \in \Sigma$. By the definition of the translation, $\langle\psi[\tau]\rangle^+ \in \Sigma'$.

$\Sigma \models \mathit{Poss}(\alpha, \sigma, \tau)$: By construction, $\mathit{Occurs}(\alpha, \sigma, \tau) \in \Sigma$ iff $\mathit{Occurs}(\alpha, \sigma, \tau) \in \Sigma'$. The induction hypothesis and the form of the precondition axioms now imply $\Sigma' \models \mathit{Poss}(\alpha, \sigma, \tau)$ and thus τ is reachable in Σ'. It remains to show $\Sigma' \models \langle\psi[\tau]\rangle^+$. We do a case distinction on why ψ holds at τ according to Σ.

1. $\Sigma \models \mathit{Dir}(\psi, \alpha, \sigma, \tau)$. Then $\mathit{Dir}(\psi, \alpha, \sigma, \tau) \in \Sigma$ which in turn means $\mathit{Dir}(\psi, \alpha, \sigma, \tau) \in \Sigma'$ by construction. The effect axiom's clause $\neg\mathit{Poss}(\alpha, \sigma, \tau) \vee \neg\mathit{Dir}(\psi, \alpha, \sigma, \tau) \vee \psi[\tau]$ yields the claim.
2. $\Sigma \not\models \mathit{Dir}(\psi, \alpha, \sigma, \tau)$. By the form of the direct effect formulas, this means $\Sigma \models \neg\mathit{Dir}(\psi, \alpha, \sigma, \tau)$. Then according to the presumption $\Sigma \models \psi[\tau]$, we have $\Sigma \models \mathit{Frame}(\psi, \sigma, \tau) \vee \mathit{Def}(\psi, \sigma, \tau)$. Due to the default closure axiom $\psi[\sigma] \vee \neg\mathit{Def}(\psi, \sigma, \tau)$ and the frame axiom $\neg\mathit{Frame}(\psi, \sigma, \tau) \vee \psi[\sigma]$ it follows that $\Sigma \models \psi[\sigma]$. By the induction hypothesis $\Sigma' \models \langle\psi[\sigma]\rangle^+$. Additionally, $\Sigma' \models \langle\mathit{Frame}(\neg\psi, \sigma, \tau)\rangle^-$ by the respective frame axiom, $\Sigma' \models \langle\mathit{Def}(\neg\psi, \sigma, \tau)\rangle^-$ by the default closure axiom $\neg\psi[\sigma] \vee \neg\mathit{Def}(\neg\psi, \sigma, \tau)$ for ψ and $\Sigma' \models \langle\mathit{Dir}(\neg\psi, \sigma, \tau)\rangle^-$ by the form of the direct effect formulas and since Σ is consistent. Unit-resolving these conclusions with the effect axiom's clause $\psi[\tau] \vee \mathit{Frame}(\neg\psi, \sigma, \tau) \vee \mathit{Dir}(\neg\psi, \alpha, \sigma, \tau) \vee \mathit{Def}(\neg\psi, \sigma, \tau)$ yields $\Sigma' \models \langle\psi[\tau]\rangle^+$. □

Due to the restricted form of the defaults $\mathit{Init}(t) : \psi[t]/\psi[t]$ or $\psi[s] : \mathit{Def}(\psi, s, t)/\mathit{Def}(\psi, s, t)$, this correspondence can be generalised to extensions of consistent action default theories.

Theorem 4.17. *Let Θ be an admissible action domain specification with a consistent action default theory (Σ, Δ) with admissible instance information, and let its corresponding definite Horn default theory be (Σ', Δ').*

1. *For every extension E for (Σ, Δ) and time point τ which is reachable in E, there is an extension E' for (Σ', Δ') such that τ is reachable in E' and for each ground fluent literal ψ, we have $E \models \psi[\tau]$ iff $E' \models \langle\psi[\tau]\rangle^+$;*
2. *for every extension E' for (Σ', Δ') and time point τ which is reachable in E', there is an extension E for (Σ, Δ) such that τ is reachable in E and for each ground fluent literal ψ, we have $E \models \psi[\tau]$ iff $E' \models \langle\psi[\tau]\rangle^+$.*

Proof. 1. Let E be an extension for (Σ, Δ). We construct an extension E' for (Σ', Δ') as follows. Set $E'_0 \stackrel{\text{def}}{=} \Sigma'$ and $E'_{i+1} \stackrel{\text{def}}{=} \mathit{Th}(E'_i) \cup \mathit{Consequents}(\Delta'_i)$ for $i \geq 0$, where[3]

$$\Delta'_i \stackrel{\text{def}}{=} \left\{ \frac{\langle\beta\rangle^+ : \neg\langle\omega\rangle^-}{\langle\omega\rangle^+} \;\middle|\; \frac{\beta : \omega}{\omega} \in \Delta, \langle\beta\rangle^+ \in E'_i \text{ and } \neg\omega \notin E \right\}$$

We will show by induction that for all $i \geq 0$, time points σ, τ where σ is reachable in E_i, ground actions α and ground fluent literals ψ,

[3]Note that we slightly abuse notation in the proof: in the first part, we use an existing extension E for (Σ, Δ) and its corresponding E_i to construct a theory E' which is later shown to be an extension for (Σ', Δ'); in the second part, we take an extension E' for (Σ', Δ') and its E'_i to define an extension E for (Σ, Δ).

(a) $Poss(\alpha,\sigma,\tau) \in E_i$ iff $Poss(\alpha,\sigma,\tau) \in E'_i$,

(b) $\psi[\tau] \in E_i$ iff $\langle\psi[\tau]\rangle^+ \in E'_i$,

(c) $Def(\psi,\sigma,\tau) \in E_i$ iff $Def(\psi,\sigma,\tau) \in E'_i$ and

(d) $\neg Def(\psi,\sigma,\tau) \in E_i$ iff $\langle Def(\psi,\sigma,\tau)\rangle^- \in E'_i$.

The induction needs a "look-back" of two steps. Thus in the induction base, we treat the cases $i = 0$ and $i = 1$; the induction step then concludes from i and $i+1$ to $i+2$.

$i = 0$: We observe that $E_0 = \Sigma$ and $E'_0 = \Sigma'$.

(a) Σ and hence Σ' contain no *Poss* atoms.

(b) The only *Holds* literals contained in Σ are of the form $\psi[\tau_0]$ for $Init(\tau_0) \in \Sigma$. By the definition of the translation, $\psi[\tau_0] \in \Sigma$ iff $\langle\psi[\tau_0]\rangle^+ \in \Sigma'$.

(c) There are no *Def* atoms in Σ or Σ'.

(d) The only $\neg$*Def* literals in Σ are trivial default closure axioms. By the definition of the translation, we have $\neg Def(\psi,\sigma,\tau) \in \Sigma$ iff $\langle\neg Def(\psi,\sigma,\tau)\rangle^+ \in \Sigma'$. $\langle\neg Def(\psi,\sigma,\tau)\rangle^+ = \langle Def(\psi,\sigma,\tau)\rangle^-$ yields the claim.

$i = 1$: By definition, $E_1 = Th(\Sigma) \cup Consequents(\Delta_0)$ and $E'_1 = Th(\Sigma') \cup Consequents(\Delta'_0)$.

(a) There are no defaults with *Poss* consequents, so we have to show $\Sigma \models Poss(\alpha,\sigma,\tau)$ iff $\Sigma' \models Poss(\alpha,\sigma,\tau)$ for all reachable σ, which follows from Theorem 4.16.

(b) $\Sigma \models \psi[\tau]$ iff $\Sigma' \models \langle\psi[\tau]\rangle^+$ is an immediate consequence of Theorem 4.16. Now let $Init(\tau) : \psi[\tau]/\psi[\tau] \in \Delta$ with $\neg\psi[\tau] \notin E$. Due to the translation we have $Init(\tau) \in E_0$ iff $Init(\tau) \in E'_0$. Hence, we have $\psi[\tau] \in Consequents(\Delta_0)$ iff $\langle\psi[\tau]\rangle^+ \in Consequents(\Delta'_0)$. In combination, we get the desired $\psi[\tau] \in E_1$ iff $\langle\psi[\tau]\rangle^+ \in E'_1$.

(c) Let $\psi[\sigma] : Def(\psi,\sigma,\tau)/Def(\psi,\sigma,\tau) \in \Delta$ with $\neg Def(\psi,\sigma,\tau) \notin E$. By case 1b for $i = 0$, we have $\psi[\sigma] \in E_0$ iff $\langle\psi[\sigma]\rangle^+ \in E'_0$. Hence, $Def(\psi,\sigma,\tau) \in Consequents(\Delta_0)$ iff $Def(\psi,\sigma,\tau) \in Consequents(\Delta'_0)$. In combination with case 1c for $i = 0$, we get $Def(\psi,\sigma,\tau) \in E_1$ iff $Def(\psi,\sigma,\tau) \in E'_1$.

(d) For trivial default closure axioms, the correspondence follows from 1d for $i = 0$. For default closure axioms $\psi[\sigma] \vee \neg Def(\psi,\sigma,\tau)$, we can use 1b for $i = 0$ on $\psi[\sigma]$ and by unit resolution get $E_0 \models \neg Def(\psi,\sigma,\tau)$ iff $E'_0 \models \langle Def(\psi,\sigma,\tau)\rangle^-$. The claim follows since there are no $\neg$*Def* literals in Δ_0.

$i \wedge i+1 \Rightarrow i+2$: We have $E_{i+2} = Th(E_{i+1}) \cup Consequents(\Delta_{i+1})$.

(a) Since there are no defaults with *Poss* consequents, we have to show $E_{i+1} \models Poss(\alpha,\sigma,\tau)$ iff $E'_{i+1} \models Poss(\alpha,\sigma,\tau)$. Σ and Σ' agree on *Occurs* atoms, hence $Occurs(\alpha,\sigma,\tau) \in E_{i+1}$ iff $Occurs(\alpha,\sigma,\tau) \in E'_{i+1}$. By the induction hypothesis for 1b, $\psi[\sigma] \in E_{i+1}$ iff $\langle\psi[\sigma]\rangle^+ \in E'_{i+1}$. By the precondition axioms of the form $Poss(\alpha,\sigma,\tau) \vee \neg\psi_1[\sigma] \vee \ldots \vee \neg\psi_m[\sigma] \vee \neg Occurs(\alpha,\sigma,\tau)$ being the only clauses with a positive occurrence of $Poss(\alpha,\sigma,\tau)$, we get the desired correspondence.

(b) $\wedge$ (c) Let $Init(\tau_0) : \psi[\tau_0]/\psi[\tau_0] \in \Delta$ with $\neg\psi[\tau_0] \notin E$. Now by construction, $Init(\tau_0) \in E_{i+1}$ iff $Init(\tau_0) \in E'_{i+1}$, hence we have that $\psi[\tau_0] \in Consequents(\Delta_{i+1})$ iff $\langle\psi[\tau_0]\rangle^+ \in Consequents(\Delta'_{i+1})$. Since statements in Σ and defaults are the only ways to conclude about the initial time point, we also have $\psi[\tau_0] \in Th(E_{i+1})$ iff $\langle\psi[\tau_0]\rangle^+ \in Th(E'_{i+1})$ by the induction hypothesis. In combination, $\psi[\tau_0] \in E_{i+2}$

iff $\langle \psi[\tau_0] \rangle^+ \in E'_{i+2}$ showing 1b for the initial time point. Now let σ be a non-initial time point that is reachable in E_i. Let $\psi[\sigma] : Def(\psi, \sigma, \tau)/Def(\psi, \sigma, \tau) \in \Delta$ with $\neg Def(\psi, \sigma, \tau) \notin E$. By the induction hypothesis of 1b for i, we have $\psi[\sigma] \in E_i$ iff $\langle \psi[\sigma] \rangle^+ \in E'_i$. Hence, $Def(\psi, \sigma, \tau) \in Consequents(\Delta_i)$ iff $Def(\psi, \sigma, \tau) \in Consequents(\Delta'_i)$. In combination with the induction hypothesis of 1c for i, we get $Def(\psi, \sigma, \tau) \in E_{i+1}$ iff $Def(\psi, \sigma, \tau) \in E'_{i+1}$, which shows 1c for $i+1$. Now by the induction hypothesis of 1a for $i+1$, we have $Poss(\alpha, \sigma, \tau) \in E_{i+1}$ iff $Poss(\alpha, \sigma, \tau) \in E'_{i+1}$. We next show $\psi[\tau] \in Th(E_{i+1})$ iff $\langle \psi[\tau] \rangle^+ \in Th(E'_{i+1})$. $\psi[\tau] \in E_{i+1}$ iff $\langle \psi[\tau] \rangle^+ \in E'_{i+1}$ is clear from the induction hypothesis for 1b, so we prove $\psi[\tau] \in Th(E_{i+1}) \setminus E_{i+1}$ iff $\langle \psi[\tau] \rangle^+ \in Th(E'_{i+1}) \setminus E'_{i+1}$ by a case distinction on the cause for $\psi[\tau]$. We note that such a case occurs whenever τ becomes newly reachable in E_{i+1}, that is, $Poss(\alpha, \sigma, \tau) \in Th(E_{i+1}) \setminus E_{i+1}$.

i. ψ is a direct effect of α. Then the result follows from the induction hypothesis of 1a for $i+1$, the correspondence of *Dir* literals and the effect axiom's conjunct $\neg Poss(\alpha, \sigma, \tau) \vee \neg Dir(\psi, \alpha, \sigma, \tau) \vee \psi[\tau]$.

ii. ψ is a default effect of α. We have $Def(\psi, \sigma, \tau) \in E_{i+1}$ iff $Def(\psi, \sigma, \tau) \in E'_{i+1}$ by 1c for $i+1$ above. Now the effect axiom's conjunct $\neg Poss(\alpha, \sigma, \tau) \vee \neg Def(\psi, \sigma, \tau) \vee \psi[\tau]$ yields the claim.

iii. ψ persists from σ to τ, that is, it holds at σ and there is no contrary direct or default effect. Then the literals $Poss(\alpha, \sigma, \tau)$, $\neg Frame(\neg\psi, \sigma, \tau)$, $\neg Dir(\neg\psi, \alpha, \sigma, \tau)$ and $\neg Def(\neg\psi, \sigma, \tau)$ (on all of which there is a correspondence) can be unit-resolved with the effect axiom's clause $\neg Poss(\alpha, \sigma, \tau) \vee \psi[\tau] \vee Frame(\neg\psi, \sigma, \tau) \vee Dir(\neg\psi, \alpha, \sigma, \tau) \vee Def(\neg\psi, \sigma, \tau)$ in both E_{i+1} and E'_{i+1} to yield the claim.

Since $Consequents(\Delta_{i+1})$ and $Consequents(\Delta'_{i+1})$ contain only *Def* atoms and *Holds* literals (or *Holds* and *-Holds* atoms, respectively) in the initial time point, we get $\psi[\tau] \in E_{i+2}$ iff $\langle \psi[\tau] \rangle^+ \in E'_{i+2}$ showing 1b for $i+2$.

(d) Can be shown as for $i = 1$.

To prove that $E' \stackrel{\text{def}}{=} \bigcup_{i \geq 0} E'_i$ is an extension for (Σ', Δ'), it remains to show that for all defaults $\beta : \omega/\omega \in \Delta$, we have $\neg\omega \notin E$ iff $\langle \omega \rangle^- \notin E'$. Now $\langle \omega \rangle^- = \langle \neg\omega \rangle^+$, so the result is straightforward for defaults of the form $Init(\tau) : \psi[\tau]/\psi[\tau]$. For the remaining defaults of the form $\psi[\sigma] : Def(\psi, \sigma, \tau)/Def(\psi, \sigma, \tau)$, the correspondence $\neg Def(\psi, \sigma, \tau) \in E_i$ iff $\langle Def(\psi, \sigma, \tau) \rangle^- \in E'_i$ for all $i \geq 0$ follows from 1d. The correspondence on reachability of time points follows from the correspondence on the initial time point and the correspondence on $Poss(\alpha, \sigma, \tau)$ atoms for reachable σ. Finally, since E is an extension, we have $E = Th(E)$ and thus $E \models \psi[\tau]$ iff $\psi[\tau] \in E$ as well as $E \models Poss(\alpha, \sigma, \tau)$ iff $Poss(\alpha, \sigma, \tau) \in E$. The same argument applies to E' and concludes the proof of 1.

2. We only sketch the proof since it is very similar to the one for 1. Let E' be an extension for (Σ', Δ'). We construct an extension E for (Σ, Δ) as follows. Set $E_0 \stackrel{\text{def}}{=} \Sigma$ and $E_{i+1} \stackrel{\text{def}}{=} Th(E_i) \cup Consequents(\Delta_i)$ for $i \geq 0$, where

$$\Delta_i \stackrel{\text{def}}{=} \left\{ \frac{\beta : \omega}{\omega} \;\middle|\; \frac{\beta : \omega}{\omega} \in \Delta, \beta \in E_i \text{ and } \langle \omega \rangle^- \notin E' \right\}$$

As above, it can be shown by induction that for all $i \geq 0$, time points σ, τ that are reachable in E'_i, ground actions α and ground fluent literals ψ,

(a) $Poss(\alpha, \sigma, \tau) \in E'_i$ iff $Poss(\alpha, \sigma, \tau) \in E_i$,

(b) $\langle \psi[\tau] \rangle^+ \in E'_i$ iff $\psi[\tau] \in E_i$,

(c) $Def(\psi, \sigma, \tau) \in E'_i$ iff $Def(\psi, \sigma, \tau) \in E_i$ and

(d) $\langle Def(\psi, \sigma, \tau) \rangle^- \in E'_i$ iff $\neg Def(\psi, \sigma, \tau) \in E_i$ and

Finally, $\neg\omega \notin E$ iff $\langle \omega \rangle^- \notin E'$ can be proved as before to show that $E \stackrel{\text{def}}{=} \bigcup_{i \geq 0} E_i$ is an extension for (Σ, Δ).

□

For an inconsistent Σ, the extensions of (Σ, Δ) and (Σ', Δ') differ: while the original default theory has a single inconsistent extension, its translation has no extension due to the integrity constraints (4.2). This restriction to consistent action default theories is not a proper limitation here since it can be effectively checked in the implementation. In any case, an inconsistent domain axiomatisation must be considered erroneous and makes determining consequences unnecessary.

With the transformation of default theories into definite Horn default theories, the bulk of translation work is done and the remaining tasks are mostly straightforward.

4.1.4 ... to Propositional Answer Set Programs

With Marek and Truszczyński's translation from Theorem 2.13 in mind, it is immediate how to turn a propositional definite Horn default theory into an answer set program. We however give a formal definition for the sake of completeness.

Definition 4.18. Let (W, D) be a propositional definite Horn default theory. Its corresponding propositional normal logic program is

$$ASP(W, D) \stackrel{\text{def}}{=} \{Q_0 \leftarrow Q_1, \ldots, Q_m \mid Q_0 \vee \neg Q_1 \vee \ldots \vee \neg Q_m \in W\} \cup \left\{ Q_0 \leftarrow Q_1, \ldots, Q_m, \mathit{not}\ R_1, \ldots, \mathit{not}\ R_n \;\middle|\; \frac{Q_1 \wedge \ldots \wedge Q_m : \neg R_1, \ldots, \neg R_n}{Q_0} \in D \right\}$$

This concludes the translation from action domain specifications into normal logic programs. The fundamental result of this chapter now states the correctness of the translation with respect to conclusions about reachable time points.

Theorem 4.19. *Let Θ be an admissible action domain specification and $(\Sigma_{dc} \cup \Sigma, \Delta)$ be its quantifier-free CNF action default theory with admissible instance information and translated logic program $\Lambda = ASP(\llbracket (Prop(\lfloor \Sigma \rfloor), Prop(\Delta)) \rrbracket)$.*

1. *For each extension E for (Σ, Δ), there is an answer set M_E such that for each time point τ reachable in E and fluent literal ψ we have $\psi[\tau] \in E$ iff $\langle \psi[\tau] \rangle^+ \in M_E$;*

2. *for each answer set M for Λ, there is an extension E_M such that for each time point τ reachable in Λ and fluent literal ψ we have $\langle \psi[\tau] \rangle^+ \in M$ iff $\psi[\tau] \in E_M$.*

Proof. This follows from Theorem 4.4 (correctness of grounding w.r.t. domain closure), Theorem 4.17 (correspondence for propositional and definite Horn default theories) and Theorem 2.13 (correspondence for definite Horn default theories and normal logic programs [Marek and Truszczyński, 1989]). □

The overall blowup in size from (Σ, Δ) to $\Lambda = ASP(\llbracket(Prop(\lfloor\Sigma\rfloor), Prop(\Delta))\rrbracket)$ is moderate. It mainly depends on the size of the domains for sorts TIME, FLUENT and ACTION, the number of variables in the default theory, the size $\|\Sigma\| + \|\Delta\|$ of the default theory. Thus for the overall translation we get

$$\|\Lambda\| \leq (\|\Sigma\| + \|\Delta\|) \cdot |\mathfrak{D}_{\text{TIME}} \cup \mathfrak{D}_{\text{FLUENT}} \cup \mathfrak{D}_{\text{ACTION}}|^{2 \cdot |Vars(\Sigma,\Delta)|}$$

The size of the default theory is linear in the size of the domain specification (in particular the number of variables); the size of the sort domains is however exponential in the size of the domain specification.

For many straightforward generalisations of *p-ule*$\mathcal{D}$, there exist examples for which the modular translation does not work.

Disjunctions in state default prerequisites. Consider the prerequisite-free state default $\delta_1 =$ `normally` G and the simple state default $\delta_2 =$ `normally` G `if` $\Phi^\top$ where $\Phi^\top$ is an arbitrarily complex tautology. Their Reiter defaults for the initial time point are logically equivalent, but to preserve this, the definite Horn translation of $\Phi^\top$ would still have to be a tautology, which cannot be guaranteed in general.

Conditional effects. With the two direct effect laws `action` A `causes` G `if` F and `action` A `causes` G `if` ¬F, it is easy to see from the resulting effect axiom that G is an unconditional effect which always occurs. Alas, without knowledge of F's truth value at the starting time point, the Horn translation of the effect axiom cannot derive the effect G.

Global effects. The specification of global effects via additional variables leads to the occurrence of existential quantifiers in the resulting action default theories. Our translation is however only defined for quantifier-free default theories. Being restricted to finite domains, we could of course express global effects via local effects, but this would not necessarily be elaboration tolerant.

Disjunctive effects. For the (unconditional) disjunctive direct effect laws `action` A `causes` F `or` G and `action` A `causes` ¬F `or` G, it is easy to see that G is a deterministic effect of the action. Again, the translation of the effect axiom is too weak to preserve this.

Disjunction-free state defaults and observations about multiple time points. For *d-ule*$\mathcal{D}$ domains where narratives may contain observations about multiple time points, the counterexample is slightly contrived. In essence, we use (1) an observed change which can only be due to a default effect and (2) the interaction of default closure axioms of different state defaults with the same prerequisite.

Example 4.20. Let the signature consist of a single constant A of sort ACTION and the FLUENTS $\mathsf{F}_0, \mathsf{F}_1, \mathsf{F}_2, \mathsf{G}_1, \mathsf{G}_2$. The domain is given by

`action` A `causes` F_0	`normally` $\neg\mathsf{F}_2$
`normally` G_1 `if` F_1	`normally` G_2 `if` $\neg\mathsf{F}_1$
`normally` G_1 `if` $\mathsf{F}_2 \wedge \mathsf{F}_0$	`normally` G_2 `if` $\mathsf{F}_2 \wedge \mathsf{F}_0$

Finally, the narrative states the occurrence of A and an observed change of fluents G_1 and G_2.

$Occurs(\mathsf{A}, 0, 1)$	$\neg Holds(\mathsf{G}_1, 0)$	$Holds(\mathsf{G}_1, 1)$
	$\neg Holds(\mathsf{G}_2, 0)$	$Holds(\mathsf{G}_2, 1)$

Let (Σ, Δ) be the resulting action default theory. Since G_1 and G_2 change from 0 to 1, we get $\Sigma \models \neg \mathit{FrameT}(\mathsf{G}_i, 0, 1) \wedge \neg \mathit{FrameF}(\mathsf{G}_i, 0, 1)$. Fluent F_0 is the only effect of A, thus we also have $\Sigma \models \neg \mathit{DirT}(\mathsf{G}_i, 0, 1)$. By the effect axiom, the only reason for G_1 and G_2 to change is therefore being a default effect: $\Sigma \models \mathit{DefT}(\mathsf{G}_1, 0, 1) \wedge \mathit{DefT}(\mathsf{G}_2, 0, 1)$. Using the default closure axiom for G_1, we can infer that one of the prerequisites must hold: $\Sigma \models \mathsf{F}_1[1] \vee (\mathsf{F}_2[1] \wedge \mathsf{F}_0[1])$. The same works for G_2 and its default closure axiom, whence we get $\Sigma \models \neg\mathsf{F}_1[1] \vee (\mathsf{F}_2[1] \wedge \mathsf{F}_0[1])$. Combining these two consequences yields $\Sigma \models \mathsf{F}_2[1] \wedge \mathsf{F}_0[1]$. While F_0 is a positive effect of A, the only possible cause for $\mathsf{F}_2[1]$ is persistence, whence we have $\Sigma \models \mathsf{F}_2[0]$ and the last state default is inapplicable at both time points. Using $\Sigma \models \mathsf{F}_2[0] \wedge \neg\mathsf{G}_1[0] \wedge \neg\mathsf{G}_2[0]$ and the default closure axioms again, we can derive $\Sigma \models \neg\mathsf{F}_0[0]$, which explains why the second and fourth state defaults were inapplicable at 0. This shows that there exists an extension $E = \mathit{Th}(\Sigma)$ where the defaults concluding $\mathsf{G}_1[1]$ and $\mathsf{G}_2[1]$ have been applied and all others are inapplicable.

However, the ASP translation of (Σ, Δ) does not admit an answer set. Roughly, the vital conclusion $\Sigma \models \mathit{Holds}(\mathsf{F}_2, 1)$ cannot be drawn in $\mathit{ASP}(\Sigma, \Delta)$ due to the reasons explained in Example 4.13. In consequence, the supernormal default concluding -$\mathit{Holds}(\mathsf{F}_2, 0)$ and thus also the one concluding $\mathit{DefF}(\mathsf{F}_2, 0, 1)$ are applicable. This together with the observations -$\mathit{Holds}(\mathsf{G}_1, 0)$ and -$\mathit{Holds}(\mathsf{G}_2, 0)$ then allows (by the default closure axioms) to infer both $\mathit{Holds}(\mathsf{F}_1, 0)$ and -$\mathit{Holds}(\mathsf{F}_1, 0)$. The integrity constraints then prevent any extension.

4.2 From Action Domain Specifications to Open Answer Set Programs

The translation from the previous section presents a sound and complete mapping from admissible action domain specifications to normal logic programs that can in principle be straightforwardly implemented. However, grounding the default theory before turning it into a definite Horn default theory has at least two disadvantages. First, we produce structurally equivalent ground formulas and afterwards turn them into definite Horn clauses. This entails repeating basically the same transformation step a number of times. Moreover, general information about the domain and information about a specific domain instance cannot necessarily be compiled in isolation: for linear time structures, the ground time points used in the narrative determine the grounding of the rest of the default theory. Finally, during grounding it is unknown which parts of the axiomatisation we need and thus all of it has to be grounded. Deferring the grounding step to a later time point, for some queries we might get away with not having to ground the whole default theory.

These reflections have led us to optimise the translation by moving the grounding step to the end of the workflow. This is possible since current answer set solvers allow to specify normal logic programs *with variables*. In a preprocessing step, the *grounder* of an ASP system then replaces rules with variables by their ground instances (provided the rules are safe). We will demonstrate below how this capability can be used to defer the grounding work to the ASP solver. Due to the modularity of our translation, this also means that the general characterisation of the domain and information about a specific instance can now be compiled independently.

But first of all, we show that this optimisation is possible without affecting the correctness of the translation. The key result here says that the functions creating the definite Horn translation and well-sorted grounding commute. For our optimisation this means that the much faster transform-ground method produces the same results as the previous, provably correct ground-transform method.

Lemma 4.21. *Let Ξ be a relational signature where all sort domains are given by sets $\mathfrak{D}$ and let $C(x_1, \ldots, x_n)$ be a clause with free variables $x_1 : \mathfrak{s}_1, \ldots, x_n : \mathfrak{s}_n$.*

$$\lfloor [\![C(x_1, \ldots, x_n)]\!] \rfloor = [\![\lfloor C(x_1, \ldots, x_n) \rfloor]\!]$$

Proof. By induction on the number of free variables. For the base case, $C(\mathfrak{o}_1, \ldots, \mathfrak{o}_n)$ is ground and hence $[\![\lfloor C(\mathfrak{o}_1, \ldots, \mathfrak{o}_n) \rfloor]\!] = [\![C(\mathfrak{o}_1, \ldots, \mathfrak{o}_n)]\!] = \lfloor [\![C(\mathfrak{o}_1, \ldots, \mathfrak{o}_n)]\!] \rfloor$ is immediate. For the induction step let

$$\lfloor [\![C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, \mathfrak{o}_i, x_{i+1}, \ldots, x_n)]\!] \rfloor = [\![\lfloor C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, \mathfrak{o}_i, x_{i+1}, \ldots, x_n) \rfloor]\!] \tag{IH}$$

and consider the clause $C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ with one additional free variable.

$$\begin{aligned}
& [\![\lfloor C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, x_i, x_{i+1}, \ldots, x_n) \rfloor]\!] \\
= & \left[\!\!\left[\bigwedge_{\mathfrak{o}_i \in \mathfrak{D}_i} \lfloor C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, \mathfrak{o}_i, x_{i+1}, \ldots, x_n) \rfloor \right]\!\!\right] && \text{(Definition 4.2)} \\
= & \bigwedge_{\mathfrak{o}_i \in \mathfrak{D}_i} [\![\lfloor C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, \mathfrak{o}_i, x_{i+1}, \ldots, x_n) \rfloor]\!] && \text{(Definition 4.7)} \\
= & \bigwedge_{\mathfrak{o}_i \in \mathfrak{D}_i} \lfloor [\![C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, \mathfrak{o}_i, x_{i+1}, \ldots, x_n)]\!] \rfloor && \text{(IH)} \\
= & \lfloor [\![C(\mathfrak{o}_1, \ldots, \mathfrak{o}_{i-1}, x_i, x_{i+1}, \ldots, x_n)]\!] \rfloor && \text{(Definition 4.2)}
\end{aligned}$$

□

It is readily observed that this result can be generalised to conjunctions of clauses, sets thereof and thus to the world knowledge part of quantifier-free default theories in CNF.

Theorem 4.22. *For any quantifier-free default theory (W, D) over a relational signature where all formulas $\Phi \in W$ are in conjunctive normal form, we have $(\lfloor [\![W]\!] \rfloor, D) = ([\![\lfloor W \rfloor]\!], D)$.*

We now define how to turn a quantifier-free definite Horn default theory into an answer set program with variables. Most of this is straightforward from Theorem 2.13, the main additional task is to ensure that variables are grounded according to their sorts and that all created rules are safe. To this end, a new unary predicate $Sort_{\mathfrak{s}}$ is introduced for each sort $\mathfrak{s}$ with the understanding that atom $Sort_{\mathfrak{s}}(\mathfrak{t})$ means that term $\mathfrak{t}$ is of sort $\mathfrak{s}$.

Definition 4.23. Let $\Xi = (\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ be a signature and for each sort $\mathfrak{s} \in \mathfrak{S}$ let $Sort_{\mathfrak{s}}$ be a fresh unary predicate symbol. Let $C(\vec{x}) = P_0(\vec{x}) \vee \neg P_1(\vec{x}) \vee \ldots \vee \neg P_m(\vec{x})$ be a definite Horn clause with free variables $\vec{x} = x_1 : \mathfrak{s}_1, \ldots, x_n : \mathfrak{s}_n$. The *corresponding logic program rule* of clause $C(\vec{x})$ is

$$ASP(C)(\vec{x}) \stackrel{\text{def}}{=} P_0(\vec{x}) \leftarrow P_1(\vec{x}), \ldots, P_m(\vec{x}), Sort_{\mathfrak{s}_1}(x_1), \ldots, Sort_{\mathfrak{s}_n}(x_n)$$

Let $\delta(\vec{x}) = Q_1(\vec{x}) \wedge \ldots \wedge Q_m(\vec{x}) : \neg R(\vec{x}) / Q_0(\vec{x})$ be a definite Horn default with free variables $\vec{x} = x_1 : \mathfrak{s}_1, \ldots, x_n : \mathfrak{s}_n$. The *corresponding logic program rule* of default $\delta(\vec{x})$ is

$$ASP(\delta)(\vec{x}) \stackrel{\text{def}}{=} Q_0(\vec{x}) \leftarrow Q_1(\vec{x}), \ldots, Q_m(\vec{x}), Sort_{\mathfrak{s}_1}(x_1), \ldots, Sort_{\mathfrak{s}_n}(x_n), \mathit{not}\ R(\vec{x})$$

Clauses C from above where the positive atom is an equality or inequality atom are discarded: in the answer set program with built-in unique-names assumption, there is no need to infer term (in-)equalities since they are fixed. It is easy to see that all the resulting rules are safe, since every variable of the Horn clause occurs positively in a sort predicate atom in

the body of the corresponding logic program rule.[4] The mapping given above is a proper generalisation of the one in Definition 4.18 since they agree on propositional signatures.

The final ingredient of our translated programs now gives a meaning to these *Sort* predicates, that is, specifies the actual sort domains. Recall that the user specifies an action domain using a signature that may contain function symbols of positive arity. Yet in our translation, we have only ever assumed the existence of finite domains for each sort of our language and moreover the absence of any positive-arity function symbols. The following definition provides the resolution: for a certain natural number n (also given by the user), all terms of the signature of the specification up to maximum depth n are viewed as constant symbols that form the signature the translation works with. That way, it is much easier for the user to write elaboration-tolerant action domain specifications, while the theory works as before.

Definition 4.24. Let $\Xi = (\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ be a signature and n be a natural number. For a sort $\mathfrak{s} \in \mathfrak{S}$, the *Herbrand domain of depth n* is denoted by $\mathfrak{D}^n_\mathfrak{s}$ and inductively defined by

$$\mathfrak{D}^0_\mathfrak{s} \stackrel{\text{def}}{=} \{C \mid C : \mathfrak{s} \in \mathfrak{F}\}$$
$$\mathfrak{D}^{i+1}_\mathfrak{s} \stackrel{\text{def}}{=} \mathfrak{D}^i_\mathfrak{s} \cup \left\{ F(\mathfrak{o}_1, \ldots, \mathfrak{o}_m) \;\middle|\; F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_m \to \mathfrak{s} \in \mathfrak{F}, \mathfrak{o}_j \in \mathfrak{D}^i_{\mathfrak{s}_j} \text{ for } 1 \leq j \leq m \right\}$$

Example 4.1 (Continued)**.** In the swipe card domain, we have $\mathfrak{D}^n_{\text{ACTION}} = \{\mathsf{Swipe}, \mathsf{Push}\}$ for all $n \in \mathbb{N}_0$. The Herbrand domains for sort TIME are

$$\mathfrak{D}^0_{\text{TIME}} = \{S_0\}\text{, and for } i \geq 0$$
$$\mathfrak{D}^{i+1}_{\text{TIME}} = \mathfrak{D}^i_{\text{TIME}} \cup \left\{ Do(\mathsf{Swipe}, \sigma), Do(\mathsf{Push}, \sigma) \;\middle|\; \sigma \in \mathfrak{D}^i_{\text{TIME}} \right\}$$

In the implementation, the creation of these domains is dynamically done by the solver. For this purpose, Definition 4.24 has been rewritten in ASP. The rules below use the numerical comparison predicate $< : \mathbb{N}_0 \times \mathbb{N}_0$ and the function $+ : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ in their standard meanings, which are provided by all established answer set solvers [Eiter et al., 1997; Gebser et al., 2011]. For each sort $\mathfrak{s}$, we utilise a binary version of the predicate $Sort_\mathfrak{s}$ to state the depth i of a term $\mathfrak{t}$ along with the sort $\mathfrak{s}$ as in $Sort_\mathfrak{s}(i, \mathfrak{t})$. Observe that below, n is a meta-level variable that represents a fixed natural number, but $i, x, x_1, \ldots, x_m$ are object-level variables of the defined program.

Definition 4.25. Let $\Xi = (\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ be a signature and n be a natural number. For a sort $\mathfrak{s} \in \mathfrak{S}$, the *domain rules for depth n* are given by the set $\Lambda^n_\mathfrak{s} \stackrel{\text{def}}{=}$

$$\{Sort_\mathfrak{s}(x) \leftarrow Sort_\mathfrak{s}(i, x)\} \cup$$
$$\{Sort_\mathfrak{s}(0, C) \mid C : \mathfrak{s} \in \mathfrak{F}\} \cup$$
$$\{Sort_\mathfrak{s}(i+1, x) \leftarrow i < n,\, Sort_\mathfrak{s}(i, x)\} \cup$$
$$\{Sort_\mathfrak{s}(i+1, F(x_1, \ldots, x_m)) \leftarrow i < n,\, Sort_{\mathfrak{s}_1}(i, x_1),\, \ldots,\, Sort_{\mathfrak{s}_m}(i, x_m) \mid F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_m \to \mathfrak{s} \in \mathfrak{F}\}$$

For the entirety of sorts, the domain rules for depth n are $\Lambda^n_\mathfrak{S} \stackrel{\text{def}}{=} \bigcup_{\mathfrak{s} \in \mathfrak{S}} \Lambda^n_\mathfrak{s}$.

This way of creating the sort domains easily allows us to reason about situations with a finite horizon. In fact, this is how it is actually done in the implementation – simply by declaring S_0 : TIME and Do : ACTION × TIME → TIME as the only function symbols into sort TIME and specifying a maximal term depth n. In this case, the value of n limits the program's lookahead into the situation tree. The lookahead can be dynamically increased by simply adjusting the meta-level variable n.

[4] In the actual implementation, we add only *Sort* atoms for variables which are not already safe.

Example 4.1 (Continued). For the sorts of $\Theta_{SwipeCard}$, the domain rules for depth $n = 3$ are

$$\begin{aligned}
Sort_{\text{TIME}}(x) &\leftarrow Sort_{\text{TIME}}(i, x)\\
Sort_{\text{TIME}}(0, S_0)&\\
Sort_{\text{TIME}}(i+1, Do(x_1, x_2)) &\leftarrow i < 3,\ Sort_{\text{ACTION}}(i, x_1),\ Sort_{\text{TIME}}(i, x)\\
Sort_{\text{ACTION}}(x) &\leftarrow Sort_{\text{ACTION}}(i, x)\\
Sort_{\text{ACTION}}(0, \mathsf{Swipe})&\\
Sort_{\text{ACTION}}(0, \mathsf{Push})&\\
Sort_{\text{FLUENT}}(x) &\leftarrow Sort_{\text{FLUENT}}(i, x)\\
Sort_{\text{FLUENT}}(0, \mathsf{HasCard})&\\
Sort_{\text{FLUENT}}(0, \mathsf{Locked})&\\
Sort_{\text{FLUENT}}(0, \mathsf{Open})&\\
Sort_{\text{FLUENT}}(0, \mathsf{Jammed})&
\end{aligned}$$

The rules in $\Lambda^n_{\mathfrak{S}}$ contain no negation as failure, which makes it straightforward to prove that they really create the Herbrand domains $\mathfrak{D}^n_{\mathfrak{s}}$ of depth n.

Proposition 4.26. *Let $\Xi = (\mathfrak{S}, \mathfrak{P}, \mathfrak{F}, \mathfrak{X})$ be a signature, $\mathfrak{s} \in \mathfrak{S}$ be a sort, $\mathfrak{t} : \mathfrak{s}$ be a ground term and n be a natural number. Then $\mathfrak{t} \in \mathfrak{D}^n_{\mathfrak{s}}$ iff $\Lambda^n_{\mathfrak{s}} \models_H Sort_{\mathfrak{s}}(\mathfrak{t})$.*

Proof. We begin by observing that due to the first and third rules, we have

$$\Lambda^n_{\mathfrak{s}} \models_H Sort_{\mathfrak{s}}(\mathfrak{t}) \quad \text{iff} \quad \Lambda^n_{\mathfrak{s}} \models_H Sort_{\mathfrak{s}}(n, \mathfrak{t}) \tag{*}$$

Furthermore, $\Lambda^{n+1}_{\mathfrak{s}}$ and $\Lambda^n_{\mathfrak{s}}$ contain the same rules with head $Sort_{\mathfrak{s}}(n, \mathfrak{t})$ and none of these rules depends on any atom defined in $\Lambda^{n+1}_{\mathfrak{s}} \setminus \Lambda^n_{\mathfrak{s}}$, hence

$$\Lambda^n_{\mathfrak{s}} \models_H Sort_{\mathfrak{s}}(n, \mathfrak{t}) \quad \text{iff} \quad \Lambda^{n+1}_{\mathfrak{s}} \models_H Sort_{\mathfrak{s}}(n, \mathfrak{t}) \tag{**}$$

The rest of the proof is by induction on n. For the base case, we have

$$\begin{aligned}
&\mathfrak{t} \in \mathfrak{D}^0_{\mathfrak{s}}\\
&\text{iff } \mathfrak{t} = C \text{ for some } C : \mathfrak{s} \in \mathfrak{F} && \text{(Definition 4.24)}\\
&\text{iff } Sort_{\mathfrak{s}}(0, C) \in \Lambda^0_{\mathfrak{s}} && \text{(Definition 4.25)}\\
&\text{iff } \Lambda^0_{\mathfrak{s}} \models_H Sort_{\mathfrak{s}}(0, C) && (Sort_{\mathfrak{s}}(0, C) \text{ is a fact})\\
&\text{iff } \Lambda^0_{\mathfrak{s}} \models_H Sort_{\mathfrak{s}}(C) && \text{(*)}
\end{aligned}$$

For the induction step, we have

$$
\begin{aligned}
& \mathfrak{t} \in \mathfrak{D}_{\mathfrak{s}}^{n+1} \\
\text{iff } & \mathfrak{t} = F(\mathfrak{o}_1, \ldots, \mathfrak{o}_m) \text{ for some } F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_m \to \mathfrak{s} \in \mathfrak{F} \text{ with} \\
& \mathfrak{o}_j \in \mathfrak{D}_{\mathfrak{s}_j}^{n} \text{ for } 1 \leq j \leq m && \text{(Definition 4.24)} \\
\text{iff } & \mathfrak{t} = F(\mathfrak{o}_1, \ldots, \mathfrak{o}_m) \text{ for some } F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_m \to \mathfrak{s} \in \mathfrak{F} \text{ with} \\
& \Lambda_{\mathfrak{s}}^{n} \models_H \mathit{Sort}_{\mathfrak{s}}(\mathfrak{o}_j) \text{ for } 1 \leq j \leq m && \text{(IH)} \\
\text{iff } & \mathfrak{t} = F(\mathfrak{o}_1, \ldots, \mathfrak{o}_m) \text{ for some } F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_m \to \mathfrak{s} \in \mathfrak{F} \text{ with} \\
& \Lambda_{\mathfrak{s}}^{n} \models_H \mathit{Sort}_{\mathfrak{s}}(n, \mathfrak{o}_j) \text{ for } 1 \leq j \leq m && \text{(*)} \\
\text{iff } & \mathfrak{t} = F(\mathfrak{o}_1, \ldots, \mathfrak{o}_m) \text{ for some } F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_m \to \mathfrak{s} \in \mathfrak{F} \text{ with} \\
& \Lambda_{\mathfrak{s}}^{n+1} \models_H \mathit{Sort}_{\mathfrak{s}}(n, \mathfrak{o}_j) \text{ for } 1 \leq j \leq m && \text{(**)} \\
\text{iff } & \Lambda_{\mathfrak{s}}^{n+1} \models_H \mathit{Sort}_{\mathfrak{s}}(n+1, \mathfrak{t}) && \text{(Definition 4.25, last rule)} \\
\text{iff } & \Lambda_{\mathfrak{s}}^{n+1} \models_H \mathit{Sort}_{\mathfrak{s}}(\mathfrak{t}) && \text{(*)}
\end{aligned}
$$

□

So from a given action domain specified by ADS Θ and instance Σ_{inst}, we first create a quantifier-free action default theory $(\Sigma \cup \Sigma_{inst}, \Delta)$. This is next translated into an answer set program with variables $ASP(W, D) \stackrel{\text{def}}{=} \{ASP(C) \mid C \in W\} \cup \{ASP(\delta) \mid \delta \in D\}$. To this program, we add logic program rules $\Lambda_{\mathfrak{S}}^{n}$ for the sorts $\mathfrak{S}$ of Θ's signature. Most importantly, at this step we can utilise the modularity of the translation and transform the general domain information and instance information separately:

$$\left(ASP(\Sigma \cup \Sigma_{inst}, \Delta) \cup \bigcup_{\mathfrak{s} \in \mathfrak{S}} \Lambda_{\mathfrak{s}}^{n}\right) = \left(ASP(\Sigma, \Delta) \cup \bigcup_{\mathfrak{s} \in \mathfrak{S} \setminus \{\text{TIME}\}} \Lambda_{\mathfrak{s}}^{n}\right) \cup (ASP(\Sigma_{inst}) \cup \Lambda_{\text{TIME}}^{n})$$

The ASP solver then grounds the domain and thereby produces the well-sorted grounding with a built-in unique-names assumption. Notably, it may ground very efficiently without even reaching the analytically predicted size of the ground program.

4.3 The Implemented System draculasp

The name draculasp stands for "**d**efault **r**easoning about **a**ctions and **c**hange **u**sing **l**ogic and **a**nswer **s**et **p**rogramming." It alludes to the system's usage for non-monotonic reasoning about action domains, the semantics of the supported input fragments of $\mathcal{D}$ being defined in terms of logic (more specifically default logic) and the actual reasoning being done via answer set programming.

The draculasp system is written in ECL^{i}PS^e Prolog[5] and prototypically implements the translation detailed in Section 4.2. It takes as input an action domain specification and instance information for that domain and transforms it into an answer set program with variables. This ASP can then be queried by invoking an external ASP solver. The system and some example domains can be downloaded at `http://informatik.uni-leipzig.de/~strass/draculasp/`.

Action domain information is stored in text files with a special syntax. The figure below shows the representation of the swipe card domain.

[5] `http://eclipseclp.org`

```
sort action:  swipe, push.
sort fluent:  hasCard, locked, open, jammed.

precondition swipe:  hasCard.
precondition push:  and(not(locked), not(jammed)).

effects swipe:  not(locked).
effects push:  open.

normally not(jammed).
```

Figure 4.2: Action domain specification file `swipecard.ads` *for* $\Theta_{SwipeCard}$ *from Example 4.1. The first two statements define the domain signature. (Subsorts are not used in this domain, but draculasp also understands statements of the form "*$\mathfrak{s}_1$ `extends` $\mathfrak{s}_2$*" expressing the subsort relationship* $\mathfrak{s}_1 \sqsubseteq \mathfrak{s}_2$*.) Next, action preconditions, action effects and state defaults are specified, where action effects are grouped by actions.*

Information about a specific domain configuration is specified in an *action domain instance* file. Each such file refers to a general domain description. In the following figure we can see a particular branching-time instance of the swipe card domain.

```
instance of "swipecard.ads".

time structure:  situations.

term depth:  3.

initially hasCard.
```

Figure 4.3: Action domain instance file `swipecard1.adi` *for* $\Theta_{SwipeCard}$*. The first line refers to the domain of which the file defines an instance. The next lines declare time structure and term depth, and the last line characterises the initial situation.*

All instance files share the reference to the domain, and declaration of time structure and term depth. Depending on the time structure, they additionally state an initial situation or a narrative.

To provide the reader with some intuition on the look-and-feel of the program, we show an example run of draculasp's user interface. After starting the program we get to see the recognised commands and can list the domains available for loading:

```
$ draculasp
Welcome!
Available commands:
        help -- print this screen
        quit -- quit
        exit -- quit
        show -- print the current domain in text format
        write -- write the current answer set program to a file
        report -- create and display a pdf report
```

```
instance of "swipecard.ads".

time structure:  linear time 0..3.

term depth:  2.

narrative:
     holds(hasCard, 0),
     occurs(swipe, 0, 1),
     occurs(push, 1, 2).
```

Figure 4.4: Action domain instance file `swipecard2.adi` *for* $\Theta_{SwipeCard}$ *using linear time. The second line expresses that the* TIME *domain is the set* $\{0,\ldots,3\}$*, the third line defines the term depth. The block at the end specifies a narrative in which the initial time point is as in* `swipecard1.adi` *and the door is unlocked from* 0 *to* 1 *before being pushed from* 1 *to* 2*.*

```
        domains -- list domains that are available for loading
        scep -- compute sceptical consequences
        test -- run regression tests
        load <domain> -- load an action domain instance
        ask <query> -- ask a query to the answer set program
}:-> domains
Available domains:
        airport.ads
        bomb.ads
        potato.ads
        shopping.ads
        swipecard.ads
        yale.ads
Available instances:
        yale1.adi
        swipecard1.adi
        swipecard2.adi
        potato.adi
0.01s CPU
```

We can now choose an available domain instance and load it (and its domain). This applies the transformation of the previous section.

```
}:-> load swipecard1.adi
Successfully compiled.
0.05s CPU
```

With the transformed answer set program at our disposal, we can now do reasoning – for example, compute all sceptical consequences of the compiled program. Not all consequences are shown here: for the benefit of the user, draculasp only prints out `holds`, `-holds` and `poss` atoms. In addition, we removed some trivial conclusions from this presentation.

```
swipecard1.adi}:-> scep
        holds(hasCard, s0).
```

```
        -holds(jammed, s0).
        poss(swipe, s0, does(swipe, s0)).
        holds(hasCard, does(swipe, s0)).
        -holds(locked, does(swipe, s0)).
        -holds(jammed, does(swipe, s0)).
        poss(push, does(swipe, s0), does(push, does(swipe, s0))).
        holds(open, does(push, does(swipe, s0))).
        holds(hasCard, does(push, does(swipe, s0))).
        -holds(jammed, does(push, does(swipe, s0))).
        -holds(locked, does(push, does(swipe, s0))).
0.0s CPU
```

So, for example, according to the program above, the door is initially not jammed by default. This persists through swiping the card, thereby enabling the agent to push the door open. Note that there is no information whether the door was initially open. Reasoning is of course restricted to the term depth specified in the instance file. For the informed user that wishes to inspect the intermediate quantifier-free action default theory from which the answer set program is created, draculasp offers to create a pdf file with the default theory in a human-readable format.[6]

```
swipecard1.adi}:-> report
This is pdfTeX, Version 3.1415926-1.40.10 (TeX Live 2009/Debian)
entering extended mode
Calling viewer, xpdf swipecard1.adi.pdf & ...
0.01s CPU
```

Finally, we can also instruct draculasp to write out the produced answer set program.

```
swipecard1.adi}:-> write
Writing swipecard1.adi.asp ...
0.0s CPU
```

The command `test` is used for development purposes. It collects all domain and instance files in draculasp's working directory and tries to compile them to answer set programs. It then calls the associated ASP solver on the logic programs and reports the results of all these tests back to the user.

In addition to the command line-based interactive mode, draculasp's functionalities can be imported by another Prolog program, for example to use draculasp as a knowledge representation engine for a reasoning agent. In a specific domain, such an agent could use a domain instance file to represent its knowledge about the current situation and employ draculasp to make predictions about what normally holds in the future. Upon action execution, the agent can progress its world knowledge and add observations to create the updated domain instance information. Notably, the agent can employ different time structures for different tasks without technical difficulties.

Despite all of its features, draculasp is still in a prototypical stage and should be seen as a proof-of-concept implementation rather than an off-the-shelf product. There is much potential for future work for using draculasp in connection with a full-fledged agent programming language such as agent logic programs [Drescher et al., 2009]. Considering the fleeting nature of time-dependent knowledge about dynamic domains, we would also like to use the incremental solving capabilities [Gebser et al., 2008] of the Potassco suite [Gebser et al., 2011].

[6] In fact, the formulas in Example 4.1 on page 62 have been created by draculasp from the files shown here.

4.4 Related Work

Propositional Default Reasoning

Using the theoretical relationship between default logic and answer set programming to implement one via the other is not new. [Junker and Konolige, 1990] provided a translation from propositional default theories into normal logic programs[7] which has recently been rediscovered by [Chen et al., 2010]. Also based on the theoretical result of [Marek and Truszczyński, 1989], the idea of this approach is to replace compound formulas Φ by new atoms P_Φ. Then, for a propositional default theory (W, D), a default $\beta : \kappa_1, \ldots, \kappa_n / \omega \in D$ becomes the normal logic program rule $P_\omega \leftarrow P_\beta$, *not* $P_{\neg\kappa_1}$, ..., *not* $P_{\neg\kappa_n}$. A formula $\Phi \in W$ is turned into a fact P_Φ. To guarantee correctness, all relevant semantical entailment relationships $\{\Phi_1, \ldots, \Phi_n\} \models \Psi$ between formulas occurring in the default theory are recorded via rules $P_\Psi \leftarrow P_{\Phi_1}$, ..., P_{Φ_n}. This translation is of course not modular – after adding new information about the domain, the whole theory has to be recompiled. In particular, the theory has to be recompiled for each query that is asked, since queries are modelled by defaults. What is even more significant, their approach only works for propositional default theories. This necessitates a first-ground-then-transform approach, which we dismissed earlier on.

As we can see below, modularity remains absent even under the heavy restriction that we add only atoms to a disjunction-free, normal default theory.

Example 4.27. Consider the propositional default theory $(\emptyset, D)$, where $D = \{A \wedge B : C/C\}$. Chen et al.'s translation produces the logic program $\mathsf{dl2asp}(D) = \{C \leftarrow P_{A\wedge B}, \mathit{not}\ P_{\neg C}\}$. For the default theory $(W, \emptyset)$ with $W = \{A, B\}$, we get $\mathsf{dl2asp}(W) = \{A, B\}$. Now although the default theory (W, D) sceptically entails C, we do not find C in any answer set of $\mathsf{dl2asp}(D) \cup \mathsf{dl2asp}(W)$. This is because the union of separate translations $\mathsf{dl2asp}(D) \cup \mathsf{dl2asp}(W)$ differs from the translation of (W, D) by the rule $P_{A\wedge B} \leftarrow A, B$.

Action Theory Implementations with Defaults

The causal calculator (CCALC) [McCain and Texas Action Group, 1997] was developed by Norman McCain as part of his PhD thesis and since then maintained by the Texas Action Group at the University of Austin. It implements the action language $\mathcal{C}+$ [Giunchiglia et al., 2004]. Like draculasp, the causal calculator offers the specification of action domains and answers queries about these domains by translation into a logical language. Indeed, the functionality of CCALC was an inspiration for draculasp. Similarly, the system $\texttt{dlv}^{\mathcal{K}}$ implements the action language $\mathcal{K}$ [Eiter et al., 2000] on top of the `dlv` answer set solver [Eiter et al., 1997]. However, the default semantics of $\mathcal{C}+$ and $\mathcal{K}$ have an underlying intuition that greatly differs from the one of $\mathcal{D}$ (see also Section 6.1.2). $\mathcal{D}$ considers default statements as saying that something normally holds, but may be exceptionally untrue, where this exception persists. $\mathcal{C}+$ and $\mathcal{K}$ regard default statements as causes on a par with all others. Fluents that have a default truth value may become true (or false) by default without an obvious immediate cause. This view allows them to use defaults to model causes that are not known, not observable or too cumbersome to axiomatise.

[Martin and Thielscher, 2001] present an implemented system that extends the agent programming language FLUX [Thielscher, 2005a] to deal with the qualification problem. The system constructs extensions of the prioritised default theories of [Thielscher, 2001], where exactly

[7] In fact, they translated default theories and theories of autoepistemic logic into truth maintenance systems, which can however equivalently be seen as normal logic programs under the stable model semantics [Reinfrank et al., 1989].

one abnormality is considered as explanation for action failure at each time point in the search. While it constitutes an implementation of an action theory combined with default reasoning, the system is restricted to the special case of supernormal defaults about non-appearance of exogenously caused abnormalities.

ASP-Based Reasoning about Actions

One of the motivations for introducing action languages was the possibility of translating them into logic programs under the stable model semantics. Such a translation was presented for $\mathcal{A}$ [Gelfond and Lifschitz, 1993] and also implemented [Lifschitz et al., 1993]. In recent years, numerous other systems have emerged that follow this initial idea and use answer set programming to reason about actions and change. We only give an overview here, since most of these systems do not employ nonmonotonic negation for default reasoning, but for other purposes.

[Kim et al., 2009] embed the circumscriptive Event Calculus into the general language of stable models and further into answer set programs. The nonmonotonic features of the stable model semantics are used to compute circumscription of *Initiates*, *Terminates*, *Releases* and *Happens*. Default reasoning about dynamic domains is not mentioned in the paper.

[Lee and Palla, 2010] reformulate the Situation Calculus in terms of the first-order stable model semantics and then into answer set programming. For a finite and fully known domain, they turn Lin's Causal Action Theories [Lin, 1995] and Reiter's Basic Action Theories [Reiter, 2001] into answer set programs. In the first translation, they use the stable model semantics to compute the circumscription of Lin's *Caused* predicate (among others). For Basic Action Theories, nonmonotonic negation is employed to solve the frame problem. Both approaches are not intended to make default assumptions about action domains.

[Gebser et al., 2010] offer the *Coala* system that translates various $\mathcal{C}$-like action languages into ASP. Although quite comprehensive in that it offers different types of encodings, the straightforward usage of incremental solving, and LTL-like queries, default reasoning beyond $\mathcal{C}$'s offerings is not part of its capabilities.

[Casolary and Lee, 2011] provide a translation from the input language of the causal calculator into answer set programs. In this respect, they extend the *Coala* system by allowing the full $\mathcal{C}+$ language. While not offering the full functionality of CCalc, it improves upon efficiency of computation by orders of magnitude. Since the system implements the $\mathcal{C}+$ approach to default reasoning about actions, the criticism of Section 6.1.2 applies.

Chapter 5

Ramification and Loop Formulas

The ramification problem of reasoning about actions has received considerable attention since its discovery in [Ginsberg and Smith, 1987]. Yet, the state of the art of research into this problem is not entirely satisfactory: many different approaches have been developed for individual action formalisms, some of which rely on non-classical logics and non-standard semantics. This makes it hard to assess individual approaches by other means than through example scenarios.

In this chapter, we provide a solution to the ramification problem that attempts to remedy this situation. Our approach integrates findings of different approaches to ramification from the last ten to fifteen years. For the first time, we present a solution that:

1. is independent of a particular time structure,
2. is formulated in classical first-order logic,
3. treats cycles – a notoriously difficult aspect – properly, and
4. is assessed against a state-transition semantics via a formal correctness proof.

This is achieved as follows: We first generalise the notion of *causal relationships* of [Thielscher, 1997] to enable us to specify ramifications that are triggered by activation of a formula rather than just an atomic effect. We characterise the intended models of these indirect effect laws by a state-transition semantics. Afterwards, we show how to compile indirect effect laws into effect axioms that then solve the ramification and frame problems. For this compilation, special care needs to be taken to deal with positive cyclic fluent dependencies, that is, self-supporting effects: using techniques from logic programming, we identify positive loops among indirect effects and build their corresponding loop formulas [Lin and Zhao, 2004] into the effect axiom. We finally prove the resulting effect axioms sound and complete with respect to the semantics defined earlier.

* * *

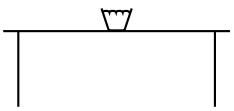
The rest of this chapter is organised as follows. In the next section, we introduce indirect effect laws and their semantics. In Section 5.2, we provide an axiomatic combined solution to the frame and ramification problems. We start out with how we axiomatise persistence and direct effects, then how we include indirect effects and finally and most importantly how we deal with circular dependencies among effects. After proving the correctness of our axiomatic

solution, we show how to express arbitrary trigger formulas in our formalism. We conclude with demonstrating how indirect effect laws can be used to model conditional and disjunctive effects.

5.1 Specifying Indirect Effects

Much like the frame problem, the ramification problem of reasoning about actions has aspects that relate to specification and representation of as well as reasoning about effects. Where the frame problem is concerned with world properties that do *not* change, the ramification problem is about world properties that change *indirectly* – that is, not as a direct result of applying the action but rather as a result of a change induced by the action. The main task here is to efficiently deal with domino effects, where a single change can initiate arbitrarily complex chains of indirect effects. Specifying these effects should be possible in a modular, elaboration tolerant fashion. For example, it should be sufficient to state indirect effect relations only for immediate neighbours in the effect chain and have the theory figure out what actually happens. After all (direct and indirect) effects have been computed, the persistence assumption can be employed to complete the knowledge about the resulting time point with those world properties that have not been affected by direct or indirect effects.

Example 5.1 (Bowl Position)**.** Consider the following simple domain that involves a table and a bowl of soup. If the soup is on the table and just one side of the table is lifted while the other stays on the ground, the soup will be spilled and the bowl falls off. If however both sides of the table are lifted at the same time, there will be no spill and the bowl stays on the table. If the soup is not on the table, there will be no spill whatsoever. With the equipment introduced in Chapter 3, we can model this domain Θ_{Spill} in *cle*$\mathcal{D}$ as follows. The FLUENT sort consists of the functions OnTable (the soup is on the table) LeftUp, RightUp (the left, resp. right side of the table is lifted), Spill (the soup is spilled); available ACTION functions are LiftLeft, LiftRight for lifting either side of the table and LiftBoth for lifting both sides simultaneously. The effects of the last action are clearly specified by `action` LiftBoth `causes` LeftUp and `action` LiftBoth `causes` RightUp.



To formalise indirect effects, we begin with how we capture the truth values of all domain-relevant fluents at a specific time point via so-called states. A state represents a snapshot of the properties of the world at a certain point in time.

Definition 5.2. Consider a fixed domain signature Ξ. A *state* is a maximal consistent set of ground fluent literals. The *satisfaction relation* $\models$ between a state S and ground fluent atoms φ and fluent formulas Φ_1, Φ_2 is recursively defined by

$S \models \top$ and $S \not\models \bot$
$S \models \varphi$ iff $\varphi \in S$
$S \models \neg\Phi_1$ iff $S \not\models \Phi_1$
$S \models \Phi_1 \wedge \Phi_2$ iff both $S \models \Phi_1$ and $S \models \Phi_2$
$S \models \Phi_1 \vee \Phi_2$ iff one of $S \models \Phi_1$ or $S \models \Phi_2$
$S \models (\forall x)\Phi_1$ iff $S \models \Phi_1\{x \mapsto \mathrm{t}\}$ for all terms t over Ξ
$S \models (\exists x)\Phi_1$ iff $S \models \Phi_1\{x \mapsto \mathrm{t}\}$ for some term t over Ξ

Whether a fluent formula is satisfied by a given state will later be used to figure out if a conditional action effect occurs. To compute the changes to a state caused by action effects, we use the concept of state update known from the Fluent Calculus. For the purpose of this definition, direct effect laws with variables are viewed as representatives of their ground instances.

Definition 5.3. Consider a fixed domain signature. For a state S and set L of fluent literals, define the *update of S with L* as $S + L \stackrel{\text{def}}{=} (S \setminus \overline{L}) \cup L$. Let Θ be a *cle*$\mathcal{D}$ action domain specification and α be a ground term of sort action. The *resulting state of α in S* is

$$S + \{\psi \mid \underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi \in \Theta, S \models \Phi\} \tag{5.1}$$

So for a given state S and an action A applied in S, the resulting state is simply the state that contains all positive and negative effects whose conditions were satisfied in S and all literals that are not affected by the action. Note that $S + L$ is a state if and only if L is consistent.

Example 5.1 (Continued)**.** Applying the action LiftBoth to the above depicted initial state $S = \{\mathsf{OnTable}, \neg\mathsf{LeftUp}, \neg\mathsf{RightUp}, \neg\mathsf{Spill}\}$, state update determines the resulting state $S + \{\mathsf{LeftUp}, \mathsf{RightUp}\} = \{\mathsf{OnTable}, \mathsf{LeftUp}, \mathsf{RightUp}, \neg\mathsf{Spill}\}$.

For the remaining actions LiftLeft and LiftRight, we could be tempted to specify their effects by, say,

$$\begin{aligned}
&\underline{\texttt{action}}\ \mathsf{LiftLeft}\ \underline{\texttt{causes}}\ \mathsf{LeftUp}\\
&\underline{\texttt{action}}\ \mathsf{LiftLeft}\ \underline{\texttt{causes}}\ \mathsf{Spill}\ \underline{\texttt{if}}\ \mathsf{OnTable} \wedge \neg\mathsf{RightUp}\\
&\underline{\texttt{action}}\ \mathsf{LiftLeft}\ \underline{\texttt{causes}}\ \neg\mathsf{OnTable}\ \underline{\texttt{if}}\ \mathsf{OnTable} \wedge \neg\mathsf{RightUp}
\end{aligned}$$

for LiftLeft and symmetrically for LiftRight. But there are several issues with this specification:

Firstly, it makes unstated assumptions. The second and third law implicitly assume that the right side of the table not being up persists through lifting the left. But this cannot be guaranteed. There may at the same time occur an effect which causes the left side to be up as well, leading the representation above to falsely predict a spill. What we want to express is that there is a spill if the two sides of the table are at different levels in the *resulting state*! This also relates to elaboration tolerance: if the specification is later changed, these laws might become incorrect and would have to be revisited. This is in contrast to the property of elaboration tolerance that we desire for our formalism, meaning that we incorporate new domain information by adding new laws. Secondly, the specification above is not a faithful representation of causality. Although the left side of the table being up can be seen as a direct effect of lifting it, this hardly holds for the bowl falling off the table and subsequently producing a spill on the floor. And now assume the chain of events possibly initiated by lifting the table does not stop there. What if a paper airplane happens to lie below the table and the spilled soup wets it, thereby impairing the plane's ability to fly? Surely, we cannot take into account all such absurd contingencies when specifying the supposed direct effects of lifting the table. It is particularly mentionable here that the cause of the paper airplane becoming wet is irrelevant to its subsequent not-flying, yet the effect would have to be duplicated for the effect specifications of LiftLeft and LiftRight.

From these reflections we can see that formulating all possible changes using direct effect laws leads to cumbersome and at last unmanageably large action specifications, if it is not outright impossible. To deal with this representational aspect of the ramification problem, we introduce here a modular specification of indirect action effects. We employ expressions that specify certain conditions under which a change of truth value of one fluent causes a change of

truth value of another fluent. These conditions do not hinge on execution of a specific action, but solely depend on change of a world property instead.

Definition 5.4. Let χ, ψ be fluent literals and Φ_1, Φ_2 be fluent formulas. An *indirect effect law* is of the form

$$\texttt{effect}\ \chi\ \texttt{causes}\ \psi\ \texttt{if}\ \Phi_1\ \texttt{before}\ \Phi_2 \tag{5.2}$$

where χ is the *trigger*, ψ is the *effect*, Φ_1 is the *initial* and Φ_2 the *terminal context*. An indirect effect law is *open* if it contains variables, otherwise it is *closed*.

The intended reading of such an indirect effect law is "whenever Φ_1 holds in the starting state, Φ_2 holds in the resulting state and χ has turned from false to true during action execution, then ψ should be an indirect effect." If both contexts are $\top$, we omit them and simply write $\texttt{effect}\ \chi\ \texttt{causes}\ \psi$. Introducing these indirect effect laws into the $\mathcal{D}$ family, we have arrived at the language *clerD* with no state defaults, and conditional, local, deterministic and indirect action effects (ramifications).

Example 5.1 (Continued)**.** In the soup-on-the-table domain, the effects propagate as follows. For causing a spill and causing the bowl to fall off the table, we have, respectively:

$$\texttt{effect}\ \mathsf{LeftUp}\ \texttt{causes}\ \mathsf{Spill}\ \texttt{if}\ \mathsf{OnTable}\ \texttt{before}\ \neg\mathsf{RightUp} \tag{5.3}$$
$$\texttt{effect}\ \mathsf{RightUp}\ \texttt{causes}\ \mathsf{Spill}\ \texttt{if}\ \mathsf{OnTable}\ \texttt{before}\ \neg\mathsf{LeftUp} \tag{5.4}$$
$$\texttt{effect}\ \mathsf{LeftUp}\ \texttt{causes}\ \neg\mathsf{OnTable}\ \texttt{if}\ \top\ \texttt{before}\ \neg\mathsf{RightUp} \tag{5.5}$$
$$\texttt{effect}\ \mathsf{RightUp}\ \texttt{causes}\ \neg\mathsf{OnTable}\ \texttt{if}\ \top\ \texttt{before}\ \neg\mathsf{LeftUp} \tag{5.6}$$

For the actions of lifting just one side of the table, it now suffices to specify their immediate, direct effects $\texttt{action}\ \mathsf{LiftLeft}\ \texttt{causes}\ \mathsf{LeftUp}$ and $\texttt{action}\ \mathsf{LiftRight}\ \texttt{causes}\ \mathsf{RightUp}$, which completes the description of the domain Θ_{Spill}.

Although it might at first glance seem to be an overkill to have two contexts, the expressiveness gained through the distinction is crucial here and in general: the context of the bowl being on the table must be checked in the starting state, since it might become false during action execution (when the bowl falls off the table); the context of the other side of the table not being up must be checked in the resulting state, since it could become true during action execution (when both sides are lifted simultaneously). It is important to note that we do not care exactly *how* the change in the trigger or the terminal context was established.

In an early attempt to overcome the restriction to direct effects from STRIPS-like systems, [Wilkins, 1988] already proposed to use statements like (5.2), that he called *domain rules*. However, their semantics was only defined operationally by his implemented system.[1] In the reasoning about actions community, on the other hand, researchers started out with simpler causation statements [Lin, 1995; Thielscher, 1995; McCain and Turner, 1995] that however had a clear-cut meaning. Here, we in a sense reunite these two lines of research by providing a declarative semantics for Wilkins' domain rules.

Using the notions of states and state update as above, we now formally define the meaning of indirect effect laws. For the following definition, we take open direct and indirect effect laws to represent the respective sets of their ground instances.

[1] To his credit: he *did* provide Situation Calculus formulas to illustrate the intended meaning of domain rules. Alas in the case of conflicts between effects, he proposed the preference-handling mechanism "take the effect that has been derived earlier."

Definition 5.5. Let Θ be a *clerD* action domain specification, α be a ground action and let S, T be states. Define

$$S_0^\alpha \stackrel{\text{def}}{=} \{\psi \mid \underline{\texttt{action}}\ \alpha\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi \in \Theta, S \models \Phi\}$$

and for $i \geq 0$

$$S_{i+1}^\alpha \stackrel{\text{def}}{=} S_i^\alpha \cup \{\psi \mid \underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2 \in \Theta, S \models \Phi_1 \wedge \neg\chi, S_i^\alpha \models \chi, T \models \Phi_2\}^2$$

$$S_\infty^\alpha \stackrel{\text{def}}{=} \bigcup_{i=0}^{\infty} S_i^\alpha$$

T is called a *successor state of S for α* iff $T = S + S_\infty^\alpha$.

So in order to verify that a given state T is indeed a successor state, we must be able to reconstruct it in a well-founded way. First, we figure out all the direct action effects in S_0^α. We then repeatedly apply indirect effect laws to construct a set S_∞^α of literals that contains the direct and indirect atomic effects of the action. It only remains to check whether updating the starting state S with these effects does lead to T.

Example 5.1 (Continued)**.** Applying the action LiftLeft to the above depicted initial state $S = \{\mathsf{OnTable}, \neg\mathsf{LeftUp}, \neg\mathsf{RightUp}, \neg\mathsf{Spill}\}$, we can by means of Definition 5.5 verify that the state where the soup has spilled and the bowl fell off the table, $T = \{\neg\mathsf{OnTable}, \mathsf{LeftUp}, \neg\mathsf{RightUp}, \mathsf{Spill}\}$, is a successor state of S for LiftLeft:

The direct effects are $S_0^{\mathsf{LiftLeft}} = \{\mathsf{LeftUp}\}$, which makes (w.r.t. S) the indirect effect laws (5.3) and (5.5) applicable. We get $S_1^{\mathsf{LiftLeft}} = S_0^{\mathsf{LiftLeft}} \cup \{\mathsf{Spill}, \neg\mathsf{OnTable}\}$. No more indirect effect laws are triggered through these effects, therefore $S_2^{\mathsf{LiftLeft}} = S_1^{\mathsf{LiftLeft}} = S_\infty^{\mathsf{LiftLeft}}$. Lastly, we verify that $S + S_\infty^{\mathsf{LiftLeft}} = T$: $\{\mathsf{OnTable}, \neg\mathsf{LeftUp}, \neg\mathsf{RightUp}, \neg\mathsf{Spill}\} + \{\mathsf{LeftUp}, \mathsf{Spill}, \neg\mathsf{OnTable}\} = \{\neg\mathsf{OnTable}, \mathsf{LeftUp}, \neg\mathsf{RightUp}, \mathsf{Spill}\}$.

5.2 An Axiomatic Solution to the Ramification Problem

We now present the general, first-order effect axiom that will be used to solve the ramification problem. It uses the same axiomatisation technique as the effect axiom from the previous chapter, with the only difference that *DirT* and *DirF* are not used as predicates but as macros for the right-hand sides of the direct effect formulas. To the basic causes, persistence and direct effects, we add a third cause, indirect effects. The idea is to express them as implications and take care that inferences in the contrapositive, non-causal direction are not possible. The macros $IndT(f, s, t)$ and $IndF(f, s, t)$ express that fluent f is an indirect (positive or negative, respectively) effect of an action occurring from s to t. For an indirect effect law $r = \underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2$, the indirect effect ψ materialises whenever the relationship has been *triggered*, that is, whenever the initial context Φ_1 holds at the starting time point s, the terminal context Φ_2 holds at the resulting time point t and the trigger χ has changed from untrue to true from s to t. As for direct effect laws, by the sign of an indirect effect law we refer to the sign of its effect, $sign(r) \stackrel{\text{def}}{=} sign(\psi)$. To access the effect literal of an indirect effect law r, we use $Effect(r) \stackrel{\text{def}}{=} \psi$ and do so similarly for the trigger: $Trigger(r) \stackrel{\text{def}}{=} \chi$.

[2] Notice the occurrence of T in the definition of the S_i^α.

Definition 5.6. Let $r(\vec{y}) = \underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2$ be an indirect effect law with free variables among $\vec{y}$, and let s, t : TIME be variables.

$$Triggered_{r(\vec{y})}(s,t) \stackrel{\text{def}}{=} \Phi_1[s] \wedge \Phi_2[t] \wedge \neg\chi[s] \wedge \chi[t] \tag{5.7}$$

Let Θ be a *clerD* domain and f be a variable of sort FLUENT.

$$IndT(f,s,t) \stackrel{\text{def}}{=} \bigvee_{r(\vec{y})\in\Theta, sign(r)=+} (\exists\vec{y})(f = Effect(r(\vec{y})) \wedge Triggered_{r(\vec{y})}(s,t)) \tag{5.8}$$

$$IndF(f,s,t) \stackrel{\text{def}}{=} \bigvee_{r(\vec{y})\in\Theta, sign(r)=-} (\exists\vec{y})(f = Effect(r(\vec{y})) \wedge Triggered_{r(\vec{y})}(s,t)) \tag{5.9}$$

According to these macros, a fluent f is an indirect effect from s to t if there is a corresponding indirect effect law with effect f that triggered from s to t. The macros are straightforwardly integrated into the effect axiom as follows.

Definition 5.7. Let A be a function into sort ACTION. The *effect axiom* Υ_A *with conditional effects, the frame assumption and ramifications* is of the form (3.16), where

$$CausedT(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} FrameT(f,s,t) \vee DirT(f, A(\vec{x}), s, t) \vee IndT(f,s,t) \tag{5.10}$$

$$CausedF(f, A(\vec{x}), s, t) \stackrel{\text{def}}{=} FrameF(f,s,t) \vee DirF(f, A(\vec{x}), s, t) \vee IndF(f,s,t) \tag{5.11}$$

While the approach presented so far works well for simple ramification domains and easily copes with instantaneous effect propagation, it still harbours a serious flaw: it cannot handle cyclic fluent dependencies.

Example 5.8 (Gear Wheels [Van Belleghem et al., 1998])**.** Two interlocked gear wheels can be separately turned and stopped. Let the fluents $\mathsf{W}_1, \mathsf{W}_2$ express that the first (respectively second) wheel is turning. The actions to initiate and stop turning are Turn_i and Stop_i with effects $\underline{\texttt{action}}\ \mathsf{Turn}_i\ \underline{\texttt{causes}}\ \mathsf{W}_i$ and $\underline{\texttt{action}}\ \mathsf{Stop}_i\ \underline{\texttt{causes}}\ \neg\mathsf{W}_i$ for $i = 1, 2$. A trivial action, Wait, has no direct effects. The causal relation between the wheels is: whenever the first wheel is turned (resp. stopped), it causes the second one to turn (resp. stop), and vice versa:

$\underline{\texttt{effect}}\ \mathsf{W}_1\ \underline{\texttt{causes}}\ \mathsf{W}_2$ $\quad$ $\underline{\texttt{effect}}\ \neg\mathsf{W}_1\ \underline{\texttt{causes}}\ \neg\mathsf{W}_2$

$\underline{\texttt{effect}}\ \mathsf{W}_2\ \underline{\texttt{causes}}\ \mathsf{W}_1$ $\quad$ $\underline{\texttt{effect}}\ \neg\mathsf{W}_2\ \underline{\texttt{causes}}\ \neg\mathsf{W}_1$

Let us compile the (positive half of the) effect axiom for Wait. There are no direct effects, so Definitions 5.6 and 5.7 yield $CausedT(f, \mathsf{Wait}, s, t) \equiv FrameT(f,s,t) \vee IndT(f,s,t)$ and

$$\begin{aligned} IndT(f,s,t) = {} & (f = \mathsf{W}_2 \wedge \neg Holds(\mathsf{W}_1, s) \wedge Holds(\mathsf{W}_1, t)) \vee \\ & (f = \mathsf{W}_1 \wedge \neg Holds(\mathsf{W}_2, s) \wedge Holds(\mathsf{W}_2, t)) \end{aligned}$$

Consider the structure $\mathfrak{M}$ with $\text{TIME}^{\mathfrak{M}} = \{\sigma, \tau\}$, $\sigma <^{\mathfrak{M}} \tau$, $Poss^{\mathfrak{M}} = \{(\mathsf{Wait}, \sigma, \tau)\}$ and $Holds^{\mathfrak{M}} = \{(\mathsf{W}_1, \tau), (\mathsf{W}_2, \tau)\}$. Together with the variable evaluation $\{s \mapsto \sigma, t \mapsto \tau\}$, $\mathfrak{M}$ is a model for effect axiom (3.16) for Wait where both wheels initially stand still and magically start turning – one being the cause for the other and vice versa. This is undesired as Wait is intended to have no effect at all.

5.2.1 Loops and Loop Formulas

Much as in the case of Clark's completion of normal logic programs, our compilation of indirect effect laws into effect axioms allows too many models for fluents (that is, reified predicates) that cyclically depend on each other. We propose a solution to this problem in the spirit of loop formulas [Lin and Zhao, 2004] for normal logic programs. In order for the approach to stay practical, we however have to restrict the syntax of the indirect effect laws in Θ:

1. For mere technical reasons we require that for any $r_1 \neq r_2 \in \Theta$, we have $Var(r_1) \cap Var(r_2) = \emptyset$.

2. For each indirect effect law $\underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2 \in \Theta$, we stipulate $Var(\chi) \subseteq Var(\psi)$, that is, there may not be local variables in triggers.

3. We do not use function symbols with arity greater zero as arguments of sort FLUENT.

The latter two constraints guarantee the existence of a finite, complete set of loops [Chen et al., 2006]. This set can be identified by a simple algorithm operating on open indirect effect laws, which makes our definition of effect axioms entirely constructive and easily automatable.

Throughout the following definitions, we will make explicit use of substitutions, unifiers and most general unifiers (*mgu*s). Their domains and ranges are understood to be built from the domain signature used for specifying the indirect effect laws. For unification, negation is treated as a unary function. The definition of loops here follows the one of [Chen et al., 2006] given in Chapter 2.

Definition 5.9. Let Θ be a *clerD* action domain specification over a domain signature Ξ. The *influence graph* G_Θ *of* Θ is the (possibly infinite) directed graph $G_\Theta \stackrel{\text{def}}{=} (V, E)$, where V is the set of all fluent literals over Ξ and for all $\underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2 \in \Theta$ and substitutions θ, there is an edge $(\chi\theta, \psi\theta) \in E$. A finite, nonempty set L of literals constitutes a *loop* iff for all $\mu, \nu \in L$, there is a directed, non-zero length path from μ to ν in the subgraph of G_Θ induced by L. An indirect effect law $r = \underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2 \in \Theta$ *leads into the loop* L iff (1) there exists a $\mu \in L$ and a substitution $\theta_r^- = mgu(\psi, \mu)$ and (2) for all substitutions θ' with $(\exists\theta'')\theta' = \theta_r^-\theta''$ we have $\chi\theta' \notin L\theta'$. Then $\Theta_L^- \stackrel{\text{def}}{=} \{r\theta_r^- \mid r \in \Theta \text{ and } r \text{ leads into the loop } L\}$.

For example, the indirect effect laws of the gear wheel domain give rise to two loops, $L_1 = \{\mathsf{W}_1, \mathsf{W}_2\}$ and $L_2 = \{\neg\mathsf{W}_1, \neg\mathsf{W}_2\}$.

From the work of [Chen et al., 2006], we know that although there may be infinitely many loops in general, there is always a finite set $Loops(\Theta)$ that captures all of them.

Theorem 5.10. *Let Θ be a clerD domain that satisfies the restrictions above. There exists a finite set $Loops(\Theta)$ such that every loop L of Θ is subsumed by some $L' \in Loops(\Theta)$.*

Proof. We define a logic program Λ_Θ such that there is a one-to-one correspondence between the loops of Θ and the loops of Λ_Θ. For each function symbol $F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n \to$ FLUENT, introduce two new predicates $P_F : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n$ and $P_{\neg F} : \mathfrak{s}_1 \times \cdots \times \mathfrak{s}_n$, and define

$$\Lambda_\Theta \stackrel{\text{def}}{=} \left\{ P_{\langle\psi\rangle^+}(\vec{x_2}) \leftarrow P_{\langle\chi\rangle^+}(\vec{x_1}) \;\middle|\; \underline{\texttt{effect}}\ \chi(\vec{x_1})\ \underline{\texttt{causes}}\ \psi(\vec{x_2})\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2 \in \Theta \right\}$$

Correspondence of the loops follows straightforwardly from the definition of loops in Chapter 2 and Definition 5.9. By Lemma 2.12, there is a finite, complete set of loops for Λ_Θ. Due to the correspondence, it is also complete for Θ. □

Having defined the loops of a given domain and being sure they can be captured finitely, we can now proceed to define the corresponding loop formulas. The idea of loop formulas is to eliminate the models that arise due to "spontaneous" activation of loops for which no *external support* exists. In the case of logic programs, the external support that counts as "legal" cause for loop activation is a program rule leading into the loop. In our case, the direct effects of an action have to be taken into account as potential reasons for loop activation, too. A loop can also be activated by another loop – but then the union of the two is again a loop, so this case is implicitly catered for. This general form of direct and indirect loop activation is illustrated in Figure 5.1.

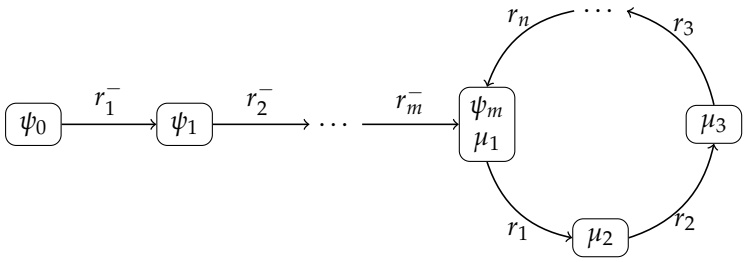


Figure 5.1: General form of loop activation. The leftmost node ψ_0 represents a direct action effect. The cycle on the right depicts a subset-maximal loop $L = \{\mu_1, \ldots, \mu_n\}$ of length n where $\psi_m = \mu_1$. For $m = 0$, we have the case of direct activation. Otherwise, for any natural m, there is a chain of m effects, and the loop is activated by the indirect effect law r_m^- leading into the loop L. Whenever r_m^- is itself part of a loop L', then $L \cup L'$ is again a loop and L is not subset-maximal.

When translating a logic program into a logical theory, the loop formulas are added to the predicate completion of the program.[3] In case of general effect axioms with their standard first-order semantics, loop formulas are "built into" the axioms, by enforcing the frame assumption for all fluent literals that could possibly change their truth value due to spontaneous loop activation. To achieve this for a given literal μ, we specify non-activation of all loops that could change μ as a sufficient cause for persistence of μ's truth value.

We use the notation $L(\vec{y})$ to explicitly refer to the free variables $\vec{y}$ mentioned in the loop L. Macro $DirActivated_{L(\vec{y})}(A(\vec{x}), s, t)$ denotes whether a loop $L(\vec{y})$ has been activated by action $A(\vec{x})$ from s to t. To find out if a loop has been activated by an indirect effect, macro $IndActivated_{L(\vec{y})}(s, t)$ checks whether the corresponding instance of an indirect effect law leading into the loop has been triggered from s to t.

Definition 5.11. Let Θ be a *clerD* action domain specification, $Loops(\Theta)$ be a finite and complete set of loops of Θ, $L(\vec{y}) \in Loops(\Theta)$, A be a function into sort ACTION and s, t be variables

[3]Meanwhile, there have been more general proposals where the loop formulas are added to the program as is, without forming the completion [Ferraris et al., 2006].

of sort TIME.

$$DirActivated_{L(\vec{y})}(A(\vec{x}),s,t) \stackrel{\text{def}}{\equiv} \bigvee_{F(\vec{y})\in L(\vec{y})} DirT(F(\vec{y}),A(\vec{x}),s,t) \vee \bigvee_{\neg F(\vec{y})\in L(\vec{y})} DirF(F(\vec{y}),A(\vec{x}),s,t) \tag{5.12}$$

$$IndActivated_{L(\vec{y})}(s,t) \stackrel{\text{def}}{\equiv} \bigvee_{r(\vec{z})\in\Theta_L^-} (\exists\vec{z})\left(Triggered_{r(\vec{z})}(s,t) \wedge \bigvee_{\mu(\vec{y})\in L(\vec{y})} Effect(r(\vec{z})) = \mu(\vec{y}) \right) \tag{5.13}$$

$$Activated_{L(\vec{y})}(A(\vec{x}),s,t) \stackrel{\text{def}}{\equiv} DirActivated_{L(\vec{y})}(A(\vec{x}),s,t) \vee IndActivated_{L(\vec{y})}(s,t) \tag{5.14}$$

Let f : FLUENT be a variable.

$$LoopFrameT(f,A(\vec{x}),s,t) \stackrel{\text{def}}{\equiv} Holds(f,s) \wedge \bigvee_{\substack{L(\vec{y})\in Loops(\Theta),\\ \neg F(\vec{y})\in L(\vec{y})}} (\exists\vec{y})\Big(f = F(\vec{y}) \wedge \neg Activated_{L(\vec{y})}(A(\vec{x}),s,t)\Big) \tag{5.15}$$

$$LoopFrameF(f,A(\vec{x}),s,t) \stackrel{\text{def}}{\equiv} \neg Holds(f,s) \wedge \bigvee_{\substack{L(\vec{y})\in Loops(\Theta),\\ F(\vec{y})\in L(\vec{y})}} (\exists\vec{y})\Big(f = F(\vec{y}) \wedge \neg Activated_{L(\vec{y})}(A(\vec{x}),s,t)\Big) \tag{5.16}$$

Macros (5.15) and (5.16) formalise the intuition that the truth value of a fluent should persist whenever there is some loop $L(\vec{y})$ that could change the truth value, but which has not been activated from s to t. If there indeed exists such a loop, then all of the loops containing the literal are activated whenever one of them is activated (cf. Figure 5.1). However, if for a fluent literal ψ there is *no* loop containing $\neg\psi$, then $LoopFrame(\psi,A(\vec{x}),s,t) = \psi[s] \wedge \bot$ is equivalent to false. This is justified since ψ may never spontaneously change, so the precaution provided by the loop formulas is unnecessary.

If the specified causal relationships do not give rise to any loops, both (5.15) and its negative version are equivalent to $\bot$. The new causes are now added to the effect axiom as usual.

Definition 5.12. Let A be a function into sort ACTION. The *effect axiom* Y_A *with conditional effects, the frame assumption and ramifications* is of the form (3.16), where

$$CausedT(f,A(\vec{x}),s,t) \stackrel{\text{def}}{\equiv} FrameT(f,s,t) \vee DirT(f,A(\vec{x}),s,t) \vee IndT(f,s,t) \vee LoopFrameT(f,A(\vec{x}),s,t) \tag{5.17}$$

Example 5.8 (Continued)**.** The Wait action has no direct effects and for neither of the loops L_1, L_2 exists an indirect effect law leading into the loop, hence $Activated_{L_1}(\mathsf{Wait},s,t) = \bot$ and $Activated_{L_2}(\mathsf{Wait},s,t) = \bot$. The new causes added to the effect axiom say that through Wait, the truth value of the loop literals must persist:

$$LoopFrameT(f,\mathsf{Wait},s,t) \equiv Holds(f,s) \wedge (f = \mathsf{W}_1 \vee f = \mathsf{W}_2)$$
$$LoopFrameF(f,\mathsf{Wait},s,t) \equiv \neg Holds(f,s) \wedge (f = \mathsf{W}_1 \vee f = \mathsf{W}_2)$$

For the undesired interpretation $\mathfrak{I}$ seen earlier we now have $\mathfrak{I} \models LoopFrameF(\mathsf{W}_1,\mathsf{Wait},\sigma,\tau)$ but $\mathfrak{I} \not\models \neg Holds(\mathsf{W}_1,\tau)$: $\mathfrak{I}$ is no model for the effect axiom any more, just as desired.

This concludes the compilation of direct and indirect effect laws into first-order effect axioms. It remains to give the immediate definition of a domain axiomatisation for a domain with indirect effect laws.

Definition 5.13. Let Θ be a *clerD* action domain specification. Its corresponding domain axiomatisation is $\Sigma = \Omega \cup \Pi \cup \Upsilon \cup \Sigma_{aux}$, where Ω defines the time structure, for each function into sort ACTION, Π contains a precondition axiom (3.4) or (3.6) depending on the time structure and Υ contains an effect axiom according to Definition 5.12, and Σ_{aux} contains the unique-names axioms for sorts FLUENT and ACTION.

5.2.2 From Trigger Formulas to Trigger Literals

Up to here, we considered only triggers that are literals. The question arises whether this is a proper limitation, especially in light of the *indirect effect rules* `initiating` X `causes` ψ `if` Φ of [Van Belleghem et al., 1998], where ψ is a fluent literal and X and Φ can be general propositional fluent formulas.

It turns out that even more general indirect effect rules where X and Φ are first-order quantifier-free (implicitly universally quantified) formulas can be transformed into indirect effect laws by the procedure below. It takes as input an indirect effect rule `initiating` X `causes` ψ `if` Φ.

1. Transform X into its disjunctive normal form (DNF) $X = X_1 \vee \ldots \vee X_m$. This might involve a (worst-case) exponential blowup, which is however also inherently present in [Van Belleghem et al., 1998], where all possible "activating sets" have to be considered. As an example, we look at the indirect effect rule

   `initiating` LeftUp $\not\equiv$ RightUp `causes` Spill `if` OnTable

   Transforming the trigger formula LeftUp $\not\equiv$ RightUp into disjunctive normal form yields $(\text{LeftUp} \wedge \neg\text{RightUp}) \vee (\neg\text{LeftUp} \wedge \text{RightUp})$.

2. Create the intermediate indirect effect rules `initiating` X_i `causes` ψ `if` Φ for $1 \leq i \leq m$. Note that all trigger formulas are conjunctions of literals. In the example, this step produces

   `initiating` LeftUp $\wedge$ $\neg$RightUp `causes` Spill `if` OnTable
   `initiating` $\neg$LeftUp $\wedge$ RightUp `causes` Spill `if` OnTable

3. For each `initiating` $\chi_1 \wedge \ldots \wedge \chi_n$ `causes` ψ `if` Φ, create for $i = 1, \ldots, n$ the indirect effect laws

   $$\underline{\texttt{effect}}\ \chi_i\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi\ \underline{\texttt{before}} \bigwedge_{j=1,\ldots,n,j\neq i} \chi_j$$

   For the example, we get the indirect effect laws (5.3) and (5.4) we already know and the two new rules

   `effect` $\neg$RightUp `causes` Spill `if` OnTable `before` LeftUp
   `effect` $\neg$LeftUp `causes` Spill `if` OnTable `before` RightUp

   for the case when both sides are up and only one is let down.

The intuition behind the procedure is straightforward. Assume the trigger formula X is activated from s to t, that is $\neg X[s]$ and $X[t]$. Then $\neg X_i[s]$ for all $1 \leq i \leq m$ and $X_j[t]$ for some $1 \leq j \leq m$. In particular $\neg X_j[s] \wedge X_j[t]$. Since $X_j = \chi_{j1} \wedge \ldots \wedge \chi_{jn}$ is a conjunction of literals, this in turn means that there is at least one χ_{jk} for $1 \leq k \leq n$ with $\neg\chi_{jk}[s] \wedge \chi_{jk}[t]$. The procedure now simply creates indirect effect laws for all possible χ_{jk}.

5.3 Correctness of the Solution

To show that the axioms just presented indeed capture the state-transition semantics given earlier, we first define how the two semantics link together. Below, we define how two time points of a first-order interpretation that are connected by an action application give rise to two states. This definition will be the basis of the correspondence result of this section.

Definition 5.14. Let Ξ be a domain signature and $\mathfrak{I} = (\mathfrak{M}, \mathfrak{V})$ be an interpretation for Ξ. Let σ and τ be ground terms of sort TIME, α be a ground term of sort ACTION and assume w.l.o.g. that $(\alpha^{\mathfrak{M}}, \sigma^{\mathfrak{M}}, \tau^{\mathfrak{M}}) \in Poss^{\mathfrak{M}}$. Define two states S and T as follows: for a ground fluent literal ψ, set $\psi \in S$ if and only if $\mathfrak{I} \models \psi[\sigma]$, and $\psi \in T$ if and only if $\mathfrak{I} \models \psi[\tau]$.

Throughout the proof, we identify α, σ and τ with their interpretations under $\mathfrak{M}$. We then write $\mathrm{Y}_\alpha[\sigma, \tau]$ to refer to effect axiom (3.16) for α where the TIME variables s and t have been replaced by σ and τ, respectively. An important first observation concerning the proof is that by the definition of S and T we have for all fluent formulas Φ that $S \models \Phi$ iff $\mathfrak{I} \models \Phi[\sigma]$ and $T \models \Phi$ iff $\mathfrak{I} \models \Phi[\tau]$. An immediate consequence thereof is this:

Lemma 5.15. *For a ground action α and direct effect law* `action` α `causes` ψ `if` Φ*, we have $\psi \in S_0^\alpha$ iff $\mathfrak{I} \models Dir(\psi, \alpha, \sigma, \tau)$.*

For the rest of the results, assume a fixed domain Θ over the signature Ξ. The next lemma is easy to prove via induction. It says that whenever $\mathfrak{I}$ is a model for α's effect axiom, the action effects S_∞^α have materialised in the resulting state T.

Lemma 5.16. $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$ *implies* $S_\infty^\alpha \subseteq T$.

Proof. We use induction to show $S_i^\alpha \subseteq T$ for all $i \geq 0$.

$i = 0$. Let $\mu \in S_0^\alpha$. By Lemma 5.15, $\mathfrak{I} \models Dir(\mu, \alpha, \sigma, \tau)$. Together with $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$, we have $\mathfrak{I} \models \mu[\tau]$. The definition of $\mathfrak{I}$ yields $\mu \in T$.

$i \to i+1$. Let $\mu \in S_{i+1}^\alpha \setminus S_i^\alpha$. According to Definition 5.5, there is an indirect effect law $r =$ `effect` χ `causes` μ `if` Φ_1 `before` $\Phi_2 \in \Theta$ such that $S \models \Phi_1 \wedge \neg\chi$, $S_i^\alpha \models \chi$ and $T \models \Phi_2$. Applying the induction hypothesis to $\chi \in S_i^\alpha$ yields $\chi \in T$ and thus, by definition of $\mathfrak{I}$, we have $\mathfrak{I} \models \chi[\tau]$, $\mathfrak{I} \models Triggered_r(\sigma, \tau)$ and $\mathfrak{I} \models Ind(\mu, \sigma, \tau)$. By the presumption $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$ we get $\mathfrak{I} \models \mu[\tau]$ and thus $\mu \in T$ by definition of $\mathfrak{I}$. □

Now any fluent literal of which the effect axiom says it is an indirect effect can be found in the set of action effects.

Lemma 5.17. *Let $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$. For any ground fluent literal μ, $\mathfrak{I} \models Ind(\mu, \sigma, \tau)$ implies $\mu \in S_\infty^\alpha$.*

Proof. Let $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$ and $\mathfrak{I} \models Ind(\mu, \sigma, \tau)$. Then there is an instance of an indirect effect law `effect` χ `causes` μ `if` Φ_1 `before` $\Phi_2 \in \Theta$ such that $\mathfrak{I} \models \Phi_1[\sigma] \wedge \neg\chi[\sigma]$ and $\mathfrak{I} \models \Phi_2[\tau] \wedge \chi[\tau]$. From $\mathfrak{I} \models \chi[\tau]$ by $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$ we get $\mathfrak{I} \models Caused(\chi, \alpha, \sigma, \tau) \wedge \neg Caused(\neg\chi, \alpha, \sigma, \tau)$. Since additionally $\mathfrak{I} \models \neg\chi[\sigma]$ yields $\mathfrak{I} \models \neg Frame(\chi, \sigma, \tau)$ and $\mathfrak{I} \models \neg LoopFrame(\chi, \alpha, \sigma, \tau)$, we must have one of $\mathfrak{I} \models Dir(\chi, \alpha, \sigma, \tau)$ or $\mathfrak{I} \models Ind(\chi, \alpha, \sigma, \tau)$.

- $\mathfrak{I} \models \mathit{Dir}(\chi, \alpha, \sigma, \tau)$. Then Lemma 5.15 shows $\chi \in S_0^\alpha$ and Definition 5.5 yields $\mu \in S_\infty^\alpha$.
- $\mathfrak{I} \models \mathit{Ind}(\chi, \alpha, \sigma, \tau)$. By $\mathfrak{I} \models \neg \mathit{CausedT}(\neg\chi, \alpha, \sigma, \tau)$, we get $\mathfrak{I} \models \neg \mathit{LoopFrame}(\neg\chi, \alpha, \sigma, \tau)$, whose macro-expansion means $\mathfrak{I} \models \neg(\neg\chi[\sigma] \wedge \bigvee_{L \in \mathit{Loops}(\Theta)_{\chi \in L}} \neg \mathit{Activated}_L(\alpha, \sigma, \tau))$. Together with $\mathfrak{I} \models \neg\chi[\sigma]$, this yields $\mathfrak{I} \models \bigwedge_{L \in \mathit{Loops}(\Theta)_{\chi \in L}} \mathit{Activated}_L(\alpha, \sigma, \tau)$, that is, all loops containing χ have been activated.

  1. χ is not contained in any loop. Due to the syntactic restriction of FLUENTS' arguments, there is a finite sequence $r_1, \ldots, r_m$ of indirect effect laws such that $\mathfrak{I} \models \mathit{Dir}(\mathit{Trigger}(r_1), \alpha, \sigma, \tau)$, $\mathit{Effect}(r_i) = \mathit{Trigger}(r_{i+1})$ for $1 \leq i \leq m-1$ and $\mathit{Effect}(r_m) = \chi$. By Definition 5.5, we get $\chi \in S_\infty^\alpha$ and thus $\mu \in S_\infty^\alpha$.
  2. Let L be a maximal, ground loop containing χ such that $\mathfrak{I} \models \mathit{Activated}_L(\alpha, \sigma, \tau)$. We make a case distinction on the macro expansion of $\mathit{Activated}_L(\alpha, \sigma, \tau)$.
     (a) $\mathfrak{I} \models \mathit{DirActivated}_L(\alpha, \sigma, \tau)$. Then for some $\mu' \in L$ we have $\mathfrak{I} \models \mathit{Dir}(\mu', \alpha, \sigma, \tau)$ and therefore $\mu' \in S_0^\alpha$. By the definition of a loop, there is a sequence $r_1, \ldots, r_n$ of indirect effect laws such that $\mu' = \mathit{Trigger}(r_1)$ and $\mathit{Effect}(r_i) = \mathit{Trigger}(r_{i+1})$ for $1 \leq i \leq n-1$. Now $L = \{\mathit{Effect}(r_i) \mid 1 \leq i \leq n\}$ and Definition 5.5 yield $L \subseteq S_\infty^\alpha$, whence $\chi \in S_\infty^\alpha$ and $\mu \in S_\infty^\alpha$.
     (b) $\mathfrak{I} \models \mathit{IndActivated}_L(\alpha, \sigma, \tau)$. Then there is an indirect effect law $r \in \Theta_L^-$ leading into the loop with $\mathfrak{I} \models \mathit{Triggered}_r(\sigma, \tau)$. Since L is maximal, r is itself not part of another loop. We now have $\mathfrak{I} \models \mathit{Ind}(\mathit{Trigger}(r), \sigma, \tau)$ where $\mathit{Trigger}(r)$ is not part of a loop. Consequently, $\mathit{Effect}(r) \in S_\infty^\alpha$ and $L \subseteq S_\infty^\alpha$ can be argued for as above (item 1). □

With the two lemmata, it becomes straightforward to prove soundness of the axiomatisation. This is actually the "hard" direction of the overall proof, since inability to deal with cyclic dependencies results in unsound prediction of effects.

Theorem 5.18 (Soundness)**.** $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$ *implies* $T = S + S_\infty^\alpha$.

Proof. Let $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$.

"$\supseteq$": Let $\psi \in S + S_\infty^\alpha = (S \setminus \overline{S_\infty^\alpha}) \cup S_\infty^\alpha$.

  1. $\psi \in S_\infty^\alpha = \bigcup_{i=0}^\infty S_i^\alpha$. Lemma 5.16 makes $\psi \in T$ immediate.
  2. $\psi \in S \setminus \overline{S_\infty^\alpha}$, i.e. $\psi \in S$ and $\neg\psi \notin S_\infty^\alpha$. Then $\mathfrak{I} \models \psi[\sigma]$ and $\mathfrak{I} \models \neg \mathit{Frame}(\neg\psi, \sigma, \tau) \wedge \neg \mathit{LoopFrame}(\neg\psi, \sigma, \tau)$. By Lemma 5.15 and the presumption $\neg\psi \notin S_\infty^\alpha$, also $\mathfrak{I} \models \neg \mathit{Dir}(\neg\psi, \alpha, \sigma, \tau)$. By Lemma 5.17, $\neg\psi \notin S_\infty^\alpha$ implies $\mathfrak{I} \not\models \mathit{Ind}(\neg\psi, \sigma, \tau)$. Hence $\mathfrak{I} \models \neg \mathit{Caused}(\neg\psi, \alpha, \sigma, \tau)$, from which we can conclude $\mathfrak{I} \models \psi[\tau]$ and thus $\psi \in T$.

"$\subseteq$": Let $\psi \in T$. The definition of $\mathfrak{I}$ tells us $\mathfrak{I} \models \psi[\tau]$; $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$ tells us $\mathfrak{I} \models \mathit{Caused}(\psi, \alpha, \sigma, \tau) \wedge \neg \mathit{Caused}(\neg\psi, \alpha, \sigma, \tau)$. Macro-expansion of *Caused* yields that $\mathfrak{I} \models \mathit{Dir}(\psi, \alpha, \sigma, \tau)$ or $\mathfrak{I} \models \mathit{Ind}(\psi, \sigma, \tau)$ or $\mathfrak{I} \models \mathit{Frame}(\psi, \sigma, \tau)$ or $\mathfrak{I} \models \mathit{LoopFrame}(\psi, \alpha, \sigma, \tau)$.

  1. $\mathfrak{I} \models \mathit{Dir}(\psi, \alpha, \sigma, \tau)$. Lemma 5.15 immediately yields $\psi \in S_0^\alpha \subseteq S + S_\infty^\alpha$.
  2. $\mathfrak{I} \models \mathit{Ind}(\psi, \sigma, \tau)$. Using Lemma 5.17, $\psi \in S_\infty^\alpha \subseteq S + S_\infty^\alpha$ is immediate.
  3. $\mathfrak{I} \models \mathit{Frame}(\psi, \sigma, \tau)$ or $\mathfrak{I} \models \mathit{LoopFrame}(\psi, \alpha, \sigma, \tau)$. Then $\mathfrak{I} \models \psi[\sigma]$ and $\psi \in S$. Now $\psi \in T$ by T being a state implies $\neg\psi \notin T$, which by Lemma 5.16 yields $\neg\psi \notin S_\infty^\alpha$. Hence $\psi \notin \overline{S_\infty^\alpha}$ and $\psi \in S + S_\infty^\alpha$. □

For the converse direction, the first lemma says that whenever T is a successor state of S for α, then the effect axiom correctly establishes all indirect effects.

Lemma 5.19. *$T = S + S^\alpha_\infty$ and $\mathfrak{I} \models Ind(\mu, \sigma, \tau)$ imply $\mathfrak{I} \models \mu[\tau]$.*

Proof. Let $T = S + S^\alpha_\infty$ and $\mathfrak{I} \models Ind(\mu, \sigma, \tau)$. Then there exists a ground instance of an indirect effect law $r = \underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \mu\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2$ such that $\mathfrak{I} \models \Phi_1[\sigma] \wedge \Phi_2[\tau] \wedge \neg\chi[\sigma] \wedge \chi[\tau]$. By definition of $\mathfrak{I}$, $S \models \Phi_1 \wedge \neg\chi$ and $T \models \Phi_2 \wedge \chi$. Since $\chi \notin S$, $\chi \in T$ and $T = S + S^\alpha_\infty$, there must be an $i \geq 0$ such that $\chi \in S^\alpha_i$. By Definition 5.5, $\mu \in S^\alpha_{i+1} \subseteq S^\alpha_\infty \subseteq T$; thus $\mathfrak{I} \models \mu[\tau]$. □

Indeed, whenever a fluent literal is found to hold at the resulting time point and it was part of the update, then the effect axiom says it was an effect of the action.

Lemma 5.20. *Let $T = S + S^\alpha_\infty$ and $\mathfrak{I} \models \mu[\tau]$. Then $\mu \in S^\alpha_\infty$ implies $\mathfrak{I} \models Dir(\mu, \alpha, \sigma, \tau) \vee Ind(\mu, \sigma, \tau)$.*

Proof. Let $\mu \in S^\alpha_\infty$ and consider the smallest i such that $\mu \in S^\alpha_i$.

1. $i = 0$, that is, $\mu \in S^\alpha_0$. Lemma 5.15 immediately yields $\mathfrak{I} \models Dir(\mu, \alpha, \sigma, \tau)$.
2. $i > 0$. By Definition 5.5, there is an indirect effect law $r = \underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \mu\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2$ such that $S \models \Phi_1 \wedge \neg\chi$, $S^\alpha_{i-1} \models \chi$ and $T \models \Phi_2$. Now $S^\alpha_{i-1} \subseteq T$ implies $T \models \chi$, hence $\mathfrak{I} \models \Phi_1[\sigma] \wedge \Phi_2[\tau] \wedge \neg\chi[\sigma] \wedge \chi[\tau]$. That means we have $\mathfrak{I} \models Triggered_r(\sigma, \tau)$ and hence $\mathfrak{I} \models Ind(\mu, \sigma, \tau)$. □

The completeness result below now says that whenever T is a successor state of S for α, the corresponding interpretation $\mathfrak{I}$ is a model for the respective instance of the effect axiom.

Theorem 5.21 (Completeness)**.** *$T = S + S^\alpha_\infty$ implies $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$.*

Proof. Let $T = S + S^\alpha_\infty$. We show $\mathfrak{I} \models \mathrm{Y}_\alpha[\sigma, \tau]$ by showing $\mathfrak{I} \models \psi[\tau]$ iff $\mathfrak{I} \models Caused(\psi, \alpha, \sigma, \tau)$.

"if": Let $\mathfrak{I} \models Caused(\psi, \alpha, \sigma, \tau)$. We make a case distinction according to the macro-expansion of $Caused(\psi, \alpha, \sigma, \tau)$.

1. $\mathfrak{I} \models Dir(\psi, \alpha, \sigma, \tau)$. Lemma 5.15 yields $\psi \in S^\alpha_0 \subseteq S + S^\alpha_\infty$. The presumption $S + S^\alpha_\infty = T$ now shows $\mathfrak{I} \models \psi[\tau]$.
2. $\mathfrak{I} \models Ind(\psi, \sigma, \tau)$. By Lemma 5.19, we have $\mathfrak{I} \models \psi[\tau]$.
3. $\mathfrak{I} \models Frame(\psi, \sigma, \tau)$. Then $\mathfrak{I} \models \psi[\tau]$ is immediate.
4. $\mathfrak{I} \models LoopFrame(\psi, \alpha, \sigma, \tau)$. This entails $\mathfrak{I} \models \psi[\sigma]$ and thus $\psi \in S$. Furthermore, $\mathfrak{I} \models \bigvee_{L \in Loops(\Theta), \neg\psi \in L} \neg Activated_L(\alpha, \sigma, \tau)$, that is, none of the loops involving $\neg\psi$ has been activated, whereby we can conclude $\neg\psi \notin S^\alpha_\infty$. Now $\psi \in T = S + S^\alpha_\infty$ yields the desired $\mathfrak{I} \models \psi[\tau]$.

"only if": Let $\mathfrak{I} \models \psi[\tau]$. We have to show $\mathfrak{I} \models Caused(\psi, \alpha, \sigma, \tau)$. The definition of $\mathfrak{I}$ yields $\psi \in T$. By the presumption, we know $\psi \in S + S^\alpha_\infty = (S \setminus \overline{S^\alpha_\infty}) \cup S^\alpha_\infty$.

1. $\psi \in S^\alpha_\infty$. By Lemma 5.20, $\mathfrak{I} \models Caused(\psi, \alpha, \sigma, \tau)$.
2. $\psi \in (S \setminus \overline{S^\alpha_\infty})$, that is, $\psi \in S$ and $\neg\psi \notin S^\alpha_\infty$. From $\psi \in S$ and $\psi \in T$ we directly get $\mathfrak{I} \models Frame(\psi, \sigma, \tau)$ and thus $\mathfrak{I} \models Caused(\psi, \alpha, \sigma, \tau)$. □

5.4 Concluding Remarks

We presented a solution to the ramification problem that unifies and generalises existing approaches in that it (1) is independent of a particular time structure, (2) is based on first-order effect axioms and (3) deals with cyclic causal dependencies among fluents. Moreover, we have formally proved our axiomatisation correct for the presented state transition semantics. We also briefly indicated how to express ramifications that are triggered by formulas in our framework. To conclude the presentation, we show how indirect effects and defaults can be combined, and discuss related work on ramification.

5.4.1 Combining Ramifications and Defaults

It is clear that the approach for default reasoning about actions presented in Chapter 3 and the approach for reasoning about indirect effects of actions of this chapter can be combined into a single formalism that gives a meaning to *n-clerD* domains. Since the two are based on the same axiomatisation technique given by effect axiom (3.16), mild modifications to the respective isolated formalisms suffice to integrate them. For one, macros $IndT, IndF$ become predicates; to retain their meaning, we use the indirect effect formulas

$$IndT(f,s,t) \equiv \bigvee_{r(\vec{y})\in\Theta, sign(r)=+} (\exists \vec{y})(f = Effect(r(\vec{y})) \wedge Triggered_{r(\vec{y})}(s,t))$$
$$IndF(f,s,t) \equiv \bigvee_{r(\vec{y})\in\Theta, sign(r)=-} (\exists \vec{y})(f = Effect(r(\vec{y})) \wedge Triggered_{r(\vec{y})}(s,t))$$

analogously to direct effect formulas (3.14) and (3.15). Secondly, we adjust the definition of Reiter defaults (3.28) to let the safety condition check for possibly conflicting indirect effects:

Definition 5.22. Let $\delta =$ `normally` ψ `if` Φ be a state default and s, t : TIME be variables.

$$Safe_\delta(s,t) \stackrel{\text{def}}{=} (\forall a)(Poss(a,s,t) \supset (\neg Dir(\neg\psi, a, s, t) \wedge \neg Ind(\neg\psi, s, t))$$

How to construct an action default theory for an *n-clerD* domain is then immediate. We do not give the definition here but employ an example to illustrate the concept.

Example 5.23 (Ramification and Casualty)**.** To Θ_{Yale} from Example 3.6 we add a fluent Walking indicating whether the turkey is walking and an indirect effect law saying that a turkey caused not to be alive is also caused not to be walking any more.

$$\begin{aligned}\Theta'_{Yale} = \{&\texttt{possible}\ \mathsf{Shoot}\ \texttt{iff}\ \mathsf{Loaded} \wedge \neg\mathsf{Broken},\\ &\texttt{action}\ \mathsf{Shoot}\ \texttt{causes}\ \neg\mathsf{Alive},\\ &\texttt{action}\ \mathsf{Load}\ \texttt{causes}\ \mathsf{Loaded},\\ &\texttt{normally}\ \neg\mathsf{Broken},\\ &\texttt{effect}\ \neg\mathsf{Alive}\ \texttt{causes}\ \neg\mathsf{Walking}\}\end{aligned}$$

We choose situations as underlying time structure and specify the initial situation by $\Sigma_0 = \{Holds(\mathsf{Alive}, S_0)\}$, leading to the domain axiomatisation Σ. Reiter defaults Δ are constructed in view of Definition 5.22 above. The only non-trivial direct effect formulas are

$$DirT(f, \mathsf{Load}, s, t) \equiv f = \mathsf{Loaded} \quad \text{and} \quad DirF(f, \mathsf{Shoot}, s, t) \equiv f = \mathsf{Alive}$$

The one meaningful indirect effect formula is derived from the only indirect effect law:

$$IndF(f, s, t) \equiv (f = \mathsf{Walking} \wedge Holds(\mathsf{Alive}, s) \wedge \neg Holds(\mathsf{Alive}, t))$$

It says that the indirect effect of not-walking occurs exactly when Fred's life is terminated. In addition to the usual conclusions known from Example 3.6, we can now additionally infer that the turkey stops walking after being shot at:

$$(\Sigma, \Delta) \models\!\!\approx \neg Holds(\mathsf{Walking}, Do([\mathsf{Load}, \mathsf{Shoot}], S_0)$$

5.4.2 Related Work on Ramification

Early treatments of ramifications [Ginsberg and Smith, 1987; Winslett, 1988] all assumed the behaviour of the world to be specified by state constraints – static laws that must hold in all states of the world – and relied on the principle of minimal change. [Brewka and Hertzberg, 1993] addressed the shortcomings of these approaches and provided an alternative formalisation which incorporated an explicit notion of causation. [Lin, 1995], [McCain and Turner, 1995] and [Thielscher, 1995] independently made the important observation that mere state constraints are insufficient for a proper treatment of ramifications, and likewise introduced explicit representations of causality to deal with indirect effects. [Thielscher, 1997] then showed how information on causal influence among fluents can be used to create a set of causal relationships from a set of domain constraints and provided a solution to the ramification problem, albeit restricted to a specific formalism. [Van Belleghem et al., 1998] provide a way of modelling indirect effects that is independent of a specific calculus; they however use a three-valued semantics. It is not immediately clear how the approach is to be embedded into general-purpose formalisms and how classical logical reasoning methods can be used in that approach. [Shanahan, 1999] shows how to handle a particular class of ramifications in the Event Calculus, limited however in that it admittedly cannot treat self-justifying cycles. In an approach basically similar to ours, [Pinto, 1999] compiles ramification constraints into effect axioms by preprocessing, yet it does not integrate causality. Causality is present in [McIlraith, 2000] (even if only implicitly in the syntax), but the approach resorts to a minimal-model policy and is not able to deal with cyclic fluent dependencies, since it requires stratification of the ramification constraints. More recently, [Herzig and Varzinczak, 2007] dealt with indirect effects through static laws in a special formalism based on modal logic. [Forth and Miller, 2007] proposed a treatment of indirect effects in the Event Calculus which is based on prioritised circumscription and bound to linear time and a narrative-based semantics.

Most recently, Lin's causal action theories [Lin, 1995] have been extended to the case of cyclic dependencies among fluents [Lin and Soutchanski, 2011]. The approach is based on a version of the Situation Calculus where action domains are characterised by precondition axioms and direct effect statements much like our (deterministic) direct effect laws. Indirect effects are specified by causal rules, formulas $\Phi[t] \supset Caused(\psi, s, t)$ meaning that literal ψ is caused to be true whenever state formula $\Phi[t]$ is true. Lin and Soutchanski give a minimisation-based approach to compile such domain specifications into successor state axioms which works even if there are cyclic fluent dependencies among the causal rules. However, the approach is limited in that its specification of indirect effects does not provide references to the starting time point of the action (at least not without some additional hacking). It would therefore be unable to straightforwardly treat Example 5.1 about the bowl of soup falling off the table, for the soup spills only when it was on the table before unevenly lifting one side. Causal rules $\Phi(t) \supset Caused(\psi, s, t)$ where $\Phi(t)$ is quantifier-free can be viewed as indirect effect rules [Denecker et al., 1998] `initiating` Φ `causes` ψ `if` $\top$, which allows to apply the procedure of

Section 5.2.2 to produce an equivalent set of indirect effect laws (5.2). In accordance with [Lin and Soutchanski, 2011], we conjecture that the approaches then yield the same results for this common fragment but leave the formal proof for future work.

In [Baumann et al., 2010], we provided a treatment of ramifications using Default Logic and the Unifying Action Calculus. Since it is close to the approach presented in this thesis, we review it here in some more detail. There, indirect effect laws $\underline{\texttt{effect}}\ \chi\ \underline{\texttt{causes}}\ \psi\ \underline{\texttt{if}}\ \Phi_1\ \underline{\texttt{before}}\ \Phi_2$ are translated into justification-free Reiter defaults[4]

$$\frac{\Phi_1[s] \wedge \Phi_2[t] \wedge \neg\chi[s] \wedge \chi[t] :}{\mathit{Indirect}(\psi, s, t)} \tag{5.18}$$

This solution is very elegant in that there are no syntactic restrictions on indirect effect laws, the laws can be modularly translated, and the groundedness of default extensions easily guarantees a proper treatment of causality for indirect effects. On the other hand, the translation of [Baumann et al., 2010] produces a default theory, where the translation in this thesis produces a classical first-order theory. Despite being computationally more complex in the propositional case, the default logic approach is not always strong enough for incompletely specified domains:

Example 5.24 (The Walking Dead)**.** Staying in domain Θ_{Yale}, we revisit the indirect effect law $r = \underline{\texttt{effect}}\ \neg\mathsf{Alive}\ \underline{\texttt{causes}}\ \neg\mathsf{Walking}$ from Example 5.23 and turn r into the two laws

$$r_1 = \underline{\texttt{effect}}\ \neg\mathsf{Alive}\ \underline{\texttt{causes}}\ \neg\mathsf{Walking}\ \underline{\texttt{if}}\ \mathsf{Walking}\ \underline{\texttt{before}}\ \top$$
$$r_2 = \underline{\texttt{effect}}\ \neg\mathsf{Alive}\ \underline{\texttt{causes}}\ \neg\mathsf{Walking}\ \underline{\texttt{if}}\ \neg\mathsf{Walking}\ \underline{\texttt{before}}\ \top$$

that say that getting killed causes the turkey not to be walking any more if it was walking before (r_1), and that the same happens if it was not walking (r_2). In the approach of this chapter, for all pairs of states S, T with $S \models \mathsf{Alive}$ and $T \models \neg\mathsf{Alive}$, we have that r is applicable and exactly one of r_1 or r_2 is applicable. Thus, even if there is uncertainty about Walking on theory level, the indirect effect $\neg\mathsf{Walking}$ occurs. Translating these laws into the Reiter defaults

$$\delta_r = \frac{\mathit{Holds}(\mathsf{Alive}, s) \wedge \neg\mathit{Holds}(\mathsf{Alive}, t) :}{\mathit{IndirectF}(\mathsf{Walking}, s, t)}$$
$$\delta_1 = \frac{\mathit{Holds}(\mathsf{Alive}, s) \wedge \neg\mathit{Holds}(\mathsf{Alive}, t) \wedge \mathit{Holds}(\mathsf{Walking}, s) :}{\mathit{IndirectF}(\mathsf{Walking}, s, t)}$$
$$\delta_2 = \frac{\mathit{Holds}(\mathsf{Alive}, s) \wedge \neg\mathit{Holds}(\mathsf{Alive}, t) \wedge \neg\mathit{Holds}(\mathsf{Walking}, s) :}{\mathit{IndirectF}(\mathsf{Walking}, s, t)}$$

reveals the difference between the approaches. While loop formulas treat $\{r\}$ and $\{r_1, r_2\}$ equivalently, the ramification defaults behave in a different way when the truth of Walking at s is unknown: δ_r always yields the indirect effect $\neg\mathit{Holds}(\mathsf{Walking}, t)$ whenever Alive changes to true from s to t; the defaults δ_1 and δ_2 do not yield the indirect effect since they depend on the underlying theory additionally entailing either $\mathit{Holds}(\mathsf{Walking}, s)$ or $\neg\mathit{Holds}(\mathsf{Walking}, s)$.

In [Baumann et al., 2010], we also showed that the default logic-based approach to the ramification problem can be straightforwardly combined with *n-ule*$\mathcal{D}$ action default theories. Alas, since normal and justification-free defaults are mixed, extension existence cannot be guaranteed for that approach. Our combined approach of Section 5.4.1, on the other hand, utilises only normal defaults, the default theories defined there thus always have an extension.

[4] Actually, [Baumann et al., 2010] have no distinction between initial and final context. It is however trivial to add this.

Chapter 6

Conclusion

We have presented a comprehensive framework for default reasoning about actions. Starting with very simple domains, we have increased the range of domains we can model to those including non-deterministic effects. Putting the approach into practice turned out to be possible; we presented an implementation of action default theories and proved its correctness for a certain class of domains. We finally presented a novel solution to the ramification problem that can deal with cyclic fluent dependencies. It is also easily integrated into the default reasoning framework seen earlier.

* * *

In the remainder, we discuss related work on combinations of action theories with defaults about states. We will see that all existing alternative approaches tie themselves to a particular time structure. Most of them use non-standard logics or propositional languages with non-standard semantics. In contrast, the approach that we have developed is independent of a specific notion of time and uses the standard semantics of first-order logic and default logic. This also implies groundedness of the conclusions that can be drawn from action default theories, a property that the majority of alternative approaches does not possess.

Finally, we explore some areas that offer potential for future work.

6.1 Related Work on Default Reasoning about Actions

There are four further major approaches for default reasoning about actions. We will address each of them separately in some more detail.

6.1.1 Defaults in the Discrete Event Calculus

[Mueller, 2006] offers simple default reasoning about time in the Event Calculus. In general, the Event Calculus already makes the default assumptions that unexpected events do not occur, and that events have no unexpected effects. In Mueller's axiomatisation, these assumptions are implemented via circumscription of *Happens* and *Initiates*/*Terminates*. He uses a similar technique for general temporal default reasoning as follows. The user fixes a set of abnormality predicates like $Ab_1(x,t)$ that expresses whether an object x is abnormal in a certain way at time t. These abnormality predicates are freely used in the axiomatisation, as in $(Holds(\mathsf{Apple}(x),t) \wedge \neg Ab_1(x,t)) \supset Holds(\mathsf{Red}(x),t)$ saying that apples are red unless they are abnormal. So-called cancellation axioms like $Holds(\mathsf{GrannySmith}(x),t) \supset Ab_1(x,t)$

and $\mathit{Holds}(\mathsf{Rotten}(x), t) \supset \mathit{Ab}_1(x, t)$ are employed by the user to state sufficient reasons for abnormality. The cancellation axioms are then circumscribed in the abnormality predicates, effecting the assumption that for any given abnormal object, one of the sufficient conditions must necessarily hold. For the above, this circumscription would produce $\mathit{Ab}_1(x, t) \equiv (\mathit{Holds}(\mathsf{GrannySmith}(x), t) \vee \mathit{Holds}(\mathsf{Rotten}(x), t))$.

However, circumscription is not suited to do the kind of default reasoning we are interested in. For one, consider the two statements (N) "if Φ, then normally ψ" and (S) "normally, if Φ then ψ" [Kraus et al., 1990]. Although they look alike, the distinction lies on the scope of normality. In (N), truth of Φ must be asserted before invoking normality; in (S), normality is invoked right at the start to assume that Φ implies ψ (and its contrapositive). In default logic, these statements are formalised by a normal (N) default $\Phi : \psi / \psi$ and a supernormal (S) default $\top : \Phi \supset \psi / \Phi \supset \psi$. These defaults employ the correct scope of normality. Using circumscription, (N) would be formalised as $(\Phi \wedge \neg \mathit{Ab}) \supset \psi$, while (S) is written as $\neg \mathit{Ab} \supset (\Phi \supset \psi)$. It is immediately clear that the two are logically equivalent. Circumscription's predicate Ab reifies abnormality and may talk about objects, but allows no distinction between the different normality scopes of (N) and (S).

Secondly, circumscription allows unjustified conclusions. To see this, look at the two state defaults `normally` Wet `if` Rain and `normally` Rain `if` Wet saying that rain and wet grass usually go hand in hand. With the axiomatisation technique of [Mueller, 2006], they become $(\mathit{Holds}(\mathsf{Rain}, t) \wedge \neg \mathit{Ab}_1(t)) \supset \mathit{Holds}(\mathsf{Wet}, t)$ for the first state default and $(\mathit{Holds}(\mathsf{Wet}, t) \wedge \neg \mathit{Ab}_2(t)) \supset \mathit{Holds}(\mathsf{Rain}, t)$ for the second one. At a perfectly normal time point τ with $\neg \mathit{Ab}_1(\tau) \wedge \neg \mathit{Ab}_2(\tau)$, the two defaults are equivalent to $\mathit{Holds}(\mathsf{Rain}, \tau) \equiv \mathit{Holds}(\mathsf{Wet}, \tau)$. Consequently, the axiomatisation allows models where both rain and wet grass materialise at τ out of thin air, one "causing" the other. Again, default logic is much better suited to model these state defaults due to the groundedness of extensions.

6.1.2 Action Language $\mathcal{C}+$

The action language $\mathcal{C}+$ [Giunchiglia et al., 2004] offers a form of default reasoning in propositional domains. However, as we shall see, the language uses the same mechanism for defaults and persistence and makes some counterintuitive predictions concerning defaults. We first recapitulate the most important definitions of $\mathcal{C}+$, alas without its distinction between simple and statically determined fluents since this is of no importance to our point.

$\mathcal{C}+$ distinguishes between different kinds of formulas: (1) fluent formulas F, that speak about time points like our fluent formulas, (2) action formulas A, that speak about occurrence of actions, (3) formulas H, that speak about both. To specify domains, $\mathcal{C}+$ provides three kinds of laws: (1) static laws **caused** F_1 **if** F_2, (2) action dynamic laws **caused** A **if** H, (3) fluent dynamic laws **caused** F_1 **if** F_2 **after** H. The latter are used to express direct action effects via the macro A **causes** $F \stackrel{\text{def}}{=}$ **caused** F **if** $\top$ **after** A. The language $\mathcal{C}+$ does not have a built-in frame assumption. The tendency of a fluent literal to persist has to be specified by a special macro **inertial** $F \stackrel{\text{def}}{=}$ **caused** F **if** F **after** F. Static default values of fluents are also specified via a macro: **default** F_1 **if** $F_2 \stackrel{\text{def}}{=}$ **caused** F_1 **if** $F_1 \wedge F_2$.

The semantics of $\mathcal{C}+$ is defined via a translation to nonmonotonic causal logic [Giunchiglia et al., 2004]. This logic uses causal rules $F \Leftarrow G$ for propositional formulas F, G to express that "there is a cause for F to be true whenever G is true." Roughly, an interpretation is then a model for a set of implications if all that is true in the interpretation is also "caused" by some implication in the theory.

When turning $\mathcal{C}+$ action descriptions into nonmonotonic causal theories, a finite sequence $0, \ldots, n$ of time points is incorporated into the signature by replacing each atom P by "time

stamped" versions $i{:}P$ for $0 \leq i \leq n$. This generalises to formulas H via structural induction. A static law or action dynamic law **caused** H_1 **if** H_2 then becomes the nonmonotonic causal logic formulas $i{:}H_1 \Leftarrow i{:}H_2$ for $0 \leq i \leq n$, a fluent dynamic law **caused** F_1 **if** F_2 **after** H becomes $i+1{:}F_1 \Leftarrow i+1{:}F_2 \wedge i{:}H$ for $0 \leq i \leq n-1$.

Example 6.1 (Learning to Fly)**.** Imagine a simple domain where an inertial fluent Flies is true by default if another inertial fluent Bird is true, and there is an action Wait without effects. The $\mathcal{C}+$ expressions for this domain and their macro expansions are

$$\begin{aligned}
\textbf{inertial}\ \mathsf{Flies} &= \textbf{caused}\ \mathsf{Flies}\ \textbf{if}\ \mathsf{Flies}\ \textbf{after}\ \mathsf{Flies}\\
\textbf{inertial}\ \neg\mathsf{Flies} &= \textbf{caused}\ \neg\mathsf{Flies}\ \textbf{if}\ \neg\mathsf{Flies}\ \textbf{after}\ \neg\mathsf{Flies}\\
\textbf{inertial}\ \mathsf{Bird} &= \textbf{caused}\ \mathsf{Bird}\ \textbf{if}\ \mathsf{Bird}\ \textbf{after}\ \mathsf{Bird}\\
\textbf{inertial}\ \neg\mathsf{Bird} &= \textbf{caused}\ \neg\mathsf{Bird}\ \textbf{if}\ \neg\mathsf{Bird}\ \textbf{after}\ \neg\mathsf{Bird}\\
\textbf{default}\ \mathsf{Flies}\ \textbf{if}\ \mathsf{Bird} &= \textbf{caused}\ \mathsf{Flies}\ \textbf{if}\ \mathsf{Flies} \wedge \mathsf{Bird}
\end{aligned}$$

Assuming just two time points 0 and 1, this is translated into the nonmonotonic causal theory

$$\begin{aligned}
T_{\mathsf{Flies}} = \{&1{:}\mathsf{Flies} \Leftarrow 0{:}\mathsf{Flies} \wedge 1{:}\mathsf{Flies},\\
&\neg 1{:}\mathsf{Flies} \Leftarrow \neg 0{:}\mathsf{Flies} \wedge \neg 1{:}\mathsf{Flies},\\
&1{:}\mathsf{Bird} \Leftarrow 0{:}\mathsf{Bird} \wedge 1{:}\mathsf{Bird},\\
&\neg 1{:}\mathsf{Bird} \Leftarrow \neg 0{:}\mathsf{Bird} \wedge \neg 1{:}\mathsf{Bird},\\
&0{:}\mathsf{Flies} \Leftarrow 0{:}\mathsf{Flies} \wedge 0{:}\mathsf{Bird},\\
&1{:}\mathsf{Flies} \Leftarrow 1{:}\mathsf{Flies} \wedge 1{:}\mathsf{Bird}\}
\end{aligned}$$

Now consider an abnormal initial time point 0 where Bird is true and Flies is false:

$$T_0 = \{0{:}\mathsf{Bird} \Leftarrow \top, \neg 0{:}\mathsf{Flies} \Leftarrow \top\}$$

What happens after we apply an action without effects? Intuition suggests that no effects change nothing, hence ¬Flies should hold at time point 1 after a dummy action like Wait.

Let us inspect the models of the nonmonotonic causal theory $T_{\mathsf{Flies}} \cup T_0 \cup \{0{:}\mathsf{Wait} \Leftarrow \top\}$. Its first model is characterised by the literals

$$0{:}\mathsf{Wait}, 0{:}\mathsf{Bird}, 1{:}\mathsf{Bird}, \neg 0{:}\mathsf{Flies}, \neg 1{:}\mathsf{Flies}$$

and corresponds to our intuitive interpretation of the domain. The last literal can be justified by the second rule in T_{Flies} saying that persisting false is a cause for falsity.

However, $\mathcal{C}+$ clashes with intuition and admits an additional model where the abnormal bird magically learns to fly (by default) during Wait:

$$0{:}\mathsf{Wait}, 0{:}\mathsf{Bird}, 1{:}\mathsf{Bird}, \neg 0{:}\mathsf{Flies}, 1{:}\mathsf{Flies}$$

Here, the last literal can be justified by the last rule of T_{Flies} invoking truth by default as a cause for Flies' truth.

The unintended model occurs because $\mathcal{C}+$ fails to recognise that the default is violated and the initial time point 0 is thus *abnormal*. It subsequently ignores this default violation and applies the default at the next time point.

But even if we restrict default reasoning to a single time point, the intuitions behind the $\mathcal{C}+$ expression $d =$ **default** Flies **if** Bird and the *a*-$\mathcal{D}$ state default normally Flies if Bird differ: while the latter translates to a Reiter default like

$$\delta^{\mathcal{D}} = \frac{0{:}\mathsf{Bird} : 0{:}\mathsf{Flies}}{0{:}\mathsf{Flies}}$$

the former corresponds [Giunchiglia et al., 2004, Section 7.1] to the Reiter default

$$\delta_d^{\mathcal{C}+} = \frac{: 0{:}\mathsf{Flies} \wedge 0{:}\mathsf{Bird}}{0{:}\mathsf{Flies}}$$

which may even conclude that something flies without any evidence that it is a bird: Translating expression d into a nonmonotonic causal theory for the time point 0 yields $T_d = \{0{:}\mathsf{Flies} \Leftarrow 0{:}\mathsf{Flies} \wedge 0{:}\mathsf{Bird}\}$. This theory has no model, thus according to [Giunchiglia et al., 2004, Proposition 10] the default theory $(\emptyset, \{\delta_d^{\mathcal{C}+}\})$ has no complete extension.[1] However, the default theory *does* have an extension, the set $Th(\{0{:}\mathsf{Flies}\})$. Here $0 : \mathsf{Flies}$ has been concluded out of the blue.

Finally, $\mathcal{C}+$ is restricted to a propositional language with a built-in linear time structure. It allows for cyclic conclusions in the same way that circumscription-based default reasoning does: The statements **default** Rain **if** Wet and **default** Wet **if** Rain expand to the formulas $0{:}\mathsf{Rain} \Leftarrow 0{:}\mathsf{Rain} \wedge 0{:}\mathsf{Wet}$ and $0{:}\mathsf{Wet} \Leftarrow 0{:}\mathsf{Wet} \wedge 0{:}\mathsf{Rain}$. Obviously, $\{0{:}\mathsf{Rain}, 0{:}\mathsf{Wet}\}$ is a model for the two formulas where rain and wet grass appear without being caused.

6.1.3 Modal Situation Calculus with Only-knowing

[Lakemeyer and Levesque, 2009] combined action theories with defaults in a modal language called $\mathcal{ES}_{\mathcal{O}}$. For reasoning about actions, $\mathcal{ES}_{\mathcal{O}}$ reformulates a fragment of the Situation Calculus in a second-order modal dialect with substitutional quantification. The fragment captures Reiter's basic action theories [Reiter, 2001] with regression-based reasoning and reasoning about knowledge. Although situations provide the underlying time structure, they are not present in their classical version as terms of the language. Instead, $\mathcal{ES}_{\mathcal{O}}$ uses time modalities – $\Box$ for any situation reachable from the current one, and $[a]$ for the situation resulting from executing action a in the current situation. For nonmonotonic reasoning, their approach employs the logic of only-knowing [Levesque, 1990] which is integrated with these situation-based action theories into a single semantical framework.

$\mathcal{ES}_{\mathcal{O}}$ allows to specify the effects of actions via Reiter-style successor state axioms, as in

$$(\forall a, x)\Box([a]\mathsf{Broken}(x) \equiv ((a = \mathsf{Drop}(x) \wedge \mathsf{Fragile}(x)) \vee (\mathsf{Broken}(x) \wedge a \neq \mathsf{Repair}(x))))$$

According to this formula, in any situation, an object x is broken after action a if a was the action of dropping it and x is fragile, or if x was already broken before and a was not the action of repairing it.

To express what is known, only-known, and additionally known and only-known by default, $\mathcal{ES}_{\mathcal{O}}$ offers the respective modalities $\mathbf{K}, \mathbf{O}, \mathbf{B}, \mathbf{\Omega}$. For example, the default that objects made of glass are fragile unless known otherwise is written as

$$(\forall x)((\mathbf{B}\mathsf{Glass}(x) \wedge \neg\mathbf{B}\neg\mathsf{Fragile}(x)) \supset \mathsf{Fragile}(x)) \tag{6.1}$$

An $\mathcal{ES}_{\mathcal{O}}$ basic action theory Σ and a conjunction Δ of Moore defaults as the above are now combined into a formula $\mathbf{O}\Sigma \wedge \Box\mathbf{\Omega}\Delta$ saying that Σ is all the agent knows and it applies

[1] A complete extension is such that for each formula it entails the formula or its negation.

defaults from Δ at all situations. Using such a basic action theory with defaults, an agent can reason about the state of the world. After action execution, it progresses its knowledge base about the current time point in the sense of [Lin and Reiter, 1997]. During this progression, it is vital that the agent can distinguish between default assumptions and hard facts. (For otherwise the agent might elevate an assumption to a hard fact, and subsequently sensing a default violation would lead to an inconsistency.) The progression semantics of [Lakemeyer and Levesque, 2009] solves this issue by forgetting the past and reapplying the defaults after action execution. This however causes them to effectively discard all default assumptions and conclusions, the connections between which make up the essence of the expansions.

Example 6.2 (Expansion Hopping)**.** We focus our attention on whether a given vase is fragile and the belief dynamics of fragility. To the default about glass objects being fragile above, we add the prerequisite-free default that objects in general are to be considered not fragile:

$$(\forall x)(\neg \mathbf{B}\mathsf{Fragile}(x) \supset \neg \mathsf{Fragile}(x)) \tag{6.2}$$

Denote the conjunction of defaults (6.1) and (6.2) by Δ. In an initial situation $\Sigma_0 = \mathsf{Glass}(\mathsf{Vase})$ where all the agent knows is that the vase is made of glass, both defaults are applicable and the autoepistemic theory given by $\Sigma_0 \wedge \Delta$ has two stable expansions. In $\mathcal{ES}_{\mathcal{O}}$ this means that both $\mathbf{O}\Sigma_0 \wedge \mathbf{\Omega}\Delta \wedge \neg \mathsf{Fragile}(\mathsf{Vase})$ and $\mathbf{O}\Sigma_0 \wedge \mathbf{\Omega}\Delta \wedge \mathsf{Fragile}(\mathsf{Vase})$ are satisfiable.

Now we add an (irrelevant) action Move that inverts a fluent InKitchen. (We assume there are only two rooms and for simplicity leave out precondition axiom and sensing result of the action.) The status of this new fluent is determined by the successor state axiom

$$\Sigma_{\mathsf{Move}} = (\forall a, x)\Box([a]\mathsf{InKitchen} \equiv ((a = \mathsf{Move} \wedge \neg \mathsf{InKitchen}) \vee (\mathsf{InKitchen} \wedge a \neq \mathsf{Move})))$$

Contrary to intuition, $\mathcal{ES}_{\mathcal{O}}$ considers it possible that changing rooms has an effect on the vase being fragile or not, which is witnessed by the satisfiability of

$$\mathbf{O}(\Sigma_0 \wedge \Sigma_{\mathsf{Move}}) \wedge \Box\mathbf{\Omega}\Delta \wedge \mathsf{Fragile}(\mathsf{Vase}) \wedge [\mathsf{Move}]\neg \mathsf{Fragile}(\mathsf{Vase})$$

Roughly, the status of $\mathsf{Fragile}(\mathsf{Vase})$ is forgotten during Move as explained above. This makes both defaults applicable in the progressed situation (the known fact $\mathsf{Glass}(\mathsf{Vase})$ persists) and again leads to two possible beliefs about fragility regardless of beliefs at earlier time points.

In our approach, defaults are applied to multiple time points simultaneously, moreover our default conclusions are subject to persistence. Combined with violation-checking, this leads to extensions where such spontaneous, uncaused changes do not occur. The straightforward $\mathcal{D}$ axiomatisation of the domain above would allow the sceptical conclusion

$$\mathit{Holds}(\mathsf{Fragile}(x), s) \equiv \mathit{Holds}(\mathsf{Fragile}(x), \mathit{Do}(\mathsf{Move}, s))$$

A further and much more grave issue with default reasoning based on autoepistemic logic [Moore, 1985] is that it allows unfounded beliefs due to the cyclic definition of stable expansions. Our action default theories are based on default logic, where such problems are not to be found. Finally, $\mathcal{ES}_{\mathcal{O}}$ is a non-standard logic (second-order modal logic with substitutional quantification) with tailor-made semantics. In contrast, our approach relies on the standard semantics of sorted first-order logic and default logic.

6.1.4 Argumentation-based Approaches to Knowledge Qualification

[Kakas et al., 2008] presented a framework that aimed at combining different forms of non-monotonic reasoning in AI: default reasoning in static domains and defeasible persistence in

temporal domains. They motivated their approach by a series of examples and then introduced a tailor-made semantics whose action-part is based on the action language $\mathcal{ME}$ [Kakas et al., 2005]. Default reasoning is treated as a black box, which makes it difficult to pin down specific properties of the framework with regard to interaction of defaults, action effects and persistence.

Their preliminary work was then extended by [Michael and Kakas, 2009], which integrates temporal and default reasoning via argumentation frameworks [Dung, 1995]. They define attacks and admissibility "in a manner that closely follows corresponding definitions in the literature." The formalism works for propositional domains and provides a progression-like semantics for a single state transition. To specify semantics for multiple state transitions, the authors require a minimal invocation of exogenous causes at each time step. Still, they do not fix a specific semantics for their state defaults:

> [...] we take a black-box approach to the syntax and semantics of default static theories, and assume simply that we have access to their models, without concerning ourselves with how these models are derived.

In particular, they assume the existence of a revision function $\texttt{rev}(\cdot,\cdot)$ that revises a theory T with a set L of literals such that the revised theory $\texttt{rev}(T, L)$ entails all literals in L. It is clear that such revision functions do exist, for example by setting $\texttt{rev}(T, L) \stackrel{\text{def}}{=} L$ to satisfy the entailment condition. However, one usually expects additional useful properties from a nonmonotonic semantics, such as no unnecessary removal of information. While intended to preserve flexibility, ignoring specificities of the underlying nonmonotonic logic only postpones semantical issues which are likely to occur. As we have seen in the previous sections about $\mathcal{C}+$ and $\mathcal{ES}_{\mathcal{O}}$, many subtleties arising in default reasoning about actions hinge on semantical and technical details of the combined languages. Not fixing one of these languages also means not addressing these problems.

The same line of research was further extended by [Michael and Kakas, 2011]. A linear time structure was now built into the syntax of the underlying language. As before, each piece of knowledge is encoded as an argument in favour of some conclusion. As a novelty, they made the underlying preferences between these arguments explicit. Where [Michael and Kakas, 2009] had fixed preferences built into the semantics, [Michael and Kakas, 2011] now modelled these preferences as arguments which could in turn be defeated by other arguments. While this is in general very flexible in terms of expressivity, the task of specifying these preferences and therefore specifying the semantics of the whole domain is deferred to the user. In general, the framework provides not much guidance in domain specification: there is no clear language for specification, the authors only sketch a pseudo-syntax on an example. Specific axiom schemas are only provided for frame axioms and the no-action axiom. Most importantly, there is no well-studied default semantics behind the framework. Indeed, [Michael and Kakas, 2011] make no ontological distinction between defaults and laws, for example the argument "$\{Bird(x) \supset CanFly(x)\}$ *at* T" stands for a state default while its syntactical variant "$\{CanFly(x) \supset Alive(x)\}$ *at* T" represents a state constraint. To achieve their respective desired semantics, the user has to specify preferences between these and other arguments. Finally, the approach is restricted to propositional logic and linear time.

6.1.5 Further Approaches

[Thielscher, 2001] used an extension of the Fluent Calculus with supernormal defaults to solve the qualification problem. The goal there is however not to make default assumptions about general fluents, but rather special predicates that refer to action (dis-)qualification.

[Denecker and Ternovska, 2007] enriched the Situation Calculus [Reiter, 2001] with inductive definitions. While they provide a nonmonotonic extension of an action calculus, the intended usage is to solve the ramification problem rather than to do the kind of defeasible reasoning we treat in this work.

The action language $\mathcal{K}$ [Eiter et al., 2004] provides simple default statements, but allows the same counterexample as $\mathcal{C}+$.

6.2 Directions for Future Work

Although there are multiple starting points to continue this work, we specifically want to mention two fields. The first one concerns a well-known problem in reasoning about actions, for which our framework will be able to give a novel solution. The second area we want to point out will allow for more flexible and expressive ways to specify how the world normally behaves.

6.2.1 State Defaults and the Qualification Problem

A robot that operates in a complex environment can never predict with absolute certainty the actions it may successfully execute. Any exceptional circumstance, however unlikely, might render an action impossible. This fundamental problem of cognitive robotics has been termed the *qualification problem* [McCarthy, 1977]. In order to solve this problem, an agent must be able to (1) assume away by default abnormal action qualifications and (2) deal with unexpected action failures. In the literature there is sometimes a distinction between *strong* qualification – an action being inapplicable contrary to prediction – and *weak* qualification – an action failing to produce one or more of its predicted effects [Gelfond et al., 1991; Thielscher, 1996; Thielscher, 2001]. Elsewhere [Kakas et al., 2011], the authors distinguish *endogenous* qualifications – that can be explained within the theory – from *exogenous* qualifications, whose explanation lies outside of the scope of the theory.[2] We illustrate the general problem with a modified version of an example from [Thielscher, 1996].

Example 6.3 (Tail Pipe Potato)**.** This domain is about starting a car whose tail pipe may or may not have been clogged with a potato. Putting the potato in the tail pipe is possible if the potato is not too heavy. The tail pipe is clogged after putting the potato in it. Starting the car is possible if we have the ignition key and the battery is full. The car runs after starting it if the tail pipe was not clogged and the engine was not broken. In *cle*$\mathcal{D}$ this is written as

$$\begin{aligned}\Theta_{Car} = \{&\texttt{possible}\ \mathsf{PutPotato}\ \texttt{iff}\ \neg\mathsf{Heavy},\\ &\texttt{action}\ \mathsf{PutPotato}\ \texttt{causes}\ \mathsf{Clogged},\\ &\texttt{possible}\ \mathsf{Start}\ \texttt{iff}\ \mathsf{Key}\wedge\mathsf{Battery},\\ &\texttt{action}\ \mathsf{Start}\ \texttt{causes}\ \mathsf{Runs}\ \texttt{if}\ \neg\mathsf{Clogged}\wedge\neg\mathsf{Broken}\}\end{aligned}$$

Our formalism for default reasoning about actions provides all the necessary ingredients for a solution to the qualification problem: for endogenous qualifications, we can specify state defaults that assume absence of abnormal action disqualifications and effect laws about known causes for disqualifications. For exogenous qualification, we introduce new fluents expressing miraculous disqualification of actions and action effects. Strong qualification can be modelled by $\mathsf{Disqualified} : \textsc{action} \to \textsc{fluent}$ saying that an action is disqualified for some

[2] [Thielscher, 1996] calls exogenous qualifications "miraculous."

reason. We then assume $\neg\mathsf{Disqualified}(A(\vec{x}))$ by default for each action A. Weak qualification can be modelled by introducing a function symbol $\mathsf{Ab} : \textsc{fluent} \to \textsc{fluent}$ saying that a specific effect is disqualified for some reason. We then add $\neg\mathsf{Ab}(\varphi)$ to the prerequisite of action effect φ and assume $\neg\mathsf{Ab}(\varphi)$ by default.

Example 6.3 (Continued)**.** Incorporating a solution to the qualification problem with a distinction between endogenous/exogenous weak and strong qualifications, we add the fluents Disqualified and Ab as sketched above. We also say that normally, the potato is not heavy, the battery is not empty, the tail pipe is not clogged and the engine is not broken.

$$\begin{aligned}\Theta'_{Car} = \{&\underline{\texttt{possible}}\ \mathsf{PutPotato}\ \underline{\texttt{iff}}\ \neg\mathsf{Heavy} \wedge \neg\mathsf{Disqualified}(\mathsf{PutPotato}),\\ &\underline{\texttt{action}}\ \mathsf{PutPotato}\ \underline{\texttt{causes}}\ \mathsf{Clogged}\ \underline{\texttt{if}}\ \neg\mathsf{Ab}(\mathsf{Clogged}),\\ &\underline{\texttt{possible}}\ \mathsf{Start}\ \underline{\texttt{iff}}\ \mathsf{Key} \wedge \mathsf{Battery} \wedge \neg\mathsf{Disqualified}(\mathsf{Start}),\\ &\underline{\texttt{action}}\ \mathsf{Start}\ \underline{\texttt{causes}}\ \mathsf{Runs}\ \underline{\texttt{if}}\ \neg\mathsf{Clogged} \wedge \neg\mathsf{Broken} \wedge \neg\mathsf{Ab}(\mathsf{Runs}),\\ &\underline{\texttt{normally}}\ \neg\mathsf{Heavy},\\ &\underline{\texttt{normally}}\ \mathsf{Battery},\\ &\underline{\texttt{normally}}\ \neg\mathsf{Clogged},\\ &\underline{\texttt{normally}}\ \neg\mathsf{Broken},\\ &\underline{\texttt{normally}}\ \neg\mathsf{Disqualified}(\mathsf{PutPotato}),\\ &\underline{\texttt{normally}}\ \neg\mathsf{Disqualified}(\mathsf{Start}),\\ &\underline{\texttt{normally}}\ \neg\mathsf{Ab}(\mathsf{Clogged}),\\ &\underline{\texttt{normally}}\ \neg\mathsf{Ab}(\mathsf{Runs})\}\end{aligned}$$

This way of modelling action qualifications satisfies [Kakas et al., 2011]'s requirement that

> [...] if a fluent has a default truth value, this should manifest itself everywhere in the reasoning process, and not just when the fluent appears as a precondition in an effect or executability law.

Furthermore, our treatment of linear time domains easily enables the important distinction between *attempted* and *accomplished* action executions. [Kakas et al., 2011]:

> A full solution to the exogenous qualification problem necessitates a view of occurrence statements as identifying only *attempts* to execute actions (even though we may choose to build into a formalism a default principle that such attempts can be assumed to result in actual action executions unless there is evidence to the contrary).

With our linear time action precondition axioms

$$\mathit{Poss}(A(\vec{x}), s, t) \equiv (\Phi_A[s] \wedge \mathit{Holds}(\mathsf{Happens}(A(\vec{x}), s, t), s) \wedge s < t)$$

the fluent $\mathsf{Happens}(A(\vec{x}), s, t)$ only refers to an attempted execution, while $\Phi_A[s]$ must be additionally inferred to ensure accomplishment. In particular, if some definite effect has been observed not to have materialised at t although there was an attempted action execution, the effect and precondition axioms allow to derive a violation of the action's precondition Φ_A at s.

Previous solutions to the qualification problem all had to extend existing formalisms [McCarthy, 1986; Lin and Reiter, 1994; Thielscher, 1996; Kvarnström and Doherty, 2000; Thielscher, 2001]. Our action default theories are strong enough to deal with the qualification problem right away. It will be the topic of future research to work out the details of the approach to qualification sketched in this section and investigate its relations to existing solutions.

6.2.2 Preferred Default Logic

The informed reader will have noticed that the approach to qualification of the previous section treats miraculous disqualifications just like any other state default. Intuitively, however, we consider a "miraculous," unexplainable action failure much less likely than a "regular" abnormal situation. To model this qualitative distinction between state defaults, we will need preferred default logic [Brewka, 1994; Brewka and Eiter, 1999; Delgrande and Schaub, 2000].

But prioritised defaults are not only needed for action qualification, they are quite useful in general. Consider the defaults $\delta_{Bird} = Bird(x) : Flies(x)/Flies(x)$ that birds usually fly and $\delta_{Penguin} = Penguin(x) : \neg Flies(x)/\neg Flies(x)$ that penguins usually do not fly. Given that penguins are birds, the second default is more specific and should be preferred over the first one in cases where both are applicable. In preferred default logics, this priority is expressed by a relationship $\delta_{Penguin} \prec \delta_{Bird}$. The exact meaning of such a preference is then determined by the underlying semantics. *Descriptive* preferences [Brewka and Eiter, 1999] disallow extensions where both defaults are applicable, but only the less preferred one has been applied. *Prescriptive* preferences [Delgrande and Schaub, 2000] constrain the order in which the defaults may be applied, where a default cannot be applied unless all preferred defaults have been either applied or found to be inapplicable. It is an important future research topic to identify a notion of preferences that is suitable for default reasoning about actions, and to incorporate these preferences into action default theories.

6.2.3 Further Future Work

Some reasoning about actions formalisms make a distinction between *primitive* and *defined* fluents (e.g. $\mathcal{C}+$ offers this). The idea is that primitive fluents would in principle suffice to describe the domain, but the additional defined fluents are allowed to ease specification. Roughly, the truth values of defined fluents at any time point are completely determined by the truth values of the defining (primitive) fluents (hence the name). This distinction is also connected to the important problem of the modularity of action theories. To date, modularity is formally defined only for classical, monotonic UAC domain axiomatisations [Thielscher, 2011]. A generalisation of this notion for nonmonotonic action formalisms would be very valuable, since modularity lays the foundation of an expressive, yet efficient implementation. As mentioned in Section 4.3, our current implementation draculasp has much potential for additional research. Apart from examining possible gains from well-defined notions of modularity of action default theories, we want to investigate draculasp's usage in combination with agent logic programs [Drescher et al., 2009] and novel enhancements thereof [Brewka et al., 2012].

Bibliography

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[Baker, 1991] Baker, A. B. (1991). Nonmonotonic Reasoning in the Framework of Situation Calculus. *Artificial Intelligence*, 49:5–23.

[Baumann et al., 2010] Baumann, R., Brewka, G., Strass, H., Thielscher, M., and Zaslawski, V. (2010). State Defaults and Ramifications in the Unifying Action Calculus. In *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 435–444, Toronto, Canada.

[Beel and Gipp, 2010] Beel, J. and Gipp, B. (2010). On the Robustness of Google Scholar Against Spam. In *Proceedings of the 21st ACM conference on Hypertext and Hypermedia*, HT '10, pages 297–298, New York, NY, USA. ACM.

[Bidoit and Froidevaux, 1987] Bidoit, N. and Froidevaux, C. (1987). Minimalism Subsumes Default Logic and Circumscription in Stratified Logic Programming. In *Symposium on Logic in Computer Science: Proceedings*. IEEE Press, New York, NY, USA.

[Bidoit and Froidevaux, 1991] Bidoit, N. and Froidevaux, C. (1991). General logical databases and programs: Default logic semantics and stratification. *Information and Computation*, 91(1):15–54.

[Brewka, 1994] Brewka, G. (1994). Adding Priorities and Specificity to Default Logic. In *Proceedings of the European Workshop on Logics in Artificial Intelligence (JELIA-94)*, pages 247–260. Springer.

[Brewka and Eiter, 1999] Brewka, G. and Eiter, T. (1999). Prioritizing Default Logic: Abridged Report. In *Festschrift on the occasion of Prof. Dr. W. Bibel's 60th birthday*. Kluwer.

[Brewka and Hertzberg, 1993] Brewka, G. and Hertzberg, J. (1993). How to Do Things with Worlds: On Formalizing Actions and Plans. *Journal of Logic and Computation*, 3(5):517–532.

[Brewka et al., 2012] Brewka, G., Strass, H., and Thielscher, M. (2012). Declarative Strategies for Agents with Incomplete Knowledge. In Rosati, R. and Woltran, S., editors, *Proceedings of the Fourteenth International Workshop on Non-Monotonic Reasoning (NMR)*.

[Casolary and Lee, 2011] Casolary, M. and Lee, J. (2011). Representing the Language of the Causal Calculator in Answer Set Programming. In *Proceedings of the Twenty-Seventh International Conference on Logic Programming (ICLP-11)*.

[Chen et al., 2006] Chen, Y., Lin, F., Wang, Y., and Zhang, M. (2006). First-Order Loop Formulas for Normal Logic Programs. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 298–307. AAAI Press.

[Chen et al., 2010] Chen, Y., Wan, H., Zhang, Y., and Zhou, Y. (2010). dl2asp: Implementing Default Logic via Answer Set Programming. In Janhunen, T. and Niemelä, I., editors, *Proceedings of the 12th European Conference on Logics in Artificial Intelligence (JELIA 2010)*, volume 6341 of *Lecture Notes in Computer Science*, pages 104–116. Springer.

[Clark, 1978] Clark, K. L. (1978). Negation as Failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum Press.

[Darwiche and Marquis, 2002] Darwiche, A. and Marquis, P. (2002). A Knowledge Compilation Map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264.

[Delgrande and Schaub, 2000] Delgrande, J. P. and Schaub, T. (2000). Expressing Preferences in Default Logic. *Artificial Intelligence*, 123(1–2):41–87.

[Denecker et al., 2003] Denecker, M., Marek, V. W., and Truszczynski, M. (2003). Uniform Semantic Treatment of Default and Autoepistemic Logics. *Artificial Intelligence*, 143(1):79–122.

[Denecker and Ternovska, 2007] Denecker, M. and Ternovska, E. (2007). Inductive Situation Calculus. *Artificial Intelligence*, 171(5–6):332–360.

[Denecker et al., 1998] Denecker, M., Theseider-Dupré, D., and Van Belleghem, K. (1998). An Inductive Definition Approach to Ramifications. *Linköping Electronic Articles in Computer and Information Science*, 3(7):1–43.

[Drescher et al., 2009] Drescher, C., Schiffel, S., and Thielscher, M. (2009). A Declarative Agent Programming Language Based on Action Theories. In *Proceedings of the International Conference on Frontiers of Combining Systems (FroCoS 2009)*, volume 5749 of *LNCS*, pages 230–245, Trento, Italy. Springer.

[Dung, 1995] Dung, P. M. (1995). On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77:321–358.

[Eiter et al., 2000] Eiter, T., Faber, W., Leone, N., Pfeifer, G., and Polleres, A. (2000). Planning under Incomplete Knowledge. In Lloyd, J. W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Palamidessi, C., Pereira, L. M., Sagiv, Y., and Stuckey, P. J., editors, *CL*, volume 1861 of *Lecture Notes in Computer Science*, pages 807–821. Springer.

[Eiter et al., 2004] Eiter, T., Faber, W., Leone, N., Pfeifer, G., and Polleres, A. (2004). A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. *ACM Transactions on Computational Logic*, 5:206–263.

[Eiter et al., 1997] Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1997). A deductive system for non-monotonic reasoning. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Computer Science*, pages 364–375, Dagstuhl Castle, Germany. Springer.

[Fahlman, 1979] Fahlman, S. E. (1979). *NETL: A System for Representing and Using Real-World Knowledge*. MIT Press, Cambridge, MA.

[Fahlman et al., 1981] Fahlman, S. E., Touretzky, D. S., and van Roggen, W. (1981). Cancellation in a Parallel Semantic Network. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI -81)*, pages 257–263, Vancouver, BC, Canada.

[Ferraris et al., 2006] Ferraris, P., Lee, J., and Lifschitz, V. (2006). A Generalization of the Lin-Zhao Theorem. *Annals of Mathematics and Artificial Intelligence*, 47:79–101.

[Ferraris et al., 2011] Ferraris, P., Lee, J., and Lifschitz, V. (2011). Stable models and circumscription. *Artificial Intelligence*, 175(1):236–263. John McCarthy's Legacy.

[Fikes and Nilsson, 1971] Fikes, R. and Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4):189–208.

[Forth and Miller, 2007] Forth, J. and Miller, R. (2007). Ramifications: An Extension and Correspondence Result for the Event Calculus. *Journal of Logic and Computation*, 17(4):639–685.

[Gebser et al., 2010] Gebser, M., Grote, T., and Schaub, T. (2010). Coala: A Compiler from Action Languages to ASP. In Janhunen, T. and Niemelä, I., editors, *Proceedings of the Twelfth European Conference on Logics in Artificial Intelligence (JELIA-10)*, volume 6341 of *Lecture Notes in Artificial Intelligence*, pages 360–364. Springer-Verlag.

[Gebser et al., 2011] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. (2011). Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, 24(2):105–124. Available at `http://potassco.sourceforge.net`.

[Gebser et al., 2008] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Thiele, S. (2008). Engineering an Incremental ASP Solver. In Garcia de la Banda, M. and Pontelli, E., editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP-08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag.

[Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. In *Proceedings of the International Conference on Logic Programming (ICLP)*, pages 1070–1080. The MIT Press.

[Gelfond and Lifschitz, 1991] Gelfond, M. and Lifschitz, V. (1991). Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385.

[Gelfond and Lifschitz, 1993] Gelfond, M. and Lifschitz, V. (1993). Representing Action and Change by Logic Programs. *Journal of Logic Programming*, 17(2/3&4):301–321.

[Gelfond and Lifschitz, 1998] Gelfond, M. and Lifschitz, V. (1998). Action Languages. *Electronic Transactions on Artificial Intelligence*, 3.

[Gelfond et al., 1991] Gelfond, M., Lifschitz, V., and Rabinov, A. (1991). What Are the Limitations of the Situation Calculus? In Boyer, R. S. and Pase, W., editors, *Automated Reasoning*, volume 1 of *Automated Reasoning Series*, pages 167–179. Springer Netherlands.

[Ginsberg and Smith, 1987] Ginsberg, M. L. and Smith, D. E. (1987). Reasoning about Action I: A Possible Worlds Approach. *Artificial Intelligence*, 35:233–258.

[Giunchiglia et al., 1997] Giunchiglia, E., Kartha, G. N., and Lifschitz, V. (1997). Representing Action: Indeterminacy and Ramifications. *Artificial Intelligence*, 95(2):409–438.

[Giunchiglia et al., 2004] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., and Turner, H. (2004). Nonmonotonic Causal Theories. *Artificial Intelligence*, 153(1-2):49–104.

[Giunchiglia and Lifschitz, 1998] Giunchiglia, E. and Lifschitz, V. (1998). An Action Language Based on Causal Explanation: Preliminary Report. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 623–630, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[Gottlob, 1992] Gottlob, G. (1992). Complexity Results for Nonmonotonic Logics. *Journal of Logic and Computation*, 2(3):397–425.

[Halpern, 1997] Halpern, J. Y. (1997). A Critical Reexamination of Default Logic, Autoepistemic Logic, and Only Knowing. *Computational Intelligence*, 13(1):144–163.

[Hanks and McDermott, 1987] Hanks, S. and McDermott, D. (1987). Nonmonotonic Logic and Temporal Projection. *Artificial Intelligence*, 33(3):379–412.

[Herzig and Varzinczak, 2007] Herzig, A. and Varzinczak, I. J. (2007). Metatheory of actions: Beyond consistency. *Artificial Intelligence*, 171(16–17):951–984.

[Hölldobler and Schneeberger, 1990] Hölldobler, S. and Schneeberger, J. (1990). A New Deductive Approach to Planning. *New Generation Computing*, 8(3):225–244.

[Jacso, 2009] Jacso, P. (2009). Google Scholar's Ghost Authors. *Library Journal*, 134(18):26–27.

[Junker and Konolige, 1990] Junker, U. and Konolige, K. (1990). Computing the Extensions of Autoepistemic and Default Logics with a Truth Maintenance System. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 278–283. AAAI Press / The MIT Press.

[Kakas et al., 2008] Kakas, A., Michael, L., and Miller, R. (2008). Fred meets Tweety. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 747–748, Amsterdam, The Netherlands. IOS Press.

[Kakas et al., 2011] Kakas, A., Michael, L., and Miller, R. (2011). Modular-e and the role of elaboration tolerance in solving the qualification problem. *Artificial Intelligence*, 175(1):49–78. John McCarthy's Legacy.

[Kakas and Miller, 1997] Kakas, A. and Miller, R. (1997). A Simple Declarative Language for Describing Narratives with Actions. *Journal of Logic Programming*, 31(1–3):157–200. Reasoning about Actions and Change.

[Kakas et al., 2005] Kakas, A. C., Michael, L., and Miller, R. (2005). Modular-epsilon: An elaboration tolerant approach to the ramification and qualification problems. In *Proceedings of the Eight International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-05)*, pages 211–226. Springer.

[Kartha, 1994] Kartha, G. N. (1994). Two Counterexamples Related to Baker's Approach to the Frame Problem. *Artificial Intelligence*, 69(1–2):379–391.

[Kautz, 1986] Kautz, H. (1986). The Logic of Persistence. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 401–405.

[Kim et al., 2009] Kim, T.-W., Lee, J., and Palla, R. (2009). Circumscriptive Event Calculus as Answer Set Programming. In *IJCAI*, pages 823–829.

[Konolige, 1994] Konolige, K. (1994). Using Default and Causal Reasoning in Diagnosis. *Annals of Mathematics and Artificial Intelligence*, 11:97–135.

[Kowalski, 1974] Kowalski, R. A. (1974). Predicate logic as programming language. In *IFIP Congress*, pages 569–574.

[Kowalski and Kuehner, 1971] Kowalski, R. A. and Kuehner, D. (1971). Linear Resolution with Selection Function. *Artificial Intelligence*, 2(3/4):227–260.

[Kowalski and Sergot, 1986] Kowalski, R. A. and Sergot, M. J. (1986). A Logic-based Calculus of Events. *New Generation Computing*, 4(1):67–95.

[Kraus et al., 1990] Kraus, S., Lehmann, D., and Magidor, M. (1990). Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44:167–207.

[Kvarnström and Doherty, 2000] Kvarnström, J. and Doherty, P. (2000). Tackling the Qualification Problem using Fluent Dependency Constraints. *Computational Intelligence*, 16(2):169–209.

[Lakemeyer and Levesque, 2009] Lakemeyer, G. and Levesque, H. (2009). A Semantical Account of Progression in the Presence of Defaults. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 842–847.

[Lee and Palla, 2010] Lee, J. and Palla, R. (2010). Situation Calculus as Answer Set Programming. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-10)*, pages 309–314.

[Levesque, 1990] Levesque, H. J. (1990). All I Know: A Study in Autoepistemic Logic. *Artificial Intelligence*, 42(2–3):263–309.

[Lifschitz, 1985] Lifschitz, V. (1985). Computing Circumscription. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 121–127. Morgan Kaufmann.

[Lifschitz, 1986] Lifschitz, V. (1986). Pointwise Circumscription: Preliminary Report. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, pages 406–410. Morgan Kaufmann.

[Lifschitz, 1990] Lifschitz, V. (1990). On Open Defaults. In *Proceedings of the Computational Logic Symposium*. Morgan Kaufmann.

[Lifschitz et al., 1993] Lifschitz, V., McCain, N., and Turner, H. (1993). Automated Reasoning about Actions: A Logic Programming Approach. In *Proceedings of the 1993 International Symposium on Logic Programming (ILPS)*, page 641.

[Lin, 1995] Lin, F. (1995). Embracing Causality in Specifying the Indirect Effects of Actions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1985–1993, Montréal, Canada. Morgan Kaufmann.

[Lin and Reiter, 1994] Lin, F. and Reiter, R. (1994). State Constraints Revisited. *Journal of Logic and Computation*, 4(5):655–677.

[Lin and Reiter, 1997] Lin, F. and Reiter, R. (1997). How to Progress a Database. *Artificial Intelligence*, 92(1–2):131–167.

[Lin and Soutchanski, 2011] Lin, F. and Soutchanski, M. (2011). Causal theories of actions revisited. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*. AAAI Press.

[Lin and Zhao, 2004] Lin, F. and Zhao, Y. (2004). ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. *Artificial Intelligence*, 157(1-2):115–137.

[Marek and Truszczyński, 1989] Marek, V. W. and Truszczyński, M. (1989). Stable semantics for logic programs and default theories. In *North American Conference on Logic Programming*, pages 243–256. The MIT Press.

[Martin and Thielscher, 2001] Martin, Y. and Thielscher, M. (2001). Addressing the Qualification Problem in FLUX. In *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI/ÖGAI-01)*, pages 290–304, Vienna, Austria.

[McCain and Texas Action Group, 1997] McCain, N. and Texas Action Group (1997). The Causal Calculator. `http://www.cs.utexas.edu/users/tag/cc/`.

[McCain and Turner, 1995] McCain, N. and Turner, H. (1995). A Causal Theory of Ramifications and Qualifications. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1978–1984, Montréal, Canada. Morgan Kaufmann.

[McCarthy, 1959] McCarthy, J. (1959). Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91.

[McCarthy, 1963] McCarthy, J. (1963). Situations and Actions and Causal Laws. Stanford Artificial Intelligence Project: Memo 2.

[McCarthy, 1977] McCarthy, J. (1977). Epistemological Problems of Artificial Intelligence. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 1038–1044.

[McCarthy, 1980] McCarthy, J. (1980). Circumscription—A Form of Non-monotonic Reasoning. *Artificial Intelligence*.

[McCarthy, 1986] McCarthy, J. (1986). Applications of Circumscription to Formalizing Common-Sense Knowledge. *Artificial Intelligence*, 28(1):89–116.

[McCarthy, 2003] McCarthy, J. (2003). Elaboration Tolerance. In progress.

[McCarthy and Hayes, 1969] McCarthy, J. and Hayes, P. J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press.

[McIlraith, 2000] McIlraith, S. (2000). Integrating Actions and State Constraints: A Closed-Form Solution to the Ramification Problem (Sometimes). *Artificial Intelligence*, 116(1–2):87–121.

[Michael and Kakas, 2011] Michael, L. and Kakas, A. (2011). A Unified Argumentation-Based Framework for Knowledge Qualification. In Davis, E., Doherty, P., and Erdem, E., editors, *Proceedings of the Tenth International Symposium on Logical Formalizations of Commonsense Reasoning*, Stanford, CA.

[Michael and Kakas, 2009] Michael, L. and Kakas, A. C. (2009). Knowledge Qualification through Argumentation. In *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 209–222.

[Minsky, 1974] Minsky, M. (1974). A Framework for Representing Knowledge. Technical Report AIM-306, MIT AI Lab.

[Moore, 1985] Moore, R. (1985). Semantical Considerations of Nonmonotonic Logic. *Artificial Intelligence*, 25(1):75–94.

[Mueller, 2006] Mueller, E. (2006). *Commonsense Reasoning*. Morgan Kaufmann.

[Pagnucco et al., 2011] Pagnucco, M., Rajaratnam, D., Strass, H., and Thielscher, M. (2011). How to Plan When Being Deliberately Misled. In *Proceedings of the Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*.

[Pearl, 1987] Pearl, J. (1987). Embracing Causality in Formal Reasoning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 369–373, Seattle, Washington.

[Pednault, 1989] Pednault, E. P. D. (1989). ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 324–332. Morgan Kaufmann.

[Pinto, 1999] Pinto, J. (1999). Compiling Ramification Constraints into Effect Axioms. *Computational Intelligence*, 15:280–307.

[Pirri and Reiter, 1999] Pirri, F. and Reiter, R. (1999). Some Contributions to the Metatheory of the Situation Calculus. *Journal of the ACM*, 46(3):325–361.

[Reinfrank et al., 1989] Reinfrank, M., Dressler, O., and Brewka, G. (1989). On the relation between truth maintenance and autoepistemic logic. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence - Volume 2*, pages 1206–1212. Morgan Kaufmann Publishers Inc.

[Reiter, 1980] Reiter, R. (1980). A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132.

[Reiter, 1991] Reiter, R. (1991). The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In *Artificial Intelligence and Mathematical Theory of Computation – Papers in Honor of John McCarthy*, pages 359–380. Academic Press.

[Reiter, 1993] Reiter, R. (1993). Proving Properties of States in the Situation Calculus. *Artificial Intelligence*, 64:337–351.

[Reiter, 2001] Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

[Sandewall, 1972] Sandewall, E. (1972). An approach to the frame problem, and its implementation. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume 7, pages 195–204. Edinburgh University Press.

[Sandewall, 1994] Sandewall, E. (1994). *Features and Fluents: The Representation of Knowledge about Dynamical Systems*. Oxford University Press, Oxford.

[Shanahan, 1997] Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. The MIT Press.

[Shanahan, 1999] Shanahan, M. (1999). The Ramification Problem in the Event Calculus. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 140–146. Morgan Kaufmann.

[Strass, 2011] Strass, H. (2011). Default Reasoning about Conditional, Non-Local and Disjunctive Effect Actions. In *Proceedings of the Ninth International Workshop on Nonmonotonic Reasoning, Action and Change (NRAC-2011)*.

[Strass, 2012] Strass, H. (2012). The draculasp System: Default Reasoning about Actions and Change Using Logic and Answer Set Programming. In Rosati, R. and Woltran, S., editors, *Proceedings of the Fourteenth International Workshop on Non-Monotonic Reasoning (NMR)*.

[Strass and Thielscher, 2009a] Strass, H. and Thielscher, M. (2009a). Defaults in Action: Nonmonotonic Reasoning About States in Action Calculi. In Lakemeyer, G., Morgenstern, L., and Williams, M.-A., editors, *Proceedings of the Ninth International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 123–128, Toronto, Canada. *Outstanding Student Paper Award*.

[Strass and Thielscher, 2009b] Strass, H. and Thielscher, M. (2009b). On Defaults in Action Theories. In *Proceedings of the 32nd German Annual Conference on Artificial Intelligence (KI-09)*, pages 298–305, Paderborn, Germany. Springer-Verlag Berlin Heidelberg.

[Strass and Thielscher, 2009c] Strass, H. and Thielscher, M. (2009c). Simple Default Reasoning in Theories of Action. In *Proceedings of the 22nd Australasian Joint Conference on Artificial Intelligence (AI-09)*, pages 31–40, Melbourne, Australia. Springer-Verlag Berlin Heidelberg.

[Strass and Thielscher, 2010a] Strass, H. and Thielscher, M. (2010a). Default Reasoning in Action Theories with Conditional, Non-Local Effect Actions. In *Short Proceedings of the Seventeenth International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*.

[Strass and Thielscher, 2010b] Strass, H. and Thielscher, M. (2010b). A General First-Order Solution to the Ramification Problem. In Meyer, T. and Ternovska, E., editors, *Proceedings of the 13th International Workshop on Non-Monotonic Reasoning*, Toronto, Canada. CEUR-WS.org.

[Strass and Thielscher, 2012] Strass, H. and Thielscher, M. (2012). A Language for Default Reasoning about Actions. In *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *LNCS*, pages 527–542. Springer-Verlag Berlin Heidelberg.

[Thielscher, 1995] Thielscher, M. (1995). Computing Ramifications by Postprocessing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1994–2000, Montréal, Canada. Morgan Kaufmann.

[Thielscher, 1996] Thielscher, M. (1996). Causality and the Qualification Problem. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 51–62, Cambridge, MA.

[Thielscher, 1997] Thielscher, M. (1997). Ramification and Causality. *Artificial Intelligence*, 89(1–2):317–364.

[Thielscher, 1999] Thielscher, M. (1999). From Situation Calculus to Fluent Calculus: State Update Axioms as a Solution to the Inferential Frame Problem. *Artificial Intelligence*, 111(1–2):277–299.

[Thielscher, 2000] Thielscher, M. (2000). Nondeterministic Actions in the Fluent Calculus: Disjunctive State Update Axioms. In *Intellectics and Computational Logic (to Wolfgang Bibel on the occasion of his 60th birthday)*, pages 327–345, Deventer, The Netherlands, The Netherlands. Kluwer, B.V.

[Thielscher, 2001] Thielscher, M. (2001). The Qualification Problem: A Solution to the Problem of Anomalous Models. *Artificial Intelligence*, 131(1-2):1–37.

[Thielscher, 2005a] Thielscher, M. (2005a). FLUX: A Logic Programming Method for Reasoning Agents. *Theory and Practice of Logic Programming*, 5(4-5):533–565.

[Thielscher, 2005b] Thielscher, M. (2005b). *Reasoning Robots: The Art and Science of Programming Robotic Agents*. Applied Logic Series. Springer, 1st edition.

[Thielscher, 2011] Thielscher, M. (2011). A Unifying Action Calculus. *Artificial Intelligence*, 175(1):120–141.

[Van Belleghem et al., 1998] Van Belleghem, K., Denecker, M., and Theseider-Dupré, D. (1998). A Constructive Approach to the Ramification Problem. In *Reasoning about Actions; Foundations and Applications*, pages 1–17.

[van Emden and Kowalski, 1976] van Emden, M. H. and Kowalski, R. A. (1976). The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742.

[Wilkins, 1988] Wilkins, D. E. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, CA.

[Winslett, 1988] Winslett, M. (1988). Reasoning about Action Using a Possible Models Approach. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 89–93.

Index

L_1 subsumes L_2, 21
$Do([a_1, \ldots, a_n], s)$, 17

action default theory, 29, 46
action domain specification (ADS) Θ, 25
action occurrence axiom, 27
actions, 2
activation of an effect law, 50, 53
activation of an effect of a disjunctive effect law, 54
admissible action domain specification, 70
admissible instance information, 70
answer set, 20
arity, 11
atom, 13
atomic formula, 13
atomic state default, 25

Basic Action Theories, 6
body of a logic program rule, 20
branching-time domain axiomatisation for Θ, 26

clause, 13
closed default, 18
closed default theory, 19
closed formula, 14
closed indirect effect law, 92
complete set of loops, 21
condition of a direct effect law, 24
conditional direct effect law for $A(\vec{x})$, 25
conflict between α and δ in E, 51
conjunctive normal form (CNF), 13
consequent of a default, 18
consequent of a state default, 25
corresponding definite Horn default, 69
corresponding definite Horn default theory, 70
corresponding logic program rule, 79
credulous consequence of (W, D), 19

deductive closure, 14

default, 18
default closure axioms for Θ, 46
default closure axioms for $F(\vec{x})$ with respect to the state defaults in Θ, 46
default theory, 19
default theory (W_Λ, D_Λ) of the logic program Λ, 21
definite Horn clause, 13
definite Horn default, 18
definite Horn translation of clause c, 67
definite logic program, 20
definite logic program rule, 20
dependency graph for a logic program, 21
dependency of program atoms, 20
deterministic direct effect law for $A(\vec{x})$, 24
direct effect law for $A(\vec{x})$, 24
direct positive and negative effect formulas, 53, 55
direct positive and negative effect formulas for $A(\vec{x})$, 42, 50
disjunction-free default, 18
disjunction-free precondition law, 24
disjunction-free state default, 25
disjunctive direct effect law for $A(\vec{x})$, 24
disjunctive normal form (DNF), 13
domain axiomatisation Σ, 43
domain closure axiom, 63
domain rules for depth n, 80
domain signature, 16

effect axiom Υ_A with conditional effects, the frame assumption and ramifications, 94, 97
effect axiom with conditional effects, the frame assumption and normal state defaults, 50
effect axiom with unconditional effects, 29
effect axiom with unconditional effects and occlusions, 36
effect axiom with unconditional effects and the frame assumption, 43
effect axiom with unconditional effects, the frame assumption and normal state defaults, 45
effect of a direct effect law, 24
effect of an indirect effect law, 92
elaboration tolerance, 4
endogenous qualification problem, 5
entailment, 14
exogenous qualification problem, 5
expansion, 9
extension for (W, D), 19

first-order loop of Λ, 21
fluent formula of FOL with equality, 24
fluents, 2
formula of FOL with equality, 12
frame problem, 4
free variables of a formula, 13

Gelfond-Lifschitz reduct, 20
generating defaults, 19

global direct effect law for $A(\vec{x})$, 25
ground logic program rule, 20
ground normal logic program, 20
ground term, 12

head of a logic program rule, 20
Herbrand base, 20
Herbrand domain of depth n, 80
Herbrand interpretation, 20
Herbrand universe, 20
Horn extension, 67

indirect effect law, 92
indirect effect rules, 98
influence graph G_Θ of Θ, 95
initial context of an indirect effect law, 92
interpretation, 14

justification-free default, 18
justifications of a default, 18

law, 25
leads into the loop L, 95
linear-time domain axiomatisation, 27
literal, 13
local conclusions about a time point, 32
local direct effect law for $A(\vec{x})$, 25
local variable in a logic program rule, 20
loop, 95

model, 14
modular translation function, 60
more general unifier, 15
most general unifier, 15

narratives, 7
negation normal form (NNF), 13
negative literal, 13
negative name, 66
negative occurrence of a variable in a logic program rule, 20
normal default, 18
normal logic program, 20
normal logic program rule, 20

occlusion completion of Θ, 37
occlusion completion of Θ for $A(\vec{x})$, 37
occlusion for $A(\vec{x})$, 25
ontologies, 1
open default, 18
open default theory, 19

open formula, 14
open indirect effect law, 92

positive literal, 13
positive name, 66
positive occurrence of a variable in a logic program rule, 20
precondition law for $A(\vec{x})$, 24
prerequisite of a default, 18
prerequisite of a state default, 25
prerequisite-free default, 18
prerequisite-free state default, 25
progressing domain axiomatisation, 17
propositional signature, 12

qualification problem, 111

ramification problem, 5
reachable in Σ, 18
reification, 6
relational signature, 12
resolvable, 15
resolvent of two clauses, 15
resulting state of α in S, 91

safe logic program rule, 20
satisfaction relation, 14
satisfaction relation $\models$, 90
satisfiable, 14
sceptical consequence of (W, D), 19
sentence, 14
sequential domain axiomatisation, 17
signature, 11
size of a formula, 13
size of a term, 13
sort, 11
stable model, 20
state, 90
state constraints, 5
state default, 25
state formula $\Phi[\vec{s}]$ in $\vec{s}$, 16
state update, 91
strongly sceptically reachable in (Σ, Δ), 31
strongly sceptically reachable in $(\Sigma, \Delta[\sigma])$, 31
structure, 14
substitution, 15
successor state axioms, 5
successor state of S for α, 93
supernormal default, 18

term, 12

term depth, 12
terminal context of an indirect effect law, 92
theory, 14
time points, 1
trigger of an indirect effect law, 92
triggered indirect effect law, 93

UAC domain axiomatisation, 16
UAC effect axiom, 16
UAC precondition axiom, 16
unconditional direct effect law for $A(\vec{x})$, 25
unifiable terms, 15
unifier of two terms, 15
unique-names axiom for $F_1, \ldots, F_n$, 15
unique-names axioms for sort $\mathfrak{s}$, 15
unit clause, 13
unsorted signature, 12
update of S with L, 91

valid, 14
variable assignment, 14
variables of a term, 12

weakly sceptically reachable in (Σ, Δ), 31
weakly sceptically reachable in $(\Sigma, \Delta[\sigma])$, 31
well-sorted grounding, 64