Bridging the Gap between Classical Logic Based Formalisms and Logic Programs

by

Ravi Palla

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2011 by the
Graduate Supervisory Committee:

Joohyung Lee, Chair
Chitta Baral
Subbarao Kambhampati
Vladimir Lifschitz

ARIZONA STATE UNIVERSITY

May 2012

ABSTRACT

Different logic-based knowledge representation formalisms have different limitations either with respect to expressivity or with respect to computational efficiency. First-order logic, which is the basis of Description Logics (DLs), is not suitable for defeasible reasoning due to its monotonic nature. The nonmonotonic formalisms that extend first-order logic, such as circumscription and default logic, are expressive but lack efficient implementations. The nonmonotonic formalisms that are based on the declarative logic programming approach, such as Answer Set Programming (ASP), have efficient implementations but are not expressive enough for representing and reasoning with open domains.

This dissertation uses the first-order stable model semantics, which extends both first-order logic and ASP, to relate circumscription to ASP, and to integrate DLs and ASP, thereby partially overcoming the limitations of the formalisms. By exploiting the relationship between circumscription and ASP, well-known action formalisms, such as the situation calculus, the event calculus, and Temporal Action Logics, are reformulated in ASP. The advantages of these reformulations are shown with respect to the generality of the reasoning tasks that can be handled and with respect to the computational efficiency. The integration of DLs and ASP presented in this dissertation provides a framework for integrating rules and ontologies for the semantic web. This framework enables us to perform nonmonotonic reasoning with DL knowledge bases. Observing the need to integrate action theories and ontologies, the above results are used to reformulate the problem of integrating action theories and ontologies as a problem of integrating rules and ontologies, thus enabling us to use the computational tools developed in the context of the latter for the former.

DEDICATION

To my family

ACKNOWLEDGEMENTS

Several people, directly or indirectly, have played a part in my successful completion of the program. Some people though, deserve a special mention.

First of all, I would like to thank Joohyung for providing me the wonderful opportunity to work with him. I truly appreciate the amount of effort he put into supervising me. I thoroughly enjoyed the numerous discussions we had and can honestly say that I have improved a lot under his supervision. I would also like to thank Chitta Baral, Subbarao Kambhampati, and Vladimir Lifschitz for agreeing to be on my committee. It has been a privilege working with them, and I learnt a lot from each of them.

I am grateful to my current and former teammates, Yunsong, Tae-Won, Michael, Mike, Sunjin, Greg, Joseph, and Jeong-Jin, for the numerous technical (and many more non-technical) discussions. I am also grateful to my friends Shankar, Raju, Luis, and Aravind, and my friends and colleagues at Siemens, Dan, Vinay, Mathaeus, and Anshuman for their help and support.

I am deeply indebted to my parents, my sister, and my brother for their amazing support and understanding. None of this would have been possible without their encouragement. I am also indebted to my cousins and other members of my family for their encouragement. I would like to specially thank my niece, Vanshika, and my nephew, Satvik, for just being so cute and for making me smile.

This work was partially supported by the National Science Foundation under Grants IIS-0839821 and IIS-0916116, the IARPA SCIL program, and Siemens Corporate Research.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Knowledge representation and reasoning (KR&R) is the area of Aritificial Intelligence (AI) that is concerned with encoding knowledge in an adequately expressive formalism and drawing conclusions effectively from the encoded knowledge. A lot of work in the area is based on formal logic, and many logic-based formalisms have been proposed. However, developing a formalism that is both adequately expressive and efficiently computable has remained a constant fundamental challenge.

First-order logic is widely used in KR&R. It forms the basis of many well-known knowledge representation languages. For example, Description Logics (DLs), which are widely studied in the context of the semantic web, are decidable fragments of first-order logic. However, since inference in first-order logic is monotonic, it is not suitable for capturing defeasible inferences (the kind of inferences in everyday reasoning in which we tentatively derive conclusions and retract them in the light of further information). In order to overcome this limitation of first-order logic, there has been extensive research on nonmonotonic reasoning, and many formalisms have been proposed.

Circumscription (McCarthy, 1980, 1986) and Default Logic (Reiter, 1980) are among the first and most well-known nonmonotonic formalisms to have been introduced. These languages extend first-order logic and have been shown to be suitable for representing various commonsense reasoning domains. They were also used to provide elegant solutions to the frame problem, thus achieving one of the important goals of the theory of nonmonotonic reasoning. Some of the well-known action formalisms that use circumscription to solve the frame problem are the situation calculus (McCarthy & Hayes, 1969; Reiter, 2001; Lin, 1995), the

event calculus (Shanahan, 1995; Mueller, 2006), and Temporal Action Logics (TAL) (Doherty, Gustafsson, Karlsson, & Kvarnström, 1998). However, circumscription and default logic are hard to compute and thus lack efficient implementations.

Nonmonotonic reasoning is also studied in the context of logic programs, and a number of declarative languages have been proposed. One of the most well-known declarative logic programming languages is Answer Set Programming (ASP), which is based on the stable model semantics (Gelfond & Lifschitz, 1988). The traditional stable model semantics was shown to be a fragment of default logic but several important extensions of the semantics have been proposed that consider more general programs than the ones considered by the traditional semantics. ASP has a wide range of applications, examples of which include automated product configuration (Tiihonen, Soininen, Niemelä, & Sulonen, 2003), decision support for the space shuttle (Nogueira, Balduccini, Gelfond, Watson, & Barry, 2001) and phylogenetic tree inference (Brooks, Erdem, Erdoğan, Minett, & Ringe, 2007). One of the main reasons for the growing popularity of ASP is the availability of efficient off-the-shelf computational tools known as answer set solvers. ASP also has a rich theory and various mathematical tools are available for the analysis of programs. However, a limitation of ASP is that the semantics considers only Herbrand interpretations (variables are merely place-holders and are eliminated by grounding). This limitation implies that all the objects in the domain being modeled often need to be explicitly specified, which in turn implies that ASP does not effectively handle open domains, as often needed for modeling ontologies. The limitation also makes the analysis of programs difficult as the programs need to be grounded in order to apply the existing mathematical tools.

The above discussion on various well-known formalisms suggests that each formalism has interesting applications but also has certain limitations either

with respect to expressivity or with respect to computational efficiency. The discussion also suggests that the limitations of some formalisms are well-addressed by some other formalisms, which implies that relating or integrating the formalisms is a viable approach to (partially) overcoming their limitations.

Relating cirumscription and ASP enables us to use efficient answer set solvers for computing circumscriptive action theories, such as theories in the event calculus, the situation calculus, and TAL. This has several advantages. First, given the efficiency of the answer set solvers, ASP-based reasoners are expected to be more efficient on several domains when compared to the existing reasoners for the circumscriptive theories. Second, since improving answer set computation is a community-wide effort, answer set solvers are constantly improved, and these improvements can be carried over to the computation of the circumscriptive theories. This implies that the computation of the theories with different underlying formalisms can be improved without specifically focusing on each of the formalisms. Third, since answer set solvers can handle recursive axioms, ASP-based reasoners can handle certain reasoning tasks that cannot be handled by the existing reasoners for the circumscriptive action theories. Fourth, it enables us to view the underlying formalisms of the circumscriptive theories as high-level languages for ASP, thus allowing us to combine the corresponding theories with ASP-rules. This is particularly useful if we want to extend the theories by adding expressive ASP-rules such as the transitive closure rules.

Similarly, integrating DLs and ASP enables us to perform nonmonotonic reasoning using DL knowledge bases. Since DLs form the basis for the Web Ontology Language (OWL), integration of DLs and ASP provides a framework for the integration of rules and ontologies, which is a key fragment of the semantic web architecture.

Interestingly, while reasoning about actions and integrating rules and ontologies have largely been treated as separate problems, there are certain applications in which it is useful to relate the two. For instance, consider a medical expert system that is required to assist physicians in diagnosis and treatment of diseases/disorders. Such a system needs to be able to reason with various *cause-effect* relationships, such as the causes of various diseases/disorders and the effects of various drugs or disorders on the human body. For this, the system needs to have sufficient access to information regarding anatomy, pathology, pharmacology, and other related domains. Since much of this information is available in the form of ontologies, the system also needs to be able to query various biomedical ontologies. So, essentially, the system needs to be able to reason with dynamic domains while using the ontologies as knowledge bases. While there have been several approaches and computational tools presented for integrating rules and ontologies, there is not much work on integrating (circumscriptive) action theories and ontologies. By using a common semantic framework for relating circumscription to ASP and integrating DLs and ASP, we can reformulate the problem of integrating action theories and ontologies as a problem of integrating ASP-rules and ontologies, thus enabling us to use the computational tools of the latter for the former.

In order to relate circumscription to ASP, and to integrate DLs and ASP, we need to bridge the gap between the underlying logics: first-order logic and the stable model semantics. These languages are syntactically and semantically very different. First, the (traditional) stable model semantics (Gelfond & Lifschitz, 1988) is restricted to rule form while first-order logic considers first-order formulas. Second, the stable model semantics refers to grounding to eliminate variables, which implies that it considers only Herbrand interpretations. On the other hand, first-order logic has no such restriction. Third, the negation under the stable model

semantics is default negation (*not*) while that in first-order logic is the usual classical negation. In addition to bridging the gap between the logics, we also need to relate circumscription to the stable model semantics.

A recent generalization of the stable model semantics makes our tasks a bit easier since it bridges some of the gaps between first-order logic and the traditional stable model semantics (Gelfond & Lifschitz, 1988). The first-order stable model semantics (FOSM) defined by Ferraris, Lee and Lifschitz (2007, 2011) extends the traditional semantics to first-order formulas. The stable models, according to this new definition, are not restricted to Herbrand interpretations. Further, the stable models are characterized as the models of a second-order sentence, which is similar to that used in the definition of circumscription. Since the first-order stable model semantics incorporates features from first-order logic, circumscription, and the traditional stable model semantics, it provides an ideal framework for relating circumscription to ASP, and for integrating DLs and ASP. Towards accomplishing these tasks and overcoming some of the limitations of the corresponding formalisms, this dissertation

1. investigates the relationship between circumscription and FOSM and presents a uniform approach to reformulate the event calculus, the situation calculus, and TAL in ASP;
2. presents a system for computing descriptions in the event calculus, the situation calculus, and TAL using existing answer set solvers, and demonstrates some of its advantages;
3. presents a FOSM-based approach to integrate DLs and ASP, and relates it to several existing approaches;
4. presents a FOSM-based approach to reformulate the problem of integrating cirumscriptive action theories and ontologies as a problem of integrating

ASP-rules and ontologies, and discusses some interesting examples with respect to the healthcare/biomedical domain.

5. investigates several properties of the first-order stable model semantics, which, in addition to being useful for accomplishing the above tasks, also have other interesting applications with respect to overcoming some limitations of ASP.

The document is organized as follows. Chapter 2 gives the necessary background information. Chapters 3-6 present various interesting properties of FOSM and discuss some of their applications. Chapter 5 also presents system F2LP (**F**ormula **t**o **L**ogic **P**rogram) that turns formulas in FOSM, under certain conditions, into the syntax of answer set programs. Chapters 7, 8, and 9 present reformulations of the event calculus, the situation calculus, and TAL, respectively, in ASP. These chapters also show how to use F2LP to compute descriptions in these formalisms using existing answer set solvers, and compare this ASP-based computation with some existing reasoning engines for the formalisms. Chapter 10 uses FOSM to integrate ASP-rules and ontologies, and relates this approach to several existing approaches. Chapter 11 presents an interesting combination of reasoning about actions and integrating rules and ontologies, with applications in the healthcare/biomedical domain. Finally, Chapter 12 presents a conclusion.

Chapter 2

BACKGROUND

2.1 The Situation Calculus

The situation calculus is one of the most well-known action formalisms. It was originally introduced by McCarthy (1963). Reiter's version (Reiter, 2001) of the situation calculus differs from the original version mainly in terms of the interpretation of situations. According to McCarthy, a situation is "the complete state of the universe at an instance of time". On the other hand, according to Reiter, a situation is the same as its history, which is the finite sequence of actions performed since the initial situation. The frame problem was extensively studied under the situation calculus and many solutions were proposed (see, for example, (McCarthy, 1986; Pednault, 1989; Schubert, 1990; Reiter, 1991)). The literature of the situation calculus is very rich and many languages have been proposed. Here, we consider two well-known languages - Basic Action Theories (BATs) (Reiter, 2001) and Lin's Causal Theories (Lin, 1995).

BATs were introduced by Reiter and use the approach in (Reiter, 1991) to solve the frame problem. They have implementations based on Prolog, which enables expressive first-order reasoning. These theories are extensively used in Golog (Levesque, Reiter, Lespérance, Lin, & Scherl, 1997) and its extensions ConGolog (Giacomo, Lespérance, & Levesque, 2000) and IndiGolog (Giacomo & Levesque, 1999; Giacomo, Levesque, & Sardiña, 2001; Sardiña, Giacomo, Lespérance, & Levesque, 2004), which are high-level languages for the situation calculus, and which have been shown to be well-suited for various applications such as *high-level control of robots* (Burgard, Cremers, Fox, Hähnel, Lakemeyer, Schulz, Steiner, & Thrun, 1999), *web service composition* (McIlraith & Son, 2002), and *vision systems* (Borzenko, Xu, Obsniuk, Chopra, Jasiobedzki, Jenkin, & Lespérance, 2006). However, a drawback of BATs is that they do not provide an

effective solution to the ramification problem. This drawback is overcome by Lin's causal theories (Lin, 1995), that use cirumscription to solve the frame problem and the ramification problem. However, since circumscription is hard to compute, the computational tools for Lin's causal theories (Lin, 2003; Lin & Wang, 1999) rely on the reduction of circumscription to first-order logic, which is only possible under certain conditions.

2.2 The Event Calculus

The event calculus was originally introduced by Kowalski and Sergot (1986) in the framework of logic programs but was later extensively developed under the classical logic setting (Shanahan, 1995; Miller & Shanahan, 1999). The work based on classical logic uses circumscription to solve the frame problem and the ramification problem. The event calculus is a very expressive language and can handle a variety of reasoning tasks such as reasoning with compound events and hierarchical planning, reasoning about continuous change, reasoning with indeterminate effects, and reasoning with indirect effects. It has been applied to various areas of science and technology, including *open interaction systems* (Fornara & Colombetti, 2008), *robotics* (Patkos & Plexousakis, 2009), *software engineering* (Classen, Heymans, & Schobbens, 2008), and *web service composition* (Rouached, Perrin, & Godart, 2006). A key difference between the event calculus and the situation calculus is with respect to the time structures used. While the situation calculus uses a branching time structure, the event calculus uses a linear time structure.

Initial implementations of the event calculus were based on logic programming and mostly handled only abduction and planning problems (see, for example, (Shanahan, 2000)[1]). Later, Shanahan and Witkowski (2004) introduced a SAT-based planner which was shown to be more efficient than the logic

[1] http://www.iis.ee.ic.ac.uk/∼mpsha/planners.html

programming based planners. However, due to its limited coverage of the event calculus, it solves only 1 out of the 14 benchmark problems (Mueller, 2004) from (Shanahan, 1997, 1999). Mueller (2004) introduced a SAT-based event calculus reasoner that not only solves abduction and planning problems, but also solves other interesting problems such as projection and postdiction. This system is called the DEC reasoner, and is available at

http://decreasoner.sourceforge.net/ .

Similar to the computational tools for Lin's causal theories, the DEC reasoner computes circumscription by reducing it to first-order logic (Lifschitz, 1994). It handles a large fragment of the event calculus (Mueller, 2006) and solves 11 out of the 14 benchmark problems. It has been used for various applications, some of which are listed on its webpage.

2.3 Temporal Action Logics

Temporal Action Logics (TAL) (Doherty et al., 1998) is a class of logics for reasoning about action and change that are based on the *Features and Fluents* framework of Sandewall (1994). Some of the languages that belong to this class are PMON (Sandewall, 1994), **PMON-RC** (Gustafsson & Doherty, 1996), **TAL 1.0** (**PMON**$^+$) (Doherty, 1996), **TAL-C** (Karlsson & Gustafsson, 1999), **TAL 2.0** (Doherty et al., 1998), and **TAL-Q** (Kvarnström & Doherty, 2000). By TAL, we refer to the language presented in (Doherty & Kvarnström, 2008).which is essentially the latest kernel of this class of logics. Like the event calculus, TAL also uses a linear time structure. In TAL, *features* represent properties of the world, and *fluents* are functions of time representing the values of the features over time. Unlike the situation calculus and the event calculus, actions in TAL are by default durative, due to which, it provides a suitable framework for representing and reasoning with problems involving *required concurrency* (Cushing, Kambhampati,

Mausam, & Weld, 2007). Among other interesting features of TAL is the support for *durational* fluents. TAL uses circumscription to solve the frame problem, ramification problem, and the qualification problem, and the language is carefully designed so that circumscription in the language can be reduced to first-order logic using "predicate completion" presented in (Lifschitz, 1994).

VITAL[2] is a tool for reasoning about actions using TAL. The tool supports rich featuers, including ramification constraints, qualification constraints, and durational fluents. It not only generates the models but also provides a visualization of the models, which makes it easier to verify the output. TALplanner[3] (Kvarnström, 2005) is a forward-chaining planner based on TAL. The planner uses domain-dependent control rules specified in the description to prune the search space, which results in a considerable improvement in the efficiency. The planner participated in the second and third international planning competitions (IPC-2000[4] and IPC-2002[5]) and won the "distinguished planner" award in the hand-tailored track of IPC-2000.

2.4 Action Languages

Action Languages are high-level languages that are used to succintly describe transition systems. Examples include STRIPS (Fikes & Nilsson, 1971), ADL (Pednault, 1989), PDDL (McDermott, Ghallab, Howe, Knoblock, Ram, Veloso, Weld, & Wilkins, 1998), $\mathcal{A}$ (Gelfond & Lifschitz, 1998), $\mathcal{B}$ (Gelfond & Lifschitz, 1998), $\mathcal{C}$ (Giunchiglia & Lifschitz, 1998), and $\mathcal{C}+$ (Giunchiglia, Lee, Lifschitz, McCain, & Turner, 2004).

The language STRIPS (Stanford Research Institute Problem Solver), which was introduced in the context of automated planning, is one of the earliest action

[2] http://www.ida.liu.se/~jonkv/vital

[3] http://www.ida.liu.se/divisions/aiics/aiicssite/projects/talplanner.en.shtml

[4] http://www.cs.toronto.edu/aips2000/

[5] http://planning.cis.strath.ac.uk/competition/

languages to be introduced. It forms the basis for several languages that are currently being used to describe planning domains. ADL (Action Description Language) is an extension of STRIPS that allows one to represent conditional effects. PDDL (Planning Domain Definition Language), which can be considered as an extension of ADL, is the official language for the international planning competitions. Over the years, several extensions of PDDL have been proposed. While PDDL (v. 1.2) (McDermott et al., 1998) had roughly the same expressiveness as ADL, PDDL (v. 2.1) (Fox & Long, 2003) introduced durative actions, *numeric expressions* (for modeling domains involving fluents with numeric values), and *plan metrics* (such as *minimize* and *maximize* for utility-driven planning). PDDL (v. 2.2) (Edelkamp & Hoffmann, 2004) introduced *derived predicates*, and *timed initial literals* for modeling exogenous events. PDDL (v. 3.0) (Gerevini & Long, 2005) introduced *state trajectory constraints* (for specifying constriants over the trajectories) and *soft constraints and preferences* (for specifying constraints that need not be satisfied but that the user would prefer to see satisfied). PDDL (v. 3.1) is an extension of PDDL (v. 3.0) that allows the range of functions to be non-numeric.

Action language $\mathcal{A}$ (Gelfond & Lifschitz, 1998), which is essentially the propositional fragment of ADL, is based on ASP. In $\mathcal{A}$, effects of actions can be specified using statements such as *"A causes B if C"*, which represents the knowledge that action *A* has effect *B* under condition *C*. There are several extensions of language $\mathcal{A}$, including languages $\mathcal{A}_C$ (Baral & Gelfond, 1997) and $\mathcal{B}$ (Gelfond & Lifschitz, 1998) that enable representation of concurrent actions and indirect effects respectively. Action language $\mathcal{C}$ (Giunchiglia & Lifschitz, 1998), which is based on causal logic (McCain & Turner, 1997), provides a convenient way to represent nondeterministic actions and concurrent actions. Further, it allows one to choose which fluent is inertial and which is not. Language

$\mathcal{C}+$ (Giunchiglia et al., 2004) extends $\mathcal{C}$ by providing support for functional fluents and by introducing features such as *action attributes* and *additive fluents*[6] (Lee & Lifschitz, 2001). System CCALC[7] is a SAT-based implementation of an expressive fragment of $\mathcal{C}+$, and system COALA[8] is a compiler from action languages to answer set programs which supports several action languages including $\mathcal{B}$, $\mathcal{C}$, and a fragment of $\mathcal{C}+$.

Comparison with Circumscriptive Action Formalisms

While action languages and circumscriptive action formalisms address several common problems in the area of reasoning about actions, there are certain key differences between the two.This section briefly discusses some differences between circumscriptive action formalisms and PDDL, and between circumscriptive action formalisms and the action languages based on ASP and causal logic.

One of the key differences between circumscriptive action formalisms (such as the event calculus, the situation calculus, and Temporal Action Logics) and PDDL is that while the former are based on first-order logic (augmented with circumscription), the latter has a transition system based semantics. Another key difference is with respect to the representation of indirect effects. As we will see in the later chapters, the circumscriptive action formalisms handle direct effects and indirect effects in a similar way. As a result, the value of a fluent can change both as a direct effect of an action and as an indirect effect of a possibly different action. This is in contrast to PDDL (v. 2.2), where "derived predicates" cannot occur in the effect lists of actions. On the other hand, PDDL also has certain interesting features that are not well-studied in the context of circumscriptive action

[6] An additive fluent is a fluent with numerical values such that the effect of concurrently executed actions on the fluent can be computed by adding the effects of the individual actions.

[7] http://www.cs.utexas.edu/~tag/cc/

[8] http://potassco.sourceforge.net/

formalisms. Examples include “plan metrics” and “soft constraints and preferences” discussed above.

Similarly, there are some key differences between circumscriptive action formalisms and the action languages based on ASP and causal logic. For example, representation of continuous change and compound events are well-studied in the event calculus but not in the framework of action languages. Another example is the representation of durative actions that is well-studied in TAL and the event calculus but not in the framework of action languages. On the other hand, representation of action attributes and additive fluents are well-studied in $\mathcal{C}+$ but not in the framework of the circumscriptive action formalisms.

2.5 Stable Model Semantics and Answer Set Programming

The traditional stable model semantics (Gelfond & Lifschitz, 1988) applies to programs consisting of rules of the form

$$A \leftarrow A_1, \ldots, A_m, \textit{not}\ A_{m+1}, \ldots, \textit{not}\ A_n \tag{2.1}$$

where $n \geq m \geq 0$, *not* is default negation, comma(,) represents conjunction and A and each A_i are ground atoms. Such programs are traditionally referred to as *normal logic programs* and logic programs under the stable model semantics are often referred to as *answer set programs*. In (2.1), A is the *head* of the rule and $A_1, \ldots, A_m$, *not* $A_{m+1}, \ldots,$ *not* A_n is the *body* of the rule. Further, $A_1, \ldots, A_m$ is the *positive* part of the body and *not* $A_{m+1}, \ldots,$ *not* A_n is the *negative* part of the body. If the body is empty, then the rule is called a *fact* and if the head is empty, the rule is called a *constraint*. A program is called *positive* if none of the rules in the program contains the negative part of the body. A set of ground atoms X is a *stable model* (*answer set*) of a normal logic program Π if it is the minimal model of Π^X, where Π^X is the positive program (*reduct*) obtained by (i) removing all the rules such that for some *not* A_i in the negative body, $A_i \in X$, and (ii) removing the

negative bodies of all the remaining rules. According to this semantics, variables in answer set programs are understood as place-holders, to be replaced by the ground terms of the underlying signature[9]. The semantics was later extended to programs with classical negation, and disjunction in the head (Gelfond & Lifschitz, 1991).

Consider the statements "Normally, birds fly." and "Penguins do not fly." These can be represented by the following program Π:

$$\textit{flies}(x) \leftarrow \textit{bird}(x), \textit{not}\ \neg\textit{flies}(x)$$
$$\neg\textit{flies}(x) \leftarrow \textit{penguin}(x).$$

Here, '$\neg$' represents classical negation. The only answer set of $\Pi \cup \{\textit{bird}(\textit{tweety})\}$ is $\{\textit{bird}(\textit{tweety}), \textit{flies}(\textit{tweety})\}$, representing the conclusion that Tweety flies. However, the only answer set of $\Pi \cup \{\textit{bird}(\textit{tweety}), \textit{penguin}(\textit{tweety})\}$ is $\{\textit{bird}(\textit{tweety}), \neg\textit{flies}(\textit{tweety}), \textit{penguin}(\textit{tweety})\}$, representing the conclusion that Tweety does not fly. As we can see, the inference here is nonmonotonic.

The stable model semantics was further extended to programs with nested expressions (head and body can contain arbitrary nesting of default negation, conjunction and disjunction) (Lifschitz, Tang, & Turner, 1999) and programs with aggregates and choice rules.

Aggregates are constructs that greatly facilitate encoding. They have been widely studied in the context of relational databases and nonmonotonic reasoning (see, for example, (Astrahan, Blasgen, Chamberlin, Eswaran, Gray, Griffiths, King, Lorie, McJones, Mehl, Putzolu, Traiger, Wade, & Watson, 1976; Mumick, Pirahesh, & Ramakrishnan, 1990; Zaniolo, Arni, & Ong, 1993; Agarwal, Agrawal, Deshpande, Gupta, Naughton, Ramakrishnan, & Sarawagi, 1996)). They were first introduced in ASP in the form of weight constraints by Simons, Niemelä and Soininen (Simons, 1999; Niemelä, Simons, & Soininen, 1999). More general

[9] The signature is usually obtained from the program.

aggregates were later considered in (Denecker, Pelov, & Bruynooghe, 2001; Pelov, Denecker, & Bruynooghe, 2004; Faber, Leone, & Pfeifer, 2004; Marek & Truszczynski, 2004), and this was followed by various proposals for defining semantics for programs with aggregates (see, for example, (Ferraris, 2005; Son & Pontelli, 2007; Lee & Meng, 2009)).

The following rule intuitively represents that p is true if there are atleast 8 elements that belong to q:

$$p \leftarrow \#count\{x : q(x)\} \geq 8$$

Here $\#count\{x : q(x)\} \geq 8$ is an aggregate expression involving the *count* aggregate. As another example, consider the following rule involving the *sum* aggregate:

$$p(x) \leftarrow \#sum\{y : q(x, y)\} \leq 5$$

This rule represents that x belongs to p if the sum of all y such that $q(x, y)$ holds is less than or equal to 5.

While most semantics for programs with aggregates agree on the treatment of monotonic aggregates such as $count$, they usually differ with respect to the treatment of nonmonotonic aggregates such as sum. In fact, the need to understand nonmonotonic aggregates under the stable model semantics is one of the main reasons for the several different proposals for semantics of programs with aggregates.

The *choice* construct is another useful construct in ASP, which is used to represent an arbitrary choice for including atoms in the answer set. For example,

consider the following program:

$$\{p(x)\} \leftarrow q(x)$$

$$r \leftarrow p(x)$$

$$q(a)$$

The first rule is a choice rule indicating that for every element in q, arbitrarily choose whether the element belongs to p. Since $q(a)$ holds, the rule generates 2 choices for $p(a)$: one in which it is true and one in which it is false. As a result, the program has 2 answer sets : $\{q(a), p(a), r\}$ and $\{q(a)\}$.

Ferraris (2005) extended the stable model semantics to arbitrary propositional formulas. According to that semantics, answer set programs without variables are a special class of propositional formulas. In that paper, he also proposed semantics for aggregates by turning the aggregates to propositional formulas.

The tools that compute answer sets are referred to as *answer set solvers*. Some of the well-known answer set solvers are SMODELS[10], CMODELS[11], CLASP[12], CLINGO, DLV[13], and ASSAT[14]. SMODELS, CMODELS, and CLASP use the systems LPARSE and GRINGO to ground the input programs. DLV has a grounder built into it, and CLINGO is GRINGO and CLASP combined in a monolithic way. Any of LPARSE, GRINGO, or DLV can be used as a grounder for ASSAT. ASSAT and CMODELS are SAT-based systems, i.e., they turn the output of the grounders into a set of clauses and invoke satisfiability solvers to compute the answer sets. CLASPD is an extension of CLASP to disjunctive programs, and we use this in place of CLASP whenever necessary.

[10] http://www.tcs.hut.fi/Software/smodels/
[11] http://www.cs.utexas.edu/~tag/cmodels/
[12] http://potassco.sourceforge.net/
[13] http://www.dlvsystem.com
[14] http://assat.cs.ust.hk/

2.6 Description Logics and Hybrid Knowledge Bases

Description Logics (DLs), which were originally introduced in order to provide precise semantics for network-based systems such as semantic networks and frame systems (Minsky, 1981), are a family of knowledge representation languages, most of which are decidable fragments of first-order logic. DLs have a wide range of applications but are probably most well-known as the basis for the ontology layer in the semantic web. DLs form the basis for the *Web Ontology Language* (OWL), which is one of the most widely used semantic web languages. OWL today is being extensively used for representing ontologies spanning many different domains. Due to its standard syntax and semantics (based on DLs), it greatly facilitates knowledge sharing across domains. However, the lack of support for nonmonotonic reasoning is a significant limitation of the language. This limitation is well-recognized by the semantic web community, and several approaches have been proposed for integrating nonmonotonic rules and ontologies (DLs). The knowledge base resulting from combining a DL knowledge base with nonmonotonic rules is usually referred to as a *hybrid knowledge base*.

A hybrid knowledge base is a pair $(\mathcal{T}, \mathcal{P})$ where $\mathcal{T}$ is a FOL knowledge base (typically in a description logic) of signature $\Sigma_{\mathcal{T}}$ and $\mathcal{P}$ is a logic program of signature $\Sigma_{\mathcal{P}}$. The existing integration approaches can be classified into three categories: *loose integration*, *tight integration with semantic separation*, and *tight integration under a unifying logic* (Nazarenko, Polo, Eiter, de Bruijn, Schwichtenberg, & Heymans, 2010). In the *loose integration* approach, $\mathcal{T}$ and $\mathcal{P}$ are viewed as separate, independent components, and are connected through minimal safe interfaces for exchanging data (usually in the form of ground atoms). Examples in this category include *nonmonotonic dl-programs* (Eiter, Ianni, Lukasiewicz, Schindlauer, & Tompits, 2008), and the combination of description

logics and defeasible logic (Wang, Billington, Blee, & Antoniou, 2004). In the *tight integration with semantic separation* approach, $\mathcal{T}$ and $\mathcal{P}$ are more tightly integrated, but the predicates in $\Sigma_{\mathcal{T}}$ and $\Sigma_{\mathcal{P}}$ are kept separate. This approach builds an integrated model I as the union of a model $I_{\mathcal{T}}$ of $\mathcal{T}$ and a model $I_{\mathcal{P}}$ of $\mathcal{P}$ with the same domain. Examples in this category are r-hybrid KB (Rosati, 2005), $\mathcal{DL} + log$ (Rosati, 2006), g-hybrid KB (Heymans, de Bruijn, Predoiu, Feier, & Nieuwenborgh, 2008), and f-hybrid KB (Feier & Heymans, 2009). Finally, in the *tight integration under a unifying logic* approach, $\mathcal{T}$ and $\mathcal{P}$ are treated uniformly by translating them into a uniform logic, and there is no principled separation between $\Sigma_{\mathcal{T}}$ and $\Sigma_{\mathcal{P}}$. Examples in this category are Hybrid MKNF KB (Motik & Rosati, 2010), the first-order Autoepistemic Logic based integration (de Bruijn, Eiter, Polleres, & Tompits, 2007a), and the Quantified Equilibrium Logic based integration (de Bruijn, Pearce, Polleres, & Valverde, 2007b). This approach is attractive since it provides a seamless integration of DLs and logic programs, and since the information flow is bi-directional.

Chapter 3

CIRCUMSCRIPTION, FIRST-ORDER STABLE MODEL SEMANTICS, AND THEIR RELATIONSHIP

In this chapter, we present a class of formulas, called *canonical formulas*, on which cirumscription and the stable model semantics coincide. As we will see in the later chapters, canonical formulas are general enough to cover theories in the event calculus, the situation calculus, and TAL. This enables us to turn theories in these action formalisms into the first-order stable model semantics.

The chapter is organized as follows. We first review the definitions of circumscription and the first-order stable model semantics. We then introduce canonical formulas and discuss the related work. Several parts of this chapter are also presented in (Kim, Lee, & Palla, 2009; Lee & Palla, 2010), which contain a stronger definition of canonical formulas.

3.1 Circumscription

We assume the following set of primitive propositional connectives and quantifiers:

$$\bot \text{ (falsity)},\ \wedge,\ \vee,\ \rightarrow,\ \forall,\ \exists\,.$$

We understand $\neg F$ as an abbreviation of $F \rightarrow \bot$; symbol $\top$ stands for $\bot \rightarrow \bot$, and $F \leftrightarrow G$ stands for $(F \rightarrow G) \wedge (G \rightarrow F)$.

Let **p** be a list of distinct predicate constants $p_1, \ldots, p_n$, and let **u** be a list of distinct predicate variables $u_1, \ldots, u_n$. By $\mathbf{u} \leq \mathbf{p}$ we denote the conjunction of the formulas $\forall \mathbf{x}(u_i(\mathbf{x}) \rightarrow p_i(\mathbf{x}))$ for all $i = 1, \ldots n$ where **x** is a list of distinct object variables whose length is the same as the arity of p_i. Expression $\mathbf{u} < \mathbf{p}$ stands for $(\mathbf{u} \leq \mathbf{p}) \wedge \neg(\mathbf{p} \leq \mathbf{u})$. For instance, if p and q are unary predicate constants then $(u, v) < (p, q)$ is

$$\forall x(u(x) \rightarrow p(x)) \wedge \forall x(v(x) \rightarrow q(x)) \wedge \neg\Big(\forall x(p(x) \rightarrow u(x)) \wedge \forall x(q(x) \rightarrow v(x))\Big).$$

Circumscription is defined in terms of the CIRC operator with *minimized* predicates. For any first-order formula F, expression $\mathrm{CIRC}[F; \mathbf{p}]$ stands for the second-order formula

$$F \wedge \neg\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F(\mathbf{u})),$$

where $F(\mathbf{u})$ is the formula obtained from F by substituting u_i for p_i. When F is a sentence (i.e., a formula with no free variables), intuitively, the models of $\mathrm{CIRC}[F; \mathbf{p}]$ are the models of F that are "minimal" on $\mathbf{p}$.

This minimization of certain predicates is what makes the inference under circumscription nonmonotonic. For instance, consider again the sentences "Normally, birds fly." and "Penguins do not fly." These can be represented using the following formula:

$$\forall x(\mathit{bird}(x) \wedge \neg \mathit{ab}(x) \rightarrow \mathit{flies}(x)) \wedge \forall x(\mathit{penguin}(x) \rightarrow \mathit{ab}(x)). \tag{3.1}$$

It follows that

$$\mathrm{CIRC}[(3.1) \wedge \mathit{bird}(\mathit{tweety}); \mathit{penguin}, \mathit{ab}, \mathit{flies}] \models \mathit{flies}(\mathit{tweety}),$$

and

$$\mathrm{CIRC}[(3.1) \wedge \mathit{bird}(\mathit{tweety}) \wedge \mathit{penguin}(\mathit{tweety}); \mathit{penguin}, \mathit{ab}, \mathit{flies}] \models \neg \mathit{flies}(\mathit{tweety}).$$

The definition of circumscription is straightforwardly extended to the case when F is a many-sorted first-order formula (Lifschitz, 1994, Section 2.4), which is the language that the event calculus, the situation calculus, and TAL are based on.

Predicate Completion

Lifschitz (1994) uses the notion of *predicate completion* to characterize circumscription by a first-order formula under certain conditions.

If p is a predicate constant and $F(\mathbf{x})$ is a formula whose only free variables are the ones in $\mathbf{x}$, then replacing the implication $F(\mathbf{x}) \rightarrow p(\mathbf{x})$ with $F(\mathbf{x}) \leftrightarrow p(\mathbf{x})$ is known as *predicate completion*.

Theorem 1 *(Lifschitz, 1994) If $F(\boldsymbol{x})$ does not contain p, then*

$$CIRC[\forall \boldsymbol{x}(F(\boldsymbol{x}) \rightarrow p(\boldsymbol{x})); p]$$

is equivalent to

$$\forall \boldsymbol{x}(F(\boldsymbol{x}) \leftrightarrow p(\boldsymbol{x})).$$

The above result was extended to the case when several predicates are minimized in parallel. A formula F is *positive relative* to a list of predicate constants **p** if every occurrence of every predicate constant $p \in \mathbf{p}$ in F is in the antecedent of an even number of implications. Following is a restricted version of Proposition 7.1.1 from (Lifschitz, 1994).

Theorem 2 *If F is positive relative to $\boldsymbol{p}$, then*

$$CIRC[F; \boldsymbol{p}]$$

is equivalent to

$$\bigwedge_{p \in \boldsymbol{p}} CIRC[F; p].$$

These results are used by the event calculus reasoner DEC reasoner, and the TAL reasoner VITAL for computing circumscription.

3.2 First-Order Stable Model Semantics

This review follows the definition by Ferraris et al. (2011).

We assume the same set of primitive propositional connectives and quantifiers as in the case of circumscription (Chapter 3.1).

The stable models are defined in terms of the SM operator with *intensional* predicates,[1] which is similar to the circumscription operator: For any first-order

[1] The intensional predicates **p** are the predicates that we "intend to characterize" by F, which are analogous to "output" predicates in Datalog; non-intensional (i.e., extensional) predicates are analogous to input predicates in Datalog. Here we use expression $\mathrm{SM}[F; \mathbf{p}]$ in place of $\mathrm{SM}_{\mathbf{p}}[F]$ used in the work of Ferraris et al. (2011).

formula F and any finite list $\mathbf{p} = (p_1, \ldots, p_n)$ of intensional predicates, $\mathrm{SM}[F; \mathbf{p}]$ is defined as

$$F \wedge \neg\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where **u** is defined the same as in $\mathrm{CIRC}[F; \mathbf{p}]$ and $F^*(\mathbf{u})$ is defined recursively as follows:

- $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ for any list **t** of terms;
- $F^* = F$ for any atomic formula F (including $\bot$ and equality) that does not contain members of **p**;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \to G)^* = (F^* \to G^*) \wedge (F \to G)$;
- $(\forall x F)^* = \forall x F^*$;
- $(\exists x F)^* = \exists x F^*$.

When F is a sentence, the models of $\mathrm{SM}[F; \mathbf{p}]$ are called the ***p**-stable* models of F. Intuitively they are the models of F that are "stable" on **p**. We will often simply write $\mathrm{SM}[F]$ in place of $\mathrm{SM}[F; \mathbf{p}]$ when **p** is the list of all predicate constants occurring in F.

According to Lee, Lifschitz, and Palla (2008a), *answer sets* are defined as a special class of stable models as follows. By $\sigma(F)$ we denote the signature consisting of the object, function and predicate constants occurring in F. If F contains at least one object constant, an Herbrand interpretation of $\sigma(F)$ that satisfies $\mathrm{SM}[F]$ is called an *answer set* of F. The answer sets of a logic program Π are defined as the answer sets of the FOL-representation of Π (i.e., the

conjunction of the universal closures of implications corresponding to the rules). For example, the FOL-representation of the program

$$p(a)$$
$$q(b)$$
$$r(x) \leftarrow p(x), \textit{not}\ q(x)$$

is

$$p(a) \wedge q(b) \wedge \forall x(p(x) \wedge \neg q(x) \rightarrow r(x)) \tag{3.2}$$

and $\mathrm{SM}[F]$ is

$$\begin{aligned} & p(a) \wedge q(b) \wedge \forall x(p(x) \wedge \neg q(x) \rightarrow r(x)) \\ & \quad \wedge \neg \exists uvw(((u,v,w) < (p,q,r)) \wedge u(a) \wedge v(b) \\ & \qquad \wedge \forall x((u(x) \wedge (\neg v(x) \wedge \neg q(x)) \rightarrow w(x)) \wedge (p(x) \wedge \neg q(x) \rightarrow r(x)))), \end{aligned}$$

which is equivalent to the first-order sentence

$$\forall x(p(x) \leftrightarrow x = a) \wedge \forall x(q(x) \leftrightarrow x = b) \wedge \forall x(r(x) \leftrightarrow (p(x) \wedge \neg q(x))) \tag{3.3}$$

by completion (presented later in the chapter). The stable models of F are any first-order models of (3.3). The only answer set of F is the Herbrand model $\{p(a),\ q(b),\ r(a)\}$.

Ferraris et al. show that this definition of an answer set, when applied to the syntax of logic programs, is equivalent to the traditional definition of an answer set that is based on grounding and fixpoints (Gelfond & Lifschitz, 1988).

Note that the definition of a stable model is more general than the definition of an answer set in the following ways: stable models are not restricted to Herbrand models, the underlying signature can be arbitrary, and the intensional predicates are not fixed to the list of predicate constants occurring in the formula. The last fact is not essential in view of the following proposition. By *pr*(F) we denote the list of all predicate constants occurring in F; by *Choice*(**p**) we denote

the conjunction of "choice formulas" $\forall \mathbf{x}(p(\mathbf{x}) \vee \neg p(\mathbf{x}))$ for all predicate constants p in **p** where **x** is a list of distinct object variables; by *False*(**p**) we denote the conjunction of $\forall \mathbf{x} \neg p(\mathbf{x})$ for all predicate constants p in **p**. We sometimes identify a list with the corresponding set when there is no confusion.

Proposition 1 *Formula*

$$SM[F; \mathbf{p}] \leftrightarrow SM[F \wedge \mathit{Choice}(\mathit{pr}(F) \setminus \mathbf{p}) \wedge \mathit{False}(\mathbf{p} \setminus \mathit{pr}(F))] \tag{3.4}$$

is logically valid.

For example, if F is $\forall x(q(x) \rightarrow p(x))$ and r is a unary predicate, then $\mathrm{SM}[F; p, r]$ is equivalent to

$$\mathrm{SM}[\forall x(q(x) \rightarrow p(x)) \wedge \forall x(q(x) \vee \neg q(x)) \wedge \forall x \neg r(x)].$$

Notice that the (implicit) intensional predicates on the right-hand side of (3.4) are those in $(\mathit{pr}(F) \cup \mathbf{p})$. The *Choice* formula makes the predicates in $(\mathit{pr}(F) \setminus \mathbf{p})$ to be exempt from the stability checking. On the other hand, the *False* formula makes the predicates in $(\mathbf{p} \setminus \mathit{pr}(F))$ to be stabilized (i.e., to have empty extents), though they do not occur in F.

The language presented so far does not consider strong (a.k.a. classical) negation. Strong negation is useful to represent the notion of a property being false. This is different from the notion of a property not known to be true, which can be represented using default negation $(\neg)$[2]. Typical uses of strong negation include explicit representation of the Closed World Assumption (CWA) and the commonsense law of inertia. For example, the following axioms represent the knowledge that the property *on*(x, y), representing that an object x is on y, is

[2]Note that this symbol is used to represent classical negation in the logic program syntax discussed in Chapter 2.5. However, in formulas under the first-order stable model semantics, this symbol represents default negation.

inertial:

$$\mathit{on}(x, y, t) \wedge \neg \sim \mathit{on}(x, y, t+1) \rightarrow \mathit{on}(x, y, t+1)$$

$$\sim \mathit{on}(x, y, t) \wedge \neg \mathit{on}(x, y, t+1) \rightarrow \sim \mathit{on}(x, y, t+1).$$

Here, '$\sim$' is a symbol for strong negation and t is a time variable. The expression $\sim$*on*(x, y, t) represents that x is not on y at time t. This is different from $\neg$*on*(x, y, t), which represents that x is not known to be on y at time t.

Ferraris et al. (2011) incorporated strong (a.k.a. classical) negation into the stable model semantics by distinguishing between intensional predicates of two kinds, *positive* and *negative*. Each negative intensional predicate has the form $\sim p$, where p is a positive intensional predicate. An interpretation of the underlying signature is *coherent* if the extent of every negative predicate $\sim p$ in it is disjoint from the extent of the corresponding positive predicate p.

According to Ferraris et al. (2011), a formula F is *strongly equivalent* to formula G if, for any formula H containing F as a subformula (and possibly containing object, function and predicate constants that do not occur in F, G), and for any list **p** of distinct predicate constants, $\mathrm{SM}[H; \mathbf{p}]$ is equivalent to $\mathrm{SM}[H'; \mathbf{p}]$, where H' is obtained from H by replacing an occurrence of F by G. In other words, replacing a subformula with a formula that is strongly equivalent to the subformula does not change the stable models of the whole formula. While strongly equivalent theories are classically equivalent, the converse does not necessarily hold. Consequently, classically equivalent transformations do not necessarily preserve stable models. For instance, consider p and $\neg\neg p$. If p is intensional, the former has stable models but the latter does not.

Like the extension of circumscription to many-sorted first-order sentences, the definition of a stable model is straightforwardly extended to many-sorted first-order sentences.

The above extension of the stable model semantics to first-order formulas is essentially the same as the extension presented by Lin and Zhou (2011). These definitions are also equivalent to the definition of Quantified Equilibrium Logic given by Pearce and Valverde (2005), which is defined in terms of the logic of Here-and-There. In the rest of the dissertation, we often use FOSM as an abbreviation of first-order stable model semantics.

Relation to Completion

Similar to circumscription, completion can be used, under certain conditions, to characterize the first-order stable model semantics by a first-order formula.

We say that an occurrence of a predicate constant, or any other subexpression, in a formula F is *positive* if the number of implications containing that occurrence in the antecedent is even, and *negative* otherwise. (Recall that we treat $\neg G$ as shorthand for $G \to \bot$.) We say that the occurrence is *strictly positive* if the number of implications in F containing that occurrence in the antecedent is 0. For instance the occurrence of q in

$$((p \to q) \to r) \to p \tag{3.5}$$

is positive, and the second occurrence of p is strictly positive. Let F be a first-order formula. A *rule* of F is an implication that occurs strictly positively in F. We say that F is negative on a list of predicates **p** if members of **p** have no strictly positive occurrences in F.[3]

The *predicate dependency graph* of F (relative to **p**) is the directed graph that

- has all members of **p** as its vertices, and
- has an edge from p to q if, for some rule $G \to H$ of F,

[3]Note the difference between a formula being negative and an occurrence being negative.

- p has a strictly positive occurrence in H, and
- q has a positive occurrence in G that does not belong to any subformula of G that is negative on **p**.

For instance, the dependency graph of (3.5) relative to $\{p\}$ has no edges since the only rule of the formula is the formula itself and $((p \to q) \to r)$ is negative on $\{p\}$. On the other hand, the dependency graph of the same formula relative to $\{p, r\}$ has an edge from p to r. Though p occurs positively in $((p \to q) \to r)$, there is no edge from p to p since $p \to q$ is negative on $\{p, r\}$.

A formula F is in *Clark normal form* (relative to the list **p** of intensional predicates) if it is a conjunction of sentences of the form

$$\forall \mathbf{x}(G \to p(\mathbf{x})), \tag{3.6}$$

one for each intensional predicate p, where **x** is a list of distinct object variables, and G has no free variables other than those in **x**. The *completion* (relative to **p**) of a formula F in Clark normal form is obtained by replacing each conjunctive term (3.6) with

$$\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow G).$$

The following theorem from (Ferraris et al., 2011) relates SM to completion. We say that F is *tight* on **p** if the predicate dependency graph of F relative to **p** is acyclic.

Theorem 3 *(Ferraris et al., 2011) For any formula F in Clark normal form that is tight on **p**, formula $SM[F; \mathbf{p}]$ is equivalent to the completion of F relative to **p**.*

3.3 Canonical Formulas

Neither the stable model semantics nor circumscription is stronger than the other. For example,

$$\mathrm{CIRC}[\forall x(p(x) \vee \neg p(x));\ p] \tag{3.7}$$

is equivalent to $\forall x \neg p(x)$, and

$$\mathrm{SM}[\forall x(p(x) \vee \neg p(x));\ p] \tag{3.8}$$

is equivalent to $\top$, so that (3.7) is stronger than (3.8). On the other hand,

$$\mathrm{CIRC}[\forall x(\neg p(x) \to q(x));\ p, q] \tag{3.9}$$

is equivalent to $\forall x(\neg p(x) \leftrightarrow q(x))$, and

$$\mathrm{SM}[\forall x(\neg p(x) \to q(x));\ p, q] \tag{3.10}$$

is equivalent to $\forall x(\neg p(x) \wedge q(x))$, so that (3.10) is stronger than (3.9).

Here, we show that the two semantics coincide on a class of formulas called *canonical formulas*, which we define below.

We say that a formula F is *canonical* relative to a list **p** of predicate constants if

- no occurrence of a predicate constant from **p** is in the antecedent of more than one implication in F, and
- every occurrence of a predicate constant from **p** that is in the scope of a strictly positive occurrence of $\exists$ or $\vee$ in F is strictly positive in F.

Example 1 *The formula*

$$\forall x(\neg p(x) \to q(x)) \tag{3.11}$$

that is shown above is not canonical relative to $\{p, q\}$ *since it does not satisfy the first clause of the definition (*p *occurs in the antecedent of two implications as* $\neg p(x)$ *is shorthand for* $p(x) \to \bot$*). On the other hand, the formula is canonical relative to* $\{q\}$*. The formula*

$$\forall x(p(x) \vee \neg p(x)) \tag{3.12}$$

is not canonical relative to $\{p\}$ *since it does not satisfy the second clause; the formula*

$$p(a) \wedge (\exists x\, p(x) \rightarrow \exists x\, q(x)) \tag{3.13}$$

is canonical relative to $\{p, q\}$*, while*

$$p(a, a) \wedge \exists x(p(x, a) \rightarrow p(b, x)) \tag{3.14}$$

is not canonical relative to $\{p, q\}$ *since it does not satisfy the second clause (the second occurrence of* p *is in the scope of a strictly positive occurrence of* $\exists$*, but is not strictly positive in formula (3.14)).*

The following theorem states that, for any canonical formula, circumscription coincides with the stable model semantics.

Theorem 4 *For any canonical formula* F *relative to* $\boldsymbol{p}$*,*

$$CIRC[F; \boldsymbol{p}] \leftrightarrow SM[F; \boldsymbol{p}] \tag{3.15}$$

is logically valid.

For instance, for formula (3.13), which is canonical relative to $\{p, q\}$, formulas $\mathrm{CIRC}[(3.13);\ p, q]$ and $\mathrm{SM}[(3.13);\ p, q]$ are equivalent to each other. Also, any sentence F is clearly canonical relative to $\emptyset$, so that $\mathrm{CIRC}[F; \emptyset]$ is equivalent to $\mathrm{SM}[F; \emptyset]$, which in turn is equivalent to F. On the other hand, such equivalence may not necessarily hold for non-canonical formulas. For instance, we observed that, for formula (3.12) that is not canonical relative to $\{p\}$, formulas (3.7) and (3.8) are not equivalent to each other. For formula (3.11) that is not canonical relative to $\{p, q\}$, formulas (3.9) and (3.10) are not equivalent to each other. We also observe that for formula (3.14) that is not canonical relative to $\{p, q\}$, $\mathrm{CIRC}[(3.14); p, q]$ is not equivalent to $\mathrm{SM}[(3.14); p, q]$: the Herbrand interpretation $\{p(a, a), p(b, a)\}$ satisfies $\mathrm{SM}[(3.14); p, q]$, but does not satisfy $\mathrm{CIRC}[(3.14); p, q]$.

Note that non-canonical formulas can often be equivalently rewritten as canonical formulas. Since any classically equivalent transformation preserves the models of circumscription, Theorem 4 can be applied to such non-canonical formulas, by first rewriting them as canonical formulas. For example, formula (3.11) is equivalent to

$$\forall x(p(x) \vee q(x)), \tag{3.16}$$

which is canonical relative to $\{p, q\}$, so that $\mathrm{CIRC}[(3.11);\ p, q]$ is equivalent to $\mathrm{SM}[(3.16);\ p, q]$. As another example, formula (3.12) is equivalent to

$$\forall x(p(x) \rightarrow p(x)), \tag{3.17}$$

which is canonical relative to $\{p\}$, so that $\mathrm{CIRC}[(3.12); p]$ is equivalent to $\mathrm{SM}[(3.17); p]$. It is clear that this treatment can be applied to any quantifier-free formula (including any propositional formula) because a quantifier-free formula can be equivalently rewritten as a canonical formula by first rewriting it into a clausal normal form and then turning each clause into the form $C \rightarrow D$ where C is a conjunction of atoms and D is a disjunction of atoms.[4]

3.4 Related Work

The relationship between circumscription and the stable model semantics has been well-studied. Proposition 8 from the work of Lee and Lin (2006) shows an embedding of propositional circumscription in logic programs under the stable model semantics. The theorem on canonical formulas is a generalization of this result to the first-order case. Janhunen and Oikarinen (2004) showed another embedding of propositional circumscription in logic programs, and implemented the system CIRC2DLP,[5] but their translation appears quite different from the one

[4] It appears unlikely that knowledge has to be encoded in a non-canonical formula such as (3.8) that cannot be easily turned into an equivalent canonical formula. c.f. "Guide to Axiomatizing Domains in First-Order Logic" (http://cs.nyu.edu/faculty/davise/guide.html). It is not a surprise that all circumscriptive action theories considered in this dissertation satisfy the canonicality assumption.

[5] http://www.tcs.hut.fi/Software/circ2dlp/.

by Lee and Lin (2006). Ferraris et al. (2007, 2011) showed a characterization of the first-order stable model semantics in terms of circumscription, and Lin and Zhou (2011) presented the same via logic of knowledge and justified assumptions (Lin & Shoham, 1992).

Recently, Zhang, Zhang, Ying, and Zhou (2011) showed that circumscription of any first-order formula F can be translated into the first-order stable model semantics. Theorem 3 from that paper is represented as follows.[6]

Theorem 5 *(Zhang et al., 2011, Theorem 3) Let F be a formula in negation normal form and let **p** be a finite list of predicate constants. Let $F^{\neg\neg}$ be the formula obtained from F by replacing every $p(\boldsymbol{t})$ by $\neg\neg p(\boldsymbol{t})$, and let F' be the formula obtained from F by replacing every $\neg p(\boldsymbol{t})$ by $p(\boldsymbol{t}) \rightarrow$ Choice$(\boldsymbol{p})$, where p is in **p** and **t** is a list of terms. Then $CIRC[F; \boldsymbol{p}]$ is equivalent to $SM[F^{\neg\neg} \wedge F'; \boldsymbol{p}]$.*

For example, consider the formula $F = \forall x(p(x) \vee \neg p(x))$. According to the above theorem, $\mathrm{CIRC}[F; p]$ is equivalent to

$$\mathrm{SM}[\forall x(\neg\neg p(x) \vee \neg\neg\neg p(x)) \wedge \forall x\Big(p(x) \vee \big(p(x) \rightarrow (p(x) \vee \neg p(x))\big)\Big); p].$$

Notice that F is not canonical relative to p, and so, Theorem 4 is not applicable to F.

The above theorem provides another insight into the relationship between the two semantics. While the result is applicable to circumscription of any formula, it requires non-trivial transformations. Our result is limited to canonical formulas, but does not require any transformation, and is still general enough to cover useful circumscriptive theories, such as the situation calculus, the event calculus, and TAL. In fact, we expect that most practical circumscriptive theories satisfy the canonicality condition.

[6] This is a bit simpler than the original statement because some redundancy is dropped.

3.5 Proofs

We will first review some results from (Ferraris et al., 2011) which will be used in our proofs.

Theorem 6 *(Ferraris et al., 2011, Theorem 2) For any first-order formula F and any disjoint lists **p**, **q** of distinct predicate constants,*

$$SM[F; \mathbf{p}] \leftrightarrow SM[F \wedge \mathit{Choice}(\mathbf{q});\ \mathbf{p}, \mathbf{q}]$$

is logically valid.

Theorem 7 *(Ferraris et al., 2011, Theorem 3) For any first-order formulas F and G, and any list of distinct predicate constants **p**, $SM[F \wedge \neg G; \mathbf{p}]$ is equivalent to $SM[F; \mathbf{p}] \wedge \neg G$.*

Theorem 8 *(Ferraris et al., 2011, Theorem 4) Let F be any first-order formula, **p** be any list of distinct predicate constants, and p be a member of **p**. If every occurrence of p in F belongs to the antecedent of an implication, then the formula*

$$SM[F; \mathbf{p}] \rightarrow \mathit{False}(p)$$

is logically valid.

Proof of Proposition 1

From Theorem 8, it follows that $\mathrm{SM}[F; \mathbf{p}]$ is equivalent to

$$\mathrm{SM}[F; \mathbf{p}] \wedge \mathit{False}(\mathbf{p} \backslash \mathit{pr}(F)).$$

From Theorem 7, it follows that the above formula is equivalent to

$$\mathrm{SM}[F \wedge \mathit{False}(\mathbf{p} \backslash \mathit{pr}(F)); \mathbf{p}].$$

The result follows from Theorem 6. □

Proof of Theorem 4

In the following, F is a formula, **p** is a list of distinct predicate constants $p_1, \ldots, p_n$, and **u** is a list of distinct predicate variables $u_1, \ldots, u_n$ of the same length as **p**.

Lemma 1 *(Ferraris et al., 2011, Lemma 5) Formula*

$$\boldsymbol{u} \leq \boldsymbol{p} \to (F^*(\boldsymbol{u}) \to F)$$

is logically valid.

Lemma 2 *If every occurrence of every predicate constant from* ***p*** *is strictly positive in* F*,*

$$(\boldsymbol{u} \leq \boldsymbol{p}) \to (F^*(\boldsymbol{u}) \leftrightarrow F(\boldsymbol{u}))$$

is logically valid.

Proof. By induction. We will show only the case when F is $G \to H$. The other cases are straightforward. Consider

$$F^*(\mathbf{u}) = (G^*(\mathbf{u}) \to H^*(\mathbf{u})) \wedge (G \to H).$$

Since every occurrence of predicate constants from **p** in F is strictly positive, G contains no predicate constants from **p**, so that $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$, which is the same as G. Also by I.H., $H^*(\mathbf{u}) \leftrightarrow H(\mathbf{u})$ is logically valid. Therefore it is sufficient to prove that under the assumption $\mathbf{u} \leq \mathbf{p}$,

$$(G \to H(\mathbf{u})) \wedge (G \to H) \leftrightarrow (G \to H(\mathbf{u}))$$

is logically valid. From left to right is clear. Assume $(\mathbf{u} \leq \mathbf{p})$, $G \to H(\mathbf{u})$, and G. We get $H(\mathbf{u})$, which is equivalent to $H^*(\mathbf{u})$ by I.H. By Lemma 1, we conclude H. □

The proof of Theorem 4 is immediate from the following lemma, which can be proved by induction.

Lemma 3 *If F is canonical relative to **p**, then formula*

$$(\boldsymbol{u} \leq \boldsymbol{p}) \wedge F \rightarrow (F^*(\boldsymbol{u}) \leftrightarrow F(\boldsymbol{u}))$$

is logically valid.

Proof.

- F is an atomic formula. Trivial.
- F = $G \wedge H$. Follows from I.H.
- F = $G \vee H$. Assume $(\mathbf{u} \leq \mathbf{p}) \wedge (G \vee H)$. Since $G \vee H$ is canonical relative to **p**, every occurrence of every predicate constant from **p** is strictly positive in G or in H, so that, by Lemma 2, $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$, and $H^*(\mathbf{u})$ is equivalent to $H(\mathbf{u})$.
- $F = G \rightarrow H$. Assume $(\mathbf{u} \leq \mathbf{p}) \wedge (G \rightarrow H)$. It is sufficient to show

  $$(G^*(\mathbf{u}) \rightarrow H^*(\mathbf{u})) \leftrightarrow (G(\mathbf{u}) \rightarrow H(\mathbf{u})). \tag{3.18}$$

  Since $G \rightarrow H$ is canonical relative to **p**, every occurrence of every predicate constant from **p** in G is strictly positive in G, so that, by Lemma 2, $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$.

  - *Case 1:* $\neg G$. By Lemma 1, $\neg G^*(\mathbf{u})$. The claim follows since $\neg G^*(\mathbf{u})$ is equivalent to $\neg G(\mathbf{u})$.
  - *Case 2:* H. By I.H. $H^*(\mathbf{u})$ is equivalent to $H(\mathbf{u})$. The claim follows since $G^*(\mathbf{u})$ is equivalent to $G(\mathbf{u})$.
- F = $\forall x G$. Follows from I.H.
- F = $\exists x G$. Since every occurrence of every predicate constant from **p** in G is strictly positive in G, the claim follows from Lemma 2.

□

Chapter 4

SPLITTING

Splitting in ASP was introduced by Lifschitz and Turner (1994). That paper showed several applications of splitting answer set programs. Among them are the simplification of the computation of answer sets and a simple characterization of locally stratified programs (Przymusinski, 1988). For example, consider the following program:

$$p \leftarrow \textit{not}\ q, t$$

$$q \leftarrow \textit{not}\ t \tag{4.1}$$

$$t \leftarrow \textit{not}\ q \tag{4.2}$$

According to (Lifschitz & Turner, 1994), the above program can be split into two parts, one part consists of the first rule, and the other part consists of the remaining rules. By splitting the program in this way, the answer sets of the program can be computed by first computing the answer sets of the (program consisting of the) last two rules and then using them in the first rule to determine if p holds or not. This kind of splitting is called *top-bottom* splitting, wherein we first evaluate the bottom part of the program and then use its answer sets in evaluating the top part. The answer sets of the last two rules are $\{t\}$ and $\{q\}$. From the first answer set, we get that p should hold and from the second answer set, we get that p should not hold. Thus we get 2 answer sets for the entire program : $\{t, p\}$ and $\{q\}$. Erdoğan and Lifschitz (2004) extended the splitting theorem to programs with nested expressions and showed how it can be used to prove correctness of the same. Oikarinen and Janhunen (2008) generalized the splitting theorem in (Lifschitz & Turner, 1994) by considering certain atoms as *input* atoms.[1] For example, consider the program consisting only of rules (4.1) and (4.2). According

[1] The notion of input atoms is similar to the notion of extensional predicates.

to the splitting theorems in (Lifschitz & Turner, 1994) and (Erdoğan & Lifschitz, 2004), the program cannot be split. However, according to the result in (Oikarinen & Janhunen, 2008), the answer sets of the program are the common answer sets of (4.1) (with t as input) and (4.2) (with q as input). Janhunen, Oikarinen, Tompits, and Woltran (2007) apply similar techniques to split disjunctive programs.

The work on splitting discussed above presents some interesting results with respect to simplifying answer set computation and proving correctness of answer set programs. However, the splitting theorems discussed above cannot be applied to programs with variables and programs with aggregates. Since variables and aggregates are integral parts of the ASP language, generalizing the splitting theorem to handle programs with these constructs will enable us to extend the results discussed above to more general classes of programs.

In this chapter, we present a generalization of the splitting theorem in the framework of the first-order stable model semantics. This generalization enables us to extend the splitting theorem to programs with variables, choice constructs, and count aggregates (Chapter 2.5). This theorem is also used in the later chapters to reformulate the event calculus, the situation calculus, and TAL in ASP, and to integrate DLs and ASP.

The chapter is organized as follows. We first present the splitting lemma followed by the splitting theorem. The splitting lemma, which is about splitting the intensional predicates, is used to prove the splitting theorem, which is about splitting a formula into its conjunctive terms. We then introduce RASPL-1 (Lee et al., 2008a), which is a function-free programming language that allows representation of the count aggregate and the choice construct, and show how the splitting theorem can be applied to programs in this language. Several parts of this chapter are also presented in (Lee et al., 2008a; Ferraris, Lee, Lifschitz, & Palla, 2009).

4.1 Generalization of the Splitting Theorem

For the generalization, we use the notion of *predicate dependency graph* presented in Chapter 3.2. We will denote the predicate dependency graph of F relative to **p** by $\mathrm{DG}_{\mathbf{p}}[F]$.

Splitting Lemma

Here, we present 3 equivalent formulations of the splitting lemma.

Theorem 9 ***[Splitting Lemma, Version 1]*** *Let F be a first-order sentence, and let **p**, **q** be disjoint lists of distinct predicate constants. If each strongly connected component of $DG_{\mathbf{p},\mathbf{q}}[F]$ is a subset of **p** or a subset of **q**, then*

$$SM[F; \mathbf{p}, \mathbf{q}] \quad \textit{is equivalent to} \quad SM[F; \mathbf{p}] \wedge SM[F; \mathbf{q}].$$

Note that the condition on $\mathrm{DG}_{\mathbf{p},\mathbf{q}}[F]$ in the statement of the theorem holds trivially if all strongly connected components of this graph are singletons.

Example 2 *F is $\neg p \wedge r \to q$, **p** is p, **q** is q. In this case, the graph $DG_{\mathbf{p},\mathbf{q}}[F]$ has two vertices p, q, and no edges, so that its strongly connected components are singletons. The splitting lemma asserts that*

$$SM[\neg p \wedge r \to q; p, q] \tag{4.3}$$

is equivalent to the conjunction of

$$SM[\neg p \wedge r \to q; p] \tag{4.4}$$

and

$$SM[\neg p \wedge r \to q; q]. \tag{4.5}$$

Each of these three expressions can be rewritten as a propositional formula using Theorem 3. Formula (4.3) becomes

$$(p \leftrightarrow \bot) \wedge (q \leftrightarrow \neg p \wedge r), \tag{4.6}$$

(4.4) becomes

$$(\neg p \wedge r \to q) \wedge \neg p, \tag{4.7}$$

and (4.5) turns into

$$q \leftrightarrow \neg p \wedge r. \tag{4.8}$$

It is clear that (4.6) is indeed equivalent to the conjunction of (4.7) and (4.8).

Example 3 *F is $r \to p \vee q$, **p** is p, **q** is q. The graph $DG_{\mathbf{p},\mathbf{q}}[F]$ is the same as in Example 2, and the splitting lemma asserts that*

$$SM[r \to p \vee q; p, q] \tag{4.9}$$

is equivalent to the conjunction of

$$SM[r \to p \vee q; p] \tag{4.10}$$

and

$$SM[r \to p \vee q; q]. \tag{4.11}$$

Theorem 3 is not directly applicable to (4.9), but it can be applied to (4.10) and (4.11) by moving the non-intensional predicates to the antecedent. The former is thus equivalent to $p \leftrightarrow \neg q \wedge r$ and the latter to $q \leftrightarrow \neg p \wedge r$. Consequently (4.9) is equivalent to the conjunction of these two formulas.

Example 3 shows that the splitting lemma allows us to expand the power of completion, as a method for describing stable models, to some disjunctive programs. This is similar to the generalization of completion to disjunctive programs described in (Lee & Lifschitz, 2003); the advantage of the splitting lemma is that it is applicable to programs with variables. For instance, using the same argument as in Example 3, we can check that

$$\mathrm{SM}[\forall xy(r(x,y) \to p(x) \vee q(y)); p, q]$$

is equivalent to the conjunction of

$$\forall x(p(x) \leftrightarrow \exists y(\neg q(y) \wedge r(x,y)))$$

and

$$\forall y(q(y) \leftrightarrow \exists x(\neg p(x) \wedge r(x,y))).$$

To illustrate the role of the condition on the predicate dependency graph in the statement of the splitting lemma, take F to be $p \leftrightarrow q$, with p as **p** and q as **q**. The graph $\mathrm{DG}_{\mathbf{p},\mathbf{q}}[F]$ in this case has two edges, from p to q and from q to p. The strongly connected component $\{p, q\}$ of this graph has a common element with **p** and a common element with **q**, so that the splitting lemma is not applicable. Accordingly, the formulas $\mathrm{SM}[p \leftrightarrow q; p, q]$ and $\mathrm{SM}[p \leftrightarrow q; p] \wedge \mathrm{SM}[p \leftrightarrow q; q]$ are not equivalent to each other. Indeed, the former can be rewritten as $\neg p \wedge \neg q$, and each conjunctive term of the latter is equivalent to $p \leftrightarrow q$.

The splitting lemma as stated above can be equivalently reformulated as follows:

Theorem 10 *[**Splitting Lemma, Version 2**] Let F be a first-order sentence, and let **p** be a list of distinct predicate constants. If $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^n$ are all the strongly connected components of $DG_{\boldsymbol{p}}[F]$, then*

$$SM[F; \boldsymbol{p}] \quad \textit{is equivalent to} \quad SM[F; \boldsymbol{c}^1] \wedge \cdots \wedge SM[F; \boldsymbol{c}^n].$$

A *loop* of a first-order formula F (relative to a list **p** of intensional predicates) is a nonempty subset **l** of **p** such that the subgraph of $\mathrm{DG}_{\mathbf{p}}[F]$ induced by **l** is strongly connected. It is clear that the strongly connected components of $\mathrm{DG}_{\mathbf{p}}[F]$ can be characterized as the maximal loops of F. For example, the loops of

$$(p \to q) \wedge (q \to p)$$

relative to $\{p, q\}$ are $\{p\}$, $\{q\}$, and $\{p, q\}$, while the only strongly connected component is the maximal loop $\{p, q\}$.

Theorem 11 *[**Splitting Lemma, Version 3**] Let F be a first-order sentence, and let **p** be a list of distinct predicate constants. If $\mathbf{l}^1, \ldots, \mathbf{l}^n$ are all the loops of F relative to **p** then*

$$SM[F; \mathbf{p}] \quad \textit{is equivalent to} \quad SM[F; \mathbf{l}^1] \wedge \cdots \wedge SM[F; \mathbf{l}^n].$$

The last two versions of the splitting lemma are equivalent to each other in view of the fact that the operator SM is monotone with respect to the intensional predicates **p**: if **p** contains **q** then $\mathrm{SM}[F; \mathbf{p}]$ entails $\mathrm{SM}[F; \mathbf{q}]$.

Splitting Theorem

We now present the splitting theorem, which can be easily proved using the splitting lemma.

Theorem 12 *[**Splitting Theorem**] Let F, G be first-order sentences, and let **p**, **q** be disjoint lists of distinct predicate constants. If*

*(a) each strongly connected component of $DG_{\mathbf{p},\mathbf{q}}[F \wedge G]$ is either a subset of **p** or a subset of **q**,*

*(b) F is negative on **q**, and*

*(c) G is negative on **p***

then

$$SM[F \wedge G;\ \mathbf{p}, \mathbf{q}] \leftrightarrow SM[F;\ \mathbf{p}] \wedge SM[G;\ \mathbf{q}]$$

is logically valid.

Example 4 *F is $(\neg q \rightarrow p)$, G is $(\neg p \rightarrow q)$, **p** is p , and **q** is q. $DG_{\mathbf{p},\mathbf{q}}[F \wedge G]$ has two vertices p and q, and no edges. So condition (a) in the splitting theorem is trivially satisfied. Further, F is negative on q, and G is negative on p. The splitting theorem asserts that*

$$SM[(\neg q \rightarrow p) \wedge (\neg p \rightarrow q); p, q] \tag{4.12}$$

is equivalent to

$$SM[(\neg q \rightarrow p); p] \wedge SM[(\neg p \rightarrow q); q]. \tag{4.13}$$

One can easily verify this by applying completion (Theorem 3) to each of the formulas under SM. As discussed earlier in this chapter, similar result can be obtained by using the splitting theorem by Oikarinen and Janhunen (2008).

4.2 RASPL-1

The language RASPL-1 is based on the observation that the choice construct and the count aggregate can be intuitively represented as first-order formulas, which enables us to straightforwardly use the first-order stable model semantics to provide semantics for programs with these constructs. This implies that we can use the splitting theorem discussed above to split RASPL-1 programs, thus overcoming some of the limitations of the earlier splitting approaches.

RASPL-1 is a function-free programming language that allows representation of the *count* aggregate and the *choice* construct.

In RASPL-1, an *aggregate expression* is an expression of the form

$$b\,\{\mathbf{x} : F_1, \ldots, F_k\} \tag{4.14}$$

$(k \geq 1)$, where b is a positive integer ("the bound"), **x** is a list of variables (possibly empty), and each F_i is an atom possibly preceded by *not*. This expression is similar to the *count* aggregate expression presented in Chapter 2.5 and reads:

there are at least b values of **x** such that $F_1, \ldots, F_k$ hold. Since comma represents conjunction, we can view $F_1, \ldots, F_k$ as a conjunction of literals.

A rule in RASPL-1 is an expression of the form

$$A_1 \,;\ldots;\, A_l \leftarrow\ E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n \tag{4.15}$$

($l \geq 0$; $n \geq m \geq 0$), where each A_i is an atom, each E_i is an aggregate expression, and semi-colon(;) represents disjunction. A program is a finite set of rules. If an aggregate expression E_i in (4.15) has the form $1\{: A\}$, where A is an atom (so that the list of variables in front of the semi-colon is empty) then we will write it as A. If an aggregate expression E_i in (4.15) with $i > m$ has the form $1\{:\ \mathit{not}\ A\}$ then we will write it as *not* A. If E_i in (4.15) with $i > m$ is

$$b\,\{\mathbf{x} : F(\mathbf{x})\}$$

then the term *not* E_i can be written as

$$\{\mathbf{x} : F(\mathbf{x})\}\,b-1$$

Finally, an expression of the form

$$\{A\} \leftarrow\ E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n$$

where A is an atom, stands for

$$A \leftarrow\ E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n, \mathit{not\ not}\ A.$$

The semantics of RASPL-1 is defined by a procedure that turns every aggregate, every rule, and every program into a formula of first-order logic, called its FOL-representation.

The FOL-representation of an aggregate expression $b\,\{\mathbf{x} : F(\mathbf{x})\}$ is the formula

$$\exists \mathbf{x}^1 \cdots \mathbf{x}^b \left[\bigwedge_{1 \leq i \leq b} F(\mathbf{x}^i) \wedge \bigwedge_{1 \leq i < j \leq b} \neg(\mathbf{x}^i = \mathbf{x}^j) \right] \tag{4.16}$$

where $\mathbf{x}^1, \ldots \mathbf{x}^b$ are lists of new variables of the same length as $\mathbf{x}$, and $\mathbf{x} = \mathbf{y}$, where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, stands for $x_1 = y_1 \wedge \cdots \wedge x_n = y_n$. For example, the FOL-representation of $2\{x : p(x), \mathit{not}\ q(x)\}$ is

$$\exists xy((p(x) \wedge \neg q(x)) \wedge (p(y) \wedge \neg q(y)) \wedge x \neq y).$$

The FOL-representation of a RASPL-1 rule *Head* $\leftarrow$ *Body* is the universal closure of the implication *Body* $\rightarrow$ *Head* with each aggregate expression in *Body* replaced by its FOL-representation. The FOL-representation of a RASPL-1 program is the conjunction of the FOL-representations of its rules.

For example, the FOL-representation of the rule

$$p(x); q(y) \leftarrow 2\{x : r(x)\}, s(x, y)$$

is

$$\forall xy(s(x, y) \wedge \exists xy(r(x) \wedge r(y) \wedge x \neq y) \rightarrow p(x) \vee q(y)).$$

For any RASPL-1 program Π containing at least one object constant, an *answer set* of Π is an answer set of the FOL-representation of Π.

RASPL-1 provides a succint representation for many known NP-complete problems. For example, the following RASPL-1 program computes cliques of size greater than or equal to n in a graph G=(V, E):

$$\begin{array}{ll}
\mathit{vertex}(a) & (a \in V), \\
\mathit{edge}(a_i, b_j) & ((a_i, b_j) \in E), \\
\{\mathit{in}(x)\} \leftarrow \mathit{vertex}(x) & \\
\leftarrow \mathit{in}(x),\ \mathit{in}(y),\ \mathit{not}\ \mathit{edge}(x, y),\ \mathit{not}\ x = y & \\
\leftarrow \{x : \mathit{in}(x)\}\, n - 1 &
\end{array} \tag{4.17}$$

The first two rules simply declare all the vertices and edges, and the third rule arbitrarily chooses if a vertex belongs to in or not. Thus the answer sets of the first

three rules of (4.17) are in a 1–1 correspondence with arbitrary sets W of vertices of the graph. The first of the two constraints in (4.17) eliminates the sets W that are not cliques, and the second constraint eliminates the sets that contain fewer than n vertices.

4.3 Splitting RASPL-1 Programs

Since the semantics of RASPL-1 programs is defined in terms of the first-order stable model semantics, the splitting theorem can be easily extended to these programs. Here, we show how the splitting theorem helps us in proving the correctness of the program (4.17) shown above, which computes cliques of size greater than or equal to n in a graph G=(V, E).

Let the program consisting of the last three rules of (4.17) be Π. It follows from the splitting theorem that

$$\mathrm{SM}[\Pi^{FOL}; in] \Leftrightarrow \mathrm{SM}[first^{FOL}; in] \wedge rest^{FOL}$$

where Π^{FOL} is the FOL-representation of Π, $first^{FOL}$ is the FOL-representation of the first rule of Π and $rest^{FOL}$ is the FOL-representation of the remaining rules of Π. Applying completion (Theorem 3) to the theory on the right hand side, it follows that the right hand side is equivalent to

$$\begin{aligned}
&\forall x(\mathit{in}(x) \leftrightarrow (\mathit{vertex}(x) \wedge \mathit{in}(x))) \\
&\wedge \forall xy \neg(\mathit{in}(x) \wedge \mathit{in}(y) \\
&\qquad\qquad \wedge \neg \mathit{edge}(x, y) \wedge x \neq y) \\
&\wedge \exists_n x(\mathit{in}(x)),
\end{aligned}$$

which is in turn equivalent to

$$\begin{aligned}
&\forall x(\mathit{in}(x) \rightarrow \mathit{vertex}(x)) \\
&\wedge \forall xy(\mathit{in}(x) \wedge \mathit{in}(y) \wedge x \neq y \rightarrow \mathit{edge}(x, y)) \\
&\wedge \exists_n x(\mathit{in}(x)).
\end{aligned}$$

If we consider any interpretation I that interprets $vertex$ and $edge$ according to the vertices and edges in the graph, then I satisfies the above formula if and only if in^I represents a clique of size greater than or equal to n.

4.4 Relevance to Later Chapters

Earlier, we briefly mentioned that the splitting theorem is used to reformulate the event calculus, the situation calculus, and TAL in ASP, and to integrate DLs and ASP. We will now discuss a bit more about this by considering the event calculus. As we will see later in Chapter 7.1, an event calculus description is of the form

$$\mathrm{CIRC}[\Sigma\ ;\ \textit{Initiates}, \textit{Terminates}, \textit{Releases}] \wedge \mathrm{CIRC}[\Delta\ ;\ \textit{Happens}]$$
$$\wedge\ \mathrm{CIRC}[\Theta\ ;\ \textit{Ab}_1, \ldots, \textit{Ab}_n] \wedge\ \Xi.$$

Using the result on canonical formulas (Chapter 3.3) and the splitting theorem, we can turn the above formula into

$$\mathrm{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi\ ;\ \textit{Initiates}, \textit{Terminates}, \textit{Releases}, \textit{Happens}, \textit{Ab}_1, \ldots, \textit{Ab}_n].$$

In Chapter 7.3, a further transformation is shown that turns the above formula into an answer set program. Similar approach is used to reformulate the situation calculus and TAL in ASP.

4.5 Proofs

In this section, we present the proofs of the splitting lemma and the splitting theorem.

Proof of the Splitting Lemma (Theorem 11)

We will prove the splitting lemma for a slightly different definition of a predicate dependency graph, and then show how the result can be extended to the definition presented in Chapter 3.2 (which is the one used in the above sections of this chapter).

We say that an occurrence of a predicate constant in a formula is *negated* if it belongs to a subformula of the form $\neg F$, and *nonnegated* otherwise. Recall that

a *rule* of a formula is any strictly positive occurrence of an implication in the formula. For any first-order formula F, the *predicate dependency graph* of F (relative to the list **p** of intensional predicates), represented by $\mathrm{DG}_{\mathbf{p}}[F]$, is the directed graph that

- has all intensional predicates as its vertices, and
- has an edge from p to q if, for some rule $G \to H$ of F,
  - p has a strictly positive occurrence in H, and
  - q has a positive nonnegated occurrence in G.

Lemma 4 below can be easily proved by induction (Ferraris et al., 2011).

Recall that, about a formula F we say that it is *negative* on a list **p** of predicate constants if members of **p** have no strictly positive occurrences in F.

Lemma 4 *If F is negative on **p** then*

$$(\boldsymbol{u} \leq \boldsymbol{p}) \to (F^*(\boldsymbol{u}) \leftrightarrow F)$$

is logically valid.

The following lemma extends Lemma 3 from (Ferraris, Lee, & Lifschitz, 2006) to first-order formulas.

Lemma 5 *Let $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ be disjoint lists of distinct predicate constants, and let $\boldsymbol{u}_1$, $\boldsymbol{u}_2$ be disjoint lists of distinct predicate variables of the same length as $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ respectively.*

(a) If every positive occurrence of every predicate constant from $\boldsymbol{p}_2$ in F is negated then

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \to F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$$

is logically valid.

(b) If every nonpositive occurrence of every predicate constant from $\boldsymbol{p}_2$ *in* F *is negated then*

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \rightarrow F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$$

is logically valid.

Proof. Both parts are proved simultaneously by induction on F. Consider the case when F is $G \rightarrow H$; the other cases are straightforward. Then $F^*(\mathbf{u}_1, \mathbf{u}_2)$ is

$$(G^*(\mathbf{u}_1, \mathbf{u}_2) \rightarrow H^*(\mathbf{u}_1, \mathbf{u}_2)) \wedge (G \rightarrow H). \tag{4.18}$$

(a) Every nonpositive occurrence of every predicate constant from $\mathbf{p}_2$ in G is negated, and so is every positive occurrence of every predicate constant from $\mathbf{p}_2$ in H. By the induction hypothesis, it follows that the formulas

$$((\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge G^*(\mathbf{u}_1, \mathbf{u}_2) \rightarrow G^*(\mathbf{u}_1, \mathbf{p}_2) \tag{4.19}$$

and

$$(\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2) \wedge H^*(\mathbf{u}_1, \mathbf{p}_2) \rightarrow H^*(\mathbf{u}_1, \mathbf{u}_2) \tag{4.20}$$

are logically valid. Assume $(\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)$,

$$(G^*(\mathbf{u}_1, \mathbf{p}_2) \rightarrow H^*(\mathbf{u}_1, \mathbf{p}_2)) \wedge (G \rightarrow H) \tag{4.21}$$

and $G^*(\mathbf{u}_1, \mathbf{u}_2)$. By (4.19), we conclude $G^*(\mathbf{u}_1, \mathbf{p}_2)$. Then, by (4.21), we conclude $H^*(\mathbf{u}_1, \mathbf{p}_2)$. Then, by (4.20), we conclude $H^*(\mathbf{u}_1, \mathbf{u}_2)$. (b) Similar. □

The following assertion is a generalization of Lemma 5 from (Ferraris et al., 2006).

Lemma 6 *Let* $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ *be disjoint lists of distinct predicate constants such that* $DG_{\boldsymbol{p}_1, \boldsymbol{p}_2}[F]$ *has no edges from predicate constants in* $\boldsymbol{p}_1$ *to predicate constants in*

$\boldsymbol{p}_2$, and let $\boldsymbol{u}_1$, $\boldsymbol{u}_2$ be disjoint lists of distinct predicate variables of the same length as $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ respectively. Formula

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \to F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$$

is logically valid.

Proof. By induction on F. Consider the case when F is $G \to H$, so that $F^*(\mathbf{u}_1, \mathbf{u}_2)$ is (4.18); the other cases are straightforward. Assume $(\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)$ and $F^*(\mathbf{u}_1, \mathbf{u}_2)$. Our goal is to prove

$$G^*(\mathbf{u}_1, \mathbf{p}_2) \to H^*(\mathbf{u}_1, \mathbf{p}_2).$$

Assume $G^*(\mathbf{u}_1, \mathbf{p}_2)$. By Lemma 1, the formula

$$((\mathbf{u}_1, \mathbf{p}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge G^*(\mathbf{u}_1, \mathbf{p}_2) \to G \tag{4.22}$$

is logically valid. Consequently, from the assumptions above we can conclude G, and, by (4.18), H. *Case 1:* H is negative on $\mathbf{p}_1$. It follows from Lemma 4 that the formula

$$((\mathbf{u}_1, \mathbf{p}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \to (H^*(\mathbf{u}_1, \mathbf{p}_2) \leftrightarrow H)$$

is logically valid, and we can conclude that $H^*(\mathbf{u}_1, \mathbf{p}_2)$. *Case 2:* H is not negative on $\mathbf{p}_1$, that is to say, H contains a strictly positive occurrence of a predicate constant from $\mathbf{p}_1$. Then every positive occurrence of every predicate constant from $\mathbf{p}_2$ in G is negated, because otherwise there would exist an edge from $\mathbf{p}_1$ to $\mathbf{p}_2$ in $\mathrm{DG}_{\mathbf{p}_1,\mathbf{p}_2}[F]$. By Lemma 5(a), the formula

$$((\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge G^*(\mathbf{u}_1, \mathbf{p}_2) \to G^*(\mathbf{u}_1, \mathbf{u}_2)$$

is logically valid. Consequently from the assumptions above we can conclude that $G^*(\mathbf{u}_1, \mathbf{u}_2)$. By (4.18), it follows that $H^*(\mathbf{u}_1, \mathbf{u}_2)$. Since every edge in $\mathrm{DG}_{\mathbf{p}_1,\mathbf{p}_2}[H]$ belongs to $\mathrm{DG}_{\mathbf{p}_1,\mathbf{p}_2}[F]$, by the induction hypothesis applied to H, the formula

$$((\mathbf{u}_1, \mathbf{u}_2) \leq (\mathbf{p}_1, \mathbf{p}_2)) \wedge H^*(\mathbf{u}_1, \mathbf{u}_2) \rightarrow H^*(\mathbf{u}_1, \mathbf{p}_2)$$

is logically valid. We can thus conclude that $H^*(\mathbf{u}_1, \mathbf{p}_2)$. □

Lemma 7 *For any formula F and any nonempty set Y of intensional predicates, there exists a subset Z of Y such that*

(a) Z is a loop of F, and

(b) the predicate dependency graph of F has no edges from predicate constants in Z to predicate constants in $Y \setminus Z$.

The proof is essentially the same as the proof of Lemma 4 in (Ferraris et al., 2006).

Proof of Version 3 of the Splitting Lemma (Theorem 11). It is sufficient to prove the logical validity of the formula

$$\begin{aligned}
&\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})) \\
&\quad \leftrightarrow \ \exists \mathbf{u}^1((\mathbf{u}^1 < \mathbf{l}^1) \wedge F^*(\widetilde{\mathbf{u}^1})) \\
&\qquad \vee \cdots \vee \exists \mathbf{u}^n((\mathbf{u}^n < \mathbf{l}^n) \wedge F^*(\widetilde{\mathbf{u}^n})),
\end{aligned}$$

where each $\mathbf{u}^i$ is the part of $\mathbf{u}$ that corresponds to the part $\mathbf{l}^i$ of $\mathbf{p}$, and $\widetilde{\mathbf{u}^i}$ is the list of symbols obtained from $\mathbf{p}$ by replacing every intensional predicate p that belongs to $\mathbf{l}^i$ with the corresponding predicate variable u. *Right to left:* Clear. *Left to right:* Assume $\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u}))$ and take $\mathbf{u}$ such that $(\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})$. Consider several cases, each corresponding to a nonempty subset Y of $\mathbf{p}$. The assumption characterizing each case is that $u < p$ for each member p of $\mathbf{p}$ that belongs to Y, and that $u = p$ for each p that does not belong to Y. By Lemma 7, there is a loop $\mathbf{l}^i$ of F that is contained in Y such that the dependency graph has no edges from predicate constants in $\mathbf{l}^i$ to predicate constants in $Y \setminus \mathbf{l}^i$. Since $\mathbf{l}^i$ is contained in Y,

from the fact that $u < p$ for each p in Y we can conclude that

$$\mathbf{u}^i < \mathbf{l}^i. \tag{4.23}$$

Let $\mathbf{u}'$ be the list of symbols obtained from $\mathbf{p}$ by replacing every member p that belongs to Y with the corresponding variable u. Under the assumption characterizing each case, $\mathbf{u} = \mathbf{u}'$, so that $F^*(\mathbf{u}) \leftrightarrow F^*(\mathbf{u}')$. Consequently, we can derive $F^*(\mathbf{u}')$. It follows from Lemma 6 that the formula

$$(\mathbf{u}' \leq \mathbf{p}) \wedge F^*(\mathbf{u}') \rightarrow F^*(\widetilde{\mathbf{u}^i})$$

is logically valid, so that we further conclude that $F^*(\widetilde{\mathbf{u}^i})$. In view of (4.23), it follows that $\exists \mathbf{u}^i((\mathbf{u}^i < \mathbf{l}^i) \wedge F^*(\widetilde{\mathbf{u}^i}))$. □

The splitting lemma can be extended to the predicate dependency graph presented in Chapter 3.2 using the following theorem:

Theorem 13 (Theorem on Double Negations) *Let H be a sentence, F a subformula of H, and H^- the sentence obtained from H by inserting $\neg\neg$ in front of F. If F is contained in a subformula G of H that is negative on **p** then $SM[H^-; \mathbf{p}]$ is equivalent to $SM[H; \mathbf{p}]$.*

Proof. Let G^- be the formula obtained from G by inserting $\neg\neg$ in front of F. By Lemma 4, the formulas

$$\mathbf{u} \leq \mathbf{p} \rightarrow (G^*(\mathbf{u}) \leftrightarrow G)$$

and

$$\mathbf{u} \leq \mathbf{p} \rightarrow ((G^-)^*(\mathbf{u}) \leftrightarrow G^-)$$

are logically valid. Consequently

$$\mathbf{u} \leq \mathbf{p} \rightarrow (G^*(\mathbf{u}) \leftrightarrow (G^-)^*(\mathbf{u}))$$

is logically valid also, and so is

$$\mathbf{u} \leq \mathbf{p} \rightarrow (H^*(\mathbf{u}) \leftrightarrow (H^-)^*(\mathbf{u})).$$

It follows that $\mathrm{SM}[H^-; \mathbf{p}]$ is equivalent to $\mathrm{SM}[H; \mathbf{p}]$. □

Now, let F be any formula, and **p** be any list of intensional predicates. Let F^- be the formula obtained from F by replacing every maximal subformula G of F that is negative on the intensional predicates **p** by $\neg\neg G$. It is clear that the predicate dependency graph of F according to the definition in Chapter 3.2 coincides with the predicate dependency graph of F^- according to the definition above. So, it follows that

$$\mathrm{SM}[F^-; \mathbf{p}] \quad \text{is equivalent to} \quad \mathrm{SM}[F^-; \mathbf{l}^1] \wedge \cdots \wedge \mathrm{SM}[F^-; \mathbf{l}^n].$$

The splitting lemma for the dependency graph presented in Chapter 3.2 follows from Theorem 13 above.

Proof of the Splitting Theorem (Theorem 12)

By the splitting lemma, $\mathrm{SM}[F \wedge G;\ \mathbf{p}, \mathbf{q}]$ is equivalent to

$$\mathrm{SM}[F \wedge G; \mathbf{p}] \wedge \mathrm{SM}[F \wedge G; \mathbf{q}].$$

Since G is negative on **p**, from Lemma 4, it follows that the first conjunctive term can be rewritten as

$$\mathrm{SM}[F;\ \mathbf{p}] \wedge G. \tag{4.24}$$

Similarly, the second conjunctive term can be rewritten as

$$\mathrm{SM}[G;\ \mathbf{q}] \wedge F. \tag{4.25}$$

It remains to observe that the second conjunctive term of each of the formulas (4.24), (4.25) is entailed by the first conjunctive term of the other. □

Chapter 5

REDUCING THEORIES IN THE FIRST-ORDER STABLE MODEL SEMANTICS TO ANSWER SET PROGRAMS

While the first-order stable model semantics is an expressive language, it is undecidable in general. An important question to consider is whether there are certain interesting fragments of the language that can be efficiently computed. In this chapter, we show how answer sets of first-order formulas can be computed using existing answer set solvers. In particular, we present translation F2LP (**F**ormula **t**o **L**ogic **P**rogram) that turns formulas in the first-order stable model semantics, under certain conditions, into the syntax of answer set programs. This translation is also used in the later chapters to reformulate the event calculus, the situation calculus, and TAL in ASP.

The chapter is organized as follows. We first present an approach to eliminate quantifiers in certain formulas under the first-order stable model semantics. We then introduce translation F2LP, that uses the quantifier elimination approach along with the approach in (Cabalar, Pearce, & Valverde, 2005) that turns quantifier-free formulas into logic programs, to turn formulas under the first-order stable model semantics into the syntax of ASP. We then present system F2LP that implements this translation and produces programs that can be processed by the grounders LPARSE and GRINGO, which are front-ends to various answer set solvers such as SMODELS, CLASP(D), CMODELS, etc. Finally, we discuss the related work. Several parts of this chapter are also presented in (Lee & Palla, 2007; Kim et al., 2009; Lee & Palla, 2010).

5.1 Quantifier Elimination

We introduce a quantifier elimination method that distinguishes between two kinds of occurrences of quantifiers: *singular* and *non-singlar*. Any non-singular

occurrence of a quantifier is easy to eliminate, while a singular occurrence is eliminated under a certain assumption.

We say that an occurrence of QxG in F is *singular* if

- Q is $\exists$, and the occurrence of QxG is positive in F, or
- Q is $\forall$, and the occurrence of QxG is negative in F.

For example, the occurrence of $\exists x\, q(x)$ in

$$\exists x\ p(x) \to \exists x\ q(x) \tag{5.1}$$

is singular, but the occurrence of $\exists x\, p(x)$ is not.

Non-singular occurrences of quantifiers can be eliminated in view of the fact that every first-order sentence can be rewritten in prenex form. The prenex form conversion rules given in Section 6.3.1 of Pearce and Valverde (2005) preserve strong equivalence.[1]

Theorem 14 *Every first-order formula is strongly equivalent to a formula in prenex form.*

For example, the formula (5.1) is strongly equivalent to

$$\forall x\ \exists y\ (p(x) \to q(y)). \tag{5.2}$$

As we can see in the above example, the standard prenex form conversion turns a non-singular occurrence of a quantifier into an outermost $\forall$ while preserving strong equivalence. Consequently, if a sentence contains no singular occurrences of quantifiers, then the above result can be used to turn it into a

[1] Pearce and Valverde (2005) show that a sentence in $\mathbf{QN}^{c}_{5}$, the monotonic basis of Quantified Equilibrium Logic, can be turned into prenex form, from which the result follows.

universal sentence. However, in the presence of a singular occurrence of a quantifier, such as $\exists x\ q(x)$ in (5.1), the standard prenex form conversion turns the occurrence into an outermost $\exists$, which is not allowed in logic programs. Below we consider how to handle such occurrences.

Obviously, if the Herbrand universe is finite, and if we are interested in Herbrand stable models (i.e., answer sets) only, quantified formulas can be rewritten as multiple disjunctions and conjunctions. We do not even need to consider turning the formula into prenex form. For example, for a formula

$$r \wedge \neg\exists x(p(x) \wedge q(x)) \rightarrow s \tag{5.3}$$

occurring in a theory whose signature contains $\{1, \ldots, n\}$ as the only object constants (and no other function constants), if we replace $\exists x(p(x) \wedge q(x))$ with multiple disjunctions and then turn the resulting program with nested expression into a usual disjunctive program (Lifschitz et al., 1999), 2^n rules are generated. For instance, if $n = 3$, the resulting logic program is

$$\begin{aligned}
s \leftarrow\ & r, \mathit{not}\ p(1), \mathit{not}\ p(2), \mathit{not}\ p(3)\\
s \leftarrow\ & r, \mathit{not}\ p(1), \mathit{not}\ p(2), \mathit{not}\ q(3)\\
s \leftarrow\ & r, \mathit{not}\ p(1), \mathit{not}\ q(2), \mathit{not}\ p(3)\\
s \leftarrow\ & r, \mathit{not}\ p(1), \mathit{not}\ q(2), \mathit{not}\ q(3)\\
s \leftarrow\ & r, \mathit{not}\ q(1), \mathit{not}\ p(2), \mathit{not}\ p(3)\\
s \leftarrow\ & r, \mathit{not}\ q(1), \mathit{not}\ p(2), \mathit{not}\ q(3)\\
s \leftarrow\ & r, \mathit{not}\ q(1), \mathit{not}\ q(2), \mathit{not}\ p(3)\\
s \leftarrow\ & r, \mathit{not}\ q(1), \mathit{not}\ q(2), \mathit{not}\ q(3).
\end{aligned}$$

However, this translation is not modular as it depends on the underlying domain; the multiple disjunctions or conjunctions need to be updated when the domain changes. More importantly, this method is not applicable if the language contains function constants of positive arity, as its Herbrand universe is infinite.

One may also consider introducing Skolem constants as in first-order logic, presuming that, for any sentence F and its "Skolem form" F' , $\mathrm{SM}[F; \mathbf{p}]$ is satisfiable iff $\mathrm{SM}[F'; \mathbf{p}]$ is satisfiable. However, this method does not work.[2]

Example 5 *For formula*

$$F = (\forall x\, p(x) \to q) \land \neg\neg\exists x(q \land \neg p(x)),$$

$SM[F; q]$ *is equivalent to the first-order sentence*

$$(q \leftrightarrow \forall x\, p(x)) \land \exists x(q \land \neg p(x)),$$

which is unsatisfiable (the equivalence can be established using Theorems 3 and 11 from the work of Ferraris et al., 2011). Formula F *is strongly equivalent to its prenex form*

$$\exists x \exists y\big((p(x) \to q) \land \neg\neg(q \land \neg p(y))\big). \tag{5.4}$$

However, if we introduce new object constants a *and* b *to replace the existentially quantified variables as in*

$$F' = (p(a) \to q) \land \neg\neg(q \land \neg p(b)),$$

formula $SM[F'; q]$ *is equivalent to*

$$(q \leftrightarrow p(a)) \land (q \land \neg p(b)),$$

which is satisfiable.

Here we present a method of eliminating singular occurrences of quantifiers by introducing auxiliary predicates. Our idea is a generalization of the practice in logic programming that simulates negated existential quantification in the body of a

[2] Pearce and Valverde (2005) show that Skolemization works with $\mathbf{QN}_5^c$, the monotonic basis of Quantified Equilibrium Logic, but as our example shows, this does not imply that Skolemization works with Quantified Equilibrium Logic.

rule by introducing auxiliary predicates. For instance, in order to eliminate $\exists$ in (5.3), we will introduce a new predicate constant p', and turn (5.3) into

$$(r \wedge \neg p' \rightarrow s) \wedge \forall x(p(x) \wedge q(x) \rightarrow p'), \tag{5.5}$$

which corresponds to the logic program

$$\begin{aligned} s \ &\leftarrow \ r, \mathit{not}\, p' \\ p' \ &\leftarrow \ p(x), q(x). \end{aligned} \tag{5.6}$$

The models of $\mathrm{SM}[(5.3);\ p, q, r, s]$ are the same as those of $\mathrm{SM}[(5.5);\ p, q, r, s, p']$ if we disregard p'. This method does not involve grounding, so that the translation does not depend on the domain and is not restricted to Herbrand models. The method is formally justified by the following proposition.

Proposition 2 *Let F be a sentence of a signature σ, let **p** be a finite list of distinct predicate constants, and let q be a new predicate constant that does not belong to σ. Consider any non-strictly positive occurrence of $\exists y G(y, \boldsymbol{x})$ in F that is contained in a subformula of F that is negative on **p**, where **x** is the list of all free variables of $\exists y G(y, \boldsymbol{x})$. Let F' be the formula obtained from F by replacing that occurrence with $q(\boldsymbol{x})$. Then*

$$SM[F; \boldsymbol{p}] \wedge \forall \boldsymbol{x}(q(\boldsymbol{x}) \leftrightarrow \exists y G(y, \boldsymbol{x}))$$

is equivalent to

$$SM[F' \wedge \forall \boldsymbol{x} y(G(y, \boldsymbol{x}) \rightarrow q(\boldsymbol{x}));\ \boldsymbol{p}, q].$$

Proposition 2 tells us that $\mathrm{SM}[F; \mathbf{p}]$ and $\mathrm{SM}[F' \wedge \forall \mathbf{x} y(G(y, \mathbf{x}) \rightarrow q(\mathbf{x}); \mathbf{p}, q]$ have the same models if we disregard the new predicate q. Notice that F' does not retain the occurrence of $\exists y$.

Example 6 *In formula (5.3), $\exists x(p(x) \wedge q(x))$ is contained in a negative formula (relative to any set of intensional predicates). In accordance with Proposition 2,*

$SM[(5.3);\ p, q, r, s]$ *has the same models as* $SM[(5.5); p, q, r, s, p']$ *if we disregard* p'.

Any singular occurrence of a formula $\forall y G(y, \mathbf{x})$ that is *contained in a subformula of* F *that is negative on* ***p*** [3] can also be eliminated using Proposition 2 by first rewriting $\forall y G(y, \mathbf{x})$ as $\neg\exists y \neg G(y, \mathbf{x})$. Note that $\forall y G(y, \mathbf{x})$ is not strongly equivalent to $\neg\exists y \neg G(y, \mathbf{x})$, and in general the transformation may not necessarily preserve stable models. But the condition that $\forall y G(y, \mathbf{x})$ is negative on **p** ensures that it preserves **p**-stable models.

Now we are ready to present our quantifier elimination method. We say that a formula F is *almost universal relative to* ***p*** if every singular occurrence of $Qx G$ in F is contained in a subformula of F that is negative on **p**. For example, formula (5.3) is almost universal relative to any set of predicates because the only singular occurrence of $\exists x(p(x) \wedge q(x))$ in it is contained in $\neg\exists x(p(x) \wedge q(x))$, which is negative on any list of predicates. Formula F in Example 5 is almost universal relative to $\{q\}$ because the singular occurrence of $\forall x\, p(x)$ is contained in the formula itself, which is negative on $\{q\}$, and the singular occurrence of $\exists x(q \wedge \neg p(x))$ is contained in $\neg\exists x(q \wedge \neg p(x))$, which is also negative on $\{q\}$.

The following procedure can be used to eliminate all (possibly nested) quantifiers in any almost universal sentence.

Definition 1 (Translation ELIM-QUANTIFIERS) *Given a formula* F*, first prepend* $\neg\neg$ *to every maximal strictly positive occurrence of a formula of the form* $\exists y H(y, \boldsymbol{x})$,[4] *and then repeat the following until there are no occurrences of quantifiers remaining: Select a maximal occurrence of a formula of the form* $Qy G(y, \boldsymbol{x})$ *in* F *where* Q *is* $\forall$ *or* $\exists$*, and* ***x*** *is the list of all free variables in* $Qy G(y, \boldsymbol{x})$.

[3] Recall the definition of a *negative formula on* ***p*** given in Chapter 4.

[4] The maximality is in terms of the subformula relation. That is, here we select a strictly positive occurrence of a formula of the form $\exists y H(y, \mathbf{x})$ that is not a strict subformula of the same form.

(a) *If the occurrence of* $QyG(y, \boldsymbol{x})$ *in* F *is non-singular in* F*, then set* F *to be the formula obtained from* F *by replacing the occurrence of* $QyG(y, \boldsymbol{x})$ *with* $G(z, \boldsymbol{x})$ *where* z *is a new variable.*

(b) *If* Q *is* $\exists$ *and the occurrence of* $QyG(y, \boldsymbol{x})$ *in* F *is positive, then set* F *to be*

$$F' \wedge (G(y, \boldsymbol{x}) \rightarrow p_G(\boldsymbol{x}))$$

where p_G *is a new predicate constant and* F' *is the formula obtained from* F *by replacing the occurrence of* $QyG(y, \boldsymbol{x})$ *with* $p_G(\boldsymbol{x})$*.*

(c) *If* Q *is* $\forall$ *and the occurrence of* $QyG(y, \boldsymbol{x})$ *in* F *is negative, then set* F *to be the formula obtained from* F *by replacing the occurrence of* $QyG(y, \boldsymbol{x})$ *with* $\neg\exists y\neg G(y, \boldsymbol{x})$*.*

We assume that the new predicate constants introduced by the translation do not belong to the signature of the input formula F. It is clear that this process terminates, and yields a formula that is quantifier-free. Since the number of times step (b) is applied is no more than the number of quantifiers in the input formula, and the new formulas added have the size polynomial to the input formula, it follows that the size of the resulting quantifier-free formula is polynomial in the size of the input formula.

The following theorem tells us that any almost universal sentence F can be turned into the form $\forall \mathbf{x} G$ where G is a quantifier-free formula. For any (second-order) sentences F and G of a signature and any subset σ of that signature, we say that F *is* σ*-equivalent to* G, denoted by $F \Leftrightarrow_\sigma G$, if the class of models of F restricted to σ is identical to the class of models of G restricted to σ.

Theorem 15 *Let* F *be a sentence of a signature* σ*, let* F' *be the universal closure of the formula obtained from* F *by applying translation ELIM-QUANTIFIERS, and*

let ***q*** *be the list of new predicate constants introduced by the translation. If* F *is almost universal relative to* ***p****, then* $SM[F; \mathbf{p}]$ *is* σ*-equivalent to* $SM[F'; \mathbf{p}, \mathbf{q}]$*.*

The statement of the theorem becomes incorrect if F is not required to be almost universal relative to **p**. For instance, if ELIM-QUANTIFIERS is applied to $\exists x\, p(x)$, it results in $\neg\neg q \wedge (p(x) \rightarrow q)$. However, $\mathrm{SM}[\exists x\, p(x);\ p]$ is not $\{p\}$-equivalent to $\mathrm{SM}[\forall x(\neg\neg q \wedge (p(x) \rightarrow q));\ p, q]$. The former means that p is a singleton. The latter is equivalent to $q \wedge \forall x \neg p(x) \wedge (q \leftrightarrow \exists x p(x))$, which is inconsistent.

5.2 Turning Quantifier-Free Formulas into the Syntax of Logic Programs

Cabalar et al. (2005) present two transformations to turn arbitrary propositional formulas into logic programs: one is vocabulary-preserving and the other introduces new atoms but is polynomial. Following is the vocabulary-preserving transformation shown in Section 3 of that paper.

- Left side rules:

$$\top \wedge F \rightarrow G \quad \mapsto \quad \{F \rightarrow G\} \tag{L1}$$

$$\bot \wedge F \rightarrow G \quad \mapsto \quad \emptyset \tag{L2}$$

$$\neg\neg F \wedge G \rightarrow H \quad \mapsto \quad \{G \rightarrow \neg F \vee H\} \tag{L3}$$

$$(F \vee G) \wedge H \rightarrow K \quad \mapsto \quad \left\{ \begin{array}{l} F \wedge H \rightarrow K \\ G \wedge H \rightarrow K \end{array} \right\} \tag{L4}$$

$$(F \rightarrow G) \wedge H \rightarrow K \quad \mapsto \quad \left\{ \begin{array}{l} \neg F \wedge H \rightarrow K \\ G \wedge H \rightarrow K \\ H \rightarrow F \vee \neg G \vee K \end{array} \right\} \tag{L5}$$

- Right side rules:

$$F \to \bot \vee G \quad \mapsto \quad \{F \to G\} \tag{R1}$$

$$F \to \top \vee G \quad \mapsto \quad \emptyset \tag{R2}$$

$$F \to \neg\neg G \vee H \quad \mapsto \quad \{\neg G \wedge F \to H\} \tag{R3}$$

$$F \to (G \wedge H) \vee K \quad \mapsto \quad \left\{ \begin{array}{l} F \to G \vee K \\ F \to H \vee K \end{array} \right\} \tag{R4}$$

$$F \to (G \to H) \vee K \quad \mapsto \quad \left\{ \begin{array}{l} G \wedge F \to H \vee K \\ \neg H \wedge F \to \neg G \vee K \end{array} \right\} \tag{R5}$$

Before applying this transformation to each formula on the lefthand side, it is assumed that the formula is already written in *negation normal form*, in which negation is applied to atoms only, by using the following transformation:

- Negation normal form conversion:

$$\begin{array}{lcl} \neg\top & \mapsto & \bot \\ \neg\bot & \mapsto & \top \\ \neg\neg\neg F & \mapsto & \neg F \\ \neg(F \wedge G) & \mapsto & \neg F \vee \neg G \\ \neg(F \vee G) & \mapsto & \neg F \wedge \neg G \\ \neg(F \to G) & \mapsto & \neg\neg F \wedge \neg G \end{array}$$

According to Cabalar et al. (2005), successive application of the rewriting rules above turn any propositional formula into a disjunctive logic program. This result can be simply extended to turn any quantifier-free formula into a logic program.

As noted by Cabalar et al. (2005), this translation may involve an exponential blowup in size, and Theorem 1 from their paper shows that indeed

there is no vocabulary-preserving polynomial time algorithm to convert general propositional theories under the stable model semantics into disjunctive logic programs. Alternatively, one can use another translation from the same paper, which is linear in size but involves auxiliary atoms and is more complex.

5.3 F2LP: Turning Formulas in the First-Order Stable Model Semantics into the Syntax of ASP

Using the translations discussed in the previous sections, we introduce translation F2LP that turns an almost universal formula into a logic program.

Definition 2 ***[Translation F2LP]***

1. *Given a formula F and a list of intensional predicates **p**, apply translation ELIM-QUANTIFIERS (Definition 1) to F;*
2. *Add choice formulas $(q(\boldsymbol{x}) \vee \neg q(\boldsymbol{x}))$ for all non-intensional predicates q.*
3. *Turn the resulting quantifier-free formula into a logic program by applying the translation from (Cabalar et al., 2005, Section 3), which was reviewed in Chapter 5.2.*

Due to the third step, this transformation may involve an exponential blowup in size. One can obtain a polynomial time translation by replacing Step 3 with an alternative translation given in (Cabalar et al., 2005, Section 4).

The following theorem asserts the correctness of translation F2LP.

Theorem 16 *Let F be a sentence of a signature σ, let **p** be a list of intensional predicates, and let F' be the FOL representation of the program obtained from F by applying translation F2LP with **p** as intensional predicates. If F is almost universal relative to **p**, then $SM[F; \boldsymbol{p}]$ is σ-equivalent to*

$$SM[F' \wedge \mathit{False}(\boldsymbol{p} \setminus \mathit{pr}(F'))].$$

Example 7 *Consider one of the domain independent axioms in the discrete event calculus (DEC5 axiom):*

$$\begin{array}{l} \mathit{HoldsAt}(f,t) \wedge \neg \mathit{ReleasedAt}(f,t{+}1) \wedge \\ \qquad \neg\exists e(\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t)) \rightarrow \mathit{HoldsAt}(f,t{+}1). \end{array} \tag{5.7}$$

Step 1 of translation F2LP introduces the formula

$$\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t) \rightarrow \mathit{q}(f,t),$$

and replaces (5.7) with

$$\mathit{HoldsAt}(f,t) \wedge \neg \mathit{ReleasedAt}(f,t{+}1) \wedge \neg \mathit{q}(f,t) \rightarrow \mathit{HoldsAt}(f,t{+}1).$$

Step 3 turns these formulas into rules

$$\mathit{q}(f,t) \leftarrow \mathit{Happens}(e,t),\ \mathit{Terminates}(e,f,t)$$

$$\mathit{HoldsAt}(f,t{+}1) \leftarrow \mathit{HoldsAt}(f,t), \mathit{not\ ReleasedAt}(f,t{+}1),\ \mathit{not\ q}(f,t).$$

System F2LP

System F2LP is an implementation of translation F2LP, which turns a first-order formula into the languages of LPARSE and GRINGO. The system can be downloaded from its home page

http://reasoning.eas.asu.edu/f2lp/ .

First-order formulas can be encoded in F2LP using the following ASCII representation for the quantifiers and connectives.

Symbol	$\neg$	$\sim$	$\wedge$	$\vee$	$\rightarrow$	$\bot$	$\top$	$\forall xyz$	$\exists xyz$
ASCII	`-`	`~`	`&`	`|`	`->`	`false`	`true`	`![X,Y,Z]:`	`?[X,Y,Z]:`

The system also allows extended rule form $F \leftarrow G$ where F and G are first-order formulas. In this case, `not` is used to represent $\neg$, `-` is used to represent $\sim$, and `<-` is used to represent $\leftarrow$.

The usual LPARSE and GRINGO rules (which have rule arrow `:-`) are also allowed in F2LP. Such rules are simply copied to the output. The program returned by F2LP can be passed to ASP grounders and solvers that accept LPARSE and GRINGO languages.

For example, formula $\exists xy\, p(x,y) \rightarrow \forall z\, (\neg q(z) \wedge r(z))$ can be represented as

```
?[X,Y]: p(X,Y) ->  ![Z]: (-q(Z) & r(Z)).
```

Here, the dot(.) indicates the end of the formula. Each formula should end with a dot(.).

Example 8 *Consider F =*

$$\forall x\, p(x,y) \rightarrow \forall x\, q(x).$$

where p is extensional. Here y is assumed to be universally quantified. This is represented as

```
![X]: p(X,Y) -> ![X]: q(X).
#extensional p(X,Y).
```

in F2LP syntax. F2LP turns this formula into

```
q(_NV_1) :- not _new_pred_1(Y).
{p(X,Y)}.
_new_pred_1(Y) :- not p(X,Y).
```

Here, `_NV_1` and `_new_pred_1` are the new variable and new predicate introduced respectively. Further, "`{p(X,Y)}.`" is the choice rule introduced since p is extensional.

Example 9 *The input*

```
![X]: p(X,Y) -> ![X]: q(X) & r.
#extensional p(X,Y).
```

is turned to

```
q(_NV_1) :- not _new_pred_1(Y).
r :- not _new_pred_1(Y).
{p(X,Y)}.
_new_pred_1(Y) :- not p(X,Y).
```

Example 10 *The input*

```
p(X,Y) -> -q(X) | X = Y.
```

is turned to

```
 :- X!=Y,{not q(X)}0,p(X,Y).
```

This example shows how F2LP handles negation and equality in the head. F2LP turns negation in the head to double negation in the body (according to LPARSE and GRINGO languages, `{not q(X)}0` represents `not not q(X)`. This is the same as in RASPL-1 (Chapter 4.2)).

In addition to usual first-order formulas, F2LP also allows the choice construct and *aggregate formulas* (Lee & Meng, 2009). Aggregate formulas are

formulas built from normal atomic formulas (including equality) and aggregate expressions using the standard connectives and quantifiers in first-order logic. In other words, aggregate expressions are treated as atomic formulas. F2LP currently supports both GRINGO and DLV aggregates. A detailed manual of F2LP can be found from its homepage.

Example 11 *The input*

```
s(Y) & not 2{t(X,Z,Z):s(Z)} -> {p(X):q(X)} & r(Y).
#sum[t(X,Y,Z):r(Z):q(Y),s(X)]5 | s(X) -> {p(X)}.
#count{X:p(X,Y)} >= 5 | s(Y) -> q(Y).
```

is turned to

```
{p(X):q(X)} :- s(Y),not 2{t(X,Z,Z):s(Z)}.
r(Y) :- s(Y),not 2{t(X,Z,Z):s(Z)}.
{p(X)} :- #sum[t(X,Y,Z):r(Z):q(Y),s(X)]5.
{p(X)} :- s(X).
q(Y) :- #count{X:p(X,Y)}>=5.
q(Y) :- s(Y).
```

As an example of the extended rule form representation, consider the following input

```
![X]: q(X) & r <- not ![X]: p(X,Y).
#extensional p(X,Y).
```

When the above input is given to F2LP, it produces the following output:

```
q(_NV_2) :- not p(_NV_1,Y).
```

```
r :- not p(_NV_1,Y).
{p(X,Y)}.
```

where `_NV_1` and `_NV_2` are the new variables introduced.

5.4 Related Work

The idea of the translation F2LP is similar to the one independently given by Cabalar (2009) that eliminates existential quantifiers in the scope of negation in the body of a rule. The difference is that our translation applies to the larger class of almost universal formulas and also differentiates between intensional and extensional predicates. Zhang et al. (2011) introduce a translation that turns arbitrary first-order formulas into logic programs, but this work is limited to finite structures only. On the other hand, our translation works for almost universal formulas only, but is not limited to finite structures.

5.5 Relevance to Later Chapters

Translation F2LP is used in the later chapters to reformulate the event calculus, the situation calculus, and TAL in ASP. While the complete reformulations are provided in the respective chapters on each of the formalisms, we will briefly discuss the reformulation of the event calculus here. As we will see in Chapter 7.2, an event calculus description can be turned into the following formula by using the result on canonical formulas (Chapter 3.3) and the splitting theorem (Chapter 4.1):

$$\mathrm{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi\ ;\ \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}, \mathit{Ab}_1, \ldots, \mathit{Ab}_n].$$

Translation F2LP is used to turn the above formula into the syntax of ASP. As a result, system F2LP can be used to compute event calculus descriptions using answer set solvers.

Proof of Theorem 14

One can verify this by checking that for any formula F, and the formula F' obtained from F by applying one of the prenex form conversion rules given in Section 6.3.1 of (Pearce & Valverde, 2005), formula

$$\mathbf{u} \leq \mathbf{p} \rightarrow (F^*(\mathbf{u}) \leftrightarrow F'^*(\mathbf{u}))$$

is logically valid where **p** is any list of predicate constants and **u** is the corresponding list of predicate variables. □

Proof of Proposition 2

Lemma 8 *(Ferraris et al., 2011, Lemma 6) Formula*

$$\boldsymbol{u} \leq \boldsymbol{p} \rightarrow ((\neg F)^*(\boldsymbol{u}) \leftrightarrow \neg F)$$

is logically valid.

Lemma 9 *Let F be a formula, let **p** be a list of distinct predicate constants, let G be a subformula of F and let G' be any formula that is classically equivalent to G. Let F' be the formula obtained from F by substituting G' for G. If the occurrence of G is in a subformula of F that is negative on **p** and the occurrence of G' is in a subformula of F' that is negative on **p**, then*

$$SM[F; \boldsymbol{p}] \leftrightarrow SM[F'; \boldsymbol{p}]$$

is logically valid.

Proof. Let $F^{\neg\neg}$ be the formula obtained from F by prepending $\neg\neg$ to G, and let $(F')^{\neg\neg}$ be the formula obtained from F' by prepending $\neg\neg$ to G'. By the Theorem

on Double Negations (Theorem 13), the following formulas are logically valid.

$$\mathrm{SM}[F; \mathbf{p}] \leftrightarrow \mathrm{SM}[F^{\neg\neg}; \mathbf{p}],$$
$$\mathrm{SM}[F'; \mathbf{p}] \leftrightarrow \mathrm{SM}[(F')^{\neg\neg}; \mathbf{p}].$$

From Lemma 8, it follows that

$$(\mathbf{u} \leq \mathbf{p} \wedge (G \leftrightarrow G')) \rightarrow ((F^{\neg\neg})^*(\mathbf{u}) \leftrightarrow ((F')^{\neg\neg})^*(\mathbf{u}))$$

is logically valid, where **u** is a list of predicate variables corresponding to **p**. Consequently,

$$\mathrm{SM}[F^{\neg\neg}; \mathbf{p}] \leftrightarrow \mathrm{SM}[(F')^{\neg\neg}; \mathbf{p}]$$

is logically valid. □

Proof of Proposition 2. In formula

$$\mathrm{SM}[F' \wedge \forall \mathbf{x} y(G(y, \mathbf{x}) \rightarrow q(\mathbf{x})); \mathbf{p}, q], \tag{5.8}$$

clearly, F' is negative on q and $\forall \mathbf{x} y(G(y, \mathbf{x}) \rightarrow q(\mathbf{x}))$ is negative on **p**. Let H be any subformula of F that is negative on **p** and contains the occurrence of $\exists y G(y, \mathbf{x})$. Consider two cases.

- Case 1: the occurrence of $\exists y G(y, \mathbf{x})$ in H is not strictly positive. Thus the dependency graph (Chapter 3.2) of $F' \wedge \forall \mathbf{x} y(G(y, \mathbf{x}) \rightarrow q(\mathbf{x}))$ relative to $\{\mathbf{p}, q\}$ has no incoming edges into q.
- Case 2: the occurrence of $\exists y G(y, \mathbf{x})$ in H is strictly positive. Since H is negative on **p**, $\exists y G(y, \mathbf{x})$ is negative on **p** as well, so that the dependency graph of $F' \wedge \forall \mathbf{x} y(G(y, \mathbf{x}) \rightarrow q(\mathbf{x}))$ relative to $\{\mathbf{p}, q\}$ has no outgoing edges from q.

Therefore, every strongly connected component in the dependency graph belongs to either **p** or $\{q\}$. Consequently, by Theorem 12, (5.8) is equivalent to

$$\mathrm{SM}[F'; \mathbf{p}] \wedge \mathrm{SM}[\forall \mathbf{x} y(G(y, \mathbf{x}) \to q(\mathbf{x})); q] \tag{5.9}$$

Since $\exists y G(y, \mathbf{x})$ is negative on q, formula $\forall \mathbf{x} y(G(y, \mathbf{x}) \to q(\mathbf{x}))$ is tight on $\{q\}$. By Theorem 3, (5.9) is equivalent to

$$\mathrm{SM}[F'; \mathbf{p}] \wedge \forall \mathbf{x}(\exists y G(y, \mathbf{x}) \leftrightarrow q(\mathbf{x})). \tag{5.10}$$

By Lemma 9, it follows that (5.10) is equivalent to

$$\mathrm{SM}[F; \mathbf{p}] \wedge \forall \mathbf{x}(\exists y G(y, \mathbf{x}) \leftrightarrow q(\mathbf{x})).$$

Consequently, the claim follows. □

Proof of Theorem 15

It is clear that the algorithm terminates and yields a quantifier-free formula K. We will prove that $\mathrm{SM}[F; \mathbf{p}]$ is σ-equivalent to $\mathrm{SM}[\forall \mathbf{x} K; \mathbf{p} \cup \mathbf{q}]$ where **x** is the list of all (free) variables of K.

Let $F^{\neg\neg}$ be the formula obtained from the initial formula F by prepending double negations in front of every maximal strictly positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$. Since F is almost universal relative to **p**, such an occurrence is in a subformula of F that is negative on **p**. Thus by the Theorem on Double Negations (Theorem 13), $\mathrm{SM}[F; \mathbf{p}]$ is equivalent to $\mathrm{SM}[F^{\neg\neg}; \mathbf{p}]$. Note that $F^{\neg\neg}$ contains no strictly positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$.

For each iteration, let us assume that the formula before the iteration is

$$H_0 \wedge \cdots \wedge H_n$$

where H_0 is transformed from $F^{\neg\neg}$ by the previous iterations, and each H_i $(i > 0)$ is a formula of the form $G(\mathbf{x}, y) \to p_G(\mathbf{x})$ that is introduced by Step (b). Initially H_0

is $F^{\neg\neg}$ and $n = 0$. Let $\mathbf{r}_0$ be $\mathbf{p}$, and let $\mathbf{r}_i$ be each p_G for H_i ($i > 0$). By induction we can prove that

(i) every positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$ in H_i is not strictly positive, and is in a subformula of H_i that is negative on $\mathbf{r}_i$;

(ii) every negative occurrence of formulas of the form $\forall y G(\mathbf{x}, y)$ in H_i is in a subformula of H_i that is negative on $\mathbf{r}_i$.

We will prove that if Step (a) or Step (c) is applied to turn H_k into H'_k, then

$$\mathrm{SM}[\forall \mathbf{x}_0 H_0; \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}_n H_n; \mathbf{r}_n] \tag{5.11}$$

is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}'_0 H'_0; \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}'_n H'_n; \mathbf{r}_n] \tag{5.12}$$

where $H'_j = H_j$ for all j different from k, and $\mathbf{x}_i$ ($i \geq 0$) is the list of all free variables of H_i, and $\mathbf{x}'_i$ ($i \geq 0$) is the list of all free variables of H'_i.

Indeed, Step (a) is a part of prenex form conversion, which preserves strong equivalence (Theorem 14). So it is clear that (5.11) is equivalent to (5.12).

When Step (c) is applied to turn (5.11) into (5.12), since $\forall y H(\mathbf{x}, y)$ is in a subformula of H_k that is negative on $\mathbf{r}_k$, the equivalence between (5.11) and (5.12) follows from Lemma 9.

When Step (b) is applied to turn H_k into H'_k and introduces a new conjunctive term H'_{n+1}, formula (5.11) is $(\sigma, \mathbf{r}_1, \ldots, \mathbf{r}_n)$-equivalent to

$$\mathrm{SM}[\forall \mathbf{x}'_0 H'_0; \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}'_n H'_n; \mathbf{r}_n] \wedge \mathrm{SM}[\forall \mathbf{x}'_{n+1} H'_{n+1}; \mathbf{r}_{n+1}] \tag{5.13}$$

by Proposition 2 due to condition (i).

Let

$$H''_0 \wedge \cdots \wedge H''_m \tag{5.14}$$

be the final quantifier-free formula where H_0'' is transformed from $F^{\neg\neg}$. By the induction, it follows that $\mathrm{SM}[F; \mathbf{p}]$ is σ-equivalent to

$$\mathrm{SM}[\forall \mathbf{x}_0'' H_0'';\ \mathbf{r}_0] \wedge \cdots \wedge \mathrm{SM}[\forall \mathbf{x}_m'' H_m'';\ \mathbf{r}_m], \tag{5.15}$$

where each $\mathbf{x}_i''$ $(0 \leq i \leq m)$ is the list of all free variables of H_i''.

Since every non-strictly positive occurrence of new predicate $\mathbf{r}_i$ $(i > 0)$ in any H_j'' $(0 \leq j \leq m)$ is positive, there is no incoming edge into $\mathbf{r}_i$ in the dependency graph of (5.14) relative to $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_m$. Consequently, every strongly connected component of the dependency graph belongs to one of $\mathbf{r}_i$ $(i \geq 0)$. Moreover, it is clear that each H_i'' $(i \geq 0)$ is negative on every $\mathbf{r}_j$ for $j \neq i$. (In the case of H_0'', recall that the occurrence of $\mathbf{r}_j$ for any $j > 0$ is not strictly positive since $F^{\neg\neg}$, from which H_0'' is obtained, contains no strictly positive occurrence of formulas of the form $\exists y G(\mathbf{x}, y)$.) Thus by the splitting theorem (Theorem 12), formula (5.15) is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}_0'' H_0'' \wedge \cdots \wedge \forall \mathbf{x}_m'' H_m'';\ \mathbf{r}_0 \cup \cdots \cup \mathbf{r}_m]. \tag{5.16}$$

□

Proof of Theorem 16

We use the notations introduced in the proof of Theorem 15. By Theorem 15, $\mathrm{SM}[F; \mathbf{p}]$ is σ-equivalent to (5.16) and, by Theorem 6, (5.16) is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}_0'' H_0'' \wedge \cdots \wedge \forall \mathbf{x}_m'' H_m'' \wedge \mathit{Choice}(\sigma^{pred} \setminus \mathbf{p});\ \sigma^{pred} \cup \mathbf{r}_1 \cup \cdots \cup \mathbf{r}_m] \tag{5.17}$$

($\mathbf{r}_0$ is $\mathbf{p}$) where σ^{pred} is the set of all predicate constants in signature σ. It follows from Proposition 3 from (Cabalar et al., 2005) that (5.17) is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}_0'' H_0''' \wedge \cdots \wedge \forall \mathbf{x}_m'' H_m''' \wedge \mathit{Choice}(\sigma^{pred} \setminus \mathbf{p});\ \sigma^{pred} \cup \mathbf{r}_1 \cup \cdots \cup \mathbf{r}_m] \tag{5.18}$$

where H_i''' is obtained from H_i'' by applying the translation from (Cabalar et al., 2005, Section 3) (reviewed in Chapter 5.2) that turns a quantifier-free formula into

a set of rules. It is easy to see that F' is the same as the formula

$$\forall \mathbf{x}_0'' H_0''' \wedge \cdots \wedge \forall \mathbf{x}_m'' H_m''' \wedge \mathit{Choice}(\sigma^{pred} \setminus \mathbf{p})$$

and $\sigma^{pred} \cup \mathbf{r}_1 \cup \cdots \cup \mathbf{r}_m$ is the same as $\mathbf{p} \cup \mathit{pr}(F')$, so that (5.18) can be written as

$$\mathrm{SM}[F';\ \mathbf{p} \cup \mathit{pr}(F')],$$

which is equivalent to

$$\mathrm{SM}[F' \wedge \mathit{False}(\mathbf{p} \setminus \mathit{pr}(F'))].$$

by Proposition 1. □

Chapter 6

SAFETY

The safety property was studied in datalog as a condition under which the set of answers for a given query is finite (see, for example, (Ullman, 1985; Zaniolo, 1986; Ramakrishnan, Bancilhon, & Silberschatz, 1987; Krishnamurthy, Ramakrishnan, & Shmueli, 1996)). Safety thus guarantees the decidability of query answering. The study of the safety property has been carried over to ASP (see, for example, (McCain & Turner, 1994)) and it plays an important role in the design of answer set solvers. According to the traditional definition of safety in ASP, a rule in a program is *safe* if every variable occurring in it also occurs in the positive part of the body. A program is safe if all the rules in it are safe. Answer set solvers accept only safe rules as input. There are a couple of reasons for this.

The first reason is mainly from a semantic point of view but is also related to decidability. Consider the following unsafe program:

$$p(x) \leftarrow \textit{not}\ q(y)$$

$$q(a)$$

This program is not safe since the first rule is not safe (x and y do not occur in the positive body). Recall that the traditional way to compute the answer sets of programs with variables involves grounding the program with the Herbrand universe of the signature obtained from the program. In this case, the signature obtained from the program is $\{p/1, q/1, a\}$ and the program has one answer set: $\{q(a)\}$. However, if we extend the signature by an object constant b and consider the additional object constant for grounding, we get $\{q(a), p(a), p(b)\}$ as the answer set of the resulting ground program. This implies that answer sets of a program with variables actually depend on the signature of the program. However, we only consider the signature that can be obtained from the program in order to

compute the answer sets. Safety actually justifies this choice of the signature since the answer sets of a safe program depend only on the constants that occur in the program. This in turn implies that safety not only enables domain-independent reasoning but also guarantees decidability in the absence of function constants.

The second reason is from the point of view of computational efficiency. The safety property is used in the answer set solvers to reduce the size of the ground program and also to enhance the speed of grounding. For example, consider the following safe program:

$$p(x) \leftarrow \textit{not}\ q(y), r(x, y)$$
$$r(a, b)$$

The ground program generated by answer set solvers does not include rules where x is substituted with b and y is substituted with a (in the absence of $r(x, y)$ in the first rule, both x and y need to be substituted with a and b).

The definition of safety can be straightforwardly extended to disjunctive programs, and Bria, Faber, and Leone (2008) extended safety to a special class of programs with nested expressions (Lifschitz et al., 1999) called *Normal Form Nested Programs*. However, these extensions are not general enough to cover programs with aggregates. Though there has been some work done in extending safety to programs with aggregates, these extensions impose more stringent conditions than necessary. For example, the following safe rule is not covered by the definition of safety in (Faber et al., 2004):

$$p(x) \leftarrow \#count\{y : q(x, y)\} \geq 2 \tag{6.1}$$

In this chapter, we present a generalization of safety to formulas under the first-order stable model semantics, and show how the generalization can be used to define safety for RASPL-1 programs (Chapter 4.2), which allow representation

of the count aggregate. We discuss the related work at the end of the chapter. Several parts of this chapter are also presented in (Lee et al., 2008a; Lee, Lifschitz, & Palla, 2008b, 2009).

6.1 Generalization of Safety

We consider first-order formulas that may contain object constants and equality but no function constants of arity > 0. The definition of a safe formula generalizes the safety condition to arbitrary sentences in prenex form. The assumption that the formula is in prenex form is not a significant limitation in view of Theorem 14 (Chapter 5.1), which implies that all steps involved in the standard process of converting a formula to prenex form are strongly equivalent transformations.

To every quantifier-free formula F we assign a set $\mathrm{RV}(F)$ of its *restricted variables* as follows:[1]

- For an atomic formula F,
  - if F is an equality between two variables then $\mathrm{RV}(F) = \emptyset$;
  - otherwise, $\mathrm{RV}(F)$ is the set of all variables occurring in F;
- $\mathrm{RV}(\bot) = \emptyset$;
- $\mathrm{RV}(F \wedge G) = \mathrm{RV}(F) \cup \mathrm{RV}(G)$;
- $\mathrm{RV}(F \vee G) = \mathrm{RV}(F) \cap \mathrm{RV}(G)$;
- $\mathrm{RV}(F \to G) = \emptyset$.

We say that a variable x is restricted in F if x belongs to $\mathrm{RV}(F)$. For instance, consider a rule of the form

$$A \leftarrow A_1, \ldots, A_n, \mathit{not}\ A_{n+1}, \ldots, \mathit{not}\ A_m \tag{6.2}$$

[1] Some clauses of this definition are similar to parts of the definition of an allowed formula in (Topor & Sonenberg, 1988).

which can be viewed as

$$A_1 \wedge \cdots \wedge A_n \wedge \neg A_{n+1} \wedge \cdots \wedge \neg A_m \rightarrow A.$$

A variable x is restricted in the body of the rule iff it occurs in the positive body.

Consider a sentence F in prenex form:

$$Q_1 x_1 \cdots Q_n x_n M \tag{6.3}$$

(each Q_i is $\forall$ or $\exists$; $x_1, \ldots, x_n$ are distinct variables; the matrix M is quantifier-free). We say that F is *semi-safe* if every occurrence of each of the variables x_i in M is contained in a subformula $G \rightarrow H$ where x_i is restricted in G. For example, consider

$$\forall x \exists y (p(x) \wedge \neg q(y) \rightarrow r(x)). \tag{6.4}$$

This formula is semi-safe since both the occurrences of x in the matrix belong to the matrix itself (which is an implication) and x is restricted in the antecedent of the matrix, and the only occurrence of y in the matrix belongs to the implication $q(y) \rightarrow \bot$[2] and y is restricted in the antecedent of the implication. We identify a formula containing free variables with its universal closure so that a rule (6.2) is semi-safe iff every variable occurring in the head also occurs in the positive body. Formula

$$\forall x \exists y (p(x) \wedge \neg q(y) \rightarrow r(x, y)) \tag{6.5}$$

is not semi-safe since the strictly positive occurrence of y in the matrix is not restricted in antecedent of the matrix ($\mathrm{RV}(\neg q(x)) = \emptyset$).

The definition of a *safe formula* adds an additional restriction to the definition of a semi-safe formula. We say a sentence (6.3) is *safe* if every occurrence of each of the variables x_i in M is contained in a subformula $G \rightarrow H$ that satisfies two conditions:

[2]Recall that we treat $\neg F$ as shorthand for $F \rightarrow \bot$.

(a) the subformula is positive in M if Q_i is $\forall$, and negative in M if Q_i is $\exists$; and

(b) x_i is restricted in G.

Consider the sentence (6.4). This sentence is safe because of the following reasons. Both the occurrences of x in the matrix belong to the matrix itself (which is positive in itself) and x is restricted in the antecedent of the matrix. The only occurrence of y in the matrix belongs to the implication $q(y) \to \bot$, which is negative in the matrix, and y is restricted in the antecedent of the implication.

Now, consider a rule (6.2). If every variable occurs in the positive body, then every occurrence of every variable belongs to the rule and every variable is restricted in the body. So, such a rule is safe. On the other hand, consider a variable that occurs only in the negative body. Clearly, that variable is not restricted in the body. So, any rule with such a variable is not safe. So, a rule (6.2) is safe iff every variable occurring in it also occurs in the positive body.

Similarly, one can check that

$$\forall x(\neg p(x) \vee \neg q(x) \to r)$$

is semi-safe but not safe.

Since the prenex form transformation is strongly equivalent, we can consider a formula to be safe if its prenex form is safe. For instance, the formula

$$p(a) \wedge \forall x(p(x) \wedge \neg\exists y q(y) \to r(x)) \tag{6.6}$$

is safe since

$$\forall x\exists y(p(a) \wedge (p(x) \wedge \neg q(y) \to r(x)))$$

is safe.

In the remaining part of the section, we present some interesting properties of safe sentences, which ensure domain-independent and decidable reasoning.

The Small Predicate Property

We say that a stable model of a sentence F has the *small predicate property* if, for every predicate constant p_i, if the relation represented by it holds for a tuple of arguments, then each member of the tuple is represented by an object constant occurring in F. To make this idea precise, we will use the following notation: for any finite set **c** of object constants, $\mathit{in}_{\mathbf{c}}(x_1, \ldots, x_m)$ stands for the formula

$$\bigwedge_{1 \leq j \leq m} \bigvee_{c \in \mathbf{c}} x_j = c.$$

The small predicate property can be expressed by the conjunction of the sentences

$$\forall \mathbf{x}(p_i(\mathbf{x}) \rightarrow \mathit{in}_{\mathbf{c}}(\mathbf{x}))$$

for all predicate constants p_i occurring in F, where **x** is a list of distinct variables. We will denote this sentence by $\mathit{SPP}_{\mathbf{c}}$. By $c(F)$ we denote the set of all object constants occurring in F.

Proposition 3 *For any semi-safe sentence F, $SM[F]$ entails $\mathit{SPP}_{c(F)}$.*

For instance, if F is (6.6), then the proposition asserts that $\mathrm{SM}[F]$ entails

$$\forall x(p(x) \rightarrow x = a) \wedge \forall x(q(x) \rightarrow x = a).$$

Characterizing the Stable Models of a Safe Sentence

Proposition 3 seems to suggest that the stable models of a safe sentence F are closely related to the stable models of the sentence obtained by grounding F with the object constants occurring in it. In order to show that this is indeed the case, we define $\mathrm{Ground}_{\mathbf{c}}[F]$ for any safe sentence F in prenex form as follows. If F is quantifier-free, then $\mathrm{Ground}_{\mathbf{c}}[F]$ is F. Otherwise,

$$\mathrm{Ground}_{\mathbf{c}}[\forall x F(x)] = \bigwedge_{c \in \mathbf{c}} \mathrm{Ground}_{\mathbf{c}}[F(c)],$$
$$\mathrm{Ground}_{\mathbf{c}}[\exists x F(x)] = \bigvee_{c \in \mathbf{c}} \mathrm{Ground}_{\mathbf{c}}[F(c)].$$

Theorem 17 *For any safe sentence F and any nonempty finite set **c** of object constants containing $c(F)$, $SM[Ground_{\mathbf{c}}[F]]$ is equivalent to $SM[F]$.*

This theorem asserts that the stable models of a safe sentence F are the stable models of the variable-free sentence obtained by grounding F with respect to the object constants occurring in F. This result also establishes the domain-independence of the answer sets of safe formulas, i.e, adding object constants to the signature obtained from the formula does not change the Herbrand stable models. Further, the above theorem also enables us to characterize the stable models of a safe sentence by a propositional formula under the unique name assumption (UNA).

For example, consider the safe formula (6.6). The above theorem asserts that $\mathrm{SM}[F; p, q, r]$ is equivalent to

$$\mathrm{SM}[p(a) \wedge (p(a) \wedge \neg q(a) \rightarrow r(a)); p, q, r].$$

Since there is only one object constant in F, UNA holds by default. So, by treating the formula under SM as a propositional formula, it follows from the relationship of SM to completion (Theorem 3, Chapter 3.2) that the models of the above formula can be represented by the models of the propositional formula

$$p(a) \wedge \neg q(a) \wedge (p(a) \wedge \neg q(a) \leftrightarrow r(a)).$$

So, questions such as "does $\mathrm{SM}[F; pr(F)] \models \forall x \neg q(x)$?" can be answered by simply computing the models of the above propositional formula. In other words,

entailment checking w.r.t safe formulas reduces to the problem of propositional satisfiability.

We now show that the stable models of a safe sentence can be characterized by a first-order sentence.

Theorem 18 *For every safe sentence F there exists a variable-free formula G such that $SM[F]$ is equivalent to $G \wedge SPP_{c(F)}$.*

Extending a Stable Model

In the beginning of the chapter, we mentioned that the answer sets of a safe program do not change upon adding object constants to the signature obtained from the program. This result is extended to safe formulas by Theorem 17. Here, we prove a similar result by considering first-order stable models instead of answer sets.

Let I be an interpretation of a set of object and predicate constants, and let X be a superset of the universe of I. By the *extension of I to X* we mean the interpretation of the same constants with the universe X such that each object constant represents the same object under both interpretations, and each predicate constant represents the same set of tuples.

Theorem 19 *For any safe sentence F, any interpretation I of the object and predicate constants from F, and any superset X of the universe of I, the extension of I to X is a stable model of F iff I is a stable model of F.*

6.2 Safety for RASPL-1

In view of the reductive semantics of RASPL-1 (Chapter 4.2), this definition of safety can be used to define safety for RASPL-1 programs, thus overcoming some of the limitations with the earlier definitions and contributing to the design of

answer set solvers. We adapt the definition of safety provided above to RASPL-1 programs as follows.

Recall that an RASPL-1 program is a finite set of rules of the form

$$A_1 \; ; \ldots ; \; A_l \leftarrow \; E_1, \ldots, E_m, \textit{not}\; E_{m+1}, \ldots, \textit{not}\; E_n \tag{6.7}$$

where each A_i is an atom, semi-colon(;) represents disjunction, and each E_i is an aggregate expression of the form

$$b\,\{\mathbf{x} : F_1, \ldots, F_k\} \tag{6.8}$$

$(k \geq 1)$, where b is a positive integer ("the bound"), **x** is a list of variables (possibly empty), and each F_i is an atom possibly preceded by *not*. Since comma (,) represents conjunction, we can view (6.8) as

$$b\,\{\mathbf{x} : F\} \tag{6.9}$$

where F stands for $F_1 \wedge \cdots \wedge F_k$.

We say that an aggregate expression $b\{\mathbf{x} : F\}$ is *allowed* if every member of **x** is restricted in F. For instance, $2\{x : p(x, y)\}$ is allowed; $2\{x : p(y)\}$ and $2\{x : \textit{not}\; p(x, y)\}$ are not allowed.

We say that a variable v is *restricted* in an aggregate expression $b\{\mathbf{x} : F\}$ if v is restricted in F and does not belong to **x**. For instance, y is restricted in $2\{x : p(x, y)\}$ and in $2\{x : p(y)\}$, but is not restricted in $2\{x : \textit{not}\; p(x, y)\}$.

A variable v is *free* in a rule (6.7) if

- v occurs in the head $A_1 \; ; \ldots ; \; A_l$ of the rule, or
- the body $E_1, \ldots, \textit{not}\; E_n$ of the rule contains an aggregate expression $b\{\mathbf{x} : F\}$ such that v occurs in F and does not belong to **x**.

A rule (6.7) is *safe* if

- each aggregate expression in its body is allowed, and
- each of its free variables is restricted in one of the aggregate expressions $E_1, \ldots, E_m$.

A RASPL-1 program is *safe* if each of its rules is safe. For example, the RASPL-1 program (4.17) that computes cliques of size greater than or equal to n is safe and is accepted by some answer set solvers. However, any program containing (6.1)[3] or the rule[4]

$$q \leftarrow \mathit{not}\ 1\{\mathit{not}\ p(x)\}$$

is not accepted by answer set solvers as they consider them unsafe. On the other hand, the definition of safety presented here asserts that these rules are safe and answer set solvers can be extended to accept such rules.

As an example of an unsafe rule, consider

$$p(x) \leftarrow 2\{x : q(x)\}.$$

This rule is not safe since the free variable x is not restricted in $2\{x : q(x)\}$.

The following theorem states that the answer sets of a safe RASPL-1 program do not change upon adding object constants to the signature obtained from the program.

Theorem 20 *Let Π be a safe RASPL-1 program containing at least one object constant, and let F be its FOL-representation. For any signature σ obtained by adding object constants to $\sigma(F)$, an Herbrand interpretation of σ satisfies $SM[F]$ iff it is an answer set of Π.*

[3] This is not a RASPL-1 rule but can be straightforwardly rewritten as one.

[4] Such rules are quite useful when we want to compute answer sets of arbitrary formulas.

6.3 Related Work

The definition of safety by Cabalar, Pearce, and Valverde (2009) generalizes our definition provided above. We review here a slightly different version of their definition. We say that a formula (6.3) is *CPV-semi-safe* if every strictly positive occurrence of each of the variables x_i in M belongs to a subformula $G \to H$ where x_i is restricted in G. It is easy to see that any sentence that is semi-safe according to our definition is also CPV-semi-safe. For example, formula (6.4), which is semi-safe according to our definition, is also CPV-semi-safe. Since the definition of a CPV-semi-safe formula imposes restrictions on only the strictly positive occurrences of variables, any formula in which no variable occurs strictly positively is trivially CPV-semi-safe. For example, the formula

$$q(x) \vee r(y) \to r$$

is CPV-semi-safe, but is not semi-safe according to our definition. Formula (6.5) is not semi-safe according to both the definitions since the strictly positive occurrence of y in the matrix is not restricted in antecedent of the matrix ($\mathrm{RV}(\neg q(x)) = \emptyset$).

Following (Cabalar et al., 2009), we define the following transformations.

- $\neg\bot \mapsto \top, \quad \neg\top \mapsto \bot,$
- $\bot \wedge F \mapsto \bot, \quad F \wedge \bot \mapsto \bot, \quad \top \wedge F \mapsto F, \quad F \wedge \top \mapsto F,$
- $\bot \vee F \mapsto F, \quad F \vee \bot \mapsto F, \quad \top \vee F \mapsto \top, \quad F \vee \top \mapsto \top,$
- $\bot \to F \mapsto \top, \quad F \to \top \mapsto \top, \quad \top \to F \mapsto F.$

We say that a variable x is *positively weakly restricted* in a formula G if the formula obtained from G by

- first replacing every atomic formula A in it such that x is restricted in A by $\bot$,

- and then applying the transformations above

is $\top$. Similarly, we say that x is *negatively weakly restricted* in G if the formula obtained from G by the same procedure is $\bot$.

We say that a CPV-semi-safe sentence (6.3) is *CPV-safe* if, for every occurrence of each of the variables x_i in the matrix M,

(a) if Q_i is $\forall$, then the occurrence belongs to

  – a positive subformula of (6.3) in which x_i is positively weakly restricted, or

  – a negative subformula of (6.3) in which x_i is negatively weakly restricted;

(b) if Q_i is $\exists$, then the occurrence belongs to

  – a negative subformula of (6.3) in which x_i is positively weakly restricted, or

  – a positive subformula of (6.3) in which x_i is negatively weakly restricted.

It follows that if a sentence is safe according to our definition, it is also CPV-safe. For, example, consider the sentence (6.4), which is safe according to our definition. This sentence is CPV-safe because of the following reasons. By replacing $p(x)$ and $r(x)$ by $\bot$, the matrix reduces to $\top$ upon applying the transformations shown above. This implies that x is positively weakly restricted in the matrix, which is a positive subformula of (6.4). Also, the variable y, which is existentially quantified, is clearly negatively weakly restricted in $q(y)$, which is a positive subformula of (6.4). On the other hand, a CPV-safe sentence may not be safe according to our definition. For example, formula

$$\neg p(x) \rightarrow \neg q(x)$$

is CPV-safe but not safe according to our definition.

One can check that

$$\exists x \forall y((p(x) \to q(y)) \to r)$$

and

$$\exists x(\neg p(x) \to q)$$

are CPV-safe, but

$$\forall x(\neg p(x) \vee \neg q(x) \to r)$$

is CPV-semi-safe but not CPV-safe.

6.4 Proofs

Proof of Proposition 3

The notation that we use in the proof involves *predicate expressions* of the form

$$\lambda \mathbf{x} F(\mathbf{x}), \tag{6.10}$$

where $F(\mathbf{x})$ is a formula. If e is (6.10) and $G(p)$ is a formula containing a predicate constant p of the same arity as the length of **x** then $G(e)$ stands for the result of replacing each atomic part of the form $p(\mathbf{t})$ in $G(p)$ with $F(\mathbf{t})$, after renaming the bound variables in $G(p)$ in the usual way, if necessary. For instance, if $G(p)$ is $p(a) \vee p(b)$ then $G(\lambda y(x = y))$ is $x = a \vee x = b$. Substituting a tuple **e** of predicate expressions for a tuple **p** of predicate constants is defined in a similar way.

For any finite set **c** of object constants, by $\mathbf{e}_\mathbf{c}$ we denote the list of predicate expressions

$$\lambda \mathbf{x}(p_i(\mathbf{x}) \wedge \mathit{in}_\mathbf{c}(\mathbf{x}))$$

for all predicate constants p_i.

The following lemma can be proved by induction on F.

Lemma 10 *For any quantifier-free formula F and any finite set **c** of object constants containing $c(F)$,*

$$F^*(\boldsymbol{e_c}) \to \mathit{in}_{\mathbf{c}}(RV(F))$$

is logically valid.

About a variable x occurring in a quantifier-free formula F we say that it is *safe* in F if every occurrence of x in F belongs to a subformula $G \to H$ such that x is restricted in G. It is clear that a sentence in prenex form is semi-safe iff all variables in its matrix are safe. By $\mathrm{NS}(F)$ we will denote the set of the variables of F that are *not* safe.

Lemma 11 *For any quantifier-free formula F and any finite set **c** of object constants containing $c(F)$,*

$$(F \wedge \mathit{in}_{\mathbf{c}}(NS(F))) \to F^*(\boldsymbol{e_c}) \tag{6.11}$$

is logically valid.

Proof. By induction on F. We only consider the case when F is $G \to H$; the other cases are straightforward. By the induction hypothesis,

$$(H \wedge \mathit{in}_{\mathbf{c}}(\mathrm{NS}(H))) \to H^*(\mathbf{e_c}) \tag{6.12}$$

is logically valid. By Lemma 1,

$$G^*(\mathbf{e_c}) \to G \tag{6.13}$$

is logically valid. By Lemma 10,

$$G^*(\mathbf{e_c}) \to \mathit{in}_{\mathbf{c}}(\mathrm{RV}(G)) \tag{6.14}$$

is logically valid. Assume the antecedent of (6.11)

$$(G \to H) \wedge \mathit{in}_{\mathbf{c}}(\mathrm{NS}(G \to H)). \tag{6.15}$$

Assume $G^*(\mathbf{e_c})$; our goal is to derive $H^*(\mathbf{e_c})$. By (6.13), G; by the first conjunctive term of (6.15), H. By (6.14),

$$\mathit{in}_{\mathbf{c}}(\mathrm{RV}(G)). \tag{6.16}$$

Note that $\mathrm{NS}(H) \subseteq \mathrm{NS}(G \rightarrow H) \cup \mathrm{RV}(G)$. Consequently, from the second conjunctive term of (6.15), and (6.16),

$$\mathit{in}_{\mathbf{c}}(\mathrm{NS}(H)). \tag{6.17}$$

From H, (6.17) and (6.12), $H^*(\mathbf{e_c})$. □

Lemma 12 *For any semi-safe sentence F, F entails $F^*(\boldsymbol{e_c})$.*

Proof. Immediate from Lemma 11.

□

Proof of Proposition 3 Assume F and $\neg \mathit{SPP}_{c(F)}$; we will derive

$$\exists \mathbf{u}(\mathbf{u} < \mathbf{p} \wedge F^*(\mathbf{u})).$$

To this end, we will prove

$$(\mathbf{e}_{c(F)} < \mathbf{p}) \wedge F^*(\mathbf{e}_{c(F)}).$$

By Lemma 12, it is sufficient to prove the first conjunctive term, that is,

$$\begin{aligned} \bigwedge_{p \in \mathbf{p}} \Big(\forall \mathbf{x} \Big(p(\mathbf{x}) \wedge \mathit{in}_{c(F)}(\mathbf{x}) \rightarrow \mathit{p}(\mathbf{x}) \Big) \Big) \\ \wedge \neg \bigwedge_{p \in \mathbf{p}} \forall \mathbf{x} \Big(p(\mathbf{x}) \rightarrow \Big(p(\mathbf{x}) \wedge \mathit{in}_{c(F)}(\mathbf{x}) \Big) \Big). \end{aligned} \tag{6.18}$$

The first conjunctive term of (6.18) is logically valid, and the second is equivalent to $\neg \mathit{SPP}_{c(F)}$. □

Proof of Theorem 17

As in (Lifschitz, Pearce, & Valverde, 2007), by **INT**$^=$ we denote intuitionistic predicate logic with equality, and DE stands for the decidable equality axiom

$$x = y \vee x \neq y.$$

The importance of the logical system **INT**$^=$ + DE is determined by the fact that it is a part of **SQHT**$^=$ ("*s*tatic *q*uantified logic of *h*ere-and-*t*here with equality") (Ferraris et al., 2011), so that the provability of a sentence $F \leftrightarrow G$ in this system implies that $\mathrm{SM}[F]$ is equivalent to $\mathrm{SM}[G]$.

We will first prove the following proposition.

Proposition 4 *For any safe sentence F and any nonempty finite set **c** of object constants containing $c(F)$, the equivalence*

$$Ground_{\mathbf{c}}[F] \leftrightarrow F$$

*is derivable from SPP$_{\mathbf{c}}$ in **INT**$^=$ + DE.*

Lemma 13 *If any of the sentences $\forall x F(x)$, $\exists x F(x)$ is safe then so is $F(c)$ for any object constant c.*

Proof. Immediate from the fact, easily verified by induction, that if a variable other than x is restricted in a formula $G(x)$ then it is restricted in $G(c)$ as well. □

Lemma 14 *If x is restricted in a quantifier-free formula $F(x)$, and **c** is a nonempty finite set of object constants containing $c(F)$, then the formula*

$$F(x) \rightarrow in_{\mathbf{c}}(x)$$

*is derivable from SPP$_{\mathbf{c}}$ in **INT**$^=$.*

Proof. Immediate by induction on $F(x)$. □

Lemma 15 *For any formula $F(x)$ in prenex form that has no free variables other than x, and for any nonempty finite set **c** of object constants containing $c(F)$,*

(a) if the sentence $\forall x F(x)$ is safe then the equivalence

$$\forall x F(x) \leftrightarrow \bigwedge_{c \in \mathbf{c}} F(c)$$

*is derivable from SPP$_{\mathbf{c}}$ in **INT**$^{=} + DE$;*

(b) if the sentence $\exists x F(x)$ is safe then the equivalence

$$\exists x F(x) \leftrightarrow \bigvee_{c \in \mathbf{c}} F(c)$$

*is derivable from SPP$_{\mathbf{c}}$ in **INT**$^{=} + DE$.*

Proof. (a) Assume that $\forall x F(x)$ is safe. In **INT**$^{=} + \mathrm{DE}$, this formula can be equivalently written as

$$\forall x((\mathit{in}_{\mathbf{c}}(x) \rightarrow F(x)) \wedge (\neg \mathit{in}_{\mathbf{c}}(x) \rightarrow F(x))),$$

and consequently as

$$\bigwedge_{c \in \mathbf{c}} F(c) \ \wedge \ \forall x(\neg \mathit{in}_{\mathbf{c}}(x) \rightarrow F(x)). \tag{6.19}$$

Consider the maximal subformulas $G(x) \rightarrow H(x)$ of $F(x)$, positive in $F(x)$, such that x is restricted in $G(x)$. From Lemma 14 we conclude that for each of these subformulas, the implication

$$G(x) \rightarrow \mathit{in}_{\mathbf{c}}(x)$$

is derivable from *SPP*$_{\mathbf{c}}$ in **INT**$^{=}$, and consequently so is

$$\neg \mathit{in}_{\mathbf{c}}(x) \rightarrow (G(x) \rightarrow H(x)).$$

It follows that, under the assumption $SPP_{\mathbf{c}}$, (6.19) can be equivalently rewritten as

$$\bigwedge_{c\in\mathbf{c}} F(c) \ \wedge \ \forall x(\neg \mathit{in}_{\mathbf{c}}(x) \to S), \tag{6.20}$$

where S is the formula obtained from $F(x)$ by replacing each of these maximal subformulas $G(x) \to H(x)$ with $\top$. Since $\forall x F(x)$ is safe, x does not occur in S. It follows that S can be obtained from $F(c)$ in the same way as it was obtained from $F(x)$, that is, by replacing some subformulas that are positive in $F(c)$ with $\top$. Consequently, the formula $F(c) \to S$ is intuitionistically provable, and so is

$$F(c) \to \forall x(\neg \mathit{in}_{\mathbf{c}}(x) \to S).$$

It follows that the second conjunctive term of (6.20) can be dropped.

(b) Assume that $\exists x F(x)$ is safe. In $\mathbf{INT}^{=} + \mathrm{DE}$, this formula can be equivalently written as

$$\exists x((\mathit{in}_{\mathbf{c}}(x) \wedge F(x)) \vee (\neg \mathit{in}_{\mathbf{c}}(x) \wedge F(x))),$$

and consequently as

$$\bigvee_{c\in\mathbf{c}} F(c) \ \vee \ \exists x(\neg \mathit{in}_{\mathbf{c}}(x) \wedge F(x)). \tag{6.21}$$

Consider the maximal subformulas $G(x) \to H(x)$ of $F(x)$, negative in $F(x)$, such that x is restricted in $G(x)$. As before, the implications

$$\neg \mathit{in}_{\mathbf{c}}(x) \to (G(x) \to H(x))$$

are derivable from $SPP_{\mathbf{c}}$ in $\mathbf{INT}^{=}$. Consequently, under the assumption $SPP_{\mathbf{c}}$, (6.21) can be equivalently rewritten as

$$\bigvee_{c\in\mathbf{c}} F(c) \ \vee \ \exists x(\neg \mathit{in}_{\mathbf{c}}(x) \wedge S), \tag{6.22}$$

where S is the formula obtained from $F(x)$ by replacing each of these subformulas with $\top$. Since $\exists x F(x)$ is safe, x does not occur in S. It follows that S can be

obtained from $F(c)$ in the same way as it was obtained from $F(x)$, that is, by replacing some subformulas that are negative in $F(c)$ with $\top$. Consequently, the formula $S \to F(c)$ is intuitionistically provable, and so is

$$\exists x(\neg \mathit{in}_{\mathbf{c}}(x) \wedge S) \to F(c).$$

It follows that the second disjunctive term of (6.22) can be dropped. □

Proof of Proposition 4. By induction on the length of the prefix. The base case is trivial. Assume that $QxF(x)$ is safe. Case 1: Q is $\forall$. In view of Lemma 13, from the induction hypothesis we can conclude that

$$\mathrm{Ground}_{\mathbf{c}}[F(c)] \leftrightarrow F(c)$$

is derivable from $\mathit{SPP}_{\mathbf{c}}$ in $\mathbf{INT}^{=} + \mathrm{DE}$ for every $c \in \mathbf{c}$. Consequently

$$\bigwedge_{c \in \mathbf{c}} \mathrm{Ground}_{\mathbf{c}}[F(c)] \leftrightarrow \bigwedge_{c \in \mathbf{c}} F(c)$$

is derivable from $\mathit{SPP}_{\mathbf{c}}$ as well. By the definition of $\mathrm{Ground}_{\mathbf{c}}$, the left-hand side is $\mathrm{Ground}_{\mathbf{c}}[\forall x F(x)]$. By Lemma 15(a), under the assumption $\mathit{SPP}_{\mathbf{c}}$, the right-hand side is equivalent in $\mathbf{INT}^{=} + \mathrm{DE}$ to $\forall x F(x)$. Case 2: Q is $\exists$. Similar, using Lemma 15(b). □

Proof of Theorem 17. By Proposition 4 proved above, the equivalence

$$\mathrm{Ground}_{\mathbf{c}}[F] \wedge \mathit{SPP}_{\mathbf{c}} \leftrightarrow F \wedge \mathit{SPP}_{\mathbf{c}}$$

is provable in $\mathbf{INT}^{=} + \mathrm{DE}$. Consequently

$$\mathrm{SM}[\mathrm{Ground}_{\mathbf{c}}[F] \wedge \mathit{SPP}_{\mathbf{c}}] \text{ is equivalent to } \mathrm{SM}[F \wedge \mathit{SPP}_{\mathbf{c}}].$$

Since $\mathit{SPP}_{\mathbf{c}}$ is negative on all predicates, it follows from Lemma 4 that

$$\mathrm{SM}[\mathrm{Ground}_{\mathbf{c}}[F]] \wedge \mathit{SPP}_{\mathbf{c}} \text{ is equivalent to } \mathrm{SM}[F] \wedge \mathit{SPP}_{\mathbf{c}}.$$

In view of Proposition 3 and the fact that $c(F) \subseteq \mathbf{c}$, the conjunctive term $\mathrm{SM}[F]$ in the second conjunction entails its other conjunctive term *SPP*$_{\mathbf{c}}$, and the latter can be dropped. Furthermore, $\mathrm{Ground}_{\mathbf{c}}[F]$ is variable-free and consequently safe. It follows by similar reasoning that in the first conjunction the term *SPP*$_{\mathbf{c}}$ can be dropped also. □

Proof of Theorem 18

In view of Proposition 3, we need to find a variable-free formula G such that *SPP*$_{c(F)}$ entails $\mathrm{SM}[F] \leftrightarrow G$.

Case 1: $c(F) = \emptyset$. Under the assumption *SPP*$_\emptyset$, every atomic part of $\mathrm{SM}[F]$ that contains a predicate constant or variable of arity > 0 can be equivalently replaced by $\bot$. The result is a second-order propositional formula, so that it is equivalent to a propositional formula.

Case 2: $c(F) \neq \emptyset$ and F is variable-free. The only quantifiers in the definition of SM are the second-order quantifiers $\exists \mathbf{u}$. Clearly *SPP*$_{c(F)}$ entails

$$u_i \leq p_i \ \rightarrow \ u_i \leq \lambda \mathbf{x} \left(\bigvee_{\mathbf{c}} \mathbf{x} = \mathbf{c} \right)$$

where **c** ranges over the tuples of members of $c(F)$ of the same length as **x**. Consequently it entails also

$$\mathbf{u} < \mathbf{p} \ \rightarrow \ u_i \leq \lambda \mathbf{x} \left(\bigvee_{\mathbf{c}} \mathbf{x} = \mathbf{c} \right)$$

and

$$\mathbf{u} < \mathbf{p} \ \rightarrow \ \bigvee_{C} \left(u_i = \lambda \mathbf{x} \bigvee_{\mathbf{c} \in C} \mathbf{x} = \mathbf{c} \right),$$

where C ranges over all sets of such tuples. It follows that under the assumption *SPP*$_{c(F)}$ the quantifiers $\exists \mathbf{u}$ can be equivalently replaced by finite disjunctions, with expressions of the form $\lambda \mathbf{x} \bigvee_{\mathbf{c} \in C} \mathbf{x} = \mathbf{c}$ substituted for the variables u_i. The result is a variable-free formula with the required properties.

Case 3: $c(F) \neq \emptyset$ and F is not variable-free. The part of Theorem 18 corresponding to Case 2 can be applied to $\mathrm{Ground}_{c(F)}[F]$. Since the formulas F and $\mathrm{Ground}_{c(F)}[F]$ contain the same object constants, we can assert that, for some variable-free formula G, ***SPP***$_{c(F)}$ entails

$$\mathrm{SM}[\mathrm{Ground}_{c(F)}[F]] \leftrightarrow G.$$

It remains to observe that, by Theorem 17, the left-hand side is equivalent to $\mathrm{SM}[F]$. □

Proof of Theorem 19

From Theorem 18, it follows that there is a variable-free formula G such that $\mathrm{SM}[F]$ is equivalent to $G \wedge$ ***SPP***$_{c(F)}$. The result follows from the observation that $I \models G \wedge$ ***SPP***$_{c(F)}$ iff the extension of I to X satisfies $G \wedge$ ***SPP***$_{c(F)}$. □

Proof of Theorem 20

From Theorem 17, it follows that $\mathrm{SM}[\mathrm{Ground}_{c(F)}[F]]$ is equivalent to $\mathrm{SM}[F]$. The result follows from Proposition 3. □

Chapter 7

EVENT CALCULUS IN ANSWER SET PROGRAMMING

In this chapter, we use the results presented in the previous chapters to reformulate the event calculus in ASP. We begin with a review of the event calculus and then show how the theorem on canonical formulas (Theorem 4) and the splitting theorem (Theorem 12) can be used to reformulate the event calculus in the first-order stable model semantics. We then show how translation F2LP (Chapter 5.3) can be used to further transform event calculus theories into the syntax of ASP. Based on this, we show how system F2LP can be used for computing event calculus descriptions and discuss some of the advantages of this approach. We then compare this computational approach to some existing reasoners, primarily the DEC reasoner[1] (Mueller, 2004).

7.1 Review of the Event Calculus

Here we review the syntax of circumscriptive event calculus described in (Mueller, 2006, Chapter 2).

The language of the event calculus is a many-sorted first-order language, which contains an *event* sort, a *fluent* sort, and a *timepoint* sort. A *fluent term* is a term whose sort is a fluent; an *event term* and a *timepoint term* are defined similarly. Some of the key event calculus predicates that are used to represent the knowledge about the occurrences of events and the values of the fluents are as follows:

- *HoldsAt*(f, t): fluent f is true at timepoint t;
- *Happens*(e, t): event e occurs at timepoint t;
- *Initiates*(e, f, t): if event e occurs at timepoint t, then fluent f is true after t;

[1] http://decreasoner.sourceforge.net/.

- *Terminates*(e, f, t): if event e occurs at timepoint t, then fluent f is false after t;

- *Releases*(e, f, t): if event e occurs at timepoint t, then fluent f is released from the commonsense law of inertia after t;

- *ReleasedAt*(f, t): fluent f is released from the commonsense law of inertia at timepoint t.

A *condition* is defined recursively as follows:

- If τ_1 and τ_2 are terms, then comparisons $\tau_1 < \tau_2$, $\tau_1 \leq \tau_2$, $\tau_1 \geq \tau_2$, $\tau_1 > \tau_2$, $\tau_1 = \tau_2$, $\tau_1 \neq \tau_2$ are conditions;

- If f is a fluent term and t is a timepoint term, then *HoldsAt*(f, t) and $\neg$*HoldsAt*(f, t) are conditions;

- If γ_1 and γ_2 are conditions, then $\gamma_1 \wedge \gamma_2$ and $\gamma_1 \vee \gamma_2$ are conditions;

- If v is a variable and γ is a condition, then $\exists v \gamma$ is a condition.

In the following, we will use e and e_i to denote event terms, f and f_i to denote fluent terms, t and t_i to denote timepoint terms, and γ and γ_i to denote conditions.

In the event calculus, *Initiates*, *Terminates*, and *Releases* are circumscribed to minimize unexpected effects of events, *Happens* is circumscribed to minimize unexpected events, and every abnormality predicate *Ab*$_i$ is circumscribed to minimize abnormalities. Formally, an *event calculus description* is a circumscriptive theory of the form

$$\begin{aligned}\mathrm{CIRC}[\Sigma\ ;\ \textit{Initiates}, \textit{Terminates}, \textit{Releases}] \wedge \mathrm{CIRC}[\Delta\ ;\ \textit{Happens}] \\ \wedge\ \mathrm{CIRC}[\Theta\ ;\ \textit{Ab}_1, \ldots, \textit{Ab}_n] \wedge\ \Xi\end{aligned} \tag{7.1}$$

where

- Σ is a conjunction of universal closures of axioms of the form

  $\gamma \rightarrow \textit{Initiates}(e, f, t)$

  $\gamma \rightarrow \textit{Terminates}(e, f, t)$

  $\gamma \rightarrow \textit{Releases}(e, f, t)$

  $\gamma \wedge \pi_1(e, f_1, t) \rightarrow \pi_2(e, f_2, t)$ ("effect constraint")

  $\gamma \wedge [\neg]Happens(e_1, t) \wedge \cdots \wedge [\neg]Happens(e_n, t) \rightarrow \textit{Initiates}(e, f, t)$

  $\gamma \wedge [\neg]Happens(e_1, t) \wedge \cdots \wedge [\neg]Happens(e_n, t) \rightarrow \textit{Terminates}(e, f, t)$

  where each of π_1 and π_2 is either *Initiates* or *Terminates* ('$[\neg]$' means that '$\neg$' is optional);

- Δ is a conjunction of universal closures of *temporal ordering formulas* (comparisons between timepoint terms) and axioms of the form

  $\gamma \rightarrow \textit{Happens}(e, t)$

  $\sigma(f, t) \wedge \pi_1(f_1, t) \wedge \cdots \wedge \pi_n(f_n, t) \rightarrow \textit{Happens}(e, t)$ ("causal constraints")

  $\textit{Happens}(e, t) \rightarrow$
  $\textit{Happens}(e_1, t) \vee \cdots \vee \textit{Happens}(e_n, t)$ ("disjunctive event axiom")

  where σ is *Started* or *Stopped* and each π_j $(1 \leq j \leq n)$ is either *Initiated* or *Terminated*;

- Θ is a conjunction of universal closures of *cancellation* axioms of the form

  $$\gamma \rightarrow Ab_i(..., t);$$

- Ξ is a conjunction of first-order sentences (outside the scope of CIRC) including unique name axioms, state constraints, action precondition

axioms, event occurrence constraints, trajectory/antitrajectory axioms, observations, and the set of domain-independent axioms in the event calculus, such as *EC* and *DEC* axioms (Mueller, 2006). It also includes the following definitions of the predicates used in the causal constraints in Δ:

$$\textit{Started}(f,t) \overset{def}{\leftrightarrow}$$
$$(\textit{HoldsAt}(f,t) \vee \exists e(\textit{Happens}(e,t) \wedge \textit{Initiates}(e,f,t))) \quad (CC_1)$$
$$\textit{Stopped}(f,t) \overset{def}{\leftrightarrow}$$
$$(\neg\textit{HoldsAt}(f,t) \vee \exists e(\textit{Happens}(e,t) \wedge \textit{Terminates}(e,f,t))) \quad (CC_2)$$
$$\textit{Initiated}(f,t) \overset{def}{\leftrightarrow}$$
$$(\textit{Started}(f,t) \wedge \neg\exists e(\textit{Happens}(e,t) \wedge \textit{Terminates}(e,f,t))) \quad (CC_3)$$
$$\textit{Terminated}(f,t) \overset{def}{\leftrightarrow}$$
$$(\textit{Stopped}(f,t) \wedge \neg\exists e(\textit{Happens}(e,t) \wedge \textit{Initiates}(e,f,t))) \quad (CC_4).$$

For example, the "Yale Shooting" (Hanks & McDermott, 1987) scenario can be represented in the event calculus as follows:

$$\textit{Initiates}(\textit{Load}, \textit{Loaded}, t)$$
$$\textit{HoldsAt}(\textit{Loaded}, t) \rightarrow \textit{Terminates}(\textit{Shoot}, \textit{Alive}, t)$$
$$\textit{Terminates}(\textit{Shoot}, \textit{Loaded}, t)$$

$$\textit{HoldsAt}(\textit{Alive}, 0) \wedge \neg\textit{HoldsAt}(\textit{Loaded}, 0)$$
$$\textit{Happens}(\textit{Load}, 0) \wedge \textit{Happens}(\textit{Wait}, 1) \wedge \textit{Happens}(\textit{Shoot}, 2)$$

In addition to these axioms, there are the unique name axioms and the domain-independent axioms. The domain-independent axioms include inertial axioms, and the axioms that make a fluent true/false based on the occurrence of events that affect it. For example, in the *Discrete Event Calculus* (DEC), the following axioms make a fluent true/false based on the occurrence of events that

affect it:

$$\textit{Happens}(e,t) \wedge \textit{Initiates}(e,f,t) \rightarrow \textit{HoldsAt}(f,t+1)$$
$$\textit{Happens}(e,t) \wedge \textit{Terminates}(e,f,t) \rightarrow \neg\textit{HoldsAt}(f,t+1).$$

The following DEC axioms ensure that the value of a fluent does not change unless some event that affects it occurs or unless it is released from the commonsense law of inertia:

$$\textit{HoldsAt}(f,t) \wedge \neg\textit{ReleasedAt}(f,t+1) \wedge$$
$$\neg\exists e(\textit{Happens}(e,t) \wedge \textit{Terminates}(e,f,t)) \rightarrow \textit{HoldsAt}(f,t+1)$$
$$\neg\textit{HoldsAt}(f,t) \wedge \neg\textit{ReleasedAt}(f,t+1) \wedge$$
$$\neg\exists e(\textit{Happens}(e,t) \wedge \textit{Initiates}(e,f,t)) \rightarrow \neg\textit{HoldsAt}(f,t+1).$$

In the "Yale Shooting" scenario, since *Happens* is minimized, the only events that occur are those that are specified by the event occurrence formula (the last formula), and since *Initiates* and *Terminates* are minimized, no unexpected effects of events occur. As a result, the gun remains loaded after the *Wait* action occurs since the action does not affect any fluent.

7.2 Reformulating the Event Calculus in the First-Order Stable Model Semantics

We assume that Ξ in (7.1) was already equivalently rewritten so that it is negative on $\{$*Initiates*, *Terminates*, *Releases*, *Happens*, *Ab*$_1, \ldots,$ *Ab*$_n\}$.[2]

The following facts are easy to check from the description of the event calculus (7.1), and the definition of canonical formulas (Chapter 3.3):

- Σ is canonical relative to $\{$*Initiates*, *Terminates*, *Releases*$\}$;
- Δ is canonical relative to $\{$*Happens*$\}$;
- Θ is canonical relative to $\{$*Ab*$_1, \ldots,$ *Ab*$_n\}$.

[2]Recall the definition of a *negative formula on* ***p*** given in Chapter 4.

These facts enable us to reformulate the event calculus in the first-order stable model semantics. The following theorem shows a few equivalent reformulations of circumscriptive event calculus in the first-order stable model semantics.

Theorem 21 *For any event calculus description (7.1), the following theories are equivalent to each other:* [3]

(a) $CIRC[\Sigma; I, T, R] \wedge CIRC[\Delta; H] \wedge CIRC[\Theta; \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge \Xi$;

(b) $SM[\Sigma; I, T, R] \wedge SM[\Delta; H] \wedge SM[\Theta; \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge \Xi$;

(c) $SM[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi; I, T, R, H, \mathit{Ab}_1, \ldots, \mathit{Ab}_n]$;

(d) $SM[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi \wedge \mathit{Choice}(\mathit{pr}(\Sigma \wedge \Delta \wedge \Theta \wedge \Xi) \setminus \{I, T, R, H, \mathit{Ab}_1, \ldots, \mathit{Ab}_n\})]$.

The equivalence between (a) and (b) is immediate from the theorem on canonical formulas (Theorem 4). The equivalence between (b) and (c) can be proved using the splitting theorem (Chapter 4.1). The assumption that Ξ is negative on the intensional predicates is essential in showing this equivalence. (For more details, see the proof in the Appendix.) The equivalence between (c) and (d) follows from Proposition 1 since $\{I, T, R, H, \mathit{Ab}_1, \ldots, \mathit{Ab}_n\} \setminus \mathit{pr}(\Sigma \wedge \Delta \wedge \Theta \wedge \Xi)$ is the empty set.[4]

7.3 Reformulating the Event Calculus in Answer Set Programming

Using translation F2LP (Chapter 5.3), we can further turn the event calculus reformulation shown above (Chapter 7.2) into answer set programs.

The following procedure turns an event calculus description into an answer set program.

[3] For brevity, we abbreviate the names of circumscribed predicates.

[4] I, T, R, H occur in the domain independent axioms as part of Ξ.

Definition 3 (Translation EC2ASP) *1. Given an event calculus description (7.1), rewrite all the definitional axioms of the form*

$$\forall \boldsymbol{x}(p(\boldsymbol{x}) \stackrel{def}{\leftrightarrow} G) \tag{7.2}$$

in Ξ as $\forall \boldsymbol{x}(G^{\neg\neg} \rightarrow p(\boldsymbol{x}))$ where $G^{\neg\neg}$ is obtained from G by prepending $\neg\neg$ to all occurrences of the intensional predicates Initiates, Terminates, Releases, Happens, $Ab_1, \ldots, Ab_n$. Also prepend $\neg\neg$ to the strictly positive occurrences of the intensional predicates in the remaining axioms of Ξ. Let Ξ' be the resulting formula obtained from Ξ.

2. Apply translation F2LP on $\Sigma \wedge \Delta \wedge \Theta \wedge \Xi'$ with intensional predicates

$$\{\textit{Initiates}, \textit{Terminates}, \textit{Releases}, \textit{Happens}, \textit{Ab}_1, \ldots, \textit{Ab}_n\} \cup \boldsymbol{p}$$

*where **p** is the set of all predicate constants p of (7.2) considered in Step 1.*

The following theorem states the correctness of the translation.

Theorem 22 *Let T be an event calculus description (7.1) of signature σ that contains finitely many predicate constants, let F be the FOL-representation of the program obtained from T by applying translation EC2ASP. Then T is σ-equivalent to $SM[F]$.*

In view of the theorem, system F2LP can be used to compute event calculus descriptions by a simple rewriting as stated in translation EC2ASP.[5]

Figure 7.1 shows an F2LP encoding of the domain-independent axioms in the *Discrete Event Calculus* (DEC) (Mueller, 2006). The file is also available at http://reasoning.eas.asu.edu/f2lp, along with the F2LP encodings of the domain

[5] In (Kim et al., 2009), we presented a prototype of system F2LP called ECASP that is tailored to the event calculus computation.

independent axioms in other versions of the event calculus. Figure 7.2 shows an F2LP encoding of a simple blocks world domain. In this blocks world encoding, there are only 2 actions: *PickUp*(x, y) and *Stack*(x, y). *PickUp*(x, y) represents the action of picking up block x from the table (y) or from the top of another block y. *Stack*(x, y) represents the action of placing block x on top of another block y or on the table (y). Initially, block a is on the table, c is on a, and b is on c. The goal is to stack the blocks such that c is on the table, b is on c, and a is on b. The plan can be computed by invoking F2LP along with GRINGO and CLASPD as follows:

```
$ f2lp dec blocksWorld |  gringo -c maxstep=8 | claspD
```

where 8 is the length of the plan. This gives the following plan:

```
happens(pickUp(b),0) happens(stack(b,table),1)  happens(pickUp(c),2)
happens(stack(c,table),3) happens(pickUp(b),4) happens(stack(b,c),5)
happens(pickUp(a),6) happens(stack(a,b),7)
```

```
% File ‘dec’

% domain variable declarations
#domain fluent(F). #domain fluent(F1). #domain fluent(F2).
#domain event(E).
#domain time(T). #domain time(T1). #domain time(T2).

time(0..maxstep).

% DEC 1
happens(E,T) & T1 < T & T < T2 & terminates(E,F,T) -> stoppedIn(T1,F,T2).

% DEC 2
happens(E,T) & T1 < T & T < T2 & initiates(E,F,T) -> startedIn(T1,F,T2).

% DEC 3
happens(E,T1) & initiates(E,F1,T1) & T2 > 0 &
trajectory(F1,T1,F2,T2) & -stoppedIn(T1,F1,T1+T2) &
T1+T2 <= maxstep -> holdsAt(F2,T1+T2).

% DEC 4
happens(E,T1) & terminates(E,F1,T1) & 0 < T2 &
antiTrajectory(F1,T1,F2,T2) & -startedIn(T1,F1,T1+T2) &
T1+T2 <= maxstep -> holdsAt(F2,T1+T2).

% DEC 5
holdsAt(F,T) & -releasedAt(F,T+1) &
-?[E]:(happens(E,T) & terminates(E,F,T)) &
T < maxstep -> holdsAt(F,T+1).

% DEC 6
-holdsAt(F,T) & -releasedAt(F,T+1) &
-?[E]:(happens(E,T) & initiates(E,F,T)) &
T < maxstep -> -holdsAt(F,T+1).

% DEC 7
releasedAt(F,T) & -?[E]:(happens(E,T) &
(initiates(E,F,T) | terminates(E,F,T))) &
T < maxstep -> releasedAt(F,T+1).

% DEC 8
-releasedAt(F,T) &
-?[E]: (happens(E,T) & releases(E,F,T)) &
T < maxstep -> -releasedAt(F,T+1).

% DEC 9
happens(E,T) & initiates(E,F,T) & T < maxstep -> holdsAt(F,T+1).
```

```
% DEC 10
happens(E,T) & terminates(E,F,T) & T < maxstep -> -holdsAt(F,T+1).

% DEC 11
happens(E,T) & releases(E,F,T) & T < maxstep -> releasedAt(F,T+1).

% DEC 12
happens(E,T) &
(initiates(E,F,T) | terminates(E,F,T)) &
T < maxstep -> -releasedAt(F,T+1).

% Choice rules for non-intensional predicates
{holdsAt(F,T)}.
{releasedAt(F,T)}.
```

Figure 7.1: DEC axioms in F2LP

```
% file 'blocksWorld'

% domain declarations
objects(a;b;c;table).
time(0..maxstep).

#domain objects(X). #domain objects(Y).
#domain time(T).

% fluents and events
fluent(on(X,Y);holding(X);clear(X)).
event(pickUp(X);stack(X,Y)).

#domain event(E1). #domain event(E2).

% initiates and terminates formulas
T < maxstep -> initiates(pickUp(X),holding(X),T).
T < maxstep & holdsAt(on(X,Y),T) -> terminates(pickUp(X),on(X,Y),T).
T < maxstep -> initiates(stack(X,Y),on(X,Y),T).
T < maxstep -> terminates(stack(X,Y),holding(X),T).

% action precondition axioms
T < maxstep & happens(pickUp(X),T) ->
holdsAt(clear(X),T) & -?[Y]:holdsAt(holding(Y),T) & X != table.
T < maxstep & happens(stack(X,Y),T) ->
holdsAt(holding(X),T) & holdsAt(clear(Y),T) & X != table.

% event occurrence constraints
T< maxstep & happens(E1,T) & E1 != E2 -> -happens(E2,T).

% state constraints
X != table & holdsAt(clear(X),T) -> -?[Y]:(holdsAt(on(Y,X),T)).
-?[Y]:(holdsAt(on(Y,X),T)) -> holdsAt(clear(X),T) & X != table.
holdsAt(on(X,Y),T) -> X != table.
holdsAt(clear(table),T).

% clear is non-inertial, and the rest of the fluents are inertial
releasedAt(clear(X),0).
-releasedAt(holding(X),0).
-releasedAt(on(X,Y),0).

% happens is exempt from minimization in order to find a plan
T < maxstep -> {happens(E1,T)}.
```

```
% initial state
holdsAt(on(a,table),0). holdsAt(on(b,c),0). holdsAt(on(c,a),0).
-holdsAt(on(a,c),0). -holdsAt(on(a,b),0). -holdsAt(on(b,table),0).
-holdsAt(on(c,table),0). -holdsAt(on(c,b),0).
-holdsAt(on(b,a),0). -holdsAt(holding(X),0).

% goal
--(holdsAt(on(c,table),maxstep) &
   holdsAt(on(b,c),maxstep) &
   holdsAt(on(a,b),maxstep)).
```

Figure 7.2: Blocks World in F2LP

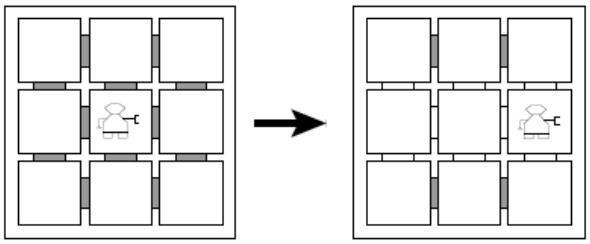

Figure 7.3: Robby's apartment in a 3×3 grid

7.4 Enhancing Event Calculus Descriptions with Answer Set Programming Rules

As mentioned in the introduction, one of the advantages of ASP-based event calculus reasoning is that we can enhance event calculus descriptions with expressive ASP rules such as the transitive closure rules. We illustrate this using the example from (Doğandağ, Ferraris, & Lifschitz, 2004). There are 9 rooms numbered 1–9 (horizontal, then vertical) and 12 doors as shown in Figure 7.3. Initially the robot "Robby" is in the middle room and all the doors are closed. The goal of the robot is to make all rooms accessible from each other. Figure 7.4 (File 'robby') shows an encoding of the problem in the language of F2LP. Atom `door(x,y)` denotes that there is a door between rooms `x` and `y`; `open(x,y)` denotes the event "Robby opening the door between rooms `x` and `y`"; `goto(x)` denotes the event "Robby going to room `x`"; `opened(x,y)` denotes that the door between `x` and `y` has been opened; `inRoom(x)` denotes that Robby is in room `x`; `accessible(x,y)` denotes that `y` is accessible from `x`. Note that the rules defining the relation `accessible` are not part of event calculus axioms (Chapter 7.1). This example illustrates an advantage of allowing ASP rules in event calculus descriptions.

The minimal number of steps to solve the given problem is 11. We can find such a plan using the combination of F2LP, GRINGO and CLASPD in the following way:

```
% File 'robby'

% objects
step(0..maxstep).
astep(0..maxstep-1).
room(1..9).

% variables
#domain step(T).
#domain room(R).
#domain room(R1).
#domain room(R2).

% position of the doors
R1 >= 1 & R2 >=1 & R1 < 4 & R2 < 4 & R2 = R1+1 -> door(R1,R2).
R1 >= 4 & R2 >= 4 & R1 < 7 & R2 < 7 & R2 = R1+1 -> door(R1,R2).
R1 >= 7 & R2 >= 7 & R1 < 10 & R2 < 10 & R2 = R1+1 -> door(R1,R2).
R2 < 10 & R2 = R1+3 -> door(R1,R2).

door(R2,R1) -> door(R1,R2).

% fluents
door(R1,R2) -> fluent(opened(R1,R2)).
fluent(inRoom(R)).
% F ranges over the fluents
#domain fluent(F).

% events
door(R1,R2) -> event(open(R1,R2)).
event(goto(R)).
% E and E1 range over the events
#domain event(E).
#domain event(E1).

% effect axioms
initiates(open(R,R1),opened(R,R1),T).
initiates(open(R,R1),opened(R1,R),T).

holdsAt(opened(R1,R2),T) & holdsAt(inRoom(R1),T) ->
initiates(goto(R2),inRoom(R2),T).

holdsAt(inRoom(R1),T) & initiates(E,inRoom(R2),T) ->
terminates(E,inRoom(R1),T).

% action precondition axioms
happens(open(R1,R2),T) -> holdsAt(inRoom(R1),T).
```

```
% event occurrence constraint
happens(E,T) & E != E1 -> -happens(E1,T).

% state constraint
holdsAt(inRoom(R1),T) & R1 != R2 -> -holdsAt(inRoom(R2),T).

% accessibility
holdsAt(opened(R,R1),T) -> accessible(R,R1,T).
accessible(R,R1,T) & accessible(R1,R2,T) -> accessible(R,R2,T).

% initial state
-holdsAt(opened(R1,R2),0).
holdsAt(inRoom(5),0).

% goal state
--accessible(R,R1,maxstep).

% happens is exempt from minimization in order to find a plan.
T < maxstep -> {happens(E,T)}.

% all fluents are inertial
-releasedAt(F,0).
```

Figure 7.4: Robby's apartment in F2LP

```
$ f2lp dec robby | gringo -c maxstep=11 | claspD
```

Following is one of the plans found:

```
happens(open(5,8),0) happens(open(5,2),1) happens(open(5,6),2)
happens(open(5,4),3) happens(goto(4),4) happens(open(4,7),5)
happens(open(4,1),6) happens(goto(5),7) happens(goto(6),8)
happens(open(6,3),9) happens(open(6,9),10)
```

7.5 Comparison with Other Event Calculus Reasoners

Shanahan (2000) introduced an event calculus planner that uses abductive logic programming to solve abduction and planning problems.[6] Later, Shanahan and Witkowski (2004) introduced a SAT-based event calculus planner that showed significant improvements over the abductive planner. Based on the reduction of

[6] http://www.iis.ee.ic.ac.uk/∼mpsha/planners.html

circumscription to completion (Chapter 3.1, Theorems 1 and 2), Mueller (2004) introduced a SAT-based event calculus reasoner that not only solves abduction and planning problems but also solves other interesting problems such as projection and postdiction. This system is called the DEC reasoner.[7] While the system handles a large fragment of the event calculus, it still cannot handle recursive and disjunctive axioms, such as effect constraints and disjunctive event axioms (Chapter 7.1), since completion cannot be applied to descriptions containing such axioms. For example, the DEC reasoner does not allow the following effect constraints which describe the indirect effects of an agent's walking on the objects that he is holding:

$$\begin{aligned} &\textit{HoldsAt}(\textit{Holding}(a,o),t) \wedge \textit{Initiates}(e,\textit{InRoom}(a,r),t) \\ &\qquad\qquad \rightarrow \textit{Initiates}(e,\textit{InRoom}(o,r),t) \\ &\textit{HoldsAt}(\textit{Holding}(a,o),t) \wedge \textit{Terminates}(e,\textit{InRoom}(a,r),t) \\ &\qquad\qquad \rightarrow \textit{Terminates}(e,\textit{InRoom}(o,r),t). \end{aligned} \tag{7.3}$$

Our ASP-based approach on the other hand can handle the *full* version of the event calculus under the assumption that the domain is given and finite.

We compared the performance of the DEC reasoner (v 1.0) running MINISAT (v 2.2) with the following:[8]

- F2LP (v 1.11) with LPARSE (v 1.0.17)+CMODELS (v 3.79) running MINISAT (v 2.0 beta),
- F2LP (v 1.11) with GRINGO (v 3.0.3)+CMODELS (v 3.79) running MINISAT (v 2.0 beta),
- F2LP (v 1.11) with GRINGO (v 3.0.3) +CLASP (v 2.0.2) (CLASP(D) (v 1.1.2) used instead for disjunctive programs), and

[7] http://decreasoner.sourceforge.net/.

[8] Similar results were observed when RELSAT (v 2.2) was used with the DEC reasoner.

Problem (max. step)	DEC reasoner (minisat)	F2LP with (LPARSE + CMODELS)	F2LP with (GRINGO + CMODELS)	F2LP with (GRINGO + CLASP(D))	F2LP with CLINGO
BusRide (15)	—	0.04s (0.03s + 0.01s) A:902/R:7779 C:0	0.00s (0.00s + 0.00s) A:355/R:555 C:0	0.01s (0.00s + 0.01s) A:448/R:647	—
Commuter (15)	—	77.29s (45.74s + 31.55s) A:32861/R:8734019 C:0	0.15s (0.07s + 0.08s) A:5269/R:24687 C:5308	0.2s (0.07s + 0.13s) A:13174/R:24687	0.14s
Kitchen Sink (25)	38.9s (38.9s + 0.00s) A:1014 C:12109	6.19s (2.99s + 3.20s) A:121621/R:480187 C:0	0.44s (0.19s + 0.25s) A:11970/R:61932 C:0	0.24s (0.18s + 0.06s) A:11970/R:61932	0.20s
Thielscher Circuit (40)	6.3s (6.3s + 0.0s) A:1394 C:42454	0.42s (0.27s + 0.15s) A:9292/R:53719 C:0	0.19s (0.09s + 0.1s) A:4899/R:35545 C:0	0.12s (0.09s + 0.03s) A:4899/R:35545	0.1s
Walking Turkey (15)	—	0.00s (0.00s + 0.00s) A:370/R:518 C:0	0.00s (0.00s + 0.00s) A:316/R:456 C:0	0.00s (0.00s + 0.00s) A:316/R:456	0.00s
Falling w/ AntiTraj (15)	141.7s (141.7s + 0.00s) A:416 C:3056	0.08s (0.05s + 0.03s) A:4994/R:9717 C:0	0.04s (0.02s + 0.02s) A:3702/R:7414 C:0	0.03s (0.03s + 0.00s) A:3702/R:7414	0.03s
Falling w/ Events (25)	59.4s (59.4s + 0.0s) A:1092 C:12351	4.95s (2.57s + 2.38s) A:1240/R:388282 C:1436	0.46s (0.20s + 0.26s) A:1219/R:71266 C:1415	0.28s (0.20s + 0.08s) A:13829/R:71266	0.22s
HotAir Baloon (15)	32.3s (32.3s + 0.0s) A:288 C:1163	0.01s (0.01s + 0.00s) A:494/R:2451 C:689	0.0s (0.0s + 0.0s) A:492/R:1835 C:681	0.0s (0.0s + 0.0s) A:1063/R:1835	0.01s
Telephone1 (40)	9.1s (9.1s + 0.0s) A:5419 C:41590	0.22s (0.13s + 0.09s) A:21414/R:27277 C:0	0.11s (0.08s + 0.03s) A:9455/R:13140 C:0	0.07s (0.06s + 0.01s) A:9455/R:13140	0.07s

A: number of atoms, C: number of clauses, R: number of ground rules

Figure 7.5: Comparison of DEC reasoner with F2LP + answer set solvers

- F2LP (v 1.11) with CLINGO (v 3.0.3 (CLASP v 1.3.5)).

F2LP turns an input theory into the languages of LPARSE and GRINGO, and LPARSE and GRINGO turn the result into a ground ASP program. CMODELS turns this ground program into a set of clauses and then invokes a SAT solver to compute answer sets, while CLASP computes answer sets using the techniques similar to those used in SAT solvers but without actually invoking them. CLINGO is a system that combines GRINGO and CLASP in a monolithic way.

The first five examples in Figure 7.5 are part of the benchmark problems from the work of Shanahan (1997, 1999). The next four are by Mueller (2006). (We increased timepoints to see more notable differences.) More examples can be found from the F2LP homepage. All experiments were done on a Pentium machine with 3.06 GHz CPU and 4GB RAM running 64 bit Linux. The reported run times are in seconds and were obtained using the Linux `time` command ("user time + sys time"), except for the DEC reasoner for which we recorded the times reported by the system. This was to avoid including the time spent by the DEC reasoner in producing output in a neat format, which sometimes takes non-negligible time. For the DEC reasoner, the times in parentheses are "(encoding time + SAT solving time)." For the others, they are the times spent by each of the grounder and the solver. CMODELS time includes the time spent in converting the ground program generated by LPARSE/GRINGO into a set of clauses, and calling the SAT solver. The time spent by F2LP in translating an event calculus description into an answer set program (with variables) is negligible for these problems.

The symbol '—' denotes that the system cannot solve the problem due to the limited expressivity. For instance, `BusRide` includes disjunctive event axioms, which results in a disjunctive program that cannot be handled by CLINGO. Similarly, the DEC reasoner cannot handle `BusRide` (disjunctive event axioms), `Commuter` (compound events) and `Walking Turkey` (effect constraints). As is evident from the experiments, the main reason for the better performance of the ASP-based approach is the efficient grounding mechanisms implemented in the ASP grounders. Though the DEC reasoner and CMODELS call the same SAT solver MINISAT, the number of atoms produced by the DEC reasoner is in general much smaller. This is because the DEC reasoner adopts an optimized encoding method (that is based on predicate completion) which avoids a large number of ground instances of atoms such as *Initiates*(e, f, t), *Terminates*(e, f, t), and

Problem (max. step)	DEC reasoner (MINISAT)	F2LP with GRINGO+ CMODELS	F2LP with GRINGO+ CLASP
ZooTest1 (16)	$>$ 2h	50.48s (6.66s + 43.82s) A:930483/R:2272288 C:3615955	29.01s (6.66s + 22.35s) A:153432/R:2271175
ZooTest2 (22)	$>$ 2h	159.51s (12.36s + 147.15s) A:2241512/R:4153670 C:8864228	210.55s (12.36s + 198.19s) A:219220/R:4152137
ZooTest3 (23)	$>$ 2h	142.68s (13.55s + 129.13s) A:2505940/R:4556928 C:9914568	196.63s (13.55s + 183.08s) A:230731/R:4555325

A: number of atoms, C: number of clauses, R: number of ground rules

Figure 7.6: Zoo World in DEC reasoner vs. Zoo World in F2LP + answer set solvers

Releases(e, f, t) (Mueller, 2004, Section 4.4). On the other hand, in several examples, the number of clauses generated by CMODELS is 0, which means that the answer sets were found without calling the SAT solver. This is because for these examples the unique answer set coincides with the well-founded model, which is efficiently computed by CMODELS in a preprocessing step before calling SAT solvers. Out of the 14 benchmark examples by Shanahan (1997, 1999), 10 of them belong to this case when LPARSE is used for grounding.

In the experiments in Figure 7.5, the solving times are negligible for most of the problems. We also experimented with some computationally hard problems, where solving takes more time than grounding. Figure 7.6 shows runs of a medium-size action domain, the Zoo World (Akman, Erdoğan, Lee, Lifschitz, & Turner, 2004). All the tests shown in the table are planning problems where "`max.step`" is the length of a minimal plan. The cut-off time was 2 hours and the DEC reasoner did not terminate within that time for any of the problems. In fact, the entire time was spent on encoding and the SAT solver was never called. On the other hand, the ASP grounder GRINGO took only a few seconds to ground the domain and, unlike in Figure 7.5, the solvers took much more time than the

grounder. As we can see, CMODELS with MINISAT performed better than CLASP on two of the problems. To check the time taken by MINISAT on the encoding generated by the DEC reasoner, we ran `ZooTest1` to completion. The DEC reasoner terminated after 116578.1 seconds (32.38 hours) and the time taken by MINISAT (solving time) was just 2 seconds.

7.6 Proofs

Proof of Theorem 21

Between (a) and (b): Follows immediately from Theorem 4.

Between (b) and (c): Note first that Ξ is equivalent to $\mathrm{SM}[\Xi; \emptyset]$. Since

- every strongly connected component in the dependency graph (Chapter 3.2) of $\Sigma \wedge \Delta$ relative to $\{I, T, R, H\}$ either belongs to $\{I, T, R\}$ or $\{H\}$,
- Σ is negative on $\{H\}$, and
- Δ is negative on $\{I, T, R\}$,

it follows from Theorem 12 that (b) is equivalent to

$$\mathrm{SM}[\Sigma \wedge \Delta;\ I, T, R, H] \wedge \mathrm{SM}[\Theta;\ Ab_1, \ldots, Ab_n] \wedge \mathrm{SM}[\Xi;\ \emptyset]$$

Similarly, applying Theorem 12 repeatedly, we can show that the above formula is equivalent to (c).

Between (c) and (d): By Proposition 1. □

Proof of Theorem 22

Assume that T is

$$\mathrm{CIRC}[\Sigma; \textit{Initiates}, \textit{Terminates}, \textit{Releases}] \wedge \mathrm{CIRC}[\Delta; \textit{Happens}]$$
$$\wedge\ \mathrm{CIRC}[\Theta; \textit{Ab}_1, \ldots, \textit{Ab}_n] \wedge \Xi,$$

which is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma; \textit{Initiates}, \textit{Terminates}, \textit{Releases}] \wedge \mathrm{SM}[\Delta; \textit{Happens}] \\ &\wedge \mathrm{SM}[\Theta; \textit{Ab}_1, \ldots, \textit{Ab}_n] \wedge \Xi\end{aligned} \tag{7.4}$$

by Theorem 21.

Let Ξ_{def} be the set of all definitions (7.2) in Ξ, and let Ξ' be the formula obtained from Ξ by applying Step 1. By Theorem 3, it follows that each formula (7.2) in Ξ_{def} is equivalent to

$$\mathrm{SM}[\forall \mathbf{x}(G' \to p(\mathbf{x}));\ p]$$

where G' is as described in Step 1. Consequently, (7.4) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma; \textit{Initiates}, \textit{Terminates}, \textit{Releases}] \wedge \mathrm{SM}[\Delta; \textit{Happens}] \\ &\wedge \mathrm{SM}[\Theta; \textit{Ab}_1, \ldots, \textit{Ab}_n] \wedge \bigwedge_{(7.2) \in \Xi_{def}} \mathrm{SM}[\forall \mathbf{x}(G' \to p(\mathbf{x}));\ p] \wedge \Xi''\end{aligned} \tag{7.5}$$

where Ξ'' is the conjunction of all the axioms in Ξ' other than the ones obtained from definitional axioms (7.2).

Applying Theorem 12 repeatedly, it follows that (7.5) is equivalent to

$$\begin{aligned}&\mathrm{SM}[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi'' \wedge \bigwedge_{(7.2) \in \Xi_{def}} \forall \mathbf{x}(G' \to p(\mathbf{x}));\\ &\textit{Initiates}, \textit{Terminates}, \textit{Releases}, \textit{Happens}, \textit{Ab}_1, \ldots, \textit{Ab}_n, \mathbf{p}]\ .\end{aligned} \tag{7.6}$$

According to the syntax of the event calculus reviewed in Section 7.1,

- every positive occurrence of a formula of the form $\exists y G(y)$ in (7.6) is contained in a subformula that is negative on $\{\textit{Initiates}, \textit{Terminates}, \textit{Releases}, \textit{Happens}, \textit{Ab}_1, \ldots, \textit{Ab}_n, \mathbf{p}\}$, and
- there are no negative occurrences of any formula of the form $\forall y G(y)$ in (7.6).

Consequently, the statement of the theorem follows from Theorem 16. □

Chapter 8

SITUATION CALCULUS IN ANSWER SET PROGRAMMING

In this chapter, we use the results from Chapters 3-5 to reformulate Lin's causal theories (Lin, 1995) and Basic Action Theories (BATs) (Reiter, 2001) in the first-order stable model semantics and in ASP. Based on the reformulations, we show how F2LP can be used to compute these theories, and discuss some of the advantages of this approach.

8.1 Lin's Causal Theories in Answer Set Programming

In this section, we show how Lin's causal theories can be reformulated in the first-order stable model semantics and in ASP. We also show how system F2LP can be used to compute these theories.

Review of Lin's Causal Theories

We assume a many-sorted first-order language which contains a *situation* sort, an *action* sort, a *fluent* sort, a *truth value* sort and an *object* sort. We do not consider functional fluents here for simplicity. Expression *Holds*$(P(\mathbf{x}), s)$ represents that fluent $P(\mathbf{x})$ is true in situation s, *Poss*(a, s) represents that action a can be executed in situation s, and *Caused*(f, v, s) represents that f is caused to have value v in situation s where $v \in \{$*true*, *false*$\}$.

According to Lin (1995), a formula $\phi(s)$ is called a *simple state formula about* s if $\phi(s)$ does not mention *Poss*, *Caused* or any situation term other than possibly the variable s.

We assume that a causal action theory $\mathcal{D}$ consists of a finite number of the following sets of axioms. We often identify $\mathcal{D}$ with the conjunction of the universal closures of all axioms in $\mathcal{D}$. In the following, F, F_i are fluent names, A is an action name, V, V_i are truth values, s, s' are situation variables, $\phi(s)$ is a simple state formula about s, symbols a, a' are action variables, f is a variable of sort fluent, v

is a variable of sort truth value, and $\mathbf{x}$, $\mathbf{x}_i$, $\mathbf{y}$, $\mathbf{y}_i$ are lists of variables.

- $\mathcal{D}_{caused}$ is a conjunction of axioms of the form

$$\textit{Poss}(A(\mathbf{x}), s) \rightarrow (\phi(s) \rightarrow \textit{Caused}(F(\mathbf{y}), V, do(A(\mathbf{x}), s)),$$

  (direct effect) and

$$\phi(s) \wedge \textit{Caused}(F_1(\mathbf{x}_1), V_1, s) \wedge \cdots \wedge \textit{Caused}(F_n(\mathbf{x}_n), V_n, s) \rightarrow \textit{Caused}(F(\mathbf{x}), V, s)$$

  (indirect effect).

- $\mathcal{D}_{poss}$ is a conjunction of precondition axioms of the form

$$\textit{Poss}(A(\mathbf{x}), s) \leftrightarrow \phi(s). \tag{8.1}$$

- $\mathcal{D}_{rest}$ is a conjunction of the following axioms:

  – The basic axioms:

$$\begin{aligned} &\textit{Caused}(f, \textit{true}, s) \rightarrow \textit{Holds}(f, s), \\ &\textit{Caused}(f, \textit{false}, s) \rightarrow \neg\textit{Holds}(f, s), \\ &\textit{true} \neq \textit{false} \wedge \forall v(v = \textit{true} \vee v = \textit{false}). \end{aligned} \tag{8.2}$$

  – The unique name assumptions for fluent names:

$$\begin{aligned} &F_i(\mathbf{x}) \neq F_j(\mathbf{y}), \quad (i \neq j) \\ &F_i(\mathbf{x}) = F_i(\mathbf{y}) \rightarrow \mathbf{x} = \mathbf{y}. \end{aligned} \tag{8.3}$$

    Similarly for action names.

  – The *foundational axioms* for the discrete situation calculus: [1]

$$s \neq do(a, s), \tag{8.4}$$

$$do(a, s) = do(a', s') \rightarrow (a = a' \wedge s = s'), \tag{8.5}$$

$$\forall p\Big(p(S_0) \wedge \forall a, s\big(p(s) \rightarrow p(do(a, s))\big) \rightarrow \forall s\, p(s)\Big). \tag{8.6}$$

[1] For simplicity we omit the two axioms regarding the partial-order among situations.

– The frame axiom:

$$\textit{Poss}(a, s) \rightarrow (\neg\exists v \textit{Caused}(f, v, do(a, s)) \\ \rightarrow (\textit{Holds}(f, do(a, s)) \leftrightarrow \textit{Holds}(f, s))).$$

– Axioms for other domain knowledge: $\phi(s)$.

A *causal action theory* is defined as

$$\mathrm{CIRC}[\mathcal{D}_{caused};\ \textit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}_{rest}. \tag{8.7}$$

For example, consider the Lin's suitcase example (Lin, 1995), wherein there is a suitcase with two locks L_1, L_2 and a spring loaded mechanism that opens the suitcase when both the locks are in the *Up* position. This can be represented as follows:

$\textit{Poss}(\textit{Flip}(x), s) \rightarrow (\textit{Holds}(\textit{Up}(x), s) \rightarrow \textit{Caused}(\textit{Up}(x), \textit{false}, \textit{do}(\textit{Flip}(x), s)))$

$\textit{Poss}(\textit{Flip}(x), s) \rightarrow (\neg\textit{Holds}(\textit{Up}(x), s) \rightarrow \textit{Caused}(\textit{Up}(x), \textit{true}, \textit{do}(\textit{Flip}(x), s)))$

$\textit{Holds}(\textit{Up}(L_1), s) \wedge \textit{Holds}(\textit{Up}(L_2), s) \rightarrow \textit{Caused}(\textit{Open}, \textit{true}, s)$

$\top \leftrightarrow \textit{Poss}(\textit{Flip}(x), s)$

The first two axioms represent the direct effects of the action *Flip*, and the third axiom is the indirect effect axiom representing that *Open* is caused to be true when both the locks are in the *Up* position. Finally, the last axiom represents that the *Flip* action can be executed in every situation.

Reformulating Lin's Causal Theories in the First-Order Stable Model Semantics

It is easy to check from the description above that $\mathcal{D}_{caused}$ is canonical relative to *Caused* (see Chapter 3.3 for definition of *canonical relative to* ***p***). This fact enables us to reformulate causal action theories in the first-order stable model semantics.

Let $\mathcal{D}_{poss\rightarrow}$ be the conjunction of axioms

$$\phi(s) \rightarrow \textit{Poss}(A(\mathbf{x}), s)$$

for each axiom (8.1) in $\mathcal{D}_{poss}$. Instead of the second-order axiom (8.6) we consider the following first-order formula $\mathcal{D}_{sit}$, which introduces a new intensional predicate constant *Sit*.[2]

$$\textit{Sit}(S_0) \wedge \forall a, s(\textit{Sit}(s) \rightarrow \textit{Sit}(do(a, s))) \wedge \neg\exists s \neg \textit{Sit}(s). \tag{8.8}$$

In the following, $\mathcal{D}^{-}_{rest}$ is the theory obtained from $\mathcal{D}_{rest}$ by dropping (8.6).

Theorem 23 *Given a causal action theory (8.7), the following theories are equivalent to each other when we disregard the auxiliary predicate Sit:*

(a) $CIRC[\mathcal{D}_{caused}; \textit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}_{rest}$*;*

(b) $SM[\mathcal{D}_{caused}; \textit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}^{-}_{rest} \wedge SM[\mathcal{D}_{sit}; \textit{Sit}]$*;*

(c) $SM[\mathcal{D}_{caused}; \textit{Caused}] \wedge SM[\mathcal{D}_{poss^{\rightarrow}}; \textit{Poss}] \wedge \mathcal{D}^{-}_{rest} \wedge SM[\mathcal{D}_{sit}; \textit{Sit}]$*;*

(d) $SM[\mathcal{D}_{caused} \wedge \mathcal{D}_{poss^{\rightarrow}} \wedge \mathcal{D}^{-}_{rest} \wedge \mathcal{D}_{sit}; \textit{Caused}, \textit{Poss}, \textit{Sit}]$*.*

The proof of the equivalence between (a) and (b) uses the theorem on canonical formulas (Theorem 4, Chapter 3.3). The equivalence between (b) and (c) follows from the theorem on completion (Ferraris et al., 2011), and the equivalence between (c) and (d) follows from the splitting theorem (Theorem 12, Chapter 4.1). A complete proof is given in the Appendix.

Reformulating Lin's Causal Theories in Answer Set Programming

Using translation F2LP, we can further turn the reformulation of Lin's causal theories shown above into the syntax of answer set programs.

Theorem 24 *Let $\mathcal{D}$ be a finite causal action theory (8.7) of signature σ that contains finitely many predicate constants, and let F be the FOL-representation of*

[2] Suggested by Vladimir Lifschitz (personal communication).

the program obtained by applying translation F2LP on

$$\mathcal{D}_{caused} \wedge \mathcal{D}_{poss\rightarrow} \wedge \mathcal{D}^{-}_{rest} \wedge \mathcal{D}_{sit} \tag{8.9}$$

with intensional predicates $\{$*Caused*, *Poss*, *Sit*$\}$*. Then* $\mathcal{D}$ *is* σ*-equivalent to* $SM[F]$*.*

In view of this theorem, system F2LP can be used to compute Lin's causal theories. The input to F2LP can be simplified if we are interested in Herbrand stable models (answer sets) only. We can drop axioms (8.2)–(8.5) as they are ensured by Herbrand models. Also, in order to ensure finite grounding, instead of $\mathcal{D}_{sit}$, we include the following set of rules $\Pi_{situation}$ in the input to F2LP.

```
nesting(0,s0).
nesting(L,S) & L < maxdepth -> nesting(L+1,do(A,S)).
               nesting(L,S) -> situation(S).
       nesting(maxdepth,S) ->  final(S).
```

$\Pi_{situation}$ is used to generate finitely many situation terms whose height is up to `maxdepth`, the value that can be given as an option in invoking GRINGO. Using the splitting theorem (Theorem 12, Chapter 4.1), it is not difficult to check that if a program Π containing the ASP-rules corresponding to $\Pi_{situation}$ has no occurrence of predicate `nesting` in any other rules and has no occurrence of predicate `situation` in the head of any other rules, then every answer set of Π contains atoms `situation(do(`a_m`,do(`a_{m-1}`,do(...,do(`a_1`,s0)))))` for all possible sequences of actions $a_1, \ldots, a_m$ for `m=0,...,maxdepth`. Though the ASP-representation of $\Pi_{situation}$ does not satisfy syntactic conditions, such as the ones corresponding to λ-restricted (Gebser, Schaub, & Thiele, 2007), ω-restricted (Syrjänen, 2004), or finite domain programs (Calimeri, Cozza, Ianni, & Leone, 2008), that answer set solvers usually impose in order to ensure finite grounding, the rules can still be finitely grounded by GRINGO Version 3.x, which

does not check such syntactic conditions.[3] It is not difficult to see why the ASP-rules corresponding to $\Pi_{situation}$ lead to finite grounding since we provide an explicit upper limit for the nesting depth of function `do`.

In addition to $\Pi_{situation}$, we use the following rules $\Pi_{executable}$ in order to represent the set of executable situations (Reiter, 2001):

```
executable(s0).
executable(S) & poss(A,S) & -final(S) &
              situation(S) & action(A) -> executable(do(A,S)).
```

Figure 8.1 shows an encoding of Lin's suitcase example (1995) in the language of F2LP (`h` is used to represent *Holds*), which describes a suitcase that has two locks and a spring loaded mechanism which will open the suitcase when both locks are up. This example illustrates how the ramification is handled in causal action theories. Since we fix the domain of situations to be finite, we require that actions not be effective in the final situations. This is done by introducing atom `final(S)`.

Consider the simple temporal projection problem by Lin (1995). Initially the first lock is down and the second lock is up. What will happen if the first lock is flipped? Intuitively, we expect both locks to be up and the suitcase to be open. We can verify this by using the combination of F2LP, GRINGO and CLASPD. First, we add $\Pi_{executable}$ and the following formulas to the theory in Figure 8.1. In order to check if the theory entails that flipping the first lock is executable, and that the suitcase is open after the flipping, we encode the negation of these facts in the last formula.

```
% initial situation
-h(up(l1),s0).
```

[3] Similarly, system DLV-COMPLEX allows us to turn off the finite domain checking (option `-nofdcheck`). This system was used in (Lee & Palla, 2010) .

```
% File 'suitcase'

% domain and domain variable declarations
lock(l1). lock(l2).
#domain lock(X).

fluent(up(X)). fluent(open).
action(flip(X)).
value(t). value(f).

#domain fluent(F).
#domain action(A).
#domain value(V).

% computing domain of situation
nesting(0,s0).
nesting(L,S) & L < maxdepth -> nesting(L+1,do(A,S)).
nesting(L,S) -> situation(S).
nesting(maxdepth,S) -> final(S).

% basic axioms
situation(S) & caused(F,t,S) -> h(F,S).
situation(S) & caused(F,f,S) -> -h(F,S).

% effect axioms (D_caused)
situation(S) & -final(S) & poss(flip(X),S) ->
      (h(up(X),S) -> caused(up(X),f,do(flip(X),S))).

situation(S) & -final(S) & poss(flip(X),S) ->
      (-h(up(X),S) -> caused(up(X),t,do(flip(X),S))).

% indirect effects
situation(S) & h(up(l1),S) & h(up(l2),S) -> caused(open,t,S).

% pre-conditions (D_poss)
situation(S) -> poss(flip(X),S).

% frame axioms
situation(S) & -final(S) & poss(A,S) ->
( -?[V]:caused(F,V,do(A,S)) ->
            (h(F,do(A,S)) <-> h(F,S)) ).

% Holds is non-intensional
situation(S) -> {h(F,S)}.
```

Figure 8.1: Lin's Suitcase example in F2LP

```
h(up(l2),s0).


% query
-(executable(do(flip(l1),s0)) & h(open,do(flip(l1),s0))).
```

We check the answer to the temporal projection problem by running the command:

```
$ f2lp suitcase | gringo -c maxdepth=1 | claspD
```

CLASPD returns no answer set as expected.

Now, consider a simple planning problem for opening the suitcase when the locks are initially down. We add $\Pi_{executable}$ and the following rules to the theory in Figure 8.1. The last rule encodes the goal.

```
% initial situation
-h(up(l1),s0).
-h(up(l2),s0).
-h(open,s0).


% goal
--( ?[S]: (executable(S) & h(open,S)) ).
```

When `maxdepth` is 1, the combined use of F2LP, GRINGO and CLASPD results in no answer sets, and when `maxdepth` is 2, it finds the unique answer set that contains both `h(open,do(flip(l2),do(flip(l1),s0)))` and `h(open,do(flip(l1),do(flip(l2),s0)))`, each of which encodes a plan. In other words, the single answer set encodes multiple plans in different branches of the situation tree, which allows us to combine information about the different branches in one model. This is an instance of hypothetical reasoning that is elegantly handled in the situation calculus due to its branching time structure.

Belleghem, Denecker, and Schreye (1997) note that the linear time structure of the event calculus is limited to handle such hypothetical reasoning allowed in the situation calculus.

8.2 Basic Action Theories in Answer Set Programming

In this section, we show how BATs can be reformulated in the first-order stable model semantics and in ASP. We also show how system F2LP can be used to compute these theories.

Review of Basic Action Theories

We understand $P(\mathbf{x}, s)$ where P is a fluent name, as shorthand for *Holds*$(P(\mathbf{x}), s)$, and do not consider functional fluents.

According to (Reiter, 2001), a formula is *uniform* in a situation term σ if it satisfies the following conditions:

- it does not contain any quantification over situations;
- it does not mention any variables for relational fluents;
- it does not mention any situation term other than σ;
- it does not mention any predicate that has a situation argument other than *Holds*;
- it does not mention any function constant that has a situation argument unless the function is a functional fluent.

Since we do not consider functional fluents, the last item above simply means that functions in uniform formulas do not have any situation argument.

A *basic action theory (BAT)* (Reiter, 2001) is of the form

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}, \tag{8.10}$$

where

- Σ is the conjunction of the foundational axioms (8.4) - (8.6);
- $\mathcal{D}_{ss}$ is a conjunction of successor state axioms of the form

$$F(\mathbf{x}, do(a, s)) \leftrightarrow \Phi_F(\mathbf{x}, a, s),$$

  where $\Phi_F(\mathbf{x}, a, s)$ is a formula that is *uniform* in s and whose free variables are among $\mathbf{x}, a, s$;
- $\mathcal{D}_{ap}$ is a conjunction of action precondition axioms of the form

$$\mathit{Poss}(A(\mathbf{x}), s) \leftrightarrow \Pi_A(\mathbf{x}, s),$$

  where $\Pi_A(\mathbf{x}, s)$ is a formula that is uniform in s and whose free variables are among $\mathbf{x}, s$;
- $\mathcal{D}_{una}$ is the conjunction of unique name axioms for fluents and actions;
- $\mathcal{D}_{S_0}$ is a conjunction of first-order formulas that are uniform in S_0.

Reformulating Basic Action Theories in the First-Order Stable Model Semantics

Similar to the reformulation of Lin's causal theories (Chapter 8.1), we use $\mathcal{D}_{sit}$ instead of the second-order axiom (8.6). The following theorem shows how a BAT (8.10) can be viewed under the first-order stable model semantics.

Theorem 25 *Given a BAT T, let T^- be the theory obtained by dropping (8.6) from T. Then, T is equivalent to the following theories if we disregard the predicate Sit:*

(a) $SM[\mathcal{D}_{sit}; Sit] \wedge T^-$*;*

(b) $SM[\mathcal{D}_{sit} \wedge T^-; Sit]$*.*

As before, the proof of the equivalence of (a) and (b) uses the splitting theorem (Theorem 12, Chapter 4.1).

In the rest of this section, we consider an alternative encoding of BAT in ASP, in which we do not need to provide explicit successor state axioms $\mathcal{D}_{ss}$. Instead, the successor state axioms are entailed by the effect axioms and the generic inertia axioms adopted in ASP by making both the positive predicate *Holds* and the negative predicate $\sim$*Holds* intensional (Chapter 3.2). In the following, we assume that the underlying signature contains both these predicates. Also, for simplicity, we disregard the second-order axiom (8.6).

An ASP-style BAT is of the form

$$\Sigma \cup \mathcal{D}_{effect} \cup \mathcal{D}_{precond} \cup \mathcal{D}_{inertia} \cup \mathcal{D}_{exogenous_0} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \tag{8.11}$$

where

- Σ, $\mathcal{D}_{una}$ and $\mathcal{D}_{S_0}$ are defined the same as before;
- $\mathcal{D}_{effect}$ is a conjunction of axioms of the form

  $$\gamma_R^+(\mathbf{x}, a, s) \rightarrow \textit{Holds}(R(\mathbf{x}), do(a, s)) \tag{8.12}$$

  or

  $$\gamma_R^-(\mathbf{x}, a, s) \rightarrow \sim\textit{Holds}(R(\mathbf{x}), do(a, s)), \tag{8.13}$$

  where $\gamma_R^+(\mathbf{x}, a, s)$ and $\gamma_R^-(\mathbf{x}, a, s)$ are formulas that are uniform in s and whose free variables are among $\mathbf{x}, a$ and s;
- $\mathcal{D}_{precond}$ is a conjunction of axioms of the form

  $$\pi_A(\mathbf{x}, s) \rightarrow \textit{Poss}(A(\mathbf{x}), s) \tag{8.14}$$

  where $\pi_A(\mathbf{x}, s)$ is a formula that is uniform in s and whose free variables are among $\mathbf{x}, s$;
- $\mathcal{D}_{inertia}$ is the conjunction of the axioms

  $$\textit{Holds}(R(\mathbf{x}), s) \wedge \neg\sim\textit{Holds}(R(\mathbf{x}), do(a, s)) \rightarrow \textit{Holds}(R(\mathbf{x}), do(a, s)),$$
  $$\sim\textit{Holds}(R(\mathbf{x}), s) \wedge \neg\textit{Holds}(R(\mathbf{x}), do(a, s)) \rightarrow \sim\textit{Holds}(R(\mathbf{x}), do(a, s))$$

for all fluent names R;

- $\mathcal{D}_{exogenous_0}$ is the conjunction of

$$\textit{Holds}(R(\mathbf{x}), S_0) \vee \sim \textit{Holds}(R(\mathbf{x}), S_0)$$

for all fluent names R.

Note that axioms in $\mathcal{D}_{inertia}$ are typically used in answer set programming to represent the common sense law of inertia (Lifschitz & Turner, 1999). Similarly, $\mathcal{D}_{exogenous_0}$ is used to represent that the initial value of a fluent is arbitrary.[4]

We will show how this ASP-style BAT is related to Reiter's BAT. First, since we use strong negation, it is convenient to define the following notions. For the signature σ of a given BAT, σ^{Holds} is the signature obtained by adding $\sim$*Holds* to σ. We say that an interpretation I of σ^{Holds} is *complete* on *Holds* if it satisfies

$$\forall \mathbf{y}(\textit{Holds}(\mathbf{y}) \vee \sim \textit{Holds}(\mathbf{y}))$$

where **y** is a list of distinct variables. Given an interpretation I of σ^{Holds}, expression $I|_\sigma$ denotes the projection of I on σ.

Let $\mathcal{D}_{ss}$ be the conjunction of successor state axioms

$$Holds(R(\mathbf{x}), do(a, s)) \leftrightarrow \Gamma_R^+(\mathbf{x}, a, s) \vee (Holds(R(\mathbf{x}), s) \wedge \neg\Gamma_R^-(\mathbf{x}, a, s))$$

where $\Gamma_R^+(\mathbf{x}, a, s)$ is the disjunction of $\gamma_R^+(\mathbf{x}, a, s)$ for all axioms (8.12) in $\mathcal{D}_{effect}$, and $\Gamma_R^-(\mathbf{x}, a, s)$ is the disjunction of $\gamma_R^-(\mathbf{x}, a, s)$ for all axioms (8.13) in $\mathcal{D}_{effect}$. By $\mathcal{D}_{ap}$ we denote the conjunction of axioms

$$\textit{Poss}(A(\mathbf{x}), s) \leftrightarrow \Pi_A(\mathbf{x}, s)$$

where $\Pi_A(\mathbf{x}, s)$ is the disjunction of $\pi_A(\mathbf{x}, s)$ for all axioms (8.14) in $\mathcal{D}_{precond}$.

[4] The axioms $\mathcal{D}_{inertia}$ and $\mathcal{D}_{exogenous_0}$ are also closely related to the translation of $\mathcal{C}$+ into nonmonotonic causal logic (Giunchiglia et al., 2004).

Theorem 26 *Let T be a theory (8.11) of signature σ^{Holds}, and I a coherent interpretation of σ^{Holds} that is complete on $Holds$. If I satisfies*

$$\neg\exists \boldsymbol{x}\, a\, s(\Gamma_R^+(\boldsymbol{x}, a, s) \wedge \Gamma_R^-(\boldsymbol{x}, a, s))$$

for every fluent name R, then I satisfies

$$SM[T;\ \mathit{Poss}, \mathit{Holds}, {\sim}\mathit{Holds}]$$

iff $I|_\sigma$ satisfies the BAT

$$\Sigma \wedge \mathcal{D}_{ss} \wedge \mathcal{D}_{ap} \wedge \mathcal{D}_{una} \wedge \mathcal{D}_{S_0}.$$

Reformulating Basic Action Theories in Answer Set Programming

Since a BAT T (not including the second-order axiom (8.6)) can be viewed as a first-order formula under the stable model semantics (with the list of intensional predicates being empty), it follows that F2LP can be used to turn $T \cup \Pi_{situation}$ into a logic program. As before, we focus on the ASP-style BAT.

Theorem 27 *Let T be a ASP-style BAT (8.11) of signature σ that contains finitely many predicate constants, and let F be the FOL-representation of the program obtained from T by applying translation F2LP with intensional predicates $\{\mathit{Holds}, {\sim}\mathit{Holds}, \mathit{Poss}\}$. Then $SM[T; \mathit{Holds}, {\sim}\mathit{Holds}, \mathit{Poss}]$ is σ-equivalent to $SM[F]$.*

Consider the "broken object" example discussed in (Reiter, 1991). An object is broken if it is fragile and someone drops it, or if a bomb next to it explodes. Also, it is no longer broken if a person repairs it. Figure 8.2 shows an encoding of the example in the language of F2LP. Consider the simple projection problem of determining if an object o, which is next to bomb b, is broken after the bomb explodes. We add $\Pi_{executable}$ and the following formulas to the theory in Figure 8.2.

```
% initial situation
-h(broken(o),s0) & h(fragile(o),s0) & h(nexto(b,o),s0).
-h(holding(p,o),s0) & -h(exploded(b),s0).

% query
-(executable(do(explode(b),s0)) & h(broken(o),do(explode(b),s0))).
```

```
$ f2lp broken | gringo -c maxdepth=1 | claspD
```

returns no answer set as expected.

```
% File: broken
% domains other than situations
person(p).    object(o).    bomb(b).

#domain person(R).
#domain object(Y).
#domain bomb(B).

fluent(holding(R,Y)).    fluent(nexto(B,Y)).    fluent(fragile(Y)).
fluent(broken(Y)).    fluent(exploded(B)).

action(drop(R,Y)).    action(explode(B)).    action(repair(R,Y)).

#domain fluent(F).    #domain action(A).

depth(0..maxdepth).    #domain depth(L).

% defining the situation domain
nesting(0,s0).
nesting(L,S) & L < maxdepth -> nesting(L+1,do(A,S)).
nesting(L,S) -> situation(S).
nesting(maxdepth,S) -> final(S).

% Effect Axioms
situation(S) & h(fragile(Y),S) & -final(S) ->
                                    h(broken(Y),do(drop(R,Y),S)).
situation(S) & h(nexto(B,Y),S) & -final(S) ->
                                    h(broken(Y),do(explode(B),S)).
situation(S) & -final(S) -> h(exploded(B),do(explode(B),S)).
situation(S) & -final(S) -> ~h(broken(Y),do(repair(R,Y),S)).
situation(S) & -final(S) -> ~h(holding(R,Y),do(drop(R,Y),S)).

% Action precondition axioms
h(holding(R,Y),S) & situation(S) -> poss(drop(R,Y),S).
situation(S) & -h(exploded(B),S) -> poss(explode(B),S).
situation(S) & h(broken(Y),S) -> poss(repair(R,Y),S).

% inertial axioms
h(F,S) & -~h(F,do(A,S)) & situation(S) & -final(S) -> h(F,do(A,S)).
~h(F,S) & -h(F,do(A,S)) & situation(S) & -final(S) -> ~h(F,do(A,S)).

% D_exogeneous_0
h(F,s0) | ~h(F,s0).

% Consider only those interpretations that are complete on Holds
-h(F,S) & -~h(F,S) & situation(S) -> false.
```

Figure 8.2: “Broken object” example in F2LP

8.3 Handling Recursive Axioms in the Situation Calculus

Chapter 7.4 shows an example of how event calculus theories can be enhanced with expressive ASP rules. Here, we show another advantage of ASP-based reasoning for circumscriptive action theories, particularly, the ability to handle recursive axioms.

Consider an extension of the Lin's suitcase example discussed in (Lin, 1995). In this extension, a new fluent `down`, which is the opposite of `up`, is introduced, and the following indirect effects of `flip` are added:

```
caused(up(X),t,S) & situation(S) -> caused(down(X),f,S).
caused(up(X),f,S) & situation(S) -> caused(down(X),t,S).
caused(down(X),t,S) & situation(S) -> caused(up(X),f,S).
caused(down(X),f,S) & situation(S) -> caused(up(X),t,S).
```

As mentioned in (Lin, 1995), Clark's completion cannot be applied to the resulting theory because of the recursion between the fluents `up` and `down`. As a result, the existing reasoners cannot handle these axioms. However, these axioms can be handled using our ASP-based approach. For example, if both the locks are down in the initial situation, one can verify that after flipping the first lock, the first lock is up and the second lock is down, by adding $\Pi_{executable}$ and the following formulas to the theory in Figure 8.1.

```
% fluent 'down'
fluent(down(X)).


% more indirect effects
caused(up(X),t,S) & situation(S) -> caused(down(X),f,S).
caused(up(X),f,S) & situation(S) -> caused(down(X),t,S).
caused(down(X),t,S) & situation(S) -> caused(up(X),f,S).
```

```
caused(down(X),f,S) & situation(S) -> caused(up(X),t,S).

% a lock has to be either in the up or down position
h(up(X),S) & h(down(X),S) & situation(S) -> false.
-h(up(X),S) & -h(down(X),S) & situation(S) -> false.

% initial situation
h(down(l1),s0).
h(down(l2),s0).

% query
-( executable(do(flip(l1),s0)) &
    h(up(l1),do(flip(l1),s0)) & h(down(l2),do(flip(l1),s0)) ).
```

```
$ f2lp suitcase | gringo -c maxdepth=1 | claspD
```

returns no answer set as expected.

8.4 Related Work

Prolog provides a natural implementation for basic action theories since definitional axioms can be represented by Prolog rules according to the Clark's theorem (Reiter, 2001, Chapter 5). The Lloyd-Topor transformation that is used to turn formulas into Prolog rules is similar to translation F2LP, but the difference is in that the former preserves the completion semantics and the latter the stable model semantics.

Lin and Wang (1999) describe a language that can be used to represent a syntactically restricted form of Lin's causal theories, called "clausal causal theories," which do not allow quantifiers. They show how to translate that language into answer set programs with strong negation, the answer sets of which are then used to obtain fully instantiated successor state axioms and action precondition

axioms. This is fundamentally different from our approach, which computes the propositional models of the full situation calculus theories directly.

Kautz and Selman (1992) introduced linear encodings that are similar to a propositionalized version of the situation calculus (McCarthy & Hayes, 1969). Lin (2003) introduced an action description language and presented a procedure to compile an action domain in that language into a complete set of successor state axioms, from which a STRIPS-like description can be extracted. This procedure is implemented in the system CCS[5]. The soundness of the procedure is shown with respect to a translation from action domain descriptions into Lin's causal theories. However, that procedure is based on completion and so, unlike our approach, cannot handle recursive axioms.

8.5 Proofs

Proof of Theorem 23

Between (a) ***and*** (b)***:*** Since $\mathcal{D}_{caused}$ is canonical relative to *Caused*, by Theorem 4, (a) is equivalent to

$$\mathrm{SM}[\mathcal{D}_{caused}; \textit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}^{-}_{rest} \wedge (8.6). \tag{8.15}$$

Consequently, it is sufficient to prove the claim that, under the assumption $\forall s\, Sit(s)$, formula (8.6) is equivalent to $\mathrm{SM}[\mathcal{D}_{sit}; \textit{Sit}]$.

First note that under the assumption, (8.6) can be equivalently rewritten as

$$\forall p\big(p(S_0) \wedge \forall a, s(p(s) \rightarrow p(do(a, s))) \rightarrow p = \textit{Sit}\big). \tag{8.16}$$

On the other hand, under $\forall s\, Sit(s)$, $\mathrm{SM}[\mathcal{D}_{sit}; \textit{Sit}]$ is equivalent to

$$\textit{Sit}(S_0) \wedge \forall a, s(\textit{Sit}(s) \rightarrow \textit{Sit}(do(a, s)))$$
$$\wedge\, \forall p\big(p < \textit{Sit} \rightarrow \neg(p(S_0) \wedge \forall a, s(p(s) \rightarrow p(do(a, s))) \wedge \forall a, s(\textit{Sit}(s) \rightarrow \textit{Sit}(do(a, s))))\big),$$

[5] http://www.cs.ust.hk/∼flin/ccp.html

which, under the assumption $\forall s\, Sit(s)$, is equivalent to

$$\forall p\big(p(S_0) \wedge \forall a, s(p(s) \rightarrow p(do(a,s))) \rightarrow \textit{Sit} \leq p\big)$$

and furthermore to (8.16).

Between (b) *and* (c)*:* Since $\phi(s)$ does not contain *Poss*, the equivalence follows from the relation between completion and the first-order stable model semantics (Theorem 3).

Between (c) *and* (d)*:* Since $\mathcal{D}_{caused}$ contains no strictly positive occurrence of *Poss* and $\mathcal{D}_{poss\rightarrow}$ contains no occurrence of *Caused*, every strongly connected component in the predicate dependency graph (Chapter 3.2) of $\mathcal{D}_{caused} \wedge \mathcal{D}_{poss\rightarrow}$ relative to $\{$*Caused*, *Poss*$\}$ either belongs to *Caused* or belongs to *Poss*. By Theorem 12, it follows that (c) is equivalent to

$$\mathrm{SM}[\mathcal{D}_{caused} \wedge \mathcal{D}_{poss\rightarrow}; \textit{Caused}, \textit{Poss}] \wedge \mathcal{D}^{-}_{rest} \wedge \mathrm{SM}[\mathcal{D}_{sit}; \textit{Sit}].$$

Similarly, applying Theorem 12 two more times, we get that the above formula is equivalent to (d). □

Proof of Theorem 24

Since (8.9) is almost universal relative to $\{$*Caused*, *Poss*, *Sit*$\}$, the result follows from Theorems 16 and 23. □

Proof of Theorem 25

As shown in the proof of Theorem 23, formula (8.6) is equivalent to $\mathrm{SM}[\mathcal{D}_{sit}; \textit{Sit}]$ under the assumption $\forall s\, Sit(s)$. So, T is equivalent to $\mathrm{SM}[\mathcal{D}_{sit}; \textit{Sit}] \wedge T^{-}$ if we disregard *Sit*.

Since *Sit* does not occur in T^{-}, and since $\mathcal{D}_{sit}$ contains no occurrence of any predicate other than *Sit*, the equivalence between (a) and (b) follows from Theorem 12. □

Proof of Theorem 26

Theory T is

$$\Sigma \wedge \mathcal{D}_{effect} \wedge \mathcal{D}_{precond} \wedge \mathcal{D}_{S_0} \wedge \mathcal{D}_{una} \wedge \mathcal{D}_{inertia} \wedge \mathcal{D}_{exogenous0},$$

and the corresponding BAT is

$$\Sigma \wedge \mathcal{D}_{ss} \wedge \mathcal{D}_{ap} \wedge \mathcal{D}_{S_0} \wedge \mathcal{D}_{una}.$$

Without loss of generality, we assume that T is already equivalently rewritten so that there is exactly one positive effect axiom and exactly one negative effect axiom for each fluent R, and that there is exactly one action precondition axiom for each action A.

Consider

$$\mathrm{SM}[\Sigma \wedge \mathcal{D}_{effect} \wedge \mathcal{D}_{precond} \wedge \mathcal{D}_{S_0} \wedge \mathcal{D}_{una} \wedge \mathcal{D}_{inertia} \wedge \mathcal{D}_{exogenous0};\ Poss, Holds, {\sim}Holds].$$

Since Σ and $\mathcal{D}_{una}$ are negative on the intensional predicates, the formula is equivalent to

$$\mathrm{SM}[\mathcal{D}_{effect} \wedge \mathcal{D}_{precond} \wedge \mathcal{D}_{S_0} \wedge \mathcal{D}_{inertia} \wedge \mathcal{D}_{exogenous0};\ Poss, Holds, {\sim}Holds] \wedge \Sigma \wedge \mathcal{D}_{una}. \tag{8.17}$$

Since $Poss$ does not occur in

$$\mathcal{D}_{effect} \wedge \mathcal{D}_{S_0} \wedge \mathcal{D}_{inertia} \wedge \mathcal{D}_{exogenous0},$$

and since $\mathcal{D}_{precond}$ is negative on $\{Holds, {\sim}Holds\}$, by Theorem 12, (8.17) is equivalent to

$$\begin{aligned} &\mathrm{SM}[\mathcal{D}_{effect} \wedge \mathcal{D}_{S_0} \wedge \mathcal{D}_{inertia} \wedge \mathcal{D}_{exogenous0};\ Holds, {\sim}Holds] \\ &\wedge \mathrm{SM}[\mathcal{D}_{precond}; Poss] \wedge \Sigma \wedge \mathcal{D}_{una}, \end{aligned} \tag{8.18}$$

which is equivalent to

$$\mathrm{SM}[\mathcal{D}_{effect} \wedge \mathcal{D}_{S_0} \wedge \mathcal{D}_{inertia} \wedge \mathcal{D}_{exogenous0};\ Holds, {\sim}Holds]$$
$$\wedge\ \mathcal{D}_{ap} \wedge \Sigma \wedge \mathcal{D}_{una}.$$

Therefore the statement of the theorem can be proven by showing the following: if

$$I \models \neg\exists \mathbf{x}\, a\, s(\Gamma^{+}_{R}(\mathbf{x}, \mathrm{a}, \mathrm{s}) \wedge \Gamma^{-}_{R}(\mathbf{x}, \mathrm{a}, \mathrm{s})) \tag{8.19}$$

for every fluent R, and

$$I \models \Sigma \tag{8.20}$$

then I satisfies

$$\mathrm{SM}[\mathcal{D}_{S_0} \wedge \mathcal{D}_{exogenous0} \wedge \mathcal{D}_{effect} \wedge \mathcal{D}_{inertia};\ Holds, {\sim}Holds] \tag{8.21}$$

iff $I|_{\sigma}$ satisfies

$$\mathcal{D}_{S_0} \wedge \mathcal{D}_{ss}.$$

From $\mathcal{D}_{exogenous0}$, it follows that (8.21) is equivalent to

$$\mathrm{SM}[\mathcal{D}^{\neg\neg}_{S_0} \wedge \mathcal{D}_{exogenous0} \wedge \mathcal{D}_{effect} \wedge \mathcal{D}_{inertia};\ Holds, {\sim}Holds], \tag{8.22}$$

where $\mathcal{D}^{\neg\neg}_{S_0}$ is the formula obtained from $\mathcal{D}_{S_0}$ by prepending $\neg\neg$ to all occurrences of *Holds*. Under the assumption (8.20),

$$\mathcal{D}^{\neg\neg}_{S_0} \wedge \mathcal{D}_{exogenous0} \wedge \mathcal{D}_{effect} \wedge \mathcal{D}_{inertia}$$

is *atomic-tight* w.r.t. I on *Holds*, [6] so that by the relationship between completion and SM that is stated in Corollary 3 of (Lee & Meng, 2011), we have that $I \models (8.22)$ iff I satisfies $\mathcal{D}_{S_0}$, and, for each fluent R,

$$Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s})) \leftrightarrow \Gamma^{+}_{R}(\mathrm{x}, \mathrm{a}, \mathrm{s}) \vee (Holds(R(\mathrm{x}, \mathrm{s}) \wedge \neg\, {\sim}Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s})))) \tag{8.23}$$

[6] See Section 4.1 of (Lee & Meng, 2011) for the definition.

and

$$\sim Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s})) \leftrightarrow \Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s}) \vee (\sim Holds(R(\mathrm{x}), \mathrm{s}) \wedge \neg Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s}))) \tag{8.24}$$

where x, a, s are any (lists of) object names of corresponding sorts.

It remains to show that, under the assumption (8.19), I satisfies $(8.23) \wedge (8.24)$ iff $I|_\sigma$ satisfies

$$Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s})) \;\leftrightarrow\; \Gamma_R^+(\mathrm{x}, \mathrm{a}, \mathrm{s}) \vee (Holds(R(\mathrm{x}), \mathrm{s}) \wedge \neg \Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})). \tag{8.25}$$

In the following we will use the following facts.

- $I \models \sim Holds(R(\mathrm{x}), \mathrm{s})$ iff $I|_\sigma \not\models Holds(R(\mathrm{x}), \mathrm{s})$.
- if F is a ground formula that does not contain $\sim$, then $I \models F$ iff $I|_\sigma \models F$.

Left to Right: Assume $I \models (8.23) \wedge (8.24)$.

- *Case 1:* $I|_\sigma \models Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s}))$. Clearly, $I \models Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s}))$, so that, from (8.23), there are two subcases to consider.
  - *Subcase 1:* $I \models \Gamma_R^+(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Clearly, $I|_\sigma$ satisfies both LHS and RHS of (8.25).
  - *Subcase 2:* $I \models Holds(R(\mathrm{x}), \mathrm{s})$. From (8.24), it follows that $I \not\models \Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})$, and consequently, $I|_\sigma \not\models \Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Clearly, $I|_\sigma$ satisfies both LHS and RHS of (8.25).
- *Case 2:* $I|_\sigma \not\models Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s}))$. It follows from (8.23) that $I \not\models \Gamma_R^+(\mathrm{x}, \mathrm{a}, \mathrm{s})$, which is equivalent to saying that $I|_\sigma \not\models \Gamma_R^+(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Also since $I \models \sim Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s}))$, from (8.24), there are two subcases to consider.

- *Subcase 1:* $I \models \Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Clearly, $I|_\sigma$ satisfies neither LHS nor RHS of (8.25).
- *Subcase 2:* $I \models \sim Holds(R(\mathrm{x}), \mathrm{s})$. This is equivalent to saying that $I|_\sigma \not\models Holds(R(\mathrm{x}), \mathrm{s})$. Clearly, $I|_\sigma$ satisfies neither LHS nor RHS of (8.25).

Right to Left: Assume $I|_\sigma \models (8.25)$.

- *Case 1:* $I \models Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s}))$. It follows from (8.25) that $I|_\sigma$ satisfies RHS of (8.25), so that there are two subcases to consider.
  - *Subcase 1:* $I|_\sigma \models \Gamma_R^+(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Clearly, I satisfies both LHS and RHS of (8.23). Also from (8.19), it follows that $I \not\models \Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Consequently, I satisfies neither LHS nor RHS of (8.24).
  - *Subcase 2:* $I|_\sigma \models Holds(R(\mathrm{x}), \mathrm{s}) \wedge \neg\Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Clearly, I satisfies both LHS and RHS of (8.23). Since $I \not\models \Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})$, I satisfies neither LHS nor RHS of (8.24).
- *Case 2:* $I \models \sim Holds(R(\mathrm{x}), do(\mathrm{a}, \mathrm{s}))$. It follows from (8.25) that $I|_\sigma \not\models \Gamma_R^+(\mathrm{x}, \mathrm{a}, \mathrm{s})$, and $I|_\sigma \not\models (Holds(R(\mathrm{x}), \mathrm{s}) \wedge \neg\Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s}))$. From the latter, consider the two subcases.
  - *Subcase 1:* $I|_\sigma \not\models Holds(R(\mathrm{x}), \mathrm{s})$. Clearly, I satisfies neither LHS nor RHS of (8.23), and satisfies both LHS and RHS of (8.24).
  - *Subcase 2:* $I|_\sigma \not\models \neg\Gamma_R^-(\mathrm{x}, \mathrm{a}, \mathrm{s})$. Clearly, I satisfies neither LHS nor RHS of (8.23), and satisfies both LHS and RHS of (8.24).

□

Proof of Theorem 27

From $\mathcal{D}_{exogenous_0}$, it follows that $\mathrm{SM}[T; \mathit{Holds}, {\sim}\mathit{Holds}, \mathit{Poss}]$ is equivalent to $\mathrm{SM}[T^{\neg\neg}; \mathit{Holds}, {\sim}\mathit{Holds}, \mathit{Poss}]$, where $T^{\neg\neg}$ is obtained from T by prepending $\neg\neg$ to all occurrences of *Holds* in $\mathcal{D}_{S_0}$. From the definition of a uniform formula (Reiter, 2001), it follows that $T^{\neg\neg}$ is almost universal relative to $\{\mathit{Holds}, {\sim}\mathit{Holds}, \mathit{Poss}\}$. The result follows from Theorem 16. □

Chapter 9

TEMPORAL ACTION LOGICS IN ANSWER SET PROGRAMMING

In this chapter, we use the results from Chapters 3-5 to reformulate Temporal Action Logics (TAL) in the first-order stable model semantics and in ASP. Based on the reformulation, we show how F2LP can be used to compute descriptions in TAL. We then show how constraint answer set solvers, which are answer set solvers enhanced with constraint processing techniques, can be used for reasoning with TAL, and discuss some of the advantages of this approach. Finally, we compare our ASP-based reasoner with VITAL[1], which is a well-known tool for reasoning about actions using TAL.

9.1 Review of Temporal Action Logics

This review is based on (Doherty & Kvarnström, 2008) and Chapter 2 of (Kvarnström, 2005).

A narrative in TAL is specified using the language $\mathcal{L}$(ND), which is referred to as the *surface language*. The language is many-sorted, consisting of a number of value sorts $\mathcal{V}_i$, a number of feature sorts $\mathcal{F}_i$, an action sort $\mathcal{A}$, and a temporal sort $\mathcal{T}$. The boolean sort $\mathcal{B}$ is a value sort with constants $\{$*true*, *false*$\}$. Each feature sort is associated with a value sort such that $dom(\mathcal{F}_i)$=$\mathcal{V}_j$ for some j.

A *temporal term*, often denoted by t or t_i or t', is a variable, or a constant, or an expression of the form $t_1 + t_2$, all of sort $\mathcal{T}$. A *value term*, often denoted by ω or ω_i, is a variable or constant of some value sort $\mathcal{V}_i$, an expression *value*(t, f) where f is a fluent term, or an expression *g* $(\omega_1, \ldots, \omega_n)$ where *g*:$\mathcal{V}_1 \times \cdots \times \mathcal{V}_n \mapsto \mathcal{V}_i$ is a value function symbol. The function *value*(t, f) returns the value of the fluent f at timepoint t.

[1] http://www.ida.liu.se/~jonkv/vital

A *fluent term* is a feature variable or a feature expression of the form $f(\omega_1, \ldots, \omega_n)$ where $f : \mathcal{V}_1 \times \cdots \times \mathcal{V}_n \mapsto \mathcal{F}_i$ is a feature symbol. An *action term* is an expression of the form $A(\omega_1, \ldots, \omega_n)$ where $A : \mathcal{V}_1 \times \cdots \times \mathcal{V}_n \mapsto \mathcal{A}$ is an action symbol. In the following, f and g are fluent terms, and a is an action term.

A *temporal formula* is any comparison between two temporal terms. A *value formula* is of the form $\omega_1 = \omega_2$, or $r(\omega_1, \ldots, \omega_n)$ where $r : \mathcal{V}_1 \times \cdots \times \mathcal{V}_n$ is a relation symbol. An *elementary fluent formula* is an expression of the form $f \hat{=} \omega$. A *fluent formula* is an elementary fluent formula or a combination of fluent formulas formed with the standard logical connectives and quantification over values. The formulas $f \hat{=} \textit{true}$ and $f \hat{=} \textit{false}$ can be abbreviated as f and $\neg f$ respectively.

A *timed formula* is any of the following, where α is a fluent formula:

- *fixed fluent formula*: $[t, t']\alpha, (t, t']\alpha, [t, t')\alpha, (t, t')\alpha, [t, \infty)\alpha, (t, \infty)\alpha$, and $[t]\alpha$.
- *fluent change formula*: $C_T([t]\alpha), C_F([t]\alpha)$, and $C([t]\alpha)$.
- *reassignment formulas*: $R([t, t']\alpha), R((t, t']\alpha), R([t, t')\alpha), R((t, t')\alpha)$, and $R([t]\alpha)$.
- *interval reassignment formulas*: $I([t, t']\alpha), I((t, t']\alpha), I([t, t')\alpha), I((t, t')\alpha)$, and $I([t]\alpha)$.
- *occlusion formulas*: $X([t, t']\alpha), X((t, t']\alpha), X([t, t')\alpha), X((t, t')\alpha)$, and $X([t]\alpha)$.

A *static formula* is a temporal formula, a value formula, a fixed fluent formula, a fluent change formula, $\top$, $\bot$, or a combination of static formulas formed using the standard logical connectives together with quantification over values and time. A *change formula* is a formula of the form $Q\overline{v}(\alpha_1 \vee \cdots \vee \alpha_n)$ where $Q\overline{v}$ is a sequence of quantifiers with variables, and each α_i is a conjunction of static,

occlusion and reassignment formulas. A change formula is *balanced* if it satisfies the following conditions:

- whenever a fluent term *f* $(\omega_1, \ldots, \omega_m)$ appears inside a reassignment or occlusion formula in one of the disjuncts α_i, it must also appear every other α_j inside a reassignment or occlusion formula with exactly the same temporal argument.
- any existentially quantified variable v in the formula, whenever appearing inside a reassignment or occlusion formula, only does so in a formula $f \hat{=} v$.

An *application formula* is (a) a balanced change formula, or (b) a formula of the form $F \rightarrow G$ where F is a static formula and G is a balanced change formula, or a combination of formulas of the form (a) and (b) formed with conjunction and universal quantification over values and time. An *occurrence formula* is of the form $[t, t']\Psi$ where Ψ is an action term. Finally, a *persistence formula* is an expression of the form $Per(t, f)$, or $Dur(t, f, \omega)$, or a combination of persistence formulas formed with conjunction and universal quantification over values and time.

A TAL narrative is formed using the following statements:

- *action type specification* (labeled **acs**): $[t, t']\Psi \rightarrow \phi$ where Ψ is an action term and ϕ is an application formula.
- *dependency constraint* (labeled **dep**): an application formula.
- *domain constraint* (labeled **dom**): a static formula.
- *persistence statement* (labeled **per**): a persistence formula.
- *observation statement* (labeled **obs**): a static formula.

- *action occurrence statement* (labeled **occ**): a variable-free occurrence formula.

Reasoning about a narrative is done by translating it into the base language $\mathcal{L}$(FL), which is an order-sorted classical first-order language with equality using a linear discrete time structure. The language uses the ternary predicates *Holds* and *Occurs*, and the binary predicate *Occlude*. *Holds*(t, f, ω) represents that fluent f has value ω at time t. *Occlude*(t, f) represents that a persistent or durational fluent f is exempt from inertia or default value assumption, respectively, at time t. *Occurs*(t, t', a), represents that action a occurs during the time interval $[t, t']$. The translation from $\mathcal{L}$(ND) to $\mathcal{L}$(FL) is given by the function $Trans$, that is defined as follows ($\overline{\omega}$ represents a list of value terms, $\odot \in \{\wedge, \vee, \rightarrow\}$ and $Q \in \{\forall, \exists\}$):

- $Trans(\top) = \top$; $Trans(\bot) = \bot$;
- $Trans([t]f(\overline{\omega})\hat{=}\omega) =$ *Holds*$(t, f(\overline{\omega}), \omega)$;
  $Trans(X([t]f(\overline{\omega})\hat{=}\omega)) =$ *Occlude*$(t, f(\overline{\omega}))$;
  $Trans([t, t']a) =$ *Occurs*(t, t', a);
- $Trans(Per(t', f))$ =
  $\forall t(t' = t + 1 \wedge \neg$*Occlude*$(t + 1, f) \rightarrow \forall v($*Holds*$(t + 1, f, v) \leftrightarrow$ *Holds*$(t, f, v)))$;
  $Trans(Dur(t, f, \omega)) = \neg$*Occlude*$(t, f) \rightarrow$ *Holds*(t, f, ω);
- $Trans(\alpha \odot \beta) = Trans(\alpha) \odot Trans(\beta)$; similar for $\neg$;
  $Trans(Qv(\alpha)) = QvTrans(\alpha)$;
- $Trans([t]\ \alpha \odot \beta) = Trans([t]\alpha) \odot Trans([t]\beta)$; similar for $\neg$ and quantifiers;
- $Trans(X([t]\ \alpha \odot \beta)) = Trans(X([t]\alpha)) \wedge Trans(X([t]\beta))$;
  $Trans(X([t]\ \neg\alpha)) = Trans(X([t]\ \alpha))$;
  $Trans(X([t]\ Qv(\alpha))) = \forall vTrans(X([t]\alpha))$;
- $Trans([t_1, t_2]\ \alpha) = \forall t(t \geq t_1 \wedge t \leq t_2 \rightarrow Trans([t]\ \alpha))$;
  $Trans([t_1, \infty)\ \alpha) = \forall t(t \geq t_1 \rightarrow Trans([t]\ \alpha))$;

$Trans(R([t_1, t_2]\ \alpha)) = Trans(X([t_1, t_2]\ \alpha)) \wedge Trans([t_2]\ \alpha)$;

$Trans(R([t]\ \alpha)) = Trans(X([t]\ \alpha)) \wedge Trans([t]\ \alpha)$;

$Trans(I([t_1, t_2]\ \alpha)) = Trans(X([t_1, t_2]\ \alpha)) \wedge Trans([t_1, t_2]\ \alpha)$;

$Trans(I([t]\ \alpha)) = Trans(X([t]\ \alpha)) \wedge Trans([t]\ \alpha)$;

other forms of intervals are treated similarly;

- $Trans(C_T([t']\ \alpha)) = \forall t(t' = t + 1 \rightarrow Trans([t]\neg\alpha) \wedge Trans([t']\ \alpha)$;

  $Trans(C_F([t']\ \alpha)) = \forall t(t' = t + 1 \rightarrow Trans([t]\alpha) \wedge Trans([t']\ \neg\alpha)$;

  $Trans(C([t]\ \alpha)) = Trans(C_T([t]\ \alpha)) \vee Trans(C_F([t]\ \alpha))$

Consider any narrative $\mathcal{N}$ and let $\mathcal{N}_{per}$, $\mathcal{N}_{obs}$, $\mathcal{N}_{occ}$, $\mathcal{N}_{acs}$, $\mathcal{N}_{domc}$, and $\mathcal{N}_{depc}$ denote the sets of persistence statements, observation statements, action occurrence statements, action type specifications, domain constraints, and dependency constraints in $\mathcal{N}$ respectively. The TAL domain description (referred to as *preferred narrative*) $\Delta_{\mathcal{N}}$ is given by

$$\Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc} \wedge \mathrm{CIRC}[\Gamma_{occ}; \textit{Occurs}] \wedge \mathrm{CIRC}[\Gamma_{depc} \wedge \Gamma_{acs}; \textit{Occlude}] \tag{9.1}$$

where Γ_{per}, Γ_{obs}, Γ_{occ}, Γ_{acs}, Γ_{domc}, and Γ_{depc} are the formulas in $\mathcal{L}$(FL) (first-order formulas) obtained by applying $Trans$ on $\mathcal{N}_{per}$, $\mathcal{N}_{obs}$, $\mathcal{N}_{occ}$, $\mathcal{N}_{acs}$, $\mathcal{N}_{domc}$, and $\mathcal{N}_{depc}$ respectively; Γ_{fnd} is the set of foundational axioms in $\mathcal{L}$(FL), containing unique name axioms, unique value axioms, etc.; and Γ_{time} is the axiomatization of the particular temporal structure (linear and discrete) in $\mathcal{L}$(FL).

9.2 Reformulating Temporal Action Logics in the First-Order Stable Model Semantics

Like in the cases of the event calculus and the situation calculus, this reformulation uses the concepts of canonical formulas and splitting, which were presented in Chapters 3.3 and 4 respectively.

Given a narrative $\mathcal{N}$ and the corresponding preferred narrative $\Delta_{\mathcal{N}}$, we can conclude the following:

(ob1) all occurrences of *Occlude* in Γ_{acs} and Γ_{depc} are strictly positive, and there are no strictly positive occurrences of *Occurs* in either of them;

(ob2) all occurrences of *Occurs* in Γ_{occ} are strictly positive, and there are no occurrences of *Occlude* in it;

(ob3) Γ_{domc}, Γ_{obs}, Γ_{fnd}, and Γ_{time} do not contain any occurrences of either *Occlude* or *Occurs*; and

(ob4) all occurrences of *Occlude* in Γ_{per} are in the scope of negation, and there are no occurrences of *Occurs* in it.

These observations lead us to the following theorem. In the following, we use Γ_{ncirc} to denote $\Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc}$.

Theorem 28 *Given a TAL narrative $\mathcal{N}$ and the corresponding preferred narrative $\Delta_{\mathcal{N}}$ ((9.1)), the following formulas are logically equivalent:*

(a) $CIRC[\Gamma_{occ}; \textit{Occurs}] \wedge CIRC[\Gamma_{depc} \wedge \Gamma_{acs}; \textit{Occlude}] \wedge \Gamma_{ncirc}$

(b) $SM[\Gamma_{occ}; \textit{Occurs}] \wedge SM[\Gamma_{depc} \wedge \Gamma_{acs}; \textit{Occlude}] \wedge \Gamma_{ncirc}$

(c) $SM[\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}; \textit{Occurs}, \textit{Occlude}]$

The equivalence between (a) and (b) follows from Theorem 4, and observations (ob1) and (ob2) above which imply that Γ_{occ} is canonical relative to *Occurs* and $\Gamma_{depc} \wedge \Gamma_{acs}$ is canonical relative to *Occlude*. The equivalence between (b) and (c) follows from Theorem 12, and observations (ob1)-(ob4) above which imply that the conditions for applying the splitting theorem (Chapter 4.1) are satisfied.

Similar to the cases of the event calculus and the situation calculus, we use translation F2LP to turn formula $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$, which is the formula within SM in Theorem 28(c) above, into the syntax of answer set programs.

Consider $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$, where Γ_{ncirc} is $\Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc}$. The only intensional predicates in these formulas are *Occurs* and *Occlude*. Among the formulas, only Γ_{depc}, Γ_{acs}, and Γ_{per} contain occurrences of *Occlude*, and only Γ_{acs} and Γ_{occ} contain occurrences of *Occurs*. From the definitions of an application formula and $Trans$, it follows that *Occlude* is outside the scope of any positive occurrence of $\exists$ and any negative occurrence of $\forall$ in $\Gamma_{depc} \wedge \Gamma_{acs}$. Further, it is clear that all occurrences of *Occlude* in Γ_{per} are in the scope of negation, and that *Occurs* is outside the scope of any positive occurrence of $\exists$ and any negative occurrence of $\forall$ in $\Gamma_{acs} \wedge \Gamma_{occ}$. This implies that $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$ is almost universal relative to $\{$*Occurs*, *Occlude*$\}$, which in turn implies that translation F2LP can be used to turn the formula into the syntax of ASP. The following theorem makes this precise.

Theorem 29 *Let F be the formula $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$ of a signature σ, and let F' be the FOL-representation of the program obtained from F by applying translation F2LP with $\{$Occurs, Occlude$\}$ as the list of intensional predicates. Then, $SM[F;$ Occurs, Occlude$]$ is σ-equivalent to $SM[F']$.*

Consider the *Russian Airplane Hijack* (RAH) scenario as discussed in (Doherty & Kvarnström, 2008).This example demonstrates the capability of TAL in handling the ramification and qualification problems. In this scenario, three businessmen Boris, Erik, and Dimiter try to board a flight to Stockholm. Boris has a comb and a gun in his pocket, Dimiter is drunk, and Erik has a comb in his pocket.

When the businessmen move, the items in their pockets should move along with them. Similarly, when a plane flies from one location to another, the people inside should move along with the plane. Since Dimiter is drunk, he may not be able to board the plane. Finally, since Boris has a gun in his pocket, he should not be able to board the plane. The (partial) solutions provided to the ramification and qualification problems in TAL use dependency constraints. Following is part of the description provided with VITAL[2] (v. 2.999.910 alpha), which is a well-known tool for reasoning about action and change using TAL. We only show the action type specifications and dependency constraints used in the description. These statements represent the direct effects, the ramifications, and the qualifications associated with the problem. The fluent *PossBoard*$(person, airplane)$ is durational with default value *true*, and the rest of the fluents are persistent. The direct effects are represented by the following action type specifications:

acs $[t_1, t_2]$*Put*$(person, pthing, pocket) \rightarrow ([t_1]$ *Loc*$(person)\hat{=}$*Loc*$(pthing) \rightarrow$
$R((t1, t2]$ *InPocket*$(person, pthing)))$

acs $[t_1, t_2]$*Travel*$(person, loc_1, loc_2) \rightarrow ([t_1]$ *Loc*$(person)\hat{=}loc_1 \rightarrow$
$R([t_2]$ *Loc*$(person)\hat{=}loc_2))$

acs $[t_1, t_2]$*Board*$(person, airplane) \rightarrow$
$([t_1]$ *PossBoard*$(person, airplane) \wedge$ *Loc*$(person)\hat{=}$*Airport* $\rightarrow$
$R([t_2]$ *Loc*$(person)\hat{=}$*Loc*$(airplane) \wedge$ *OnPlane*$(airplane, person)))$

acs $[t_1, t_2]$*Fly*$(airplane, runway_1, runway_2) \rightarrow$
$([t_1]$ *Loc*$(airplane)\hat{=}runway_1 \rightarrow$
$I((t_1, t_2)$ *Loc*$(airplane)\hat{=}$*Air*$) \wedge R([t_2]$ *Loc*$(airplane)\hat{=}runway_2))$

The qualification constraints are represented by the following dependency constraints:

dep $[t]$*InPocket*$(person, gun) \rightarrow I([t]\ \forall airplane(\neg$*PossBoard*$(person, airplane)))$

dep $[t]$*Drunk*$(person) \rightarrow X([t]\ \forall airplane(\neg$*PossBoard*$(person, airplane)))$

[2]http://www.ida.liu.se/ jonkv/vital/

The ramification constraints are represented by the following dependency constraints:

dep $[t]$*OnPlane*$(airplane, person) \wedge C_T([t]$ *Loc*$(airplane) \hat{=} loc_3) \rightarrow$
$R([t]$ *Loc*$(person) \hat{=}$ *Loc*$(airplane))$

dep $[t]$*InPocket*$(person, pthing) \wedge C_T([t]$ *Loc*$(person) \hat{=} loc_3) \rightarrow$
$R([t]$ *Loc*$(pthing) \hat{=}$ *Loc*$(person))$

To compute the description using answer set solvers, we use system F2LP. Since F2LP does not accept input in the language of $\mathcal{L}$(ND), we need to encode the corresponding first-order theory directly. Figure 9.1 shows a complete F2LP encoding of the corresponding first-order theory.

The following command can be used to compute the answer sets of the theory:

```
$ f2lp RAH | gringo -c maxstep=17 | claspD
```

The answer sets produce the expected results, including the following:

- `holds(t,loc(boris),airport)` for all `t` ≥ 10 : since `boris` has a gun in his pocket, he is unable to board the plane.
- `holds(16,loc(sas609),run609b), holds(16,onplane(sas609,erik),true), holds(16,loc(erik),run609b), holds(16,inpocket(erik,comb2),true), holds(16,loc(comb2),run609b)`: since `erik` is on the plane and `comb2` is in his pocket, the location of `erik` and `comb2` is the same as the location of the plane, which is at the destination runway.
- some of the answer sets contain `holds(16,loc(dimiter),airport)` and others contain `holds(16,loc(dimiter),run609b)`: since `dimiter` is drunk, he may or may not be able to board the plane.

```
% file RAH

% domain specification
time(0..maxstep).
airplane(sas609). person(boris;dimiter;erik).
pthing(gun;comb1;comb2;comb3). runway(run609;run609b).
location(home1;home2;home3;office;airport;air).
pocket(pocket1;pocket2;pocket3). bool(true;false).

#domain time(T). #domain time(T1). #domain time(T2).
#domain airplane(Ai). #domain airplane(Ai1). #domain airplane(Ai2).
#domain person(Pe). #domain person(Pe1). #domain person(Pe2).
#domain pthing(Pt).
#domain runway(Ru). #domain runway(Ru1). #domain runway(Ru2).
#domain pocket(Po). #domain bool(Bo).

location(Ru).
#domain location(Lo). #domain location(Lo1). #domain location(Lo2).

thing(Ai). thing(Pe). thing(Pt).
#domain thing(Th).

value(Po). value(Bo). value(Th). value(Lo).
#domain value(V). #domain value(V1). #domain value(V2).

% fluents
feature(loc(Th);inpocket(Pe,Pt);poss_board(Pe,Ai);drunk(Pe)).
feature(onplane(Ai,Pe)).
#domain feature(Fe).

%poss_board is durational with default value true
-occlude(T,poss_board(Pe,Ai)) -> holds(T,poss_board(Pe,Ai),true).

%the rest of them are persistent
-occlude(T+1,loc(Th)) & T < maxstep ->
![Lo]:(holds(T+1,loc(Th),Lo) <-> holds(T,loc(Th),Lo)).

-occlude(T+1,inpocket(Pe,Pt)) & T < maxstep ->
![Bo]:(holds(T+1,inpocket(Pe,Pt),Bo) <-> holds(T,inpocket(Pe,Pt),Bo)).

-occlude(T+1,drunk(Pe)) & T < maxstep ->
![Bo]:(holds(T+1,drunk(Pe),Bo) <-> holds(T,drunk(Pe),Bo)).

-occlude(T+1,onplane(Ai,Pe)) & T < maxstep ->
![Bo]:(holds(T+1,onplane(Ai,Pe),Bo) <-> holds(T,onplane(Ai,Pe),Bo)).
```

```
% Unique Value Axioms
1{holds(T,loc(Th),Loc):location(Loc)}1.
1{holds(T,inpocket(Pe,Pt),Boo):bool(Boo)}1.
1{holds(T,poss_board(Pe,Ai),Boo):bool(Boo)}1.
1{holds(T,drunk(Pe),Boo):bool(Boo)}1.
1{holds(T,onplane(Ai,Pe),Boo):bool(Boo)}1.

% actions
action(put(Pe,Pt,Po);travel(Pe,Lo,Lo);fly(Ai,Ru,Ru);board(Pe,Ai)).
#domain action(Ac).

% observation statements
holds(0,loc(boris),home1). holds(0,loc(gun),office).
holds(loc(comb1),home1). -holds(0,drunk(boris),true).
holds(0,loc(erik),home2). holds(0,comb2,home2).
-holds(0,drunk(erik),true). holds(0,loc(dimiter),home3).
holds(0,loc(comb3),home3). holds(0,drunk(dimiter),true).
holds(0,loc(sas609),run609).
% adding for completeness
-holds(0,inpocket(Pe,Pt),true). -holds(0,onplane(Ai,Pe),true).

% action occurrence statements
occurs(1,2,put(boris, comb1, pocket1)).
occurs(1,2,put(erik, comb2, pocket2)).
occurs(2,4,travel(dimiter, home3, office)).
occurs(3,5,travel(boris, home1, office)).
occurs(4,6,travel(erik, home2, office)).
occurs(6,7,put(boris, gun, pocket1)).
occurs(5,7,travel(dimiter, office, airport)).
occurs(7,9,travel(erik, office, airport)).
occurs(8,10,travel(boris, office, airport)).
occurs(9,10,board(dimiter, sas609)).
occurs(10,11,board(boris, sas609)).
occurs(11,12,board(erik, sas609)).
occurs(13,16,fly(sas609, run609, run609b)).

% action type specifications
occurs(T1,T2,put(Pe, Pt, Po)) ->
(?[Lo]:(holds(T1,loc(Pe),Lo) & holds(T1,loc(Pt),Lo)) ->
 (![T]:(T > T1 & T <= T2 -> occlude(T,inpocket(Pe,Pt))) &
 holds(T2,inpocket(Pe,Pt),true))).

occurs(T1,T2,travel(Pe, Lo1, Lo2)) ->
(holds(T1,loc(Pe),Lo1) -> occlude(T2,loc(Pe)) &
                          holds(T2,loc(Pe),Lo2)).
```

```
occurs(T1,T2,board(Pe, Ai)) ->
(holds(T1,poss_board(Pe,Ai),true) & holds(T1,loc(Pe),airport) ->
  (occlude(T2,loc(Pe)) & occlude(T2,onplane(Ai,Pe)) &
     ?[Lo]:(holds(T2,loc(Pe),Lo) & holds(T2,loc(Ai),Lo)) &
         holds(T2,onplane(Ai,Pe),true))).

occurs(T1,T2,fly(Ai,Ru1,Ru2)) ->
(holds(T1,loc(Ai),Ru1) ->
  (![T]:(T > T1 & T < T2 -> occlude(T,loc(Ai))) &
   ![T]:(T > T1 & T < T2 -> holds(T,loc(Ai),air)) &
   occlude(T2,loc(Ai)) & holds(T2,loc(Ai),Ru2))).

% dependency constraints reprsenting qualifications
holds(T,inpocket(Pe,gun),true) & T <= maxstep ->
![Ai]:occlude(T,poss_board(Pe,Ai)) &
![Ai]: (-holds(T,poss_board(Pe,Ai),true)).

holds(T,drunk(Pe),true) & T <= maxstep ->
![Ai]:occlude(T,poss_board(Pe,Ai)).

% dependency constraints reprsenting ramifications
holds(T,onplane(Ai,Pe),true) & T > 0 &
-holds(T-1,loc(Ai),Lo) & holds(T,loc(Ai),Lo) ->
               occlude(T,loc(Pe)) & holds(T,loc(Pe),Lo).

holds(T,inpocket(Pe,Pt),true) & T > 0 &
-holds(T-1,loc(Pe),Lo) & holds(T,loc(Pe),Lo) ->
                     occlude(T,loc(Pt)) & holds(T,loc(Pt),Lo).

% domain constraints
Pe1 != Pe2 & holds(T,inpocket(Pe1,Pt),true) ->
-holds(T,inpocket(Pe2,Pt),true).

Ai1 != Ai2 & holds(T,onplane(Ai1,Pe),true) ->
-holds(T,onplane(Ai2,Pe),true).
```

Figure 9.1: RAH example in F2LP

9.4 Using Constraint Answer Set Solvers for Reasoning with Temporal Action Logics

Notice that the translation $Trans$ involves turning formulas of the form $[t]\ f \hat{=} \omega$ into *Holds*(t, f, ω). This could cause a grounding bottleneck for large domains. In order to avoid such bottlenecks, answer set solvers are being enhanced with constraint processing (CP) (Dechter, 2003; Rossi, Beek, & Walsh, 2006) techniques (see, for example, (Baselice, Bonatti, & Gelfond, 2005; Mellarkod, Gelfond, & Zhang, 2008; Gebser, Ostrowski, & Schaub, 2009)). As mentioned in the introduction, these enhancements can be carried over to ASP-based TAL reasoning. Here, we demonstrate how CLINGCON[3] (Gebser et al., 2009), which combines the ASP solver CLINGO[4] and the CP solver GECODE[5], can be used for ASP-based TAL reasoning.

Intuitively, since *Holds* is non-intensional, we can simply replace *Holds*$(t, f(\mathbf{x}), \omega)$ in $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$, which is the formula within SM in Theorem 28(c), with $f'(t, \mathbf{x}) = \omega$, where f' is a function whose range is the same as the domain of the fluent $f(\mathbf{x})$ and **x** is a list of terms. We can then apply translation F2LP on the resulting theory to turn it into an answer set program. We make the idea precise as follows. Consider a TAL narrative $\mathcal{N}$ and any subset S of the set of feature symbols in $\mathcal{N}$. If σ is the signature of the corresponding preferred narrative $\Delta_{\mathcal{N}}$ ((9.1)), by σ^S we denote the signature obtained from σ by adding a function constant f' for every feature symbol f in S. If a feature symbol f takes n arguments, then the arity of f' is $n + 1$. If F is the formula $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$, by F_S we denote the formula of signature σ^S obtained from F by replacing every occurrence of *Holds*$(t, f(\mathbf{x}), \omega)$ in F, where $f \in S$, with

[3] http://potassco.sourceforge.net/
[4] http://potassco.sourceforge.net/
[5] http://www.gecode.org/

$f'(t, \mathbf{x}) = \omega$. By EQ_S we denote the conjunction of formulas

$\forall t, \mathbf{x}, v(\mathit{Holds}(t, f(\mathbf{x}), v) \leftrightarrow f'(t, \mathbf{x}) = v)$ for every feature symbol f in S.

Theorem 30 *Given a TAL narrative $\mathcal{N}$, let σ be the signature of the corresponding preferred narrative $\Delta_{\mathcal{N}}$ ((9.1)), and let F be the formula $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$. Then,*

$$SM[F; \mathit{Occurs}, \mathit{Occlude}]$$

is σ-equivalent to

$$SM[F_S; \mathit{Occurs}, \mathit{Occlude}] \wedge EQ_S$$

where S is any subset of the set of feature symbols in $\mathcal{N}$.

We can encode F_S, for any S, in the language of F2LP. However, the output of F2LP cannot be directly used with CLINGCON. This is because CLINGCON uses the special character `$` to distinguish between constraint and regular ASP atoms. So, we would need to prepend `$` to all comparison operators occurring in constraint atoms. Further, we also need to replace equality in constraint atoms with `$==`. For example, the constraint atoms `f(a) + g(Y) <= h(Z)` and `f(a) = X` have to be replaced with `f(a) + g(Y) $<= h(Z)` and `f(a) $== X` respectively.

Consider the "Kitchen Sink " example from the benchmark problems of the event calculus. The actions `tapOn` and `tapOff` turn the tap on and off respectively. The fluents `filling` and `spilling` represent the kitchen sink being filled and the water being spilled respectively. The fluent `waterLevel` represents the level of the water in the sink. Here, it is assumed that the height of the sink is 15 units and when the tap is on, the level of the water increases by 1 unit for every unit of time. Figure 9.2 shows an F2LP encoding of the example where `waterLevel` is used as a constraint variable. The comments (indicated by %) show the corresponding VITAL encoding. As we can observe from the encoding, we introduced constraint

variables `waterLevel(T)` corresponding to the fluent `waterLevel`. The construct `#spatom{...}` used in the encoding directs F2LP to copy the part within the parantheses to the output without any processing. The following changes need to be made to the output of F2LP in order to successfully run the description with CLINGCON (v. 0.1.2):

- since `#spatom{$domain(0..100)}` is turned to `$domain(0..100) :- true.`, this needs to be replaced with `$domain(0..100).`;
- all the comparison and equality symbols occurring in constraint atoms need to be replaced with the corresponding CLINGCON encoding as explained above; and
- since CLINGCON (v. 0.1.2) does not allow variables to begin with an underscore (_), every new variable `_NV_*` introduced by F2LP needs to be replaced with a variable not beginning with an underscore (such as `NV_*`).

The resulting program can be run using CLINGCON as follows:

```
$ clingcon -c maxstep=50 kitchensink.lp
```

One can verify the following from the output:

- since no action occurs until timepoint 15, the `waterLevel` is 0 and `filling` and `spilling` are false, for timepoints 0 to 15.
- `filling` becomes true at 16, and remains so until 45.
- starting from 17, `waterLevel` increases by 1 unit until it reaches 15 (the height of the sink), and remains so forever since the part about the water being drained out is not encoded.

- `spilling` becomes true at 31 when the `waterLevel` becomes 15, and remains so until 45.

We tested several examples with CLINGCON (v. 0.1.2), and the experimental results presented in the next section clearly show the advantages of using CLINGCON on numeric domains when compared to pure ASP solvers such as CLINGO. Since CLINGCON (v. 0.1.2) allows only integers in the domain of constraint variables, we could not test certain examples, such as the RAH scenario, which do not have any numeric domains.

```
% File 'kitchensink'

time(0..maxstep).
#domain time(T). #domain time(T1). #domain time(T2).

%#domain height :integer :lb 0 :ub 100
#spatom{$domain(0..100)}.
height(0..100).
#domain height(H).

bool(true;false).
#domain bool(B).

%#action tapon, tapoff
action(tapOn;tapOff).

%#feature filling :domain boolean :showname
%#feature waterlevel :domain height :showname
%#feature spilling :domain boolean :durational false :showname
feature(filling;spilling;waterLevel).
hFeature(waterLevel(T)).

% all fluents except spilling are persistent
-occlude(T+1,filling) & T < maxstep ->
![B]:(holds(T+1,filling,B) <-> holds(T,filling,B)).

-occlude(T+1,waterLevel) & T < maxstep ->
                    waterLevel(T+1) = waterLevel(T).

-occlude(T,spilling) -> holds(T,spilling,false).

% unique value axioms
% not required for waterLevel
1{holds(T,filling,Bo):bool(Bo)}1.
1{holds(T,spilling,Bo):bool(Bo)}1.

% effects of actions
%#acs [t1,t2] tapon -> R((t1,t2] filling)
occurs(T1,T2,tapOn) ->
![T]:(T > T1 & T <= T2 ->
         occlude(T,filling)) & holds(T2,filling,true).

%#acs [t1,t2] tapoff -> R((t1,t2] !filling)
occurs(T1,T2,tapOff) ->
![T]:(T > T1 & T <= T2 ->
         occlude(T,filling)) & holds(T2,filling,false).
```

```
%#dep forall t [ [t] filling -> X([t+1] waterlevel == 0) ]
holds(T,filling,true) & T < maxstep -> occlude(T+1,waterLevel).

%#dom forall t [ [t] filling & !spilling ->
%               [t+1] waterlevel == value(t,$plus(waterlevel,1)) ]
holds(T,filling,true) & -holds(T,spilling,true) & T < maxstep ->
waterLevel(T+1) = waterLevel(T) + 1.

%#dom forall t [ [t] spilling ->
%               [t+1] waterlevel == value(t,waterlevel) ]
holds(T,spilling,true) & T < maxstep ->
                waterLevel(T+1) = waterLevel(T).

%#dep forall t [ [t] waterlevel == 15 & [t] filling ->
%                                          R([t] spilling) ]
waterLevel(T) == 15 & holds(T,filling,true) ->
occlude(T,spilling) & holds(T,spilling,true).

%#obs [0] waterlevel == 0 & !spilling & !filling
%#occ [15,16] tapon
%#occ [45,46] tapoff
waterLevel(0) = 0.
holds(0,spilling,false) & holds(0,filling,false).
occurs(15,16,tapOn).
occurs(45,46,tapOff).
```

Figure 9.2: Kitchen Sink example in F2LP

9.5 Comparison with VITAL

VITAL[6] is a tool for reasoning about actions using TAL. The tool supports rich featuers, including ramification constraints, qualification constraints, and durational fluents. It not only generates the models but also provides a visualization of the models, which makes it easier to verify the output. VITAL is essentially a research tool mainly intended for generating and visualizing the models of the descriptions. While it has some optimizations for certain types of narratives, raw performance was not a primary factor in the design and implementation of the tool.[7] Nonetheless, we think it is useful to compare the performance of VITAL with that of our ASP-based approach since it would give us an idea about the performance of our approach.

For the comparison, we considered 3 scenarios: the *Russian Airplane Hijack* (RAH) scenario discussed earlier in the chapter, the *water tank* scenario whose TAL encoding is provided with VITAL, and the *Zoo World* scenario from (Akman et al., 2004). For the RAH scenario, we considered a few projection and planning problems. For the water tank scenario, we considered projection problems, and for the zoo world scenario, we considered projection and postdiction problems. All experiments were done on a Pentium machine with 3.06 GHz CPU and 4GB RAM running 64 bit Linux.

Figure 9.3 shows the comparison of VITAL (v. 2.999.910 alpha) with the following for the RAH and water tank scenarios:

- F2LP (v 1.11) with GRINGO (v 3.0.3) +CLASP (v 2.0.2),
- F2LP (v 1.11) with CLINGCON (v 0.1.2) (GRINGO v 2.0.2 and CLASP v 1.1.1) for the water tank scenario, and

[6] http://www.ida.liu.se/∼jonkv/vital

[7] This information was provided to us by Jonas Kvarnström in a personal communication.

Problem (max. step)	VITAL	F2LP with GRINGO + CMODELS	F2LP with GRINGO + CLASP	F2LP with CLINGCON
RAH-proj (16)	0.01s	0.23s (0.08s + 0.15s)	0.13s (0.08s + 0.05s)	NA
RAH-proj (50)	0.05s	0.99s (0.32s + 0.67s)	0.53s (0.32s + 0.21s)	NA
RAH-plan-unit (7)	10.56s	0.13s (0.03s + 0.1s)	0.08s (0.03s + 0.05s)	NA
RAH-plan-unit (10)	$>$ 30min	0.45s (0.1s + 0.35s)	0.29s (0.1s + 0.19s)	NA
RAH-plan-dur (6)	540.59s	0.3s (0.07s + 0.23s)	0.11s (0.07s + 0.04s)	NA
RAH-plan-dur (9)	$>$ 30min	0.66s (0.13s + 0.53s)	0.26s (0.13s + 0.13s)	NA
WaterTank-proj-100 (16)	0.07s	$>$ 30min	$>$ 30min	8.92s
WaterTank-proj-200 (25)	0.03s	$>$ 30min	$>$ 30min	119.44s
WaterTank-proj-300 (30)	0.01s	$>$ 30min	$>$ 30min	591.15s
WaterTank^s-proj-100 (16)	NA	0.06s (0.05s + 0.01s)	0.05s (0.05s + 0.00s)	NA
WaterTank^s-proj-200 (25)	NA	0.14s (0.13s + 0.01s)	0.13s (0.13s + 0.00s)	NA
WaterTank^s-proj-300 (30)	NA	0.22s (0.21s + 0.01s)	0.21s (0.21s + 0.00s)	NA

Figure 9.3: Comparison of VITAL with F2LP + answer set solvers

- F2LP (v 1.11) with GRINGO (v 3.0.3)+CMODELS (v 3.79) running MINISAT (v 2.0 beta).

In the figure, "`max.step`" indicates the maximum timepoint for which the tests were run. The times shown are in seconds, and were obtained using the linux `time` command for our ASP-based approach. The value shown is the sum of "user time" and "sys time" returned by the `time` command. The times shown in the parantheses are "(grounding time + solving time)". For VITAL, the time shown is the one provided by the tool itself. Since there was a non-negligible variation in the time returned by VITAL for different executions of the same problem instance, we ran each problem instance 3 times and took the lowest among the times returned. If VITAL crossed the time limit on the first run, then we did not run that problem

instance again. The cut-off time we used for the tools to terminate and return 1 model is 30 minutes. "NA" against a tool implies that that particular problem instance was either not tested with that tool or does not apply to that tool.

For the RAH scenario, the first two problems are projection problems and the next 4 are planning problems. Among the planning problems, the first two consider all actions to be of unit duration, and the next two consider actions with fixed lower and upper bounds on their durations. For the planning problems, "`max.step`" is also the length of a minimum plan. While VITAL does not support goal statements, planning problems can be encoded using dependency and domain constraints.[8] The idea is to introduce fluents representing actions, and to use dependency constraints and domain constraints to represent the effects of actions and the constraints on the action occurrences respectively. Domain constraints are also used to specify the goal. While VITAL is in general not as efficient as TALplanner on planning problems, it allows us to represent domains, such as the RAH scenario, which involve ramification and qualification constraints,[9] that are not allowed in TALplanner.

For the water tank scenario, the number in the problem name is the maximum value in the domain of the numbers considered. For instance, the number domain used for "WaterTank-proj-100" is `0..100`. The F2LP encoding of the water tank scenario is different when CLINGCON is used from when CLASP or CMODELS is used. In the F2LP encoding for CLINGCON, the fluents with numeric domains are treated as constraint variables, similar to how `waterLevel` was treated in the "kitchen sink" example discussed in the previous section. The last three problems in the water tank scenario correspond to the first 3 problems but they use an alternative (non-CSP) F2LP encoding where the unique value axioms

[8] This was suggested by Jonas Kvarnström in a personal communication.

[9] Recall that the (partial) solutions to the ramification and qualification problems use dependency constraints.

Problem (max. step)	VITAL	F2LP with GRINGO + CMODELS	F2LP with GRINGO + CLASP
ZooWorld-proj (20)	0.43s	0.97s (0.39s + 0.58s)	0.80s (0.39s + 0.41s)
ZooWorld-proj (50)	$>$ 30min	4.85s (2.28s + 2.57s)	4.33s (2.28s + 2.05s)
ZooWorld-post (15)	61.63s	0.62s (0.24s + 0.38s)	0.51s (0.24s + 0.27s)
ZooWorld-post (15)	17.40s	0.59s (0.24s + 0.25s)	0.51s (0.24s + 0.27s)
ZooWorld-post (20)	477.4s	0.99s (0.41s + 0.58s)	0.82s (0.41s + 0.41s)

Figure 9.4: Zoo World in VITAL vs. Zoo World in F2LP + answer set solvers

are not explicitly specified. This is because, for this particular scenario, the F2LP encoding can be modified so that the value of each fluent is uniquely determined at every timepoint. As the results show, this modification has a significant impact on the times taken for GRINGO+ CLASP and GRINGO+ CMODELS to return an answer set.

Figure 9.4 shows the comparison with VITAL for the zoo world scenario. The first two problems in this figure are projection problems, and the remaining are postdiction problems. For the postdiction problems, we gave an incomplete initial state along with certain constraints on the final state, and ran the tools to determine the complete initial state.

As we can see from the results, VITAL performed quite well on the projection problems but is not very efficient on the planning and postdiction problems we considered. On the other hand, our ASP-based approach performed equally well on all the problems, except on those involving numeric domains, such as the water tank scenario. The results also clearly show the usefulness of hybrid answer set solvers such as CLINGCON on numeric domains.

9.6 Proofs

Proof of Theorem 28

Given a narrative $\mathcal{N}$ and the corresponding preferred narrative $\Delta_{\mathcal{N}}$, we can conclude the following:

(ob1) all occurrences of *Occlude* in Γ_{acs} and Γ_{depc} are strictly positive, and there are no strictly positive occurrences of *Occurs* in either of them;

(ob2) all occurrences of *Occurs* in Γ_{occ} are strictly positive, and there are no occurrences of *Occlude* in it;

(ob3) Γ_{domc}, Γ_{obs}, Γ_{fnd}, and Γ_{time} do not contain any occurrences of either *Occlude* or *Occurs*; and

(ob4) all occurrences of *Occlude* in Γ_{per} are in the scope of negation, and there are no occurrences of *Occurs* in it.

The equivalence between (a) and (b) follows from Theorem 4, and observations (ob1) and (ob2) above which imply that Γ_{occ} is canonical relative to *Occurs* and $\Gamma_{depc} \wedge \Gamma_{acs}$ is canonical relative to *Occlude*. The equivalence between (b) and (c) follows from Theorem 12, and observations (ob1)-(ob4) above which imply that the conditions for applying the splitting theorem (Theorem 12, Chapter 4.1) are satisfied. □

Proof of Theorem 29

Consider $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$, where Γ_{ncirc} is $\Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc}$. The only intensional predicates in these formulas are *Occurs* and *Occlude*. Among the formulas, only Γ_{depc}, Γ_{acs}, and Γ_{per} contain occurrences of *Occlude*, and only Γ_{acs} and Γ_{occ} contain occurrences of

Occurs. From the definitions of an application formula and $Trans$, it follows that *Occlude* is outside the scope of any positive occurrence of $\exists$ and any negative occurrence of $\forall$ in $\Gamma_{depc} \wedge \Gamma_{acs}$. Further, it is clear that all occurrences of *Occlude* in Γ_{per} are in the scope of negation, and that *Occurs* is outside the scope of any positive occurrence of $\exists$ and any negative occurrence of $\forall$ in $\Gamma_{acs} \wedge \Gamma_{occ}$. This implies that $\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$ is almost universal relative to $\{$*Occurs*, *Occlude*$\}$. The result follows from Theorem 16. □

Proof of Theorem 30

Follows from Lemma 9. □

Chapter 10

INTEGRATING RULES AND ONTOLOGIES IN THE FIRST-ORDER STABLE MODEL SEMANTICS

Integrating nonmonotonic rules and ontologies is an important area of the semantic web research. Since the Web Ontology Language (OWL), which has been endorsed by the World Wide Web Consortium (W3C), is based on Description Logics (DLs), much of the work in this area focuses on integrating nonmonotonic rules and DLs. The knowledge base resulting from combining a DL knowledge base with nonmonotonic rules is usually referred to as a *hybrid knowledge base*.

A hybrid knowledge base is a pair $(\mathcal{T}, \mathcal{P})$ where $\mathcal{T}$ is a FOL knowledge base (typically in a description logic) of signature $\Sigma_{\mathcal{T}}$ and $\mathcal{P}$ is a logic program of signature $\Sigma_{\mathcal{P}}$. As discussed in Chapter 2.6, the existing integration approaches can be classified into three categories: *loose integration*, *tight integration with semantic separation*, and *tight integration under a unifying logic* (Nazarenko et al., 2010). In the *loose integration* approach, $\mathcal{T}$ and $\mathcal{P}$ are viewed as separate, independent components, and are connected through minimal safe interfaces for exchanging data (usually in the form of ground atoms). Examples in this category include *nonmonotonic dl-programs* (Eiter et al., 2008), and the combination of description logics and defeasible logic (Wang et al., 2004). In the *tight integration with semantic separation* approach, $\mathcal{T}$ and $\mathcal{P}$ are more tightly integrated, but the predicates in $\Sigma_{\mathcal{T}}$ and $\Sigma_{\mathcal{P}}$ are kept separate. This approach builds an integrated model I as the union of a model $I_{\mathcal{T}}$ of $\mathcal{T}$ and a model $I_{\mathcal{P}}$ of $\mathcal{P}$ with the same domain. Examples in this category are r-hybrid KB (Rosati, 2005), $\mathcal{DL} + log$ (Rosati, 2006), g-hybrid KB (Heymans et al., 2008), and f-hybrid KB (Feier & Heymans, 2009). Finally, in the *tight integration under a unifying logic* approach, $\mathcal{T}$ and $\mathcal{P}$ are treated uniformly by translating them into a uniform logic, and there is no principled separation between $\Sigma_{\mathcal{T}}$ and $\Sigma_{\mathcal{P}}$. Examples in this category are

Hybrid MKNF KB (Motik & Rosati, 2010), the first-order Autoepistemic Logic based integration (de Bruijn et al., 2007a), and the Quantified Equilibrium Logic based integration (de Bruijn et al., 2007b). This approach is attractive since it provides a seamless integration of DLs and logic programs, and since the information flow is bi-directional.

In this chapter, we use the framework of the first-order stable model semantics to integrate Description Logics (DLs) and ASP. We show how our approach can capture several approaches belonging to each of the categories discussed above. We also show how the research on the first-order stable model semantics can be used to strengthen certain decidability results for $\mathcal{DL} + log$ (Rosati, 2006) and to define the notion of strong equivalence (Chapter 3.2) for hybrid knowledge bases. Several parts of this chapter are also presented in (Lee & Palla, 2011a, 2011b).

10.1 Integrating Description Logic Knowledge Bases and Answer Set Programming Rules

DL knowledge bases can be viewed as theories in first-order logic. Since the first-order stable model semantics generalizes both ASP and first-order logic, it provides an ideal framework for integrating ASP-rules and DL knowledge bases. Given a DL knowledge base $\mathcal{T}$ of signature $\Sigma_\mathcal{T}$ and a logic program $\mathcal{P}$ of signature $\Sigma_\mathcal{P}$, our approach is to identify the models of the hybrid knowledge base $(\mathcal{T}, \mathcal{P})$ with the interpretations of signature $\Sigma_\mathcal{T} \cup \Sigma_\mathcal{P}$ (in the sense of classical logic) that satisfy $\mathrm{SM}[\mathit{FO}(\mathcal{T}) \wedge \mathit{FO}(\mathcal{P}); \mathbf{p}]$, where $\mathit{FO}(\mathcal{T})$ and $\mathit{FO}(\mathcal{P})$ are the first-order representations of $\mathcal{T}$ and $\mathcal{P}$ respectively, and **p** is a list of intensional predicates. We assume that $\mathcal{T}$ and $\mathcal{P}$ are finite, and so are the predicate constants in $\Sigma_\mathcal{P}$. Typically, existing integration approaches assume that the signatures do not contain function constants of positive arity, and $\Sigma_\mathcal{T}$ and $\Sigma_\mathcal{P}$ share the same set of object constants, but have disjoint sets of predicate constants.

Note here that we do not place any restriction on the syntax of $\mathcal{P}$ or on the signatures of $\mathcal{T}$ and $\mathcal{P}$. Further, the interpretations considered are first-order interpretations. This shows that both $\mathcal{T}$ and $\mathcal{P}$ are treated uniformly, which in turn indicates that our approach is faithful and tight. The intensional predicates are usually the predicate constants in $\Sigma_{\mathcal{P}}$ that do not belong to $\Sigma_{\mathcal{T}}$. However, the list of intensional predicates can be increased or decreased as necessary. Since both $\mathcal{T}$ and $\mathcal{P}$ are viewed uniformly under the first-order stable model semantics, our approach belongs to the *tight integration under a unifying logic* category discussed above.

Example 12 *(de Bruijn et al., 2007b, Example 1) Consider a hybrid knowledge base consisting of a classical theory $\mathcal{T}$:*

$$\forall x(\textit{Person}(x) \rightarrow (\textit{Agent}(x) \wedge (\exists y \textit{HasMother}(x, y))))$$
$$\forall x((\exists y \textit{HasMother}(x, y)) \rightarrow \textit{Animal}(x))$$

which says that every Person is an Agent and has some (unknown) mother, and everyone who has a mother is an Animal, and a nonmonotonic logic program $\mathcal{P}$:

$$\textit{Person}(x) \leftarrow \textit{Agent}(x), \textit{not machine}(x)$$
$$\textit{Agent}(\textit{DaveB})$$

which says that Agents are by default Persons, unless known to be machines, and DaveB is an Agent. The predicate constants starting in capital letters belong to $\Sigma_{\mathcal{T}}$, and the rest of the predicate constants belong to $\Sigma_{\mathcal{P}} \setminus \Sigma_{\mathcal{T}}$. It follows that $SM[\textit{FO}(\mathcal{T}) \wedge \textit{FO}(\mathcal{P});\ \textit{machine}]$ entails $\textit{Person}(\textit{DaveB})$, $\exists y \textit{HasMother}(\textit{DaveB}, y)$, and $\textit{Animal}(\textit{DaveB})$.

Since we do not place any restrictions on the syntax of the logic programs, we can even use quantifiers if necessary. Sometimes, quantifiers also enable succint representation.

Example 13 *Consider a hybrid knowledge base where $\mathcal{T}$ is*

$$\forall x(\textit{Student}(x) \rightarrow \textit{Person}(x))$$

$$\forall x(\textit{Professor}(x) \rightarrow \textit{Person}(x))$$

$$\forall x(\textit{Professor}(x) \rightarrow \exists y(\textit{Teaches}(x, y) \wedge \textit{Course}(y)))$$

$$\forall x(\textit{unregistered}(x) \rightarrow \textit{Professor}(x))$$

$$\forall x(\exists y(\textit{registered}(x, y) \wedge \textit{Course}(y)) \rightarrow \textit{Student}(x))$$

$$\textit{Course}(\textit{A})$$

$$\textit{Course}(\textit{B})$$

and $\mathcal{P}$ is

$$\textit{unregistered}(x) \leftarrow \textit{not}\ \exists y(\textit{registered}(x, y) \wedge \textit{Course}(y)) \wedge \textit{Person}(x)$$

$$\textit{registered}(\textit{Joe}, \textit{A})$$

$$\textit{Person}(\textit{Mary}).$$

It is not difficult to verify that $SM[\textit{FO}(\mathcal{T}) \wedge \textit{FO}(\mathcal{P}); \textit{registered}, \textit{unregistered}\,]$ entails $\textit{Professor}(\textit{Mary})$, $\exists y(\textit{Teaches}(\textit{Mary}, y) \wedge \textit{Course}(y))$, $\textit{Student}(\textit{Joe})$, $\textit{Person}(\textit{Joe})$, and $\neg\exists xy(\textit{registered}(x, y) \wedge \textit{unregistered}(x))$.

10.2 Relation to $\mathcal{DL} + log$

In $\mathcal{DL} + log$, predicate constants are partitioned into *DL predicates* $P_{\mathcal{T}}$ and *Datalog predicates* $P_{\mathcal{P}}$. DL predicates are further partitioned into *concept names* and *role names*. Additionally, $\mathcal{DL} + log$ assumes a countably infinite set of object constants, denoted by C.

A $\mathcal{DL} + log$ knowledge base is denoted by $(\mathcal{T}, \mathcal{P})$, where $\mathcal{T}$ is a DL knowledge base of signature $\langle C, P_{\mathcal{T}}\rangle$ and $\mathcal{P}$ is a Datalog program of signature $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}}\rangle$ consisting of rules R of the form

$$\begin{aligned} p_1(X_1)\ ;\ \ldots\ ;\ p_n(X_n) \leftarrow & \\ & r_1(Y_1), \ldots, r_m(Y_m), s_1(Z_1), \ldots, s_k(Z_k), \\ & \textit{not}\ u_1(W_1), \ldots, \textit{not}\ u_h(W_h) \end{aligned} \tag{10.1}$$

($n \geq 0, m \geq 0, k \geq 0, h \geq 0$) where X_i, Y_i, Z_i, W_i are lists of object variables and object constants, and

- each p_i is either a DL predicate or a Datalog predicate;
- each r_i, u_i is a Datalog predicate;
- each s_i is a DL predicate;
- (*Datalog safety*) every variable occurring in R must also occur in at least one of the atoms $r_1(Y_1), \ldots, r_m(Y_m), s_1(Z_1), \ldots, s_k(Z_k)$;
- (*Weak safety*) every variable occurring in the head of R must also occur in at least one of the atoms $r_1(Y_1), \ldots, r_m(Y_m)$.

Rosati (2006) presents two semantics of $\mathcal{DL} + log$ KB: the monotonic and the nonmonotonic semantics. The monotonic semantics of $\mathcal{DL} + log$ is given by simply viewing $\mathcal{T}$ and $\mathcal{P}$ as theories in first-order logic: given a $\mathcal{DL} + log$ knowledge base $(\mathcal{T}, \mathcal{P})$ of signature $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}} \rangle$, an interpretation I is a *monotonic model* of $(\mathcal{T}, \mathcal{P})$ if I satisfies *FO*$(\mathcal{T}) \wedge$ *FO*$(\mathcal{P})$. Since a first order theory can be characterized in the first-order stable model semantics by making the list of intensional predicates empty, the monotonic semantics of $\mathcal{DL} + log$ can be expressed by $\mathrm{SM}[$*FO*$(\mathcal{T}) \wedge$ *FO*$(\mathcal{P}); \emptyset]$.

The nonmonotonic semantics of $\mathcal{DL} + log$ is based on the stable model semantics for disjunctive logic programs. The notation $gr(\mathcal{P}, C)$ represents the ground program obtained by replacing every variable in every rule of $\mathcal{P}$ with every object constant in C.

Given $gr(\mathcal{P}, C)$ and an interpretation I of signature $\langle C, P_{\mathcal{T}} \rangle$, the *projection* of $gr(\mathcal{P}, C)$ with respect to I, denoted by $\Pi(gr(\mathcal{P}, C), I)$, is obtained as follows. For every rule $R \in gr(\mathcal{P}, C)$,

- delete R if $I \models r(\mathbf{t})$ for some head atom $r(\mathbf{t})$ such that $r \in P_{\mathcal{T}}$;
- delete every atom $r(\mathbf{t})$ in the head such that $r \in P_{\mathcal{T}}$ and $I \not\models r(\mathbf{t})$;
- delete R if $I \not\models r(\mathbf{t})$ for some atom $r(\mathbf{t})$ in the body such that $r \in P_{\mathcal{T}}$;
- delete every atom $r(\mathbf{t})$ in the body such that $r \in P_{\mathcal{T}}$ and $I \models r(\mathbf{t})$.

The $\mathcal{DL} + log$ approach imposes the *standard name assumption*: every interpretation is over the same fixed, countably infinite, domain Δ, and in addition, the set C of object constants is such that it is in the same one-to-one correspondence with Δ in every interpretation. As a result, for simplicity, we assume that the domain with respect to every interpretation is C.

An interpretation I (in the sense of classical logic) of a signature σ can be represented as a pair $\langle I^f, X\rangle$, where I^f is the restriction of I to function constants (including object constants) from σ, and X is the set of atoms, formed using predicate constants from σ and the names of elements of $|I|$, which are satisfied by I.

Given a $\mathcal{DL} + log$ knowledge base $(\mathcal{T}, \mathcal{P})$ of signature $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}}\rangle$, an interpretation I is a *nonmonotonic model* of $(\mathcal{T}, \mathcal{P})$ if

- $I|_C$ (the restriction of I on C) is an identity function that maps every constant in C to itself;
- $\langle I|_C, I|_{P_{\mathcal{T}}}\rangle$ satisfies $\mathcal{T}$;
- $\langle I|_C, I|_{P_{\mathcal{P}}}\rangle$, identified with a set of ground atoms, is an answer set of $\Pi(gr(\mathcal{P}, C), \langle I|_C, I|_{P_{\mathcal{T}}}\rangle)$.

The following proposition shows how the nonmonotonic semantics of $\mathcal{DL} + log$ can be reformulated in terms of the first-order stable model semantics.

Theorem 31 *For any $\mathcal{DL}+log$ knowledge base $(\mathcal{T},\mathcal{P})$, under the standard name assumption, the nonmonotonic models of $(\mathcal{T},\mathcal{P})$ according to (Rosati, 2006) are precisely the interpretations of $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}}\rangle$ that satisfy*

$$SM[\textit{FO}(\mathcal{T}) \wedge \textit{FO}(\mathcal{P});\ P_{\mathcal{P}}].$$

Since the reformulation does not refer to grounding, arguably, it provides a simpler account of $\mathcal{DL}+log$.

Discarding Datalog Safety

$\mathcal{DL}+log$ imposes weak safety (every variable occurring in the head of a rule also occurs in a Datalog atom in the positive body) and Datalog safety (every variable occurring in a rule also occurs in the positive body), which, even when combined, yields a condition that is weaker than DL-safety (Motik, Sattler, & Studer, 2005), where every variable occurring in a rule is also required to occur in a datalog atom in the positive body. Here, we use the concept of *semi-safety* discussed in Chapters 6.1 and 6.3 to show that the assumption of weak safety is a sufficient condition for guaranteeing decidability of reasoning with $\mathcal{DL}+log$.

First, we slightly generalize the definition of semi-safety to the case where the list of intensional predicates can be arbitrary. We start with generalizing the definition of $\mathrm{RV}(F)$ (*restricted variables of* F) to differentiate between intensional and extensional predicates. As before, we assume that the signature contains no function constants of positive arity. To every quantifier-free formula F, we assign a set $\mathrm{RV}_{\mathbf{p}}(F)$ of *restricted variables relative to* $\boldsymbol{p}$ as follows.

- For an atomic formula F (including equality and $\bot$),
  - if F is an equality between two variables, or is an atom whose predicate constant is not in $\mathbf{p}$, then $\mathrm{RV}_{\mathbf{p}}(F) = \emptyset$;
  - otherwise, $\mathrm{RV}_{\mathbf{p}}(F)$ is the set of all variables occurring in F;

- $\mathrm{RV}_{\mathbf{p}}(G \wedge H) = \mathrm{RV}_{\mathbf{p}}(G) \cup \mathrm{RV}_{\mathbf{p}}(H)$;
- $\mathrm{RV}_{\mathbf{p}}(G \vee H) = \mathrm{RV}_{\mathbf{p}}(G) \cap RV_{\mathbf{p}}(H)$;
- $\mathrm{RV}_{\mathbf{p}}(G \to H) = \emptyset$.

We say that a variable x is ***p**-restricted* in a quantifier-free formula F if $x \in \mathrm{RV}_{\mathbf{p}}(F)$.

Recall that an occurrence of a predicate constant, a variable, or any other subexpression in a formula F is *strictly positive* if that occurrence is not in the antecedent of any implication.

Consider a sentence F in prenex form:

$$Q_1 x_1 \cdots Q_n x_n M \tag{10.2}$$

(each Q_i is $\forall$ or $\exists$; $x_1, \ldots, x_n$ are distinct variables; the matrix M is quantifier-free). We say that F is *semi-safe relative to **p*** if every strictly positive occurrence of every variable x_i in M belongs to a subformula $G \to H$ where x_i is **p**-restricted in G.

The *small predicate property* (Chapter 6.1) is generalized as follows. A **p**-stable model of F has the *small predicate property* if, for every predicate constant $p_i \in \mathbf{p}$, if the relation represented by it holds for a tuple of arguments, then each member of the tuple is represented by an object constant occurring in F. As before, the idea can be made precise as follows. For any finite set **c** of object constants, $in_{\mathbf{c}}(x)$ stands for the formula

$$\bigvee_{c \in \mathbf{c}} x = c.$$

*The small predicate property relative to **p***, denoted by $SPP^{\mathbf{p}}_{\mathbf{c}}$, is the conjunction of the sentences

$$\forall v_1, \ldots, v_n \Big(p(v_1, \ldots, v_n) \to \bigwedge_{i=1,\ldots,n} in_{\mathbf{c}}(v_i) \Big)$$

for all predicate constants p in **p**, where $v_1, \ldots, v_n$ are distinct variables.

The following proposition is an extension of Proposition 3 in Chapter 6.1.[1] By $c(F)$ we denote the set of all object constants occurring in F.

Proposition 5 *For any semi-safe sentence F relative to **p**, formula $SM[F; \mathbf{p}]$ entails $SPP^{\mathbf{p}}_{c(F)}$.*

Now, we will show how this proposition can be used to drop the condition of datalog safety in $\mathcal{DL} + log$. Since we identify Datalog predicates with intensional predicates, and DL predicates with non-intensional predicates, the definition of semi-safety presented above coincides with the definition of weak-safety for programs whose rules have the form (10.1). Therefore, from Proposition 5 and Theorem 31, we get that the relations represented by the datalog predicates can hold for a tuple of arguments only if each member of the tuple is a constant occurring in the program $\mathcal{P}$. Since the universe is a countably infinite set of constants C, it follows that if a literal in a rule (10.1) contains a variable that occurs only in the negative body, then that literal can be simply replaced with $\top$. The following theorem makes this precise.

Theorem 32 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ be a $\mathcal{DL} + log$ knowledge base such that $\mathcal{P}$ is weakly safe but is not necessarily datalog safe. Let $\mathcal{P}'$ be the program obtained from $\mathcal{P}$ by removing in every rule, all the negative datalog literals that contain a variable that occurs only in the negative body. Then $\mathcal{K}$ is equivalent (under the nonmonotonic semantics) to the $\mathcal{DL} + log$ knowledge base $(\mathcal{T}, \mathcal{P}')$.*

Since the complexity of the transformation required to obtain $\mathcal{P}'$ is polynomial in the size of $\mathcal{P}$, the decidability results (Theorems 11 and 12 from (Rosati, 2006)) and the complexity results (Theorem 13 from (Rosati, 2006)) with respect to the nonmonotonic semantics can be straightforwardly carried over

[1] This extension is presented in (Bartholomew & Lee, 2010).

to $\mathcal{DL} + log$ knowledge bases $(\mathcal{T}, \mathcal{P})$ where $\mathcal{P}$ is weakly safe but not necessarily datalog safe. In other words, in terms of decidability and complexity results mentioned above, the requirement of datalog safety can be dropped.

10.3 Relation to Quantified Equilibrium Logic with Hybrid Rules

Review of Quantified Equilibrium Logic

Quantified Equilibrium Logic (QEL) (Pearce & Valverde, 2005) is based on the *Quantified Here-and-There Logic* (QHT). Here we will review QHT and QEL without function constants and strong negation as presented in (de Bruijn et al., 2007b). Consider a function-free signature Σ consisting of a set of object constants C and a set of predicate constants P. A *here-and-there* Σ-structure with static domains (**QHT**$^s(\Sigma)$-structure) is a tuple $\mathcal{M} = \langle (D, \sigma), I_h, I_t \rangle$ where

- D is a non-empty set, called the domain of $\mathcal{M}$,
- σ is a mapping $C \cup D \rightarrow D$ such that $\sigma(d) = d$ for all $d \in D$,
- I_h and I_t are interpretations of Σ over D such that for every $p^{I_h} \subseteq p^{I_t}$ for all $p \in P$.

Here $\mathcal{M}$ can be seen as a first-order model having two components, h and t, that correspond to the "here" world and "there" world respectively in the sense of Kripke semantics for intuitionistic logic (van Dalen, 1983), such that whatever is verified in h remains true at t. For any sentence F, $\mathcal{M}, w \models F$, where $w = \{h, t\}$, is defined as follows:

- $\mathcal{M}, w \models p(t_1, \ldots, t_n)$ iff $(\sigma(t_1), \ldots, \sigma(t_n)) \in p^{I_w}$,
- $\mathcal{M}, w \models G \wedge H$ iff $\mathcal{M}, w \models G$ and $\mathcal{M}, w \models H$,
- $\mathcal{M}, w \models G \vee H$ iff $\mathcal{M}, w \models G$ or $\mathcal{M}, w \models H$,
- $\mathcal{M}, t \models G \rightarrow H$ iff $\mathcal{M}, t \not\models G$ or $\mathcal{M}, t \models H$,

- $\mathcal{M}, h \models G \rightarrow H$ iff $\mathcal{M}, t \models G \rightarrow H$ and $\mathcal{M}, h \not\models G$ or $\mathcal{M}, h \models H$,
- $\mathcal{M}, t \models \forall x G(x)$ iff $\mathcal{M}, t \models G(d)$ for all $d \in D$,
- $\mathcal{M}, h \models \forall x G(x)$ iff $\mathcal{M}, t \models \forall x G(x)$ and $\mathcal{M}, h \models G(d)$ for all $d \in D$,
- $\mathcal{M}, w \models \exists G(x)$ iff $\mathcal{M}, w \models G(d)$ for some $d \in D$.

The truth of a sentence F in a model $\mathcal{M}$ is defined as follows: $\mathcal{M} \models F$ iff $\mathcal{M}, w \models F$ for each $w \in \{h, t\}$. The resulting logic is called the *Quantified Here-and-There Logic with static domains*.

In order to define QEL, the notion of a "minimal" model is needed. Among two **QHT**$^s(\Sigma)$ structures $\langle (D, \sigma), I_h, I_t \rangle$ and $\langle (D', \sigma'), I'_h, I'_t \rangle$, $\langle (D, \sigma), I_h, I_t \rangle \trianglelefteq \langle (D', \sigma'), I'_h, I'_t \rangle$ if $D = D'$, $\sigma = \sigma'$, I_t agrees with I'_t on Σ, and $p^{I_h} \subseteq p^{I'_h}$ for all $p \in P$. A model $\mathcal{M} = \langle (D, \sigma), I_h, I_t \rangle$ of F is *total* if I_h agrees with I_t on Σ. $\mathcal{M}$ is an *equilibrium model* of F if it is minimal under $\trianglelefteq$ among models of F, and it is total.

Relation to the Quantified Equilibrium Logic Based Approach

Recall that *Choice*(**p**) denotes the conjunction of "choice formulas" $\forall \mathbf{x}(p(\mathbf{x}) \vee \neg p(\mathbf{x}))$ for all predicate constants p in **p** where **x** is a list of distinct object variables whose length is the same as the arity of p. The approach in (de Bruijn et al., 2007b) uses QEL to integrate rules and ontologies. According to that approach, a **QHT**$^s(\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}} \rangle)$ interpretation is a model of the hybrid knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ iff it is an equilibrium model of *FO*$(\mathcal{T}) \wedge$ *FO*$(\mathcal{P}) \wedge$ *Choice*$(P_{\mathcal{T}})$. Formula *FO*$(\mathcal{T}) \wedge$ *FO*$(\mathcal{P}) \wedge$ *Choice*$(P_{\mathcal{T}})$ is called the *stable closure* of $\mathcal{K}$. The following proposition shows the relationship between the QEL-based approach and our approach.

Proposition 6 *For any hybrid knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ of signature $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}} \rangle$, a* **QHT**$^s(\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}} \rangle)$ *interpretation $I = \langle (D, \sigma), I_t, I_t \rangle$ is an*

equilibrium model of $\mathcal{K}$ in the sense of (de Bruijn et al., 2007b) iff $\langle I|_C, I|_{P_\mathcal{T} \cup P_\mathcal{P}} \rangle$ satisfies

$$SM[\mathit{FO}(\mathcal{T}) \wedge \mathit{FO}(\mathcal{P});\ P_\mathcal{P}].$$

The proof of the proposition is immediate from Lemma 9 from (Ferraris et al., 2011), which establishes the relationship between QEL and $\mathrm{SM}[F; \mathbf{p}]$ for the special case when **p** is the list of all predicate constants in the signature, and Proposition 1, which tells us that the set of intensional predicates can be increased by using choice formulas.

10.4 Relation to g-hybrid Knowledge Bases

de Bruijn et al. (2007b) relate the QEL-based approach to r-hybrid (Rosati, 2005) and g-hybrid knowledge bases (Heymans et al., 2008). As a corollary of Proposition 6 in this paper, we can thus relate our approach to r-hybrid and g-hybrid knowledge bases. Here we present the result only with respect to g-hybrid knowledge bases since we already covered the relationship to $\mathcal{DL} + log$, which is an extension of r-hybrid knowledge bases.

g-hybrid knowledge bases are based on the *open answer set programming* (Heymans, Nieuwenborgh, & Vermeir, 2005) approach. More specifically, a g-hybrid knowledge base is a pair $(\mathcal{T}, \mathcal{P})$, where $\mathcal{T}$ is a DL knowledge base of signature $\langle C, P_\mathcal{T} \rangle$ and $\mathcal{P}$ is a *guarded* program of signature $\langle C, P_\mathcal{T} \cup P_\mathcal{P} \rangle$ such that $P_\mathcal{T} \cap P_\mathcal{P} = \emptyset$. A program is said to be *guarded* if, for all rules R that are not of the form

$$p(\mathbf{t}) \vee \mathit{not}\, p(\mathbf{t}) \leftarrow\ , \tag{10.3}$$

there exists an atom A in the positive body (known as the *guard*) such that all the variables occurring in R also occur in A. This implies that, in addition to such safe rules R, guarded programs also allow unsafe choice rules of the form (10.3). Also,

guarded programs allow negation in the head but with the restriction that there can be at most one non-negated atom in the head.

Given an interpretation I of signature $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}} \rangle$, program $\mathcal{P}_I$ is defined as the ground program obtained from $\mathcal{P}$ by first replacing every occurrence of c from C in it with c^I and then grounding the resulting program with respect to the universe of I. Interpretation I is a model of the g-hybrid knowledge base $(\mathcal{T}, \mathcal{P})$ if

- the restriction of I to $\langle C, P_{\mathcal{T}} \rangle$ is a model of $\mathcal{T}$, and
- the restriction of I to $\langle C, P_{\mathcal{P}} \rangle$, viewed as a set of ground atoms, is an answer set of $\Pi(\mathcal{P}_I, I)$.[2]

The following proposition is a corollary of Theorem 2 from (de Bruijn et al., 2007b) and Proposition 6.

Proposition 7 *For any g-hybrid knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{P})$, an interpretation I of signature $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}} \rangle$ is a model of $\mathcal{K}$ in the sense of (de Bruijn et al., 2007b) iff I is a model of*

$$SM[\mathit{FO}(\mathcal{T}) \wedge \mathit{FO}(\mathcal{P});\ P_{\mathcal{P}}].$$

10.5 Relating to Nonmonotonic dl-programs

Review of Nonmonotonic dl-programs

We first review the syntax and the semantics of dl-programs. For simplicity, we do not allow strong negation. A nonmonotonic $dl\text{-}program$ (Eiter et al., 2008) is a pair $(\mathcal{T}, \mathcal{P})$, where $\mathcal{T}$ is a DL knowledge base of signature $\langle C, P_{\mathcal{T}} \rangle$ and $\mathcal{P}$ is a *generalized* normal logic program of signature $\langle C, P_{\mathcal{P}} \rangle$ such that $P_{\mathcal{T}} \cap P_{\mathcal{P}} = \emptyset$. A generalized normal logic program is a set of *dl-rules* that can contain queries to $\mathcal{T}$ in their bodies, in the form of *dl-atoms*.

[2] The definition of projection Π given earlier is straightforwardly extended to cover a rule like (10.3) that allows *not* in the head.

A *dl-atom* is of the form

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m;\ Q](\mathbf{t}) \quad (m \geq 0) \tag{10.4}$$

where each S_i is a concept, a role or a special symbol $\theta \in \{=, \neq\}$, symbol p_i is a unary predicate constant if S_i is a concept and a binary predicate constant otherwise and $op_i \in \{\oplus, \odot, \ominus\}$; $Q(\mathbf{t})$ is a dl-query (Eiter et al., 2008).

A *dl-rule* is of the form

$$a \leftarrow b_1, \ldots, b_k, \textit{not}\ b_{k+1}, \ldots, \textit{not}\ b_m \tag{10.5}$$

where a is an atom and each b_i is either an atom, equality, or a dl-atom. We identify rule (10.5) with

$$a \leftarrow B, N \tag{10.6}$$

where B is $b_1, \ldots, b_k$ and N is *not* $b_{k+1}, \ldots,$ *not* b_m.

The semantics of dl-programs is defined by extending the answer set semantics to generalized programs. In order to do this, the definition of satisfaction is extended to ground dl-atoms. An Herbrand interpretation I *satisfies* a ground atom A *relative to* $\mathcal{T}$ if I satisfies A. An Herbrand interpretation I *satisfies* a ground dl-atom (10.4) *relative to* $\mathcal{T}$ if $\mathcal{T} \cup \bigcup_{i=1}^{m} A_i(I)$ entails $Q(\mathbf{t})$, where $A_i(I)$ is

- $\{S_i(\mathbf{e}) \mid\ p_i(\mathbf{e}) \in I\}$ if op_i is $\oplus$,
- $\{\neg S_i(\mathbf{e}) \mid\ p_i(\mathbf{e}) \in I\}$ if op_i is $\odot$,
- $\{\neg S_i(\mathbf{e}) \mid\ p_i(\mathbf{e}) \notin I\}$ if op_i is $\ominus$,

and **t** is any list of ground terms. The satisfaction relation is extended to allow connectives in the usual way.

Given a dl-program $(\mathcal{T}, \mathcal{P})$, the *weak dl-transform* of $\mathcal{P}$ relative to $\mathcal{T}$ and an Herbrand interpretation I of $\langle C, P_{\mathcal{P}} \rangle$, denoted by $w\mathcal{P}^I_{\mathcal{T}}$, is the logic program obtained from $gr(\mathcal{P}, C)$ by deleting

- each rule (10.6) in $gr(\mathcal{P}, C)$ such that
  - $I \not\models_{\mathcal{T}} b$ for some dl-atom b in B, or
  - $I \models_{\mathcal{T}} b$ for some literal *not* b in N;
- from each remaining dl-rule (10.6), all the dl-atoms in B and all the literals in N.

I is a *weak answer set* of $(\mathcal{T}, \mathcal{P})$ if I is the minimal model of $w\mathcal{P}^I_{\mathcal{T}}$.

By $DL^?_{\mathcal{P}}$ we denote the set of dl-atoms in $gr(\mathcal{P}, C)$ that are not known to be monotonic. The *strong dl-transform* of $\mathcal{P}$ relative to $\mathcal{T}$ and I, denoted by $s\mathcal{P}^I_{\mathcal{T}}$, is the logic program obtained from $gr(\mathcal{P}, C)$ by deleting

- each rule (10.6) in $gr(\mathcal{P}, C)$ such that
  - $I \not\models_{\mathcal{T}} b$ for some dl-atom b in $B \cap DL^?_{\mathcal{P}}$ or
  - $I \models_{\mathcal{T}} b$ for some *not* b in N;
- from each remaining dl-rule (10.6), all the dl-atoms in $B \cap DL^?_{\mathcal{P}}$ and all the literals in N.

I is a *strong answer set* of $(\mathcal{T}, \mathcal{P})$ if I is the minimal model of $s\mathcal{P}^I_{\mathcal{T}}$.

Relation to Nonmonotonic dl-programs

In order to relate our approach to the semantics of dl-programs, we define *dl-formulas* of signature $\langle C, P_{\mathcal{T}} \cup P_{\mathcal{P}} \rangle$ as an extension of first-order formulas by treating dl-atoms as a base case in addition to standard atomic formulas formed from $\langle C, P_{\mathcal{P}} \rangle$.[3] Note that any generalized normal logic program can be viewed as a dl-formula: *FO*$(\mathcal{P})$ can be extended to a generalized normal logic program $\mathcal{P}$ in a

[3] The extension is similar to the extension of first-order formulas to allow aggregate expressions as given in (Lee & Meng, 2009).

straightforward way. Let F be a variable-free dl-formula.[4] We define F^{w*} the same as F^* except for a new clause for a dl-atom:

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{c})^{w*}(\mathbf{u}) = DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{c}).$$

$\mathrm{SM}^w[F]$ is defined the same as formula $\mathrm{SM}[F]$ except that F^{w*} is used in place of F^*. The following theorem shows how weak answer sets can be characterized by this extension.

Theorem 33 *For any dl-program $(\mathcal{T}, \mathcal{P})$ such that $\mathcal{P}$ is variable-free, the weak answer sets of $(\mathcal{T}, \mathcal{P})$ are precisely the Herbrand interpretations of signature $\langle C, P_{\mathcal{P}} \rangle$ that satisfy $SM^w[$FO$(\mathcal{P});\ P_{\mathcal{P}}]$ relative to $\mathcal{T}$.*

In order to capture strong answer sets, we define F^{s*} the same as F^* except for a new clause for a dl-atom:

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{c})^{s*}(\mathbf{u}) = DL[S_1 op_1 u_1, \ldots, S_m op_m u_m; Q](\mathbf{c})$$

($u_1, \ldots, u_m$ are the elements of **u** that correspond to $p_1, \ldots, p_m$) if the dl-atom is monotonic; otherwise

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{c})^{s*}(\mathbf{u}) = DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{c}).$$

$\mathrm{SM}^s[F]$ is defined the same as $\mathrm{SM}[F]$ except that F^{s*} is used in place of F^*. The following theorem shows how strong answer sets can be characterized by this extension.

Theorem 34 *For any dl-program $(\mathcal{T}, \mathcal{P})$ such that $\mathcal{P}$ is variable-free, the strong answer sets of $(\mathcal{T}, \mathcal{P})$ are precisely the Herbrand interpretations of signature $\langle C, P_{\mathcal{P}} \rangle$ that satisfy $SM^s[$FO$(\mathcal{P});\ P_{\mathcal{P}}]$ relative to $\mathcal{T}$.*

[4] We require F to be variable-free because strong answer set semantics distinguishes if a ground dl-atom is monotonic or nonmonotonic.

Above, we presented two extensions of F^* to cover weak and strong semantics respectively. Further, the relationship was shown only for variable-free dl-programs. Below, we show how strong and weak semantics can be captured with a single extension of F^*, which also applies to dl-formulas with variables and does not differentiate between monotonic and nonmonotonic dl-atoms. The tradeoff is that the strong semantics can be captured only under certain conditions.

Consider any dl-formula F. We define F^{v*} by adding the following clause to the definition of F^*:

$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{t})^*(\mathbf{u}) =$$
$$DL[S_1 op_1 p'_1, \ldots, S_m op_m p'_m; Q](\mathbf{t}) \wedge$$
$$DL[S_1 op_1 p_1, \ldots, S_m op_m p_m; Q](\mathbf{t})$$

where symbol p'_i is u_i if p_i is intensional and p_i otherwise. Further, we define $\mathrm{SM}^v[F]$ the same as $\mathrm{SM}[F]$ except that F^{v*} is used in place of F^*.

The following theorem shows how strong answer sets can be characterized by this extension of SM.

Theorem 35 *For any dl-program $(\mathcal{T}, \mathcal{P})$ such that every occurrence of $\ominus$ is in the scope of negation, the strong answer sets of $(\mathcal{T}, \mathcal{P})$ are precisely the Herbrand interpretations of $\langle C, P_{\mathcal{P}} \rangle$ that satisfy $SM^v[FO(\mathcal{P});\ P_{\mathcal{P}}]$ relative to $\mathcal{T}$.*

The syntactic condition about $\ominus$ in Theorem 35 ensures that all dl-atoms in the positive bodies of the ground program are monotonic. The statement does not hold if the condition is dropped. For example, consider the dl-program $(\mathcal{T}, \mathcal{P})$ where the axioms in $\mathcal{T}$ (written as first-order formulas) are

$$Q(b),$$
$$\forall x(\neg S(x) \rightarrow Q(x))$$

and $\mathcal{P}$ is

$$p(x) \leftarrow DL[S \ominus p; Q](x).$$

For $C = \{a, b\}$, $DL[S \ominus p; Q](b)$ is monotonic but $DL[S \ominus p; Q](a)$ is not. One can check that $(\mathcal{T}, \mathcal{P})$ has no strong answer sets, but $\{p(a), p(b)\}$ is an Herbrand model of $\mathrm{SM}^v[FO(\mathcal{P}); p]$.

In the case of weak answer set semantics, the condition is not required, but instead we need to prepend $\neg\neg$ to all dl-atoms.

Theorem 36 *For any dl-program $(\mathcal{T}, \mathcal{P})$, the weak answer sets of $(\mathcal{T}, \mathcal{P})$ are precisely the Herbrand interpretations of signature $\langle C, P_{\mathcal{P}}\rangle$ that satisfy $SM^v[$FO$(\mathcal{P})^{\neg};\ P_{\mathcal{P}}]$ relative to $\mathcal{T}$, where FO$(\mathcal{P})^{\neg}$ is obtained from FO$(\mathcal{P})$ by prepending $\neg\neg$ to all occurrences of dl-atoms.*

10.6 Strong Equivalence of Hybrid Knowledge Bases

Given two classically equivalent theories F and G of the same signature, and a theory H containing F, replacing F in the theory with G will not change the models of H. However, this property does not hold in general for answer set programs, and as a result, it also does not hold in general for hybrid knowledge bases. For instance, consider a hybrid knowledge base where $\mathcal{T}$ is empty and $\mathcal{P}$ consists of the following rules:

$$p$$

$$p \leftarrow p.$$

$\mathrm{SM}[$FO$(\mathcal{T}) \wedge$ FO$(\mathcal{P}); p]$ is consistent. Let $\mathcal{P}'$ be the theory obtained from $\mathcal{P}$ by replacing the second rule with $\bot \leftarrow p$. Even though $p \leftarrow p$ and $\bot \leftarrow p$ have the same stable models, $\mathrm{SM}[$FO$(\mathcal{T}) \wedge$ FO$(\mathcal{P}'); p]$ is inconsistent.

The notion of strong equivalence (Ferraris et al., 2011), reviewed in Chapter 3.2, ensures that the stable models of a formula F do not change if a subformula G is replaced with a formula that strongly equivalent to G. Similar to (de Bruijn et al., 2007b), we carry over this notion to hybrid knowledge bases.

We say that two hybrid knowledge bases $(\mathcal{T}_1, \mathcal{P}_1)$ and $(\mathcal{T}_2, \mathcal{P}_2)$ of the same signature $\Sigma_\mathcal{T} \cup \Sigma_\mathcal{P}$ are strongly equivalent if the formula

$$\mathbf{u} \leq \mathit{pr}(\Sigma_\mathcal{P} \setminus \Sigma_\mathcal{T}) \rightarrow \big((\mathit{FO}(\mathcal{T}_1) \wedge \mathit{FO}(\mathcal{P}_1))^*(\mathbf{u}) \leftrightarrow (\mathit{FO}(\mathcal{T}_2) \wedge \mathit{FO}(\mathcal{P}_2))^*(\mathbf{u})\big)$$

is logically valid where $\mathit{pr}(\Sigma_\mathcal{P} \setminus \Sigma_\mathcal{T})$ is the set of all predicates constants in $\Sigma_\mathcal{P}$ that do not belong to $\Sigma_\mathcal{T}$.

10.7 Related Work

Like our extension, the QEL-based approach (de Bruijn et al., 2007b) was extended to cover dl-programs in (Fink & Pearce, 2010). In that paper, the authors capture the weak (strong, respectively) semantics of dl-programs by defining weak (strong, respectively) QHT models of dl-atoms. The extensions F^{w*} and F^{s*} of F^* we presented above are syntactic counterparts of these definitions of QHT models. In (de Bruijn, Pearce, Polleres, & Valverde, 2010), the authors slightly modify the definition of SM and show how the modified definition can be used to integrate rules and ontologies.

Motik and Rosati (2010) present an unification of DL knowledge bases and logic programs in the framework of the logic of Minimal Knowledge and Negation as Failure (MKNF) (Lifschitz, 1991). They also show how their framework captures several of the existing integration approaches. In order to capture the semantics of dl-programs, they unfold dl-atoms into MKNF formulas. For instance,

$$DL[Q \oplus p; R](a)$$

is unfolded into the formula (Motik & Rosati, 2010, Definition 7.5)

$$\mathbf{K}\Big(\big(\mathit{FO}(\mathcal{T}) \wedge \forall x(p(x) \rightarrow Q(x))\big) \rightarrow R(a)\Big)$$

where $\mathcal{T}$ is the DL knowledge base. Such an unfolding does not work with our semantics. For example, consider $\mathcal{K} = (\mathcal{T}, \mathcal{P})$ such that $\mathcal{T}$ is empty, and $\mathcal{P}$ is the

following:

$$\begin{array}{rcl} p(a) & \leftarrow & DL[Q \oplus p; R](a) \\ & \leftarrow & \textit{not}\, p(a). \end{array} \tag{10.7}$$

Since $DL[Q \oplus p; R](a)$ cannot be satisfied by any set of atoms formed using $\langle a, p/1\rangle$, $\mathcal{P}$ has neither strong nor weak answer sets. On the other hand, if we represent $DL[Q \oplus p; R](a)$ by

$$(\forall x(p(x) \rightarrow Q(x))) \rightarrow R(a),$$

the FOL-representation of (10.7) is

$$F = \Big(\big((\forall x(p(x) \rightarrow Q(x))) \rightarrow R(a) \big) \rightarrow p(a) \Big) \wedge \neg\neg p(a)$$

and $\mathrm{SM}[F; p]$ is satisfiable.

10.8 Proofs

Proof of Theorem 31

Since *FO*($\mathcal{T}$) contains no occurrences of predicates from $P_{\mathcal{P}}$, from the splitting theorem (Theorem 12), it follows that

$$\mathrm{SM}[\textit{FO}(\mathcal{T}) \wedge \textit{FO}(\mathcal{P});\ P_{\mathcal{P}}]$$

is equivalent to

$$\textit{FO}(\mathcal{T}) \wedge \mathrm{SM}[\textit{FO}(\mathcal{P});\ P_{\mathcal{P}}].$$

Hence the result. □

Proof of Theorem 32

If $\mathcal{P}$ is weakly safe and datalog safe, then $\mathcal{P}'$ is the same as $\mathcal{P}$. Assume that $\mathcal{P}$ is weakly safe but not datalog safe. Then, there is a rule that contains some variable y that occurs only in a negative datalog literal. $\mathcal{P}'$ is obtained from $\mathcal{P}$ by removing all negative datalog literals that contain such a variable y. By Theorem 12, since *FO*($\mathcal{T}$) contains no predicate from $P_{\mathcal{P}}$,

$$\mathrm{SM}[\textit{FO}(\mathcal{T}) \wedge \textit{FO}(\mathcal{P}); P_{\mathcal{P}}]$$

is equivalent to

$$\mathit{FO}(\mathcal{T}) \wedge \mathrm{SM}[\mathit{FO}(\mathcal{P}); P_{\mathcal{P}}].$$

Similarly

$$\mathrm{SM}[\mathit{FO}(\mathcal{T}) \wedge \mathit{FO}(\mathcal{P}'); P_{\mathcal{P}}]$$

is equivalent to

$$\mathit{FO}(\mathcal{T}) \wedge \mathrm{SM}[\mathit{FO}(\mathcal{P}'); P_{\mathcal{P}}].$$

So, it is sufficient to show that $\mathrm{SM}[\mathit{FO}(\mathcal{P}); P_{\mathcal{P}}]$ is equivalent to $\mathrm{SM}[\mathit{FO}(\mathcal{P}'); P_{\mathcal{P}}]$. Since $\mathcal{P}$ is semi-safe relative to $P_{\mathcal{P}}$, $\mathcal{P}'$ is also semi-safe relative to $P_{\mathcal{P}}$. By Proposition 5 and Theorem 9 from (Ferraris et al., 2011), it is sufficient to show that under the assumption $\mathit{SPP}^{P_{\mathcal{P}}}_{c(\mathcal{P})}$, where $c(\mathcal{P})$ is the set of object constants occurring in $\mathcal{P}$,

$$(\mathbf{q} \leq P_{\mathcal{P}}) \rightarrow (\mathit{FO}(\mathcal{P})^*(\mathbf{q}) \leftrightarrow \mathit{FO}(\mathcal{P}')^*(\mathbf{q})) \tag{10.8}$$

is logically valid. Given a rule, let $F(\mathbf{y})$ be the conjunction of negative datalog literals that contain a variable occurring only in a negative datalog literal, where $\mathbf{y}$ is the list of all such variables. Formula (10.8) is logically valid, since $(\exists \mathbf{y} F(\mathbf{y}))^*(\mathbf{q})$ is equivalent to $\exists \mathbf{y} F(\mathbf{y})$ under the assumption $\mathbf{q} \leq P_{\mathcal{P}}$, and $\exists \mathbf{y} F(\mathbf{y})$ is equivalent to $\top$ under the assumption $\mathit{SPP}^{P_{\mathcal{P}}}_{c(\mathcal{P})}$ (the extents of predicates in $P_{\mathcal{P}}$ are finite) and the standard name assumption (there are infinitely many objects in the domain). □

Proof of Theorem 34

In the following, X and Y are Herbrand interpretations of $\langle C, P_{\mathcal{P}} \rangle$ such that Y is a subset of X (we identify an Herbrand interpretation with the set of ground atoms that are true in it), $\mathbf{q}$ is a list of new predicate constants of the same length as $P_{\mathcal{P}}$, and $Y^{P_{\mathcal{P}}}_{\mathbf{q}}$ is obtained from Y by replacing every predicate constant in $P_{\mathcal{P}}$ with the corresponding predicate constant in $\mathbf{q}$.

Proof of Theorem 34 It is clear that $X \models_{\mathcal{T}} \mathcal{P}$ iff $X \models_{\mathcal{T}} s\mathcal{P}^X_{\mathcal{T}}$. If $X \not\models_{\mathcal{T}} \mathcal{P}$, then X is not a strong answer set of $\langle \mathcal{T}, \mathcal{P} \rangle$, and X does not satisfy $\mathrm{SM}^s[\mathit{FO}(\mathcal{P});\ P_{\mathcal{P}}]$ relative to $\mathcal{T}$.

Assume $X \models_{\mathcal{T}} \mathcal{P}$. It is sufficient to prove that, for any rule (10.6) in $\mathcal{P}$, and any Herbrand interpretation Y that is a subset of X,

$$Y \models_{\mathcal{T}} s(a \leftarrow B, N)^X_{\mathcal{T}}$$

iff

$$X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \models_{\mathcal{T}} (B \wedge N)^{s*}(\mathbf{q}) \rightarrow a^{s*}(\mathbf{q}).$$

Case 1: $s(a \leftarrow B, N)^X_{\mathcal{T}}$ is empty. Clearly, $Y \models_{\mathcal{T}} s(a \leftarrow B, N)^X_{\mathcal{T}}$. It also follows that $X \models_{\mathcal{T}} b$ for some *not* b in N, or $X \not\models_{\mathcal{T}} b$ for some $b \in \mathit{DL}^?_{\mathcal{P}} \cap B$.

Subcase 1: Assume that $X \models_{\mathcal{T}} b$ for some *not* b in N. It follows from the definition of F^{s*} that $X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \not\models_{\mathcal{T}} (\neg b)^{s*}(\mathbf{q})$. Consequently, it follows that $X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \models_{\mathcal{T}} (B \wedge N)^{s*}(\mathbf{q}) \rightarrow a^{s*}(\mathbf{q})$.

Subcase 2: Assume that $X \not\models_{\mathcal{T}} b$ for some $b \in \mathit{DL}^?_{\mathcal{P}} \cap B$. It follows from the definition of F^{s*} that $X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \not\models_{\mathcal{T}} (b)^{s*}(\mathbf{q})$. Consequently, it follows that $X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \models_{\mathcal{T}} (B \wedge N)^{s*}(\mathbf{q}) \rightarrow a^{s*}(\mathbf{q})$.

Case 2: $s(a \leftarrow B, N)^X_{\mathcal{T}}$ is not empty. Assume that $s(a \leftarrow B, N)^X_{\mathcal{T}}$ is $a \leftarrow B_1$. This implies that $X \not\models_{\mathcal{T}} b$ for every *not* b in N, and $X \models_{\mathcal{T}} b$ for every $b \in \mathit{DL}^?_{\mathcal{P}} \cap B$. If b is monotonic, then $b^{s*}(\mathbf{q}) \rightarrow b$ is logically valid, otherwise, $b^{s*}(\mathbf{q}) = b$. This implies that $X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \models_{\mathcal{T}} N^{s*}(\mathbf{q})$, and $X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \models_{\mathcal{T}} b^{s*}(\mathbf{q})$ for all $b \in \mathit{DL}^?_{\mathcal{P}} \cap B$. Since a is an atom and B_1 is a set of atoms and monotonic dl-atoms, it is clear that $Y \models_{\mathcal{T}} a \leftarrow B_1$ iff $X \cup Y^{P_{\mathcal{P}}}_{\mathbf{q}} \models_{\mathcal{T}} B^{s*}(\mathbf{q}) \rightarrow a^{s*}(\mathbf{q})$. □

Proof of Theorem 33

Similar to the proof of Theorem 34 shown above, using the fact that for every dl-atom b, $b^{w*}(\mathbf{q}) = b$.

Proof of Theorem 35

The following lemma extends Lemma 8 to dl-formulas.

Lemma 16 *For any dl-formula F, $X \cup Y_{\boldsymbol{q}}^{P_{\mathcal{P}}} \models_{\mathcal{T}} (\neg F)^*(\boldsymbol{q})$ iff $X \models_{\mathcal{T}} \neg F$.*

Proof. By induction on F. □

Proof of Theorem 35 Without loss of generality, let us assume that $\mathcal{P}$ is a variable-free program obtained by grounding. It is clear that $X \models_{\mathcal{T}} \mathcal{P}$ iff $X \models_{\mathcal{T}} s\mathcal{P}_{\mathcal{T}}^X$. If $X \not\models_{\mathcal{T}} \mathcal{P}$, then X is not a strong answer set of $\langle \mathcal{T}, \mathcal{P} \rangle$, and X does not satisfy $\mathrm{SM}^v[\mathit{FO}(\mathcal{P});\ P_{\mathcal{P}}]$ relative to $\mathcal{T}$.

Assume $X \models_{\mathcal{T}} \mathcal{P}$. It is sufficient to prove that, for any rule (10.6) in $\mathcal{P}$, and any Herbrand interpretation Y that is a subset of X,

$$Y \models_{\mathcal{T}} s(a \leftarrow B, N)_{\mathcal{T}}^X$$

iff

$$X \cup Y_{\mathbf{q}}^{P_{\mathcal{P}}} \models_{\mathcal{T}} (B \wedge N)^*(\mathbf{q}) \rightarrow a^*(\mathbf{q}).$$

Case 1: $s(a \leftarrow B, N)_{\mathcal{T}}^X$ is empty. Clearly, $Y \models_{\mathcal{T}} s(a \leftarrow B, N)_{\mathcal{T}}^X$. Since no dl-atom in B mentions $\ominus$, B contains no dl-atoms from $\mathit{DL}_{\mathcal{P}}^?$, and it follows that $X \models_{\mathcal{T}} b$ for some *not* b in N. Consequently, by Lemma 16, it follows that $X \cup Y_{\mathbf{q}}^{P_{\mathcal{P}}} \not\models_{\mathcal{T}} (\neg b)^*(\mathbf{q})$, so that $X \cup Y_{\mathbf{q}}^{P_{\mathcal{P}}} \models_{\mathcal{T}} (B \wedge N)^*(\mathbf{q}) \rightarrow a^*(\mathbf{q})$.

Case 2: $s(a \leftarrow B, N)_{\mathcal{T}}^X$ is not empty. Since no dl-atom in B mentions $\ominus$, B contains no dl-atoms from $\mathit{DL}_{\mathcal{P}}^?$, and it follows that $s(a \leftarrow B, N)_{\mathcal{T}}^X$ is $a \leftarrow B$. Also it

follows that $X \not\models_{\mathcal{T}} b$ for every *not* b in N, so that by Lemma 16, we get $X \cup Y_{\mathbf{q}}^{P_{\mathcal{P}}} \models_{\mathcal{T}} N^*(\mathbf{q})$. Since a is an atom and B is a set of atoms and monotonic dl-atoms, it is clear that $Y \models_{\mathcal{T}} a \leftarrow B$ iff $X \cup Y_{\mathbf{q}}^{P_{\mathcal{P}}} \models_{\mathcal{T}} B^*(\mathbf{q}) \rightarrow a^*(\mathbf{q})$. □

Proof of Theorem 36

Similar to the proof of Theorem 35, using the fact that, by Lemma 16, for any dl-atom A, $X \cup Y_{\mathbf{q}}^{P_{\mathcal{P}}} \models_{\mathcal{T}} (\neg\neg A)^*(\mathbf{q})$ iff $X \models_{\mathcal{T}} \neg\neg A$. □

Chapter 11

INTEGRATING ACTION THEORIES AND ONTOLOGIES

In this chapter, we present an approach to integrate circumscriptive actions theories with ontologies, and show how this approach can be related to our approach for integrating rules and ontologies. We present an application of integrating action theories and ontologies in the context of a medical expert system that is required to assist physicians in diagnosis, treatment, and drug prescription. We also demonstrate our approach by using DLVHEX[1], which is a well-known tool for reasoning with HEX-programs (Eiter, Ianni, Schindlauer, & Tompits, 2005).

11.1 Integrating Circumscriptive Action Theories and Ontologies

As before, we consider ontologies that are based on Description Logics (DLs). Since circumscription extends first-order logic, integrating circumscriptive action theories and ontologies is rather straightforward. Below, we present our approach.

First, since the underlying signatures of the circumscriptive theories considered in this article are many-sorted, we define the union of two many-sorted signatures as follows. We denote a many-sorted signature by a pair (S, C) where S is a set of sorts including the *boolean* sort $\{$*true*, *false*$\}$, and C is a set of constants such that each constant in C of arity n is associated with *argument* sorts $s_1, \ldots, s_n \in S$ and a *value* sort $s \in S$. Given two many-sorted signatures $\Sigma_1 = (S_1, C_1)$ and $\Sigma_2 = (S_2, C_2)$ where each constant in $C_1 \cap C_2$ is associated with the same arguments sorts and the same value sort in both the signatures, by $\Sigma_1 \cup \Sigma_2$, we denote the signature $\Sigma_{12} = (S_1 \cup S_2, C_1 \cup C_2)$ such that for every constant $c \in C_1 \cup C_2$, if $c \in C_i$, then c is associated with the same argument and value sorts in Σ_{12} and Σ_i.

[1] http://www.kr.tuwien.ac.at/research/systems/dlvhex/

Given a circumscriptive action theory $\mathcal{CA}$ of a signature $\Sigma_{\mathcal{CA}}$, and a DL knowledge base $\mathcal{T}$ of a signature $\Sigma_{\mathcal{T}}$ that does not contain any predicate constant from $\Sigma_{\mathcal{CA}}$ that takes a timepoint/situation argument, the *hybrid action theory* $(\mathcal{T}, \mathcal{CA})$ can simply be identified with the theory

$$\mathcal{CA} \wedge \mathit{FO}(\mathcal{T})$$

of signature $\Sigma_{\mathcal{CA}} \cup \Sigma_{\mathcal{T}}$ where $\mathit{FO}(\mathcal{T})$ is the first-order representation of $\mathcal{T}$. Below, we make this precise for each of the circumscriptive action theories considered in this article.

Definition 4 *Consider an event calculus theory (Chapter 7.1) $\mathcal{EC}$ (of a signature $\Sigma_{\mathcal{EC}}$) of the form*

$$\begin{aligned} CIRC[\Sigma\ ;\ \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge CIRC[\Delta\ ;\ \mathit{Happens}] \\ \wedge\ CIRC[\Theta\ ;\ \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge\ \Xi, \end{aligned} \tag{11.1}$$

and a DL knowledge base $\mathcal{T}$ of a signature $\Sigma_{\mathcal{T}}$ that does not contain predicate constants from $\Sigma_{\mathcal{EC}}$ that take arguments of sort timepoint *(for example, HoldsAt, Initiates, Terminates, Releases, Happens, etc).*

The hybrid action theory $(\mathcal{T}, \mathcal{EC})$ is the theory

$$\begin{aligned} CIRC[\Sigma\ ;\ \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge CIRC[\Delta\ ;\ \mathit{Happens}] \\ \wedge\ CIRC[\Theta\ ;\ \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge\ \Xi \wedge \mathit{FO}(\mathcal{T}) \end{aligned} \tag{11.2}$$

of signature $\Sigma_{\mathcal{EC}} \cup \Sigma_{\mathcal{T}}$.

Definition 5 *Consider a Lin's causal situation calculus theory (Chapter 8.1) $\mathcal{SC}$ (of a signature $\Sigma_{\mathcal{SC}}$) of the form*

$$CIRC[\mathcal{D}_{caused};\ \mathit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}_{rest}, \tag{11.3}$$

and a DL knowledge base $\mathcal{T}$ of a signature $\Sigma_{\mathcal{T}}$ that does not contain predicate constants from $\{$Holds, Caused, Poss$\}$.

The hybrid action theory $(\mathcal{T}, \mathcal{SC})$ is the theory

$$CIRC[\mathcal{D}_{caused};\ \textit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}_{rest} \wedge \textit{FO}(\mathcal{T}) \tag{11.4}$$

of signature $\Sigma_{\mathcal{SC}} \cup \Sigma_{\mathcal{T}}$.

Definition 6 *Consider a Temporal Action Logics (TAL) theory (Chapter 9.1) $\mathcal{TAL}$ (of a signature $\Sigma_{\mathcal{TAL}}$) of the form*

$$CIRC[\Gamma_{occ}; \textit{Occurs}] \wedge CIRC[\Gamma_{depc} \wedge \Gamma_{acs}; \textit{Occlude}] \wedge$$
$$\Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc}, \tag{11.5}$$

and a DL knowledge base $\mathcal{T}$ of a signature $\Sigma_{\mathcal{T}}$ that does not contain predicate constants from $\{$Holds, Occurs, Occlude$\}$.

The hybrid action theory $(\mathcal{T}, \mathcal{TAL})$ is the theory

$$CIRC[\Gamma_{occ}; \textit{Occurs}] \wedge CIRC[\Gamma_{depc} \wedge \Gamma_{acs}; \textit{Occlude}] \wedge$$
$$\Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc} \wedge \textit{FO}(\mathcal{T}) \tag{11.6}$$

of signature $\Sigma_{\mathcal{TAL}} \cup \Sigma_{\mathcal{T}}$.

11.2 Relating Hybrid Action Theories to Hybrid Knowledge Bases

We now show how the above integration approach can be related to our approach for integrating DLs and ASP-rules presented in Chapter 10.1.

Since we have already related the circumscriptive action theories to ASP (Chapters 7.3, 8.1, and 9.3), we can use those results to relate hybrid action theories to hybrid knowledge bases. The following theorems give the precise relationship.

Theorem 37 *Let $(\mathcal{T}, \mathcal{EC})$ be a hybrid action theory (11.2) of signature $\Sigma_{\mathcal{EC}} \cup \Sigma_{\mathcal{T}}$, and let F be the FOL-representation of the program obtained from $\mathcal{EC}$ by applying translation EC2ASP (Chapter 7.3). Then the following theories are equivalent $(\Sigma_{\mathcal{EC}} \cup \Sigma_{\mathcal{T}})$-equivalent to (11.2) :* [2]

(a) $SM[\Sigma\ ;\ \mathit{I}, \mathit{T}, \mathit{R}] \wedge SM[\Delta\ ;\ \mathit{H}] \wedge\ SM[\Theta\ ;\ \mathit{Ab}_1, \ldots, \mathit{Ab}_n] \wedge\ \Xi \wedge \mathit{FO}(\mathcal{T})$*;*

(b) $SM[\Sigma \wedge \Delta \wedge \Theta \wedge \Xi \wedge \mathit{FO}(\mathcal{T})\ ;\ \mathit{I}, \mathit{T}, \mathit{R}, \mathit{H}, \mathit{Ab}_1, \ldots, \mathit{Ab}_n]$*;*

(c) $SM[F \wedge \mathit{FO}(\mathcal{T})\ ;\ \mathit{pr}(F)]$*.*

Notice that the formula (c) above is essentially how the hybrid knowledge base $(\mathcal{T}, \mathcal{P})$, where $\mathcal{P}$ is the ASP-representation of F, is identified according to our approach for integrating rules and ontologies (Chapter 10.1). The theorems below show similar results with respect to the situation calculus and TAL.

Theorem 38 *Let $(\mathcal{T}, \mathcal{SC})$ be a hybrid action theory (11.4) of signature $\Sigma_{\mathcal{SC}} \cup \Sigma_{\mathcal{T}}$, and let F be the FOL-representation of the program obtained by applying translation F2LP on*

$$\mathcal{D}_{caused} \wedge \mathcal{D}_{poss\rightarrow} \wedge \mathcal{D}^{-}_{rest} \wedge \mathcal{D}_{sit} \tag{11.7}$$

(Chapter 8.1) with intensional predicates $\{\mathit{Caused}, \mathit{Poss}, \mathit{Sit}\}$. Then the following theories are $(\Sigma_{\mathcal{SC}} \cup \Sigma_{\mathcal{T}})$-equivalent to (11.4) :

(a) $SM[\mathcal{D}_{caused};\ \mathit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}_{rest} \wedge \mathit{FO}(\mathcal{T})$*;*

(b) $SM[\mathcal{D}_{caused} \wedge \mathcal{D}_{poss\rightarrow} \wedge \mathcal{D}^{-}_{rest} \wedge \mathcal{D}_{sit} \wedge \mathit{FO}(\mathcal{T})\ ;\ \mathit{Caused}, \mathit{Poss}, \mathit{Sit}]$*;*

(c) $SM[F \wedge \mathit{FO}(\mathcal{T})\ ;\ \mathit{pr}(F)]$*.*

[2] For simplicity, the names of the circumscribed predicates are abbreviated. As before, we assume that Ξ is equivalently rewritten so that it does not contain strictly positive occurrences of the intensional predicates.

In the following theorem, we use Γ_{ncirc} to denote the conjunction $\Gamma_{fnd} \wedge \Gamma_{time} \wedge \Gamma_{per} \wedge \Gamma_{obs} \wedge \Gamma_{domc}$.

Theorem 39 *Let $(\mathcal{T}, \mathcal{TAL})$ be a hybrid action theory (11.6) of signature $\Sigma_{\mathcal{TAL}} \cup \Sigma_{\mathcal{T}}$, and let F be the FOL-representation of the program obtained by applying translation F2LP on*

$$\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc}$$

with intensional predicates $\{$Occurs, Occlude$\}$. Then the following theories are $(\Sigma_{\mathcal{TAL}} \cup \Sigma_{\mathcal{T}})$-equivalent to (11.6) :

(a) $SM[\Gamma_{occ}; \textit{Occurs}] \wedge SM[\Gamma_{depc} \wedge \Gamma_{acs}; \textit{Occlude}] \wedge \Gamma_{ncirc} \wedge \textit{FO}(\mathcal{T})$*;*

(b) $SM[\Gamma_{occ} \wedge \Gamma_{depc} \wedge \Gamma_{acs} \wedge \Gamma_{ncirc} \wedge \textit{FO}(\mathcal{T}) \ ; \ \textit{Occurs}, \textit{Occlude}]$*;*

(c) $SM[F \wedge \textit{FO}(\mathcal{T}) \ ; \ \textit{pr}(F)]$.

11.3 A Simple Application in the Healthcare/Biomedical Domain

Consider a medical expert system that is required to assist physicians in diagnosis and treatment of diseases/disorders. One of the important tasks of such a system would be to determine if a particular drug can be administered to the patient given his/her condition. To accomplish this task, an important question that the system needs to consider is "given the current condition of the patient, will prescribing a particular drug be beneficial to the patient or will it have adverse effects?". A lot of knowledge about the positive effects and contraindications of various drugs is available in the form of ontologies such as the National Drug File Ontology[3]. One possible way to answer the above question is to simply query the ontologies to check if a drug may treat the patient and if the patient is exhibiting any symptoms that are contraindications for prescribing the drug. However, such *static* inference

[3]http://bioportal.bioontology.org/ontologies/40402

might be insufficient in certain cases and one might need to also consider if the current patient condition might lead to another condition which is a contraindication for prescribing the drug.

For example, consider the case where a patient is suffering from *Gastritis* and complains of abdominal *pain*. The job of the system is to determine if *Aspirin* can be recommended in this context. Assume that the biomedical ontology being queried has the information that "Aspirin may treat pain" and "Gastrointestinal Bleeding is a contraindication for administering Aspirin". Further, assume that we have the knowledge that Gastritis usually causes Gastrointestinal Bleeding. Now, if the system simply considers the current patient symptoms, it will simply conclude that Aspirin can be prescribed. However, since Gastritis usually causes Gastrointestinal Bleeding, this might not be the preferred treatment. On the other hand, if the system considers the knowledge about the effects of Gastritis, then it will be able to conclude that Aspirin is not recommended.

From the above example, it is clear that the system needs to consider what conditions the current patient condition might *cause* and then check whether any of the conditions is a contraindication to prescribing the drug. So, in addition to being able to query ontologies, the system also needs to be able to reason about various *cause-effect* relationships.

Following is a partial event calculus description for reasoning about the above context. In the description, there are three fluent names and 2 action names. Fluent *Administered*(d) is used to represent whether a drug d has been administered to the patient, fluent *Condition*(c) is used to represent if the patient is currently suffering from disease/disorder c, and fluent *SideEffect*(d) is used to represent if the drug d has a side-effect on the patient. Action *Admin*(d) is used to represent the action of administering drug d, and *DummyEvent*(c) is used to simulate the effects of condition c. The predicates in all capitals (`MAY_TREAT` and

CONTRAINDICATION) are DL-predicates. $\texttt{MAY\_TREAT}(d, c)$ represents that drug d may treat c, and $\texttt{CONTRAINDICATION}(d, c)$ represents that c is a contraindication for administering drug d.

$$\textit{Initiates}(\textit{Admin}(d), \textit{Administered}(d), t)$$

$$\textit{HoldsAt}(\textit{Condition}(c), t) \wedge \texttt{MAY\_TREAT}(d, c) \wedge \neg \textit{Ab}(d, t) \rightarrow \textit{Terminates}(\textit{Admin}(d), \textit{Condition}(c), t)$$

$$\textit{Ab}(d, t) \rightarrow \textit{Initiates}(\textit{Admin}(d), \textit{SideEffect}(d), t)$$

$$\textit{HoldsAt}(\textit{Condition}(c), t) \wedge \texttt{CONTRAINDICATION}(d, c) \rightarrow \textit{Ab}(d, t)$$

These formulas represent the direct effects of administering a drug. The first formula represents that $\textit{Admin}(d)$ causes $\textit{Administered}(d)$ to be true after t. The second formula represents the positive effect of drug d w.r.t to treating the patient who is suffering from condition c. The third formula represents that administering a drug has a side-effect on the patient if the drug is contraindicated by a condition c from which the patient is suffering.

$$\neg \textit{Ab}_1(\textit{Condition}(\textit{Gastritis}), t) \rightarrow \textit{Initiates}(\textit{DummyEvent}(\textit{Gastritis}), \textit{Condition}(\textit{GastrointestinalBleeding}), t)$$

$$\textit{HoldsAt}(\textit{Condition}(c), t) \rightarrow \textit{Happens}(\textit{DummyEvent}(c), t)$$

$$\textit{Initiates}(e, \textit{Condition}(c), t) \wedge \textit{HoldsAt}(\textit{Administered}(d), t) \wedge \texttt{CONTRAINDICATION}(d, c) \rightarrow \textit{Initiates}(e, \textit{SideEffect}(d), t)$$

The first two formulas above represent that “Gastritis usually causes Gastrointestinal Bleeding". The last formula represents the indirect effects of an event e that causes condition c. Essentially it says that if an event e causes condition c, and if a drug which is contraindicated by c has been administered, then the indirect effect of e is a side-effect.

$$\neg \textit{ReleasedAt}(f, 0)$$

$$\textit{HoldsAt}(\textit{Condition}(\textit{Gastritis}), 0) \wedge \neg \textit{HoldsAt}(\textit{Administered}(d), 0) \wedge$$
$$\textit{HoldsAt}(\textit{Condition}(\textit{Pain}), 0) \wedge \neg \textit{HoldsAt}(\textit{SideEffect}(d), 0) \wedge$$

$$\neg \textit{HoldsAt}(\textit{Condition}(\textit{GastrointestinalBleeding}, \textit{Tom}), 0)$$

The first formula above represents that all the fluents are initially inertial, and the second formula represents the initial state. In addtion to the above formulas, there are other axioms such as the unique name axioms for the fluents and actions, and the domain-independent axioms of the discrete event calculus (Chapter 7.1). Now, assuming that the ontology being queried entails the facts

- MAY_TREAT(*Aspirin*, *Pain*),
- MAY_TREAT(*Acetaminophen*, *Pain*), and
- CONTRAINDICATION(*Aspirin*, *GastrointestinalBleeding*),

one can verify the following from the above description:

- *HoldsAt*(*Condition*(*GastrointestinalBleeding*), τ) holds for all $\tau > 0$;
- if *Happens*(*Admin*(*Aspirin*), τ) holds for some timepoint $\tau > 0$, then *HoldsAt*(*SideEffect*(*Aspirin*), τ_1) holds for all $\tau_1 > \tau$;
- if *Happens*(*Admin*(*Aspirin*), τ) holds for timepoint $\tau = 0$, then *HoldsAt*(*SideEffect*(*Aspirin*), τ_1) holds for all $\tau_1 \geq 2$;
- if *Happens*(*Admin*(*Acetaminophen*), τ) holds for some timepoint $\tau \geq 0$, then $\neg$*HoldsAt*(*Condition*(*Pain*), τ_1) holds for all $\tau_1 > \tau$, and $\neg$*HoldsAt*(*SideEffect*(*Acetaminophen*), τ_1) holds for all $\tau_1 \geq 0$.

From this, the system will be able to conclude that Aspirin is not recommended but Acetaminophen can be recommended.

Due to the indirect effect axiom in the above description, if Aspirin is administered at some timepoint τ, and some contraindication to Aspirin is caused at some other timepoint $\tau_1 > \tau$, then the system concludes that a side-effect is caused irrespective of the difference in the values of τ and τ_1. However, such a

conclusion might not be appropriate since the effects of Aspirin might have weared off before the contraindication is caused. One way to get over this problem is to treat action *Admin*(d) as a durative action so that a side-effect can be caused only during the duration of the action. By setting the duration of the action to the time it takes for the effects of the drug to wear off, we can avoid any inappropriate conclusions about the side-effects.

Since TAL provides an ideal framework for representing durative actions, we will use it to encode the domain. Following is a partial TAL description in the surface language. Here, fluent *Ab* is durational with default value *false*. The rest of the fluents are persistent.

acs1 $[t_1, t_2]$ *Admin*$(d) \rightarrow \big([t_1]$ *Condition*$(c) \wedge$ MAY_TREAT$(d, c) \wedge$
$\neg\exists c_1, t(t_1 \leq t \leq t_2 \wedge [t]$ *Condition*$(c_1) \wedge$ CONTRAINDICATION$(d, c_1)) \rightarrow$
$I((t_1, t_2]\ \neg$*Condition*$(c))\big)$

acs2 $[t_1, t_2]$ *Admin*$(d) \rightarrow \forall t\big(t_1 \leq t \leq t_2 \wedge [t]\ \exists c($*Condition*$(c) \wedge$
CONTRAINDICATION$(d, c)) \rightarrow I([t, t_2]$ *SideEffect*$(d))\big)$

dep1 $\forall t([t]$ *Condition*$($*Gastritis*$) \wedge \neg$*Ab*$($*Gastritis*$) \rightarrow$
$R([t+1]$ *Condition*$($*GastrointestinalBleeding*$)))$

The first statement above represents that if no contraindication to the drug d holds in the interval $[t_1, t_2]$, then administering d, which may treat c, has a positive effect on the patient who is suffering from c. The second statement represents that if a contraindication to drug d holds in the interval $[t_1, t_2]$, then administering d in that interval causes a side-effect. Finally, the dependency constraint represents that *Gastritis* usually causes Gastroinstestinal Bleeding. Note here that due to the dependency constraint, we do not need the action *DummyEvent*(c) which was used in the event calculus description. Another difference between the two descriptions is the introduction of the additional fluent *Ab* to represent abnormality. This is because, while abnormality predicates are part of the language of the event

calculus, they are not part of the language of TAL. In addition to the above statements, there are other statements such as the unique value axioms, foundational axioms, etc. (Chapter 9.1).

Given the same initial state as before and the facts corresponding to `MAY_TREAT` and `CONTRAINDICATION`, one can verify the following from the above description:

- *Condition*(*GastrointestinalBleeding*) is true for all timepoints $\tau \geq 1$;
- if *Admin*(*Aspirin*) occurs in the interval $[0, 2]$, then *SideEffect*(*Aspirin*) is true for all timepoints $\tau \geq 1$;
- if *Admin*(*Acetaminophen*) occurs in the interval $[\tau_1, \tau_2]$, then *SideEffect*(*Acetaminophen*) is false for all $\tau \geq 0$, and *Condition*(*Pain*) is false for all $\tau > \tau_1$;

Now, consider an initial state in which only *Condition*(*Pain*) is true and the rest of the fluents are false. Further, assume that *Condition*(*GastrointestinalBleeding*) is not caused in some interval $[\tau_1, \tau_2]$. If *Admin*(*Aspirin*) occurs in the interval $[\tau_1, \tau_2]$, then *Condition*(*Pain*) is false for all $\tau > \tau_1$ and no side-effect will be caused. So, the system will be able to conclude that Aspirin can be recommended, unless a contraindication to it may be caused in the interval that it is to be administered.

11.4 A Simple Demonstration Using DLVHEX

DLVHEX[4] is a system for reasoning with HEX-programs (Eiter et al., 2005). HEX-programs generalize answer set programs by allowing *higher-order* and *external* atoms. For the case when there are no higher-order atoms and all the external atoms are dl-atoms (Chapter 10.5), the semantics of HEX-programs concides with that of non-monotonic dl-programs (Eiter & Wang, 2008) which we

[4] http://www.kr.tuwien.ac.at/research/systems/dlvhex/

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns="http://www.owl-ontologies.com/drugs.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/drugs.owl">

<owl:Ontology rdf:about=""/>
<owl:Class rdf:ID="Drug"/>
<owl:Class rdf:ID="Condition"/>
<Condition rdf:ID="Pain"/>
<Condition rdf:ID="Gastritis"/>
<Condition rdf:ID="GastroIntestinalBleeding"/>

<Drug rdf:about="http://www.owl-ontologies.com/drugs.owl#Aspirin">
 <may_treat rdf:resource=
       "http://www.owl-ontologies.com/drugs.owl#Pain"/>
 <ci_with rdf:resource=
 "http://www.owl-ontologies.com/drugs.owl#GastroIntestinalBleeding"/>
</Drug>

<Drug rdf:about=
      "http://www.owl-ontologies.com/drugs.owl#Acetaminophen">
  <may_treat rdf:resource=
      "http://www.owl-ontologies.com/drugs.owl#Pain"/>
</Drug>

</rdf:RDF>
```

Figure 11.1: Sample Ontology in OWL

discussed in Chapter 10.5. So, by replacing the atoms $\texttt{MAY\_TREAT}(d, c)$ and $\texttt{CONTRAINDICATION}(d, c)$ with the corresponding dl-atoms, we can run the example discussed in the previous section in DLVHEX. The sample ontology we use is shown in Figure 11.1, which uses OWL.

DLVHEX (v. 1.7.2) uses the description logic reasoner *RacerPro*[5] for querying ontologies. For our purpose, we use RacerPro version 1.9 for which we obtained an educational license. It is clear from the above ontology that the corresponding DL/first-order theory entails the following atoms:

- `may_treat(Aspirin,Pain)`;
- `may_treat(Acetaminophen,Pain)`; and
- `contraindication(Aspirin,GastroIntestinalBleeding)`.

In order to run the event calculus and TAL descriptions using DLVHEX, we use F2LP to turn the descriptions into answer set programs. However, since F2LP outputs programs in the language of GRINGO and LPARSE, we use the python program `f2lpdlv.py`[6] to turn the output of F2LP into the language of DLV and invoke DLVHEX on the resulting program. Also, since DLVHEX (v. 1.7.2) does not allow function constants (of arity > 0) in the input program, we cannot use reified constants to represent fluents as we do in the event calculus and TAL descriptions. To work around this issue, we *flatten* the fluents by introducing a predicate `equals`. For instance, `equals(f1,condition,pain)` represents that the constant `f1` represents the fluent `condition(pain)`. Similarly, `equals(e3,admin,aspirin)` represents that the constant `e3` represents the action/event `administer(aspirin)`. We include all these *declarations* and other facts in a data file, which is shown in Figure 11.2.

In the encoding shown in the figure, DRUGS is used as an abbreviation for the Unique Resource Identifier (URI)

http://www.owl-ontologies.com/drugs.owl

[5](Racer Systems GmbH & Co. KG) http://www.racer-systems.com
[6]http://reasoning.eas.asu.edu/f2lp

which represents the namespace for the ontology. The actual encoding uses the latter. Similarly, FILE-LOC is an abbreviation for the actual location of the file. Atoms `&dlC[...](X)` and `&dlR[...](D,C)` are the dl-atoms in the language of DLVHEX that are used to query DL concepts and roles respectively.[7] For instance, the last rule in Figure 11.2 simply retrieves all pairs `(d,c)` such that `may_treat(d,c)` is entailed by the ontology shown in Figure 11.1, and adds them to the extent of `may_treat`. Note that while `may_treat` inside the construct `&dlR[...]` is a description logic predicate, the same predicate outside the construct belongs to the signature of the event calculus/TAL description. The construct `#spatom{...}` directs F2LP to copy any thing within the construct directly into the output program without any processing. This is useful to encode constructs that are not recognized by F2LP but are supported by the answer set solvers. For example, F2LP turns the last rule in Figure 11.2 into the following rule:

```
may_treat(D,C) :- &dlR["file:FILE-LOC",a,b,c,d,"may_treat"](D,C).
```

Figure 11.3 shows the F2LP encoding of the event calculus description discussed in the previous section. To check if Aspirin can be administered to the patient at timepoint 0, we add the following to the description in Figure 11.3.

```
% query for Aspirin
#spatom{equals(E,admin,"<DRUGS#Aspirin>")} -> happens(E,0).
-(#spatom{equals(F,condition,"<DRUGS#Pain>")} &
  #spatom{equals(F1,sideEffect,"<DRUGS#Aspirin>")} &
  -?[T2]:(holdsAt(F,T2) & T2 > 0) &
  -?[T1]: (holdsAt(F1,T1) & T1 > 0)).
```

The first formula represents that Aspirin is administered at timepoint 0. The second formula is the negation of the query we want to check. The query here is to check if administering Aspirin treats the patient such that no side-effect is

[7] http://www.kr.tuwien.ac.at/research/systems/dlvhex/dlplugin.html

caused after administering it. We can run the resulting theory using `f2lpdlv.py` as shown below.

```
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:13:53)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import f2lpdlv
>>> f2lpdlv.run('f2lp usecase-data usecase-ec dec',
                                  'dlvhex --filter=holdsAt')
```

The first argument to the program is the *complete* command line to run F2LP on the description. The second argument is the (partial) command line for invoking DLVHEX. The option `--filter=holdsAt` is used to supress all the predicates except `holdsAt` in the output. This produces the following answer set (the existence of an answer set indicates that the query is not entailed):

```
{ holdsAt(f6,1), holdsAt(f6,2), holdsAt(f4,2), holdsAt(f3,1),
  holdsAt(f3,2), holdsAt(f2,0), holdsAt(f2,1), holdsAt(f2,2),
  holdsAt(f1,0) }
```

In this output, `holdsAt(f4,2)` indicates that a side-effect is caused at timepoint 2. If we change the timepoint at which Aspirin is administered to 1, then we get the following output.

```
{ holdsAt(f6,2), holdsAt(f4,2), holdsAt(f3,1), holdsAt(f3,2),
  holdsAt(f2,0), holdsAt(f2,1), holdsAt(f2,2), holdsAt(f1,0),
  holdsAt(f1,1), holdsAt(f1,2) }
```

This output shows that in addition to a side-effect being caused, the pain is also not relieved (represented by `holdsAt(f1,1), holdsAt(f1,2)`). This is because at the time of administering Aspirin, fluent `f3` (representing Gastrointestinal Bleeding) was true, thus defeating the positive effect axiom.

Running similar queries by replacing Aspirin with Acetaminophen produces no answer set as expected. Using these results, an expert system will be able to conclude that Acetaminophen can be recommended but Aspirin cannot be recommended.

```
% File 'usecase-data'

% getting symptoms from the ontology
#spatom{&dlC["file:FILE-LOC",a,b,c,d,"Condition"](X)} -> symptom(X).

% drugs we are to reason about
#spatom{drug("<DRUGS#Aspirin>")}.
#spatom{drug("<DRUGS#Acetaminophen>")}.

% fluents
fluent(f1). fluent(f2). fluent(f3). fluent(f4). fluent(f5).
fluent(f6). fluent(f7).

#spatom{equals(f1,condition,"<DRUGS#Pain>")}.
#spatom{equals(f2,condition,"<DRUGS#Gastritis>")}.
#spatom{equals(f3,condition,"<DRUGS#GastroIntestinalBleeding>")}.
#spatom{equals(f4,sideEffect,"<DRUGS#Aspirin>")}.
#spatom{equals(f5,sideEffect,"<DRUGS#Acetaminophen>")}.
#spatom{equals(f6,administered,"<DRUGS#Aspirin>")}.
#spatom{equals(f7,administered,"<DRUGS#Acetaminophen>")}.

% events/actions
event(e1). event(e2). event(e3). event(e4).

#spatom{equals(e1,dummyEvent,"<DRUGS#Gastritis>")}.
#spatom{equals(e2,dummyEvent,"<DRUGS#GastroIntestinalBleeding>")}.
#spatom{equals(e3,admin,"<DRUGS#Aspirin>")}.
#spatom{equals(e4,admin,"<DRUGS#Acetaminophen>")}.

% domain of time
time(0). time(1). time(2).

% domain variable declarations
#domain drug(D). #domain symptom(C).
#domain symptom(C1). #domain fluent(F).
#domain fluent(F1). #domain fluent(F2).
#domain event(E).
#domain time(T). #domain time(T1).
#domain time(T2). #domain time(T3).

% facts about may_treat and contraindication from ontologies
#spatom{&dlR["file:FILE-LOC",a,b,c,d,"ci_with"](D,C)} ->
contraindication(D,C).
#spatom{&dlR["file:FILE-LOC",a,b,c,d,"may_treat"](D,C)} ->
may_treat(D,C).
```

Figure 11.2: F2LP encoding of the facts

```
%File: 'usecase-ec'
% use case 1: which drug can be administered given the
% patient condition

equals(E,admin,D) & equals(F,administered,D) -> initiates(E,F,T).

% positive effect of drug
equals(F,condition,C) & equals(E,admin,D) & holdsAt(F,T) &
may_treat(D,C) & -ab(D,T) -> terminates(E,F,T).

% contraindications for drug => abnormality
equals(F,condition,C) & holdsAt(F,T) & contraindication(D,C) ->
ab(D,T).

% if abnormal, then sideEffect
equals(E,admin,D) & equals(F,sideEffect,D) & ab(D,T) ->
initiates(E,F,T).

% Gastritis "normally" causes gastrointestinalBleeding
#spatom{equals(E,dummyEvent,"<DRUGS#Gastritis>")} &
#spatom{equals(F,condition,"<DRUGS#GastroIntestinalBleeding>")} &
-#spatom{ab1(condition,"<DRUGS#Gastritis>",T)}
-> initiates(E,F,T).

% Indirect effect is a sideEffect if contraindications
% surface after the drug is administered
equals(F,condition,C) & equals(F1,administered,D) &
equals(F2,sideEffect,D) & initiates(E,F,T) &
holdsAt(F1,T) & contraindication(D,C) -> initiates(E,F2,T).

% DummyEvent is triggered everytime a condition holds
equals(F,condition,C) & equals(E,dummyEvent,C) &
holdsAt(F,T) -> happens(E,T).

% initial state
#spatom{equals(F,condition,"<DRUGS#Gastritis>")} ->
holdsAt(F,0).
#spatom{equals(F,condition,"<DRUGS#GastroIntestinalBleeding>")}
-> -holdsAt(F,0).
#spatom{equals(F,condition,"<DRUGS#Pain>")} ->
holdsAt(F,0).
equals(F,sideEffect,D) -> -holdsAt(F,0).
equals(F,administered,D) ->  -holdsAt(F,0).

-releasedAt(F,T).
```

Figure 11.3: F2LP encoding of the event calculus description

Now, consider the F2LP encoding of the TAL description for the same example shown in Figure 11.4.[8] In the encoding, for simplicity, we introduce the abnormality predicate instead of the abnormality fluent. Given the initial state in which only *Condition*(*Pain*) and *Condition*(*Gastritis*) hold, if we want to check if Aspirin can be administered in the interval $[0, 2]$, then we add the following to the theory in Figure 11.4.

```
% initial state
#spatom{equals(F,condition,"<DRUGS#Gastritis>")} -> holds(0,F,true).
#spatom{equals(F,condition,"<DRUGS#GastroIntestinalBleeding>")}
-> holds(0,F,false).
#spatom{equals(F,condition,"<DRUGS#Pain>")} -> holds(0,F,true).
equals(F,sideEffect,D) -> holds(0,F,false).



% query
% can aspirin be given to the patient?
#spatom{equals(E,admin,"<DRUGS#Aspirin>")} -> occurs(0,2,E).
-(#spatom{equals(F,condition,"<DRUGS#Pain>")} &
  #spatom{equals(F1,sideEffect,"<DRUGS#Aspirin>")} &
  holds(2,F,false) & -?[T]: (T >=0 & T <=2 & holds(T,F1,true))).
```

The first formula represents that Aspirin is administered in the interval $[0, 2]$, and the second formula is the negation of the query we want to check. The query here is to check if administering Aspirin treats the patient such that no side-effect is caused in the interval $[0, 2]$. We can run the resulting theory using `f2lpdlv.py` as follows.

```
>>> f2lpdlv.run('f2lp usecase-data usecase-tal',
                                  'dlvhex --filter=holds')
```

[8] We disregard fluents `f6` and `f7` for the TAL description.

```
%File: 'usecase-tal'

bool(true). bool(false).
#domain bool(Bo).
#domain bool(Bo1).

% all the fluents are persistent
equals(F,sideEffect,D) & #spatom{#succ(T,T1)} & -occlude(T1,F) ->
![Bo]:(holds(T1,F,Bo) <-> holds(T,F,Bo)).

equals(F,condition,C) & #spatom{#succ(T,T1)} & -occlude(T1,F) ->
![Bo]:(holds(T1,F,Bo) <-> holds(T,F,Bo)).

% Unique Value Axioms
holds(T,F,Bo) & holds(T,F,Bo1) & Bo != Bo1 -> false.
-?[Bo]:holds(T,F,Bo) -> false.

% holds is non-intensional
holds(T,F,Bo) | -holds(T,F,Bo).

% positive effect of drug
equals(E,admin,D) & occurs(T1,T2,E) & equals(F,condition,C) ->
((holds(T1,F,true) & may_treat(D,C) &
 -?[F1,C1,T]:(T >= T1 & T <= T2 & holds(T,F1,true) &
              equals(F1,condition,C1) & contraindication(D,C1)))
  -> ![T]:(T > T1 & T <= T2 -> occlude(T,F) & holds(T,F,false))).

% negative effect of drug (sideEffect)
equals(E,admin,D) & occurs(T1,T2,E) & equals(F,condition,C) &
equals(F1,sideEffect,D)  ->
((T3 >= T1 & T3 <= T2 & holds(T3,F,true) & contraindication(D,C))
  -> ![T]:(T >= T3 & T <= T2 -> occlude(T,F1) & holds(T,F1,true))).

% Gastritis "normally" causes gastrointestinalBleeding
% using dependency constraints for this instead of dummy events
#spatom{equals(F,condition,"<DRUGS#Gastritis>")} &
#spatom{equals(F1,condition,"<DRUGS#GastroIntestinalBleeding>")} &
holds(T,F,true) & -#spatom{ab1(condition,"<DRUGS#Gastritis>",T)} &
#spatom{#succ(T,T1)} ->
occlude(T1,F1) & holds(T1,F1,true).
```

Figure 11.4: F2LP encoding of the TAL description

Following is one of the answer sets produced (the existence of an answer set indicates that the query is not entailed):

```
{ holds(0,f5,false), holds(0,f4,false), holds(0,f3,false),
  holds(0,f2,true), holds(0,f1,true), holds(1,f5,false),
  holds(1,f4,true), holds(1,f3,true), holds(1,f2,true),
  holds(1,f1,true), holds(2,f5,false), holds(2,f4,true),
  holds(2,f3,true), holds(2,f2,true), holds(2,f1,true) }
```

This output indicates that a side-effect is caused at timepoint 1 and that administering Aspirin in the interval $[0, 2]$ does not relieve pain. Now, consider the initial state in which only *Condition*$($*Pain*$)$ holds. Further, assume that *Condition*$($*Gastritis*$)$ holds at timepoint 2. Running the same query as before produces no answer set, which indicates that the query is entailed. This in turn indicates that Aspirin can be administered in the interval $[0, 2]$. This is because the contraindication to Aspirin, which is Gastroinstestinal Bleeding, is caused outside the interval in which Aspirin is administered.

As before, running similar queries by replacing Aspirin with Acetaminophen produces no answer sets. From these results, an expert system will be able to conclude that Acetaminophen can be recommended. It will also be able to conclude that Aspirin can be recommended, unless Gastrointestinal Bleeding may be caused in the interval in which it is to be administered.

11.5 Related Work

There has been considerable work done on integrating action formalisms and description logics (see, for example, (Baader, Lutz, Milicic, Sattler, & Wolter, 2005; Giacomo, Lenzerini, Poggi, & Rosati, 2006; Liu, Lutz, Milicic, & Wolter, 2006; Drescher & Thielscher, 2007; Gu & Soutchanski, 2007)). The primary difference between these approaches and our approach is that while these approaches focus on DL-based reasoning about actions, our approach focuses on integrating existing

circumscriptive action theories with non-temporal DL knowledge bases. The goal of the DL-based approaches for reasoning about actions is to provide a decidable yet expressive logical framework for reasoning about actions. On the other hand, our goal is to provide a framework for ASP-based reasoning for circumscriptive action theories which is capable of using ontologies as knowledge bases.

11.6 Proofs

Proof of Theorem 37

Follows from the straightforward application of Theorems 21, 22, and 12 (splitting theorem). □

Proof of Theorem 38

Follows from the straightforward application of Theorems 23, 24, and 12 (splitting theorem). □

Proof of Theorem 39

Follows from the straightforward application of Theorems 28, 29, and 12 (splitting theorem). □

Chapter 12

CONCLUSION

Developing formalisms that offer a suitable tradeoff between expressivity and computational efficiency is one of the fundamental goals of KR&R. Different existing formalisms have different limitations either with respect to expressivity or with respect to computational efficiency. Circumscription and DLs, which belong to the classical logic tradition, have certain limitations that are well-addressed by ASP, which belongs to the logic programming tradition. The vice-versa also holds. So, in this dissertation, we relate/integrate the formalisms using the framework of the first-order stable model semantics to (partially) overcome some of their limitations.

By relating circumscription to ASP, we are able to use answer set solvers for computing the event calculus, the situation calculus, and TAL. As shown earlier, this has several advantages with respect to both expressive and efficient reasoning. By integrating DLs and ASP, we are able to perform nonmonotonic reasoning using DL knowledge bases, thus providing a framework for integrating rules and ontologies for the semantic web. Further, since we use the same framework for relating circumscription to ASP, and for integrating DLs and ASP, we are able to reformulate the problem of integrating action theories and ontologies as a problem of integrating rules and ontologies, thus enabling us to use the computational tools of the latter for the former. We showed an application of this in the area of medical expert systems.

This dissertation also highlights the usefulness of the first-order stable model semantics as a mathematical tool for relating/integrating different knowledge representation formalisms, and for extending some well-known properties of traditional ASP to more general programs. Since the first-order stable

model semantics extends both first-order logic and ASP, it provides an ideal framework for relating/integrating formalisms belonging to the different traditions of classical logic and logic programming. Since some useful constructs in ASP, such as “choice” and the “count” aggregate expression, can be viewed as abbreviations for certain first-order formulas, the safety and splitting properties can be extended to programs with these constructs by extending them to the first-order stable model semantics.

This dissertation contributes to the following areas of KR&R:

- **Reasoning about Actions**: We present effective, ASP based computational approaches for the event calculus, the situation calculus, and TAL.
- **Theory of Stable Models and Answer Set Programming:** We present some interesting properties of the first-order stable model semantics and relate the stable model semantics to circumscription and circumscriptive action theories. We also extend the traditional safety and splitting properties to a more general and useful class of programs.
- **Integrating Different Knowledge Representation Formalisms**: We present simple and effective approaches to integrate DLs and ASP, and circumscriptive action theories and ontologies.

BIBLIOGRAPHY

Agarwal, S., Agrawal, R., Deshpande, P., Gupta, A., Naughton, J. F., Ramakrishnan, R., & Sarawagi, S. (1996). On the computation of multidimensional aggregates. In *VLDB*, pp. 506–521.

Akman, V., Erdoğan, S., Lee, J., Lifschitz, V., & Turner, H. (2004). Representing the Zoo World and the Traffic World in the language of the Causal Calculator. *Artificial Intelligence*, *153(1–2)*, 105–140.

Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., & Watson, V. (1976). System r: Relational approach to database management. *ACM Trans. Database Syst.*, *1*(2), 97–137.

Baader, F., Lutz, C., Milicic, M., Sattler, U., & Wolter, F. (2005). Integrating description logics and action formalisms: First results. In *AAAI*, pp. 572–577.

Baral, C., & Gelfond, M. (1997). Reasoning about effects of concurrent actions. *Journal of Logic Programming*, *31*.

Bartholomew, M., & Lee, J. (2010). A decidable class of groundable formulas in the general theory of stable models. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 477–485.

Baselice, S., Bonatti, P. A., & Gelfond, M. (2005). Towards an integration of answer set and constraint solving. In *In Proc. of ICLP 05*, pp. 52–66.

Belleghem, K. V., Denecker, M., & Schreye, D. D. (1997). On the relation between situation calculus and event calculus. *J. Log. Program.*, *31*(1-3), 3–37.

Borzenko, O., Xu, W., Obsniuk, M., Chopra, A., Jasiobedzki, P., Jenkin, M. R. M., & Lespérance, Y. (2006). Lights and camera: Intelligently controlled multi-channel pose estimation system. In *ICVS*, p. 42.

Bria, A., Faber, W., & Leone, N. (2008). Normal form nested programs. In *JELIA*, pp. 76–88.

Brooks, D. R., Erdem, E., Erdoğan, S. T., Minett, J. W., & Ringe, D. (2007). Inferring phylogenetic trees using answer set programming. *Journal of Automated Reasoning*, *39*, 471–511.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., & Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artif. Intell.*, *114*(1-2), 3–55.

Cabalar, P. (2009). Existential quantifiers in the rule body. http://www.dc.fi.udc.es/ cabalar/eqrb.pdf.

Cabalar, P., Pearce, D., & Valverde, A. (2005). Reducing propositional theories in equilibrium logic to logic programs. In *Proceedings of Portuguese Conference on Artificial Intelligence (EPIA)*, pp. 4–17.

Cabalar, P., Pearce, D., & Valverde, A. (2009). A revised concept of safety for general answer set programs. In Erdem, E., Lin, F., & Schaub, T. (Eds.), *LPNMR*, Vol. 5753 of *Lecture Notes in Computer Science*, pp. 58–70. Springer.

Calimeri, F., Cozza, S., Ianni, G., & Leone, N. (2008). Computable functions in ASP: theory and implementation. In *Proceedings of International Conference on Logic Programming (ICLP)*, pp. 407–424.

Classen, A., Heymans, P., & Schobbens, P.-Y. (2008). What's in a feature: A requirements engineering perspective. In *FASE*, pp. 16–30.

Cushing, W., Kambhampati, S., Mausam, & Weld, D. S. (2007). When is temporal planning really temporal?. In *IJCAI*, pp. 1852–1859.

de Bruijn, J., Eiter, T., Polleres, A., & Tompits, H. (2007a). Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 304–309.

de Bruijn, J., Pearce, D., Polleres, A., & Valverde, A. (2007b). Quantified equilibrium logic and hybrid rules. In *RR*, pp. 58–72.

de Bruijn, J., Pearce, D., Polleres, A., & Valverde, A. (2010). A semantical framework for hybrid knowledge bases. *Knowl. Inf. Syst.*, *25*(1), 81–104.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann Publishers.

Denecker, M., Pelov, N., & Bruynooghe, M. (2001). Ultimate well-founded and stable semantics for logic programs with aggregates. In *ICLP*, pp. 212–226.

Doherty, P. (1996). Pmon$^+$: A fluent logic for action and change, formal specification, version 1.0[1]. Tech. rep., Department of Computer and Information Science, Linköping University.

Doherty, P., Gustafsson, J., Karlsson, L., & Kvarnström, J. (1998). TAL: Temporal action logics language specification and tutorial[2]. *Linköping Electronic Articles in Computer and Information Science ISSN 1401-9841*, *3*(015).

[1] http://www.ida.liu.se/publications/techrep/96/tr96.html

[2] http://www.ep.liu.se/ea/cis/1998/015/

Doherty, P., & Kvarnström, J. (2008). Temporal action logics. In van Harmelen, F., Lifschitz, V., & Porter, B. (Eds.), *Handbook of Knowledge Representation*, pp. 709–757. Elsevier Science.

Doğandağ, S., Ferraris, P., & Lifschitz, V. (2004). Almost definite causal theories.. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 74–86.

Drescher, C., & Thielscher, M. (2007). Integrating action calculi and description logics. In *KI*, pp. 68–83.

Edelkamp, S., & Hoffmann, J. (2004). PDDL2.2: The language for the Classical Part of the 4th International Planning Competition. Tech. rep. 195, Albert-Ludwigs-Universität, Institut für Informatik.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008). Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, *172*(12-13), 1495–1539.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005). A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *IJCAI*, pp. 90–96.

Eiter, T., & Wang, K. (2008). Semantic forgetting in answer set programming. *Artif. Intell.*, *172*(14), 1644–1672.

Erdoğan, S., & Lifschitz, V. (2004). Definitions in answer set programming. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 114–126.

Faber, W., Leone, N., & Pfeifer, G. (2004). Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*.

Feier, C., & Heymans, S. (2009). Hybrid reasoning with forest logic programs. In *ESWC*, pp. 338–352.

Ferraris, P. (2005). Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 119–131.

Ferraris, P., Lee, J., & Lifschitz, V. (2006). A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, *47*, 79–101.

Ferraris, P., Lee, J., & Lifschitz, V. (2007). A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 372–379.

Ferraris, P., Lee, J., & Lifschitz, V. (2011). Stable models and circumscription. *Artificial Intelligence*, *175*, 236–263.

Ferraris, P., Lee, J., Lifschitz, V., & Palla, R. (2009). Symmetric splitting in the general theory of stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 797–803.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3–4), 189–208.

Fink, M., & Pearce, D. (2010). A logical semantics for description logic programs. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, pp. 156–168.

Fornara, N., & Colombetti, M. (2008). Specifying and enforcing norms in artificial institutions. In *DALT*, pp. 1–17.

Fox, M., & Long, D. (2003). Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, *20*, 61–124.

Gebser, M., Ostrowski, M., & Schaub, T. (2009). Constraint answer set solving. In Hill, P. M., & Warren, D. S. (Eds.), *ICLP*, Vol. 5649 of *Lecture Notes in Computer Science*, pp. 235–249. Springer.

Gebser, M., Schaub, T., & Thiele, S. (2007). Gringo : A new grounder for answer set programming. In Baral, C., Brewka, G., & Schlipf, J. S. (Eds.), *LPNMR*, Vol. 4483 of *Lecture Notes in Computer Science*, pp. 266–271. Springer.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R., & Bowen, K. (Eds.), *Proceedings of International Logic Programming Conference and Symposium*, pp. 1070–1080. MIT Press.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, *9*, 365–385.

Gelfond, M., & Lifschitz, V. (1998). Action languages[3]. *Electronic Transactions on Artificial Intelligence*, *3*, 195–210.

Gerevini, A., & Long, D. (2005). Plan constraints and preferences in pddl3 - the language of the fifth international planning competition. Tech. rep..

Giacomo, G. D., Lenzerini, M., Poggi, A., & Rosati, R. (2006). On the update of description logic ontologies at the instance level. In *AAAI*.

Giacomo, G. D., Lespérance, Y., & Levesque, H. J. (2000). Congolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, *121*(1-2), 109–169.

[3] http://www.ep.liu.se/ea/cis/1998/016/

Giacomo, G. D., & Levesque, H. (1999). An incremental interpreter for high-level programs with sensing. In *In Logical Foundations for Cognitive Agents, Contributions in Honor of Ray Reiter*, pp. 86–102.

Giacomo, G. D., Levesque, H. J., & Sardiña, S. (2001). Incremental execution of guarded theories. *ACM Trans. Comput. Log.*, *2*(4), 495–525.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, *153(1–2)*, 49–104.

Giunchiglia, E., & Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 623–630. AAAI Press.

Gu, Y., & Soutchanski, M. (2007). Decidable reasoning in a modified situation calculus. In *IJCAI*, pp. 1891–1897.

Gustafsson, J., & Doherty, P. (1996). Embracing occlusion in specifying the indirect effects of actions. In *KR*, pp. 87–98.

Hanks, S., & McDermott, D. (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence*, *33*(3), 379–412.

Heymans, S., de Bruijn, J., Predoiu, L., Feier, C., & Nieuwenborgh, D. V. (2008). Guarded hybrid knowledge bases. *TPLP*, *8*(3), 411–429.

Heymans, S., Nieuwenborgh, D. V., & Vermeir, D. (2005). Guarded open answer set programming. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 92–104.

Janhunen, T., & Oikarinen, E. (2004). Capturing parallel circumscription with disjunctive logic programs. In *Proc. of 9th European Conference in Logics in Artificial Intelligence (JELIA-04)*, pp. 134–146.

Janhunen, T., Oikarinen, E., Tompits, H., & Woltran, S. (2007). Modularity aspects of disjunctive stable models. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 175–187.

Karlsson, L., & Gustafsson, J. (1999). Reasoning about concurrent interaction. *J. Log. Comput.*, *9*(5), 623–650.

Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pp. 359–363.

Kim, T.-W., Lee, J., & Palla, R. (2009). Circumscriptive event calculus as answer set programming. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 823–829.

Kowalski, R., & Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, *4*, 67–95.

Krishnamurthy, R., Ramakrishnan, R., & Shmueli, O. (1996). A framework for testing safety and effective computability. *J. Comput. Syst. Sci.*, *52*(1), 100–124.

Kvarnström, J. (2005). Talplanner and other extensions to temporal action logic. *PhD thesis, Linköping University, Linköping Studies in Science and Technology, Dissertation No. 937*.

Kvarnström, J., & Doherty, P. (2000). Tackling the qualification problem using fluent dependency constraints. *Computational Intelligence*, *16*(2), 169–209.

Lee, J., & Lifschitz, V. (2001). Additive fluents[4]. In *Working Notes of the AAAI Spring Symposium on Answer Set Programming*.

Lee, J., & Lifschitz, V. (2003). Loop formulas for disjunctive logic programs. In *Proceedings of International Conference on Logic Programming (ICLP)*, pp. 451–465.

Lee, J., Lifschitz, V., & Palla, R. (2008a). A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 472–479.

Lee, J., Lifschitz, V., & Palla, R. (2008b). Safe formulas in the general theory of stable models (preliminary report). In *Proceedings of International Conference on Logic Programming (ICLP)*, pp. 672–676.

Lee, J., Lifschitz, V., & Palla, R. (2009). Safe formulas in the general theory of stable models[5]. Unpublished Draft.

Lee, J., & Lin, F. (2006). Loop formulas for circumscription. *Artificial Intelligence*, *170*(2), 160–185.

Lee, J., & Meng, Y. (2009). On reductive semantics of aggregates in answer set programming. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 182–195.

Lee, J., & Meng, Y. (2011). First-order stable model semantics and first-order loop formulas. *Journal of Artificial Inteligence Research (JAIR)*, *To appear*.

Lee, J., & Palla, R. (2007). Yet another proof of the strong equivalence between propositional theories and logic programs. In *Working Notes of the Workshop on Correspondence and Equivalence for Nonmonotonic Theories*.

Lee, J., & Palla, R. (2010). Situation calculus as answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 309–314.

[4] http://www.cs.utexas.edu/users/vl/papers/additive.ps

[5] http://peace.eas.asu.edu/joolee/papers/safety.pdf

Lee, J., & Palla, R. (2011a). Integrating rules and ontologies in the first-order stable model semantics (preliminary report). In *LPNMR*, pp. 248–253.

Lee, J., & Palla, R. (2011b). Integrating rules and ontologies in the first-order stable model semantics (preliminary report). In *Working Papers of the Tenth International Symposium on Logical Formalizations of Commonsense Reasoning*.

Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., & Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *J. Log. Program.*, *31*(1-3), 59–83.

Lifschitz, V. (1991). Nonmonotonic databases and epistemic queries. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 381–386.

Lifschitz, V. (1994). Circumscription. In Gabbay, D., Hogger, C., & Robinson, J. (Eds.), *Handbook of Logic in AI and Logic Programming*, Vol. 3, pp. 298–352. Oxford University Press.

Lifschitz, V., Pearce, D., & Valverde, A. (2007). A characterization of strong equivalence for logic programs with variables. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*.

Lifschitz, V., Tang, L. R., & Turner, H. (1999). Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, *25*, 369–389.

Lifschitz, V., & Turner, H. (1994). Splitting a logic program. In Van Hentenryck, P. (Ed.), *Proceedings of International Conference on Logic Programming (ICLP)*, pp. 23–37.

Lifschitz, V., & Turner, H. (1999). Representing transition systems by logic programs. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 92–106.

Lin, F. (1995). Embracing causality in specifying the indirect effects of actions. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1985–1991.

Lin, F. (2003). Compiling causal theories to successor state axioms and STRIPS-like systems. *Journal of Artificial Intelligence Research*, *19*, 279–314.

Lin, F., & Shoham, Y. (1992). A logic of knowledge and justified assumptions. *Artificial Intelligence*, *57*, 271–289.

Lin, F., & Wang, K. (1999). From causal theories to logic programs (sometimes). In *LPNMR*, pp. 117–131.

Lin, F., & Zhou, Y. (2011). From answer set logic programming to circumscription via logic of gk. *Artificial Intelligence*, *175*, 264–277.

Liu, H., Lutz, C., Milicic, M., & Wolter, F. (2006). Reasoning about actions using description logics with general tboxes. In *JELIA*, pp. 266–279.

Marek, V. W., & Truszczynski, M. (2004). Logic programs with abstract constraint atoms. In *AAAI*, pp. 86–91.

McCain, N., & Turner, H. (1994). Language independence and language tolerance in logic programs. In Van Hentenryck, P. (Ed.), *Proceedings Eleventh Int'l Conf. on Logic Programming*, pp. 38–57.

McCain, N., & Turner, H. (1997). Causal theories of action and change. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 460–465.

McCarthy, J. (1963). Situations, actions, and causal laws. Tech. rep., Stanford University.

McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, *13*, 27–39,171–172.

McCarthy, J. (1986). Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, *26*(3), 89–116.

McCarthy, J., & Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., & Michie, D. (Eds.), *Machine Intelligence*, Vol. 4, pp. 463–502. Edinburgh University Press, Edinburgh.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

McIlraith, S. A., & Son, T. C. (2002). Adapting golog for composition of semantic web services. In *KR*, pp. 482–496.

Mellarkod, V. S., Gelfond, M., & Zhang, Y. (2008). Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, *53*, 251–287.

Miller, R., & Shanahan, M. (1999). The event calculus in classical logic - alternative axiomatisations. *Electron. Trans. Artif. Intell.*, *3*(A), 77–105.

Minsky, M. (1981). A framework for representing knowledge..

Motik, B., & Rosati, R. (2010). Reconciling description logics and rules. *J. ACM*, *57*(5).

Motik, B., Sattler, U., & Studer, R. (2005). Query answering for owl-dl with rules. *J. Web Sem.*, *3*(1), 41–60.

Mueller, E. (2006). *Commonsense reasoning*. Elsevier.

Mueller, E. T. (2004). Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, *14*(5), 703–730.

Mumick, I. S., Pirahesh, H., & Ramakrishnan, R. (1990). The magic of duplicates and aggregates. In *VLDB*, pp. 264–277.

Nazarenko, A., Polo, L., Eiter, T., de Bruijn, J., Schwichtenberg, A., & Heymans, S. (2010). How to integrate ontologies and rules?.. Tutorial, 24th Conference on Artificial Intelligence (AAAI 2010).

Niemelä, I., Simons, P., & Soininen, T. (1999). Stable model semantics for weight constraint rules. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 317–331.

Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., & Barry, M. (2001). An A-Prolog decision support system for the Space Shuttle. In *Proceedings of International Symposium on Practical Aspects of Declarative Languages (PADL)*, pp. 169–183.

Oikarinen, E., & Janhunen, T. (2008). Achieving compositionality of the stable model semantics for Smodels programs. *Theory and Practice of Logic Programming*, *5–6*, 717–761.

Patkos, T., & Plexousakis, D. (2009). Reasoning with knowledge, action and time in dynamic and uncertain domains. In *IJCAI*, pp. 885–890.

Pearce, D., & Valverde, A. (2005). A first order nonmonotonic extension of constructive logic. *Studia Logica*, *80*, 323–348.

Pednault, E. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R., Levesque, H., & Reiter, R. (Eds.), *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 324–332.

Pelov, N., Denecker, M., & Bruynooghe, M. (2004). Partial stable models for logic programs with aggregates. In *LPNMR*, pp. 207–219.

Przymusinski, T. (1988). On the declarative semantics of deductive databases and logic programs. In Minker, J. (Ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 193–216. Morgan Kaufmann, San Mateo, CA.

Ramakrishnan, R., Bancilhon, F., & Silberschatz, A. (1987). Safety of recursive horn clauses with infinite relations. In *PODS*, pp. 328–339.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, *13*, 81–132.

Reiter, R. (1991). The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V. (Ed.), *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pp. 359–380. Academic Press.

Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Rosati, R. (2005). On the decidability and complexity of integrating ontologies and rules. *J. Web Sem.*, *3*(1), 61–73.

Rosati, R. (2006). DL+log: Tight integration of description logics and disjunctive datalog. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 68–78.

Rossi, F., Beek, P. v., & Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA.

Rouached, M., Perrin, O., & Godart, C. (2006). Towards formal verification of web service composition. In *Business Process Management*, pp. 257–273.

Sandewall, E. (1994). *Features and Fluents. A Systematic Approach to the Representation of Knowledge about Dynamical Systems*, Vol. 1. Oxford University Press.

Sardiña, S., Giacomo, G. D., Lespérance, Y., & Levesque, H. J. (2004). On the semantics of deliberation in indigolog - from theory to implementation. *Ann. Math. Artif. Intell.*, *41*(2-4), 259–299.

Schubert, L. (1990). Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In Kyburg, H., Loui, R., & Carlson, G. (Eds.), *Knowledge Representation and Defeasible Reasoning*, pp. 23–67. Kluwer.

Shanahan, M. (1995). A circumscriptive calculus of events. *Artif. Intell.*, *77*(2), 249–284.

Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press.

Shanahan, M. (1999). The event calculus explained. In *Artificial Intelligence Today*, LNCS 1600, pp. 409–430. Springer.

Shanahan, M. (2000). An abductive event calculus planner. *J. Log. Program.*, *44*(1-3), 207–240.

Shanahan, M., & Witkowski, M. (2004). Event calculus planning through satisfiability. *J. Log. Comput.*, *14*(5), 731–745.

Simons, P. (1999). Extending the stable model semantics with more expressive rules. In *Logic Programming and Non-monotonic Reasoning: Proceedings Fifth Int'l Conf. (Lecture Notes in Artificial Intelligence 1730)*, pp. 305–316.

Son, T. C., & Pontelli, E. (2007). A constructive semantic characterization of aggregates in answer set programming. *TPLP*, *7*(3), 355–375.

Syrjänen, T. (2004). Cardinality constraint programs.. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, pp. 187–199.

Tiihonen, J., Soininen, T., Niemelä, I., & Sulonen, R. (2003). A practical tool for mass-customising configurable products. In *Proceedings of the 14th International Conference on Engineering Design*, pp. 1290–1299.

Topor, R. W., & Sonenberg, E. A. (1988). On domain independent databases. In Minker, J. (Ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 217–240. Morgan Kaufmann, San Mateo, CA.

Ullman, J. D. (1985). Implementation of logical query languages for databases (abstract). In *SIGMOD Conference*, p. 444.

van Dalen, D. (1983). *Logic and Structure*.

Wang, K., Billington, D., Blee, J., & Antoniou, G. (2004). Combining description logic and defeasible logic for the semantic web. In *RuleML*, pp. 170–181.

Zaniolo, C. (1986). Safety and compilation of non-recursive horn clauses. In *Expert Database Conf.*, pp. 237–252.

Zaniolo, C., Arni, N., & Ong, K. (1993). Negation and aggregates in recursive rules: the ldl++ approach. In *DOOD*, pp. 204–221.

Zhang, H., Zhang, Y., Ying, M., & Zhou, Y. (2011). Translating first-order theories into logic programs. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1126–1131.