Answer Set Programming and Other Computing Paradigms

by

Yunsong Meng

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

Approved April 2013 by the
Graduate Supervisory Committee:

Joohyung Lee, Chair
Gail-Joon Ahn
Chitta Baral
Georgios Fainekos
Vladimir Lifschitz

ARIZONA STATE UNIVERSITY

May 2013

UMI Number: 3559632

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI 3559632

Published by ProQuest LLC (2013). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

ABSTRACT

Answer Set Programming (ASP) is one of the most prominent and successful knowledge representation paradigms. The success of ASP is due to its expressive non-monotonic modeling language and its efficient computational methods originating from building propositional satisfiability solvers. The wide adoption of ASP has motivated several extensions to its modeling language in order to enhance expressivity, such as incorporating aggregates and interfaces with ontologies. Also, in order to overcome the grounding bottleneck of computation in ASP, there are increasing interests in integrating ASP with other computing paradigms, such as Constraint Programming (CP) and Satisfiability Modulo Theories (SMT).

Due to the non-monotonic nature of the ASP semantics, such enhancements turned out to be non-trivial and the existing extensions are not fully satisfactory. We observe that one main reason for the difficulties is rooted in the propositional semantics of ASP, which is limited in handling complex constructs (such as aggregates and ontologies) and functions (such as constraint variables in CP and SMT) in natural ways.

This dissertation presents a unifying view on these extensions by viewing them as instances of formulas with generalized quantifiers and intensional functions. We extend the first-order stable model semantics by Ferraris, Lee, and Lifschitz to allow generalized quantifiers, which cover aggregates, DL-atoms, constraints and SMT theory atoms as special cases. Using this unifying framework, we study and relate different extensions of ASP. We also present a tight integration of ASP with SMT, based on which we enhance action language $\mathcal{C}+$ to handle reasoning about continuous changes. Our framework yields a systematic approach to study and extend non-monotonic languages.

Dedicated to my family

ACKNOWLEDGEMENTS

There are many people who influenced me and assisted along my long journey to the completion of the Ph.D program. Among them, Joohyung Lee is the people who I would like to thank most, from the deep in my heart. He provided me the great opportunity to be working with him for many years and put in tremendous efforts and patience in mentoring me and guiding me to the right direction. Under his supervision, I have improved a lot both as a researcher and as a professional.

I am also thankful to the other committee members: Gail-Joon Ahn, Chitta Baral, Georgios Fainekos and Vladimir Lifschitz. I have greatly benefited from their invaluable insights and suggestions.

I also learned a lot from the numerous discussions with my my current and former teammates: Ravi, Mike, Joe, Greg, Yu, Michael, Tae-Won, Sunjin and Joeng-Jin. Many thanks to all of them. I am grateful to Dan from Siemens and my colleagues from Samsung, Alan, Doreen, Jie, Yongmei, Justin and Priyang for their help and support. I am also grateful to my friends Jicheng, Yang, Hongxin, Yin, Liang, Zhibin, Jun, Wenjun and many others whose names I can not enumerate.

I reserve my deepest love for my wife Hairong. It is only love that can give a girl the courage and determination to fly to the other side of the earth for me and to build a family with me in a country with different culture. I cannot complete the work without her supports. My deepest love also goes to my little boy Derek, who can make me happy for a whole day with a single kiss. I also thank my parents Liping Meng and Yonghong Peng, and my parents-in-law Kerang Zhong and Shunsong Xie for their endless support and love.

My dissertation work was partially supported by National Science Foundation under Grant IIS-0916116 and IIS-0839821, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through the US Army Research Lab and by the South Korea IT R&D program MKE/KIAT 2010-TD-300404-001.

TABLE OF CONTENTS

Page

LIST OF TABLES x

LIST OF FIGURES xi

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND 6

2.1 Answer Set Programming 6

2.2 Theorem on Loop Formulas 7

2.3 Extensions of ASP 8

Aggregates 8

Description Logics 10

Constraints & SMT 11

2.4 HEX Programs & FLP Semantics 11

2.5 First-Order Stable Model Semantics 13

2.6 Functional Stable Model Semantics 15

2.7 Action Language $\mathcal{C}+$ 16

3 FIRST-ORDER STABLE MODEL SEMANTICS AND FIRST-ORDER LOOP FORMULAS 17

3.1 First-Order Loop Formulas and Herbrand Models 19

Loop Formula for Non-Disjunctive Programs 19

Loop Formula for Disjunctive Programs 22

Extension to Arbitrary Sentences 24

3.2 Comparing First-Order Stable Model Semantics and First-Order Loop Formulas 28

Loop Formulas Relative to an Interpretation 28

A Reformulation of SM 33

3.3 Representing First-Order Stable Model Semantics by First-Order Loop Formulas 38

Bounded Formulas 39

CHAPTER Page

Bounded Formulas and Clark's Equational Theory 39

Bounded Formulas and Normal Form 42

Decidability of Boundedness and Finite Complete Set of Loops . . . 44

Semi-Safe Formulas 46

3.4 Programs with Explicit Quantifiers 48

3.5 Extension to Allow Extensional Predicates 51

3.6 Related Work 56

3.7 Conclusion 58

3.8 Proofs 59

Proof of Theorem 3 59

Proof of Equivalence between (a) and (b) of Theorem 3 60

Proof of Proposition 3 61

Proof of Proposition 4 63

Proof of Proposition 5 64

Proof of Equivalence between (b) and (c) of Theorem 3 65

Proof of Theorem 2 69

Proof of Proposition 6 72

Proof of Proposition 7 74

Proof of Proposition 8 75

Proof of Proposition 9 76

Proof of Proposition 11 77

Proof of Proposition 10 79

Proof of Proposition 13 81

Proof of Proposition 16 83

4 ON SEMANTICS OF AGGREGATES 85

4.1 Syntax and Existing Semantics of Programs with Aggregates 86

Syntax of Programs with Aggregates 86

FLP Semantics 87

Ferraris Semantics 88

PDB-SPT Semantics 89

CHAPTER Page

4.2 Reformulation and Comparison of the Semantics of Aggregates 91
A Reformulation of Ferraris Semantics 91
A Reformulation of FLP Semantics 91
A Reformulation of PDB-SPT Semantics 93
Relationship among the Semantics 94
4.3 Loop Formulas for Programs with Aggregates 96
Loop Formulas for Ferraris Semantics 97
Loop Formulas for FLP semantics 98
4.4 Syntax and Semantics of Aggregate Formulas 100
4.5 Stable Model Semantics of First-Order Aggregate Formulas 101
Programs with Aggregates as a Special Case 102
4.6 FLP Semantics of First-Order Aggregate Formulas 103
4.7 Comparing FLP and the First-Order Stable Model Semantics 104
4.8 Conclusion 107
4.9 Proofs 108
Proof of Proposition 18 115
Proof of Proposition 20 116
Proof of Proposition 21 117
Proofs of Lemma 26 and Proposition 22 117
Proofs of Proposition 23 and Proposition 24 118
Proof of Proposition 27 119
Proof of Theorem 8 120
Proof of Proposition 30 122

5 FIRST-ORDER STABLE MODEL SEMANTICS FOR GENERALIZED QUANTIFIED FORMULA 123

5.1 Stable Models of Formulas with Generalized Quantifiers 125
Syntax of Formulas with Generalized Quantifiers 125
Semantics of Formulas with Generalized Quantifiers 126
Stable Models of GQ-Formulas 129
Reduct-Based Definition 131

CHAPTER Page

5.2 Extensions of ASP as GQ formulas . . . 133
Aggregates as GQ-Formulas . . . 133
Non-monotonic DL-Programs as GQ Formulas . . . 134
Abstract Constraint Atoms as GQ-Formulas . . . 137
Explicit Constraints as GQ-Formulas . . . 139
5.3 Important Theorems . . . 141
Strong Equivalence . . . 141
Splitting Theorem . . . 142
Completion . . . 144
Safety for First-order Formulas with Generalized Quantifiers . . . 145
Semi-Safety . . . 146
Grounding . . . 148
Safety . . . 150
Loop Formulas for GQ-Formula . . . 152
5.4 First-Order FLP Semantics for Programs with Generalized Quantifiers . . . 154
5.5 Revisiting Non-Monotonic DL-programs . . . 156
5.6 Related Work . . . 159
Relation to GQ-Stable Models by Eiter *et al.* . . . 159
Relation to Infinitary Formulas . . . 160
Relation to Ferraris' Semantics . . . 162
5.7 Conclusion . . . 164
5.8 Proofs . . . 165
Useful Lemmas . . . 165
Proof of Proposition 32 . . . 168
Proof of Proposition 33 . . . 170
Proof of Theorem 10 . . . 171
Proof of Proposition 34 . . . 172
Proof of Proposition 35 . . . 173
Proof of Proposition 36 . . . 175
Proof of Proposition 37 . . . 177

CHAPTER Page

Proofs of Theorems 12 and 13 . . . 178
Proof of Theorem 14 . . . 185
Proof of Proposition 38 . . . 186
Proof of Proposition 39 . . . 191
Proof of Proposition 40 . . . 193
Proof of Theorem 15 . . . 199
Proof of Proposition 41 . . . 205
Proof of Proposition 42 . . . 208
Proof of Proposition 43 . . . 210
Proof of Proposition 44 . . . 211
Proof of Proposition 45 . . . 213

6 ANSWER SET PROGRAMMING MODULO THEORIES AND REASONING ABOUT CONTINUOUS CHANGES . . . 216

6.1 Answer Set Programming Modulo Theories . . . 218
Syntax . . . 218
Semantics . . . 219
An Example . . . 219
Completion . . . 220
Comparison with Clingcon . . . 221

6.2 Enhancing $\mathcal{C}$+ for Continuous Changes . . . 223
Syntax . . . 223
Semantics . . . 224
Reasoning about Continuous Changes in $\mathcal{C}$+ . . . 225
Reasoning about Additive Fluents . . . 228
Representing Processes in $\mathcal{C}+$ . . . 234

6.3 Hybrid Automata and $\mathcal{C}$+ . . . 236
Hybrid Automata . . . 236
Linear Hybrid Automata in $\mathcal{C}$+ Modulo Theories . . . 239

6.4 Related Work and Conclusion . . . 245

6.5 Proofs . . . 247

CHAPTER Page

Useful Lemmas and Theorem 247

Proof of Theorem 18 247

Proof of Theorem 19 250

Proof of Proposition 50 252

Proof of Proposition 51 256

Proof of Proposition 49 259

7 Conclusion 261

BIBLIOGRAPHY 263

LIST OF TABLES

Table Page

6.1 Experimental Results (Running Time) on Spacecraft Example 232

6.2 Experimental Results (Instance Size) on Spacecraft Example 234

LIST OF FIGURES

Figure Page

4.1 Loops and aggregate loop formulas for Π_1 98

4.2 Loops and aggregate loop formulas for $\mathrm{Pos}(\Pi_1)$ 99

5.1 The predicate dependency graph of the formula in Example 13 143

6.1 Analogy between SMT and ASPMT 217

6.2 Car Example in $\mathcal{C}$+ 227

6.3 Car Example in ASPMT 228

6.4 A Path in the Transition System of Car Example. 228

6.5 Spacecraft Example in Additive $\mathcal{C}$+ 232

6.6 Spacecraft Example in Basic $\mathcal{C}$+ 233

6.7 Spacecraft Example in ASPTM 233

6.8 Two Taps Water Tank Example $\mathcal{C}$+ 236

6.9 Two Taps Water Tank Example in ASPTM 237

6.10 A Path in the Transition System of Two Taps Water Tank Example. 237

6.11 Hybrid Automata for Water Tank System. 238

6.12 Hybrid Automaton of Water Tank (Figure 6.11) in $\mathcal{C}+$ 241

6.13 Water Tank Example in ASPMT 242

6.14 An execution of the Water Tank Example. 243

6.15 A Path in the Transition System of Water Tank Example. 243

Chapter 1

INTRODUCTION

Applications of Knowledge Representation and Reasoning (KR & R) techniques often motivate extensions of formalisms to allow new constructs and integrations with different formalisms. Such extensions or integrations allow one to take advantage of the expressivity of each of the component as well as their reasoning capabilities. However, the fundamental differences among different computing paradigms make the work non-trivial. How to develop an expressive yet efficient integrated system with an intuitive semantics is a constant research challenge.

The challenge inherently arises in logic programming, a subarea of KR & R. ASP is one of the most prominent and successful logic programming paradigms. It is based on the answer set (stable model) semantics of logic programs. Due to the simple syntax, the intuitive semantics to handle non-monotonic reasoning, and the availability of efficient solvers, ASP has been widely used in a variety of reasoning tasks, including building decision support systems for the Space Shuttle (Nogueira, Balduccini, Gelfond, Watson, & Barry, 2001), program configuration (Tiihonen, Soininen, Niemelä, & Sulonen, 2003), phylogenetic tree inference (Brooks, Erdem, Erdoğan, Minett, & Ringe, 2007) and formal analysis of security policies (Gelfond & Lobo, 2008). Motivated by applications, there are already many extensions of the modeling language of ASP to enhance expressivity and integrations of ASP with other computing paradigms to improve computation efficiency. However, due to the non-monotonic nature of the ASP semantics, such enhancements turned out to be non-trivial and the existing extensions are not fully satisfactory. In the following, we consider the extension of ASP with aggregates as well as the integration of ASP with description logics, constraints and SMT theory atoms.

- Aggregates significantly enhance the language of ASP by allowing natural and concise representations of many problems. On the other hand, defining a reasonable semantics of aggregates under the propositional answer set semantics (Gelfond & Lifschitz, 1988) turned out to be a non-trivial task. The difficulties result in many in-

teresting extensions of the original answer set semantics (Simons, 1999; Niemelä, Simons, & Soininen, 1999; Pelov, Denecker, & Bruynooghe, 2004; Faber, Leone, & Pfeifer, 2004; Marek & Truszczynski, 2004; Ferraris, 2005; Son, Pontelli, & Tu, 2007). However, different extensions have different understandings and characteristics of non-monotonicity, which make it difficult for one to study and compare the semantics. An aggregate is better to be understood as a first-order construct instead of a propositional one. As Lee, Lifschitz, and Palla pointed out in (Lee et al., 2008a), a COUNT aggregate can be viewed as a shorthand for first-order formulas.

- Description Logics (DLs) are a basis of the Web Ontology Language (OWL). Integrating DL with logic programs enables us to perform defeasible and closed world reasoning using existing ontology knowledge bases. For this reason, there have been many interests in such integration (Eiter, Lukasiewicz, Schindlauer, & Tompits, 2004; Rosati, 2005; Heymans, de Bruijn, Predoiu, Feier, & Nieuwenborgh, 2008; Eiter, Ianni, Lukasiewicz, Schindlauer, & Tompits, 2008a; Feier & Heymans, 2009; Fink & Pearce, 2010; Shen, 2011; Lee & Palla, 2011). However, the integration of ASP and DLs is not trivial since the semantics of ASP, which is propositional, cannot handle DL formulas, which are first-order, in a natural way.

- Grounding, which replaces variables with ground terms in all possible ways, is a process that most ASP solvers rely on. However, grounding w.r.t. a large domain produces a large set of instances, which becomes a bottleneck in ASP computation. In order to alleviate this "grounding problem," there have been several recent efforts to integrate ASP with constraint solving, or to compute ASP programs using Satisfiability Modulo Theories (SMT) solvers (Gebser, Ostrowski, & Schaub, 2009; Balduccini, 2009; Janhunen, Liu, & Niemela, 2011; Liu, Janhunen, & Niemelä, 2012), where functional fluents can be represented by variables in Constraint Satisfaction Problems or uninterpreted constants in SMT. However, like the traditional ASP, non-monotonicity of those extensions has to do with predicates only, and nothing to do with functions.

From the above discussion, we observe that one main reason for the difficulties in the extensions of ASP rooted in its propositional semantics, which cannot handle first-order

constructs and functions in natural ways.

We also observe that while the extensions were driven by different motivations and applications, a common issue is to extend the stable model semantics to incorporate “complex atoms", such as aggregates, DL-atoms, constraint atoms, and SMT theory atoms. However, a systematic study is still missing. Over the past few decades, many mathematical results on the stable model semantics have been developed, such as the splitting theorem (Lifschitz & Turner, 1994), the theorem on completion (Clark, 1978; Lloyd & Topor, 1984), the theorem on loop formulas (Lin & Zhao, 2002), the theorem on strong equivalence (Lifschitz, Pearce, & Valverde, 2001) and the theorem on safety (Faber et al., 2004). On the other hand, these results do not directly apply to the extensions and need to be reproven. E.g., the splitting theorem was extended to dl-programs (Eiter, Ianni, Lukasiewicz, Schindlauer, & Tompits, 2008b), to RASPL-1 programs (Lee et al., 2008a), and to weight constraint programs (Wang, You, Lin, Yuan, & Zhang, 2010a). The strong equivalence theorem was extended to programs with weight constraints (Turner, 2003), to monotone-constraint programs (Liu & Truszczynski, 2006), and to abstract constraints (Liu, Goebel, Janhunen, Niemelä, & You, 2011). Loop formulas were extended to constraint atoms in (You & Liu, 2008), to weight constraints and aggregates in (Lee & Meng, 2009; Liu, 2009), and to non-monotonic dl-programs in (Wang, You, Yuan, & Shen, 2010b).

Our observations identify two problems: (1) the propositional answer set semantics is too restrictive for the extensions of ASP; (2) a common logical framework that captures the extensions of ASP and a systematic way of studying the extensions are needed.

Recently, the Stable Model semantics was lifted to the first-order level (Ferraris et al., 2007; Ferraris, Lee, & Lifschitz, 2011a). The new semantics is based on second-order logic and unifies first-order logic and logic programs. In (Bartholomew & Lee, 2012), the semantics is further extended to allow intensional functions. These provide a suitable framework for integrating logic programs with aggregates, DLs, constraints and SMT theory atoms. The most straightforward integration is to reduce the extended constructs into first-order formulas. The paper (Lee et al., 2008a) presented a reductive approach to defining a special class of aggregates. However, it is not always the case that integrated constructs

are reducible to first-order formulas.

HEX programs (Eiter, Ianni, Schindlauer, & Tompits, 2005) provide an elegant approach to a unifying framework. It incorporates different extensions of the stable model semantics in a uniform framework via "external atoms" to interface with external computation sources. On the other hand, HEX programs are based on the the FLP semantics (Faber, Pfeifer, & Leone, 2011), which is also limited to the propositional case. Another issue with the FLP semantics is its semantics properties. For instance, the “conservative extensions" are not preserved in the semantics. [1] As another example, choice rules (Niemelä & Simons, 2000), which are often useful in the “generate-and-test” organization of a program, are not supported in the FLP semantics.

This dissertation presents a unifying framework that combines the semantic properties of the first-order stable model semantics and the versatility of HEX programs. We present a uniform view on the extensions of ASP by viewing them as instances of generalized quantifiers and formulas with intensional functions. Our framework yields a systematic approach to study and extend non-monotonic languages. In particular, this dissertation

- relates the first-order stable model semantics with the first-order logic and presents some decidable conditions under which the former can be represented by the latter. This gives us insight on the precise relationship between the two semantics.

- provides a uniform view on some of the representative semantics of aggregates. This helps us compare the semantics and allows us to apply the results already established for propositional theories to each of the semantics. Based on the reduction, we present the first-order stable model semantics and the first-order FLP semantics for aggregate formulas which do not involve grounding. We also study the precise relationships between the two generalized semantics.

- presents a unifying and reductive view on the extensions of ASP by viewing them as special cases of formulas with generalized quantifiers under the first-order stable model semantics. The framework provides a new perspective on the existing exten-

[1] See the related discussion in http://www.cs.utexas.edu/∼vl/tag/aggregates.

sions of the stable model semantics, a way to compare them and even allows them to be combined in a single language. The unifying framework also saves efforts in re-proving the theorems for these individual extension. We extend several important theorems in ASP to formulas with generalized quantifiers, which in turn can be applied to the particular extensions of the stable model semantics. To compare our approach with HEX programs, we generalize the first-order FLP semantics to cover formulas with generalized quantifiers and compare the two semantics in the general context.

- proposes a framework of tight integration of ASP and SMT as a special case of functional stable model semantics that assumes background theories. Based on the framework, we extend and enhance action language $\mathcal{C}$+ (Giunchiglia, Lee, Lifschitz, McCain, & Turner, 2004) to handle reasoning about continuous changes. By reformulating $\mathcal{C}$+ in terms of ASPMT, we naturally extend the language to overcome the limitation, and use SMT solvers to compute the language.

The document is organized as follows. Chapter 2 gives the necessary background information. In Chapter 3, we discuss how first-order stable model semantics is related to first-order logic. In Chapter 4, we discuss our efforts in reformulating existing semantics on aggregates and extension of the semantics to the first-order level. In Chapter 5, we extend the first-order stable model semantics and the first-order FLP semantics to formulas with generalized quantifiers. We reduce several extensions of ASP to formulas with generalized quantifiers (GQ-formulas) and generalize some important theorems to handle GQ-formulas. Chapter 6 presents the tight integration of ASP and SMT. It also discusses how to define the semantics of action language $\mathcal{C}$+ in terms of the framework. We conclude in Chapter 7.

Chapter 2

BACKGROUND

2.1 Answer Set Programming

ASP (Marek & Truszczyński, 1999; Lifschitz, 2008) is a recent form of declarative programming that has emerged from the interaction between two lines of research—nonmonotonic semantics of negation in logic programming and applications of satisfiability solvers to search problems. The idea of ASP is to represent the search problem we are interested in as a logic program whose intended models, called "stable models (a.k.a. answer sets)," correspond to the solutions of the problem, and then find these models using an answer set solver—a system for computing stable models. Like other declarative computing paradigms, such as SAT (Satisfiability Checking) and CP (Constraint Programming), ASP provides a common basis for formalizing and solving various problems, but is distinct from others in that it focuses on knowledge representation and reasoning: its language is an expressive nonmonotonic language based on logic programs under the stable model semantics (Gelfond & Lifschitz, 1988; Ferraris et al., 2007), which allows elegant representation of several aspects of knowledge such as causality, defaults, and incomplete information, and provides compact encoding of complex problems that cannot be translated into SAT and CP (Lifschitz & Razborov, 2006). As the mathematical foundation of answer set programming, the stable model semantics originated from understanding the meaning of *negation as failure* in Prolog, which has the rules of the form

$$a_1 \leftarrow a_2, \ldots, a_m, \mathrm{not}\ a_{m+1}, \ldots, \mathrm{not}\ a_n \tag{2.1}$$

where all a_i are atoms and not is a symbol for *negation as failure*, also known as *default negation*. Intuitively, under the stable model semantics, rule (2.1) means that if you have generated $a_2, \ldots, a_m$ and it is impossible to generate any of $a_{m+1}, \ldots, a_n$ then you may generate a_1. This explanation looks like it contains a vicious cycle, but the semantics is carefully defined in terms of fixpoint. Formally, a program is called *positive* if none of the rules in the program contain the symbol not. 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 $\mathrm{not}\ a_i$

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. As a result, all answer sets are Herbrand models of the signature obtained from the program. The semantics was later extended to programs with classical negation, and disjunction in the head (Gelfond & Lifschitz, 1991).

Example 1 *For program* Π *that contains three rules*

$$p(a). \qquad r(X) \leftarrow p(X), not\ q(X).$$
$$q(b).$$

Grounding replaces the variable x *with each of* $\{a, b\}$*. It results in the following program* Π'*:*

$$p(a). \qquad r(a) \leftarrow p(a), not\ q(a).$$
$$q(b). \qquad r(b) \leftarrow p(b), not\ q(b).$$

We check that $X = \{p(a),\ q(b),\ r(a)\}$ *is an answer set because* Π'^X *is*

$$p(a). \qquad r(a) \leftarrow p(a).$$
$$q(b).$$

and $\{p(a),\ q(b),\ r(a)\}$ *is the minimal set that satisfies* Π'^X*.*

The language also has useful constructs, such as strong negations, weak constraints, and preferences. What distinguishes ASP from other nonmonotonic formalisms is the availability of several efficient implementations, answer set solvers, such as smodels[1], cmodels[2], clasp[3], which led to practical nonmonotonic reasoning that can be applied to industrial level applications.

2.2 Theorem on Loop Formulas

According to the theorem on loop formulas (Lin & Zhao, 2004), the stable models of a logic program (Gelfond & Lifschitz, 1988) can be characterized as the models of the logic program that satisfy all its loop formulas. This idea has turned out to be widely applicable

[1] http://www.tcs.hut.fi/Software/smodels .
[2] http://www.cs.utexas.edu/users/tag/cmodels.html .
[3] http://potassco.sourceforge.net .

in relating the stable model semantics to propositional logic, and has resulted in an efficient method for computing answer sets using SAT solvers. Since the original invention of loop formulas for nondisjunctive logic programs by Lin and Zhao (2004), the theorem has been extended to more general classes of logic programs, such as disjunctive programs (Lee & Lifschitz, 2003b), infinite programs and programs containing classical negation (Lee, 2005; Lee, Lierler, Lifschitz, & Yang, 2010), arbitrary propositional formulas under the stable model semantics (Ferraris, Lee, & Lifschitz, 2006), and programs containing aggregates (Liu & Truszczynski, 2006; You & Liu, 2008). The theorem has also been applied to other nonmonotonic formalisms, such as nonmonotonic causal theories (Lee, 2004) and McCarthy's circumscription (Lee & Lin, 2006). The notion of a loop was further refined as an "elementary loop" (Gebser & Schaub, 2005; Gebser, Lee, & Lierler, 2006, 2011). However, all this work is restricted to the propositional case. Variables contained in the program are first eliminated by grounding—the process which replaces every variable with every object constant—and then loop formulas are obtained from the ground program. As a result, loop formulas were defined as formulas in propositional logic.

Chen, Lin, Wang, and Zhang's definition (2006) of a first-order loop formula is different in that loop formulas are directly obtained from a non-ground program, so that they are formulas in first-order logic which retain variables. However, since the semantics of a logic program that they refer to is based on grounding, their first-order loop formulas are simply understood as schemas for ground loop formulas, and only Herbrand models of the loop formulas were considered in this context.

2.3 Extensions of ASP

Wide applications of answer set programming motivated various extensions to the language and implementations, for instance, to reason about aggregates, to facilitate interface with external information source, such as ontologies and to integrate different computing paradigms such as Constraint Programming and Satisfiability Modulo Theories.

Aggregates

Aggregates are motivated by applications of ASP. They greatly enhance the modeling power of the language and simplify the representation. On the other hand, defining a reasonable

semantics of aggregates under the stable model semantics turned out to be a non-trivial task. There have been many different semantics proposed, such as a modified definition of traditional reduct (Faber et al., 2004), an extension of T_P operator with "conditional satisfaction" (Son et al., 2007), and an extension of Fitting's approximating operator Φ_P to Φ_P^{aggr} (Pelov, Denecker, & Bruynooghe, 2007). Meanwhile, a few reasonable "reductive" semantics of aggregates have also been developed. In (Pelov, Denecker, & Bruynooghe, 2003) and (Son & Pontelli, 2007), the authors provided a way to identify aggregates with nested expressions, which are more complex than the naive approach above, by involving the notions of "maximal local power set" or "unfolding w.r.t. solutions." Instead of referring to nested expressions, Ferraris (2005) proposed to identify an aggregate with conjunctions of implications under his extension of the answer set semantics for arbitrary propositional formulas.

While most semantics agree on monotone and anti-monotone aggregates, they have subtle differences in understanding arbitrary recursive aggregates. We consider three representative semantics: the semantics from (Pelov et al., 2003; Son et al., 2007) (we will call it *PDB-SPT* semantics; Lemma 6 from (Son & Pontelli, 2007) tells that the semantics from (Son et al., 2007) is essentially equivalent to the semantics from (Pelov et al., 2003)), the semantics from (Faber et al., 2004) (we will call it the *FLP* semantics), and the semantics from (Ferraris, 2005) (we will call it the *Ferraris* semantics). Besides the non-trivial reduction of aggregates to nested expressions in (Pelov et al., 2003; Son & Pontelli, 2007), non-reductive approaches for the PDB-SPT semantics have been studied by several researchers as in (Pelov et al., 2007; Son et al., 2007; Son & Pontelli, 2007; You & Liu, 2008). The FLP semantics is based on an interesting modification of the traditional reduct and has been implemented in an answer set solver DLV.[4] The Ferraris semantics appears close to the FLP semantics but is based on reduction to propositional formulas under the stable model semantics, which is essentially a reformulation of the equilibrium logic (Pearce, 1997), and is generalized to arbitrary first-order formulas in (Ferraris et al., 2007). The three semantics mentioned above do not agree with each other.

[4] http://www.dbai.tuwien.ac.at/proj/dlv/ .

Example 2 *Consider the following program* Π_1*:*

$$\begin{aligned} p(2) &\leftarrow not\ \mathsf{SUM}\langle\{x : p(x)\}\rangle < 2 \\ p(-1) &\leftarrow \mathsf{SUM}\langle\{x : p(x)\}\rangle \geq 0 \\ p(1) &\leftarrow p(-1). \end{aligned} \tag{2.2}$$

This program has no answer sets according to the PDB-SPT semantics, one answer set $\{p(-1), p(1)\}$ *according to the FLP semantics, and two answer sets* $\{p(-1), p(1)\}$ *and* $\{p(-1), p(1), p(2)\}$ *according to the Ferraris semantics.*

Description Logics

Description Logics (DLs) are a family of knowledge representation languages. They are decidable fragments of first-order logic and also the basis of the Web Ontology Language (OWL), which has greatly facilitated knowledge construction, storage and sharing across the web. While DLs describe the terminological knowledge, logic programs represent constraints and defaults over them. Integrating DL with logic programs would enable one to perform defeasible and closed world reasoning using existing ontology knowledge bases.

For this reason, there is increasing interest in such integration (Eiter et al., 2004; Rosati, 2005; Heymans et al., 2008; Eiter et al., 2008a; Feier & Heymans, 2009; Fink & Pearce, 2010; Shen, 2011; Lee & Palla, 2011). The integrations of description logics with logic programming usually result in a hybrid knowledge base which is a pair $(\mathcal{T}, \mathcal{P})$ where $\mathcal{T}$ is a knowledge base in description logics of a signature $\Sigma_{\mathcal{T}}$ and $\mathcal{P}$ is a logic program of a signature $\Sigma_{\mathcal{P}}$. DL-programs (Eiter et al., 2008a) provides an interesting way of integrating DLs with ASP. In DL-programs, the program $\mathcal{P}$ interfaces with $\mathcal{T}$ via a special construct, which represents a query to the ontologies. The queries to $\mathcal{T}$ are treated in a way such that logic programming and description logic inference are technically separated. Compared to other similar work, this increases flexibility and provides a way to utilize efficient implementations from both formalisms. DL-programs are implemented in the system DLV-HEX.

There are three alternative approaches for defining the semantics of DL-programs: the one based on strong (or weak) transformation (Eiter et al., 2004), FLP semantics (Fink & Pearce, 2010) and well-supported semantics (Shen, 2011).

Constraints & SMT

Constraint programming (CP) is another computing paradigm that provides succinct representation and a powerful way of solving combinatorial search problems. The computational methods in CP are effective in reasoning about problems with variables whose values range over various domains such as arithmetic computations. Observing the grounding bottleneck in ASP, (Baselice, Bonatti, & Gelfond, 2005) and (Mellarkod, Gelfond, & Zhang, 2008) enhanced ASP with constraints and provided a family of extensions of ASP with different constraint classes. (Balduccini, 2009) and (Gebser et al., 2009) follow the work and propose an alternative way of combining ASP and CP in a way that is similar to the “lazy approach" of SMT solvers (Nieuwenhuis, Oliveras, & Tinelli, 2006). Janhunen et al.(2011) provide an integration of ASP and SMT theories by translating the integrated program into difference logic.

Marek and Truszczynski (2004) viewed propositional aggregates as a special case of abstract constraint atoms. Son et al. (2007) generalized this semantics to account for arbitrary abstract constraint atoms. Abstract constraint atoms is a generalization of aggregates to represent arbitrary constraints on atoms. A constraint atom (or c-atom following (Son et al., 2007)) is of the form (D, C). Intuitively, a c-atom represents a constraint with a finite set C of admissible solutions over a finite domain D. The semantics of programs with abstract constraint atoms was first proposed in (Marek & Truszczynski, 2004), and was further developed in (Liu & Truszczynski, 2006; Son et al., 2007; Shen, You, & Yuan, 2009). Son, Pontelli and Tu (Son et al., 2007) proposed a semantics of programs with abstract constraints atoms based on “conditional satisfaction”, which turns out to be equivalent to the PDB-SPT semantics of aggregates.

2.4 HEX Programs & FLP Semantics

HEX programs (Eiter et al., 2005) provide an approach to a unifying framework. It incorporates different extensions of the stable model semantics in a uniform framework via "external atoms" to interface with external computation sources. HEX programs are well studied (Eiter, Ianni, Schindlauer, & Tompits, 2006a; Eiter et al., 2008a; Eiter, Fink, Ianni,

Krennwallner, & Schüller, 2011) and are implemented in the system DLV-HEX,[5] which has been applied to several applications including biomedical ontologies (Hoehndorf, Loebe, Kelso, & Herre, 2007) and web query (SPARQL) (Polleres & Schindlauer, 2007). Instead of the the traditional stable model semantics (Gelfond & Lifschitz, 1988), the semantics of HEX programs adopts the "FLP semantics" (Faber et al., 2011). The adoption was a key idea that facilitates incorporating external atoms in HEX programs.

The FLP semantics is based on an interesting modification of the traditional definition of a reduct by Gelfond and Lifschitz (1988). The FLP-reduct ($\Pi^{\underline{X}}$) of a program Π relative to a set X of atoms is obtained from Π by simply removing all rules whose bodies are not satisfied by X. X is called an *FLP-answer set* of Π if X is minimal among the sets of atoms that satisfy $\Pi^{\underline{X}}$.

Example 1 continued *The FLP-reduct of*

$$\begin{array}{ll} p(a). & r(a) \leftarrow p(a), not\ q(a). \\ q(b). & r(b) \leftarrow p(b), not\ q(b). \end{array}$$

relative to $X = \{p(a), q(b), r(a)\}$ *is*

$$\begin{array}{ll} p(a). & r(a) \leftarrow p(a), not\ q(a). \\ q(b). & \end{array} \tag{2.3}$$

and X *is minimal among the sets of atoms satisfying the reduct, and hence is an FLP answer set.*

The FLP semantics diverges from the traditional stable model semantics in some essential ways. For example, consider a program

$$p(a) \leftarrow \text{not COUNT}\langle x.p(x)\rangle < 1, \tag{2.4}$$

and another program which rewrites the first program as

$$\begin{array}{l} p(a) \leftarrow \text{not } q \\ q \leftarrow \text{COUNT}\langle x.p(x)\rangle < 1, \end{array} \tag{2.5}$$

[5] http://www.kr.tuwien.ac.at/research/systems/dlvhex/

where the second rule defines q in terms of the count aggregate. One may expect this transformation to modify the collection of answer sets in a "conservative" way. That is, each answer set of (2.5) is obtained from an answer set of (2.4) in accordance with the definition of q.[6] However, this is not the case under the FLP stable model semantics: the former has $\emptyset$ as the only FLP answer set while the latter has both $\{p(a)\}$ and $\{q\}$ as the FLP answer sets.

Related to this issue is the anti-chain property that is ensured by the FLP semantics: no FLP answer set is a proper subset of another FLP answer set. This prevents us from allowing choice rules (Niemelä & Simons, 2000), which are a useful construct in the "generate-and-test" organization of ASP programming (Lifschitz, 2002).

Also, the extensions of the FLP semantics to allow complex formulas in (Truszczyński, 2010; Bartholomew, Lee, & Meng, 2011) encounter some unintuitive cases. For example, according to the extensions, $\{p\}$ is the FLP answer set of $p \leftarrow p \vee \neg p$, but this has a circular justification.

2.5 First-Order Stable Model Semantics

The stable model semantics is extended to the first-order level in (Ferraris et al., 2007) which we will review in the following.

A *signature* is a set of *function constants* and *predicate constants*. Function constants of arity 0 are called *object constants*. The syntax of a *formula* is the same as in first-order logic. We consider the following set of primitive propositional connectives and quantifiers:

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

Note that negation is not a primitive connective because we $\neg F$ as an abbreviation of $F \rightarrow \bot$. Similarly, $\top$ is an abbreviation of $\bot \rightarrow \bot$, and $F \leftrightarrow G$ is an abbreviation of $(F \rightarrow G) \wedge (G \rightarrow F)$. An *atom* of a signature σ is of the form $p(t_1, \ldots, t_n)$ where p is an n-ary predicate constant in σ and $t_1, \ldots, t_n$ are terms that can be formed from function

[6] Indeed, this is what happens in expressing a rule with nested expressions like $p \leftarrow \text{not not } p$ into $p \leftarrow \text{not } q$, $q \leftarrow \text{not } p$ by defining q as $\text{not } p$.

constants in σ and object variables. An *atomic formula* of σ is either an atoms of σ, an equality between terms of σ, or the 0-place connective $\bot$.

Let $\boldsymbol{p}$ be a list of predicate constants and $\boldsymbol{u}$ be a list of distinct predicate variables of the same length as $\boldsymbol{p}$. By $\boldsymbol{u} = \boldsymbol{p}$ we denote the conjunction of the formulas $\forall \boldsymbol{x}(u_i(\boldsymbol{x}) \leftrightarrow p_i(\boldsymbol{x}))$, where $\boldsymbol{x}$ is a list of distinct object variables of the same length as the arity of p_i, for all $i = 1, \ldots, n$. By $\boldsymbol{u} \leq \boldsymbol{p}$ we denote the conjunction of the formulas $\forall \boldsymbol{x}(u_i(\boldsymbol{x}) \rightarrow p_i(\boldsymbol{x}))$ for all $i = 1, \ldots, n$, and $\boldsymbol{u} < \boldsymbol{p}$ stands for $(\boldsymbol{u} \leq \boldsymbol{p}) \wedge \neg(\boldsymbol{u} = \boldsymbol{p})$.

For any first-order sentence F, the *stable models of F relative to a list of predicates $\boldsymbol{p}$* are the models of second-order formula,denoted by $\mathrm{SM}[F; \boldsymbol{p}]$,

$$F \wedge \neg \exists \boldsymbol{u}((\boldsymbol{u} < \boldsymbol{p}) \wedge F^*(\boldsymbol{u})), \tag{2.6}$$

where $F^*(\boldsymbol{u})$ is defined recursively:

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

We will often simply write $\mathrm{SM}[F]$ instead of $\mathrm{SM}[F; \boldsymbol{p}]$ when $\boldsymbol{p}$ is the list of all predicate constants occurring in F, and call a model of $\mathrm{SM}[F]$ a *stable* model of F. We distinguish between the terms "stable models" and "answer sets" as follows.[7] By $\sigma(F)$ we denote the signature consisting of the function and predicate constants occurring in F. If F contains at least one object constant, an Herbrand interpretation[8] 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

[7] The distinction is useful because in the first-order setting, stable models are no longer Herbrand interpretations and may not be represented by *sets* of atoms.

[8] An *Herbrand interpretation* of a signature σ (containing at least one object constant) is an interpretation of σ such that its universe is the set of all ground terms of σ, and every ground term represents itself. An Herbrand interpretation can be identified with the set of ground atoms to which it assigns the value *true*.

the answer sets of the *FOL-representation* of Π, which is the conjunction of the universal closures of implications corresponding to the rules.

Example 1 continued *The FOL-representation F of Π is*

$$p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)) \tag{2.7}$$

and $SM[F]$ is

$$p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)) \wedge \neg\exists uvw(((u,v,w) < (p,q,r)) \wedge u(a) \wedge v(b) \\ \wedge\forall x(((u(x) \wedge (\neg v(x) \wedge \neg q(x))) \rightarrow w(x)) \wedge ((p(x) \wedge \neg q(x)) \rightarrow r(x)))),$$

which is equivalent to the first-order sentence

$$\forall x(p(x) \leftrightarrow x = a) \wedge \forall x(q(x) \leftrightarrow x = b) \wedge \forall x(r(x) \leftrightarrow (p(x) \wedge \neg q(x))) \tag{2.8}$$

The stable models of F are any first-order models of (2.8). The only answer set of F is the Herbrand model $\{p(a),\ q(b),\ r(a)\}$.

2.6 Functional Stable Model Semantics

The first-order stable model semantics is further extended to allow intensional function in (Bartholomew & Lee, 2012). We consider the same syntax as in the previous section. For lists of predicate symbols $\boldsymbol{u}$ and $\boldsymbol{p}$, expressions $\boldsymbol{u} \leq \boldsymbol{p}$ and $\boldsymbol{u} = \boldsymbol{p}$ are defined the same as before. The notation is extended to cover function symbols (constants or variables) v and c by defining the expression $u = c$ as $\forall \boldsymbol{x}(u(\boldsymbol{x}) = c(\boldsymbol{x}))$.

Let $\boldsymbol{c}$ be a list of distinct predicate and function constants, and let $\widehat{\boldsymbol{c}}$ be a list of distinct predicate and function variables corresponding to $\boldsymbol{c}$. Members of $\boldsymbol{c}$ are called *intensional* constants. By $\boldsymbol{c}^{pred}$ ($\boldsymbol{c}^{func}$, respectively) we mean the list of all predicate constants (function constants, respectively) in $\boldsymbol{c}$, and by $\widehat{\boldsymbol{c}}^{pred}$ the list of the corresponding predicate variables in $\widehat{\boldsymbol{c}}$.

For any formula F, expression $\mathrm{SM}[F;\ \boldsymbol{c}]$ is defined as

$$F \wedge \neg\exists\widehat{\boldsymbol{c}}(\widehat{\boldsymbol{c}} < \boldsymbol{c} \wedge F^*(\widehat{\boldsymbol{c}})),$$

where $\widehat{\boldsymbol{c}} < \boldsymbol{c}$ is shorthand for $(\widehat{\boldsymbol{c}}^{pred} \leq \boldsymbol{c}^{pred}) \wedge \neg(\widehat{\boldsymbol{c}} = \boldsymbol{c})$, and $F^*(\widehat{\boldsymbol{c}})$ is defined the same as in the previous section except that the first two bullets are replaced by the following bullet:

- When F is an atomic formula, F^* is $F' \wedge F$ where F' is obtained from F by replacing all intensional (function and predicate) constants $\boldsymbol{c}$ in it with the corresponding (function and predicate) variables from $\widehat{\boldsymbol{c}}$;

This definition of a stable model is a proper generalization of the one from (Ferraris et al., 2011a): in the absence of intensional function constants, it reduces to the one in (Ferraris et al., 2011a).

2.7 Action Language $\mathcal{C}+$

One of the main focuses of Artificial Intelligence research involves describing the effects of actions and automatically generating plans to achieve some given goal. Action languages (Gelfond & Lifschitz, 1998) are formal models of parts of the natural language that describe the effects of actions which corresponds to transitions of the form $\langle s, a, s' \rangle$. The transition denotes that action a is executed in state s and s' is the resulting state of the action. Action languages provide an intuitive way to describe complex problems instead of encoding the entire transition system.

There are many efforts to study different aspects of dynamic systems (Pednault, 1994; Gelfond, 1993; Baral, Gelfond, & Provetti, 1997; Giunchiglia & Lifschitz, 1998). Action language $\mathcal{C}+$ (Giunchiglia, Lee, Lifschitz, & Turner, 2001) is a well-recognized member of the action language family. It is based on non-monotonic causal logic (McCain & Turner, 1997). Language $\mathcal{C}+$ is significantly more enhanced than its predecessor $\mathcal{C}$ in several ways, such as being able to represent multi-valued formulas, defined fluents, additive fluents, rigid constants and defeasible causal laws. The Causal Calculator (CCALC) is an implementation of the $\mathcal{C}+$ language that computes the definite fragment of $\mathcal{C}+$ by turning it into propositional logic using literal completion. Models of the resulting formulas are computed using SAT solvers. CCALC has been applied to several challenging problems including to provide a group of robots with high-level reasoning (Caldiran, Haspalamutgil, Ok, Palaz, Erdem, & Patoglu, 2009), to give executable specifications of norm-governed computational societies (Artikis, Sergot, & Pitt, 2003), and to automate the analysis of business processes (Armando & Compagna, 2002).

Chapter 3

FIRST-ORDER STABLE MODEL SEMANTICS AND FIRST-ORDER LOOP FORMULAS

In this chapter, we discuss how first-order loop formulas and the first-order stable model semantics are related to each other. In general, the first-order stable model semantics is more expressive than first-order logic. Like circumscription, the concept of transitive closure can be represented in the former, but it can not be represented using any set of first-order formulas, even if the set is allowed to be infinite.[1] However, as we show in this chapter, understanding the precise relationship between them gives us insights into the semantics and computational properties of the stable model semantics.

In order to facilitate the comparison, we first extend the loop formulas for nondisjunctive programs to disjunctive programs and to arbitrary first-order theories. Then, we reformulate the SM operator in the style of loop formulas, which characterize a loop by a second-order formula. From this reformulation, we present some conditions, under which a first-order theory under the stable model semantics can be equivalently rewritten as first-order loop formulas.

Based on the studied relationship, we extend the syntax of logic programs to allow explicit quantifiers. This overcomes some limitations of traditional answer set programs in reasoning about non-Herbrand models. The semantics of such extended programs is defined by identifying them as a special class of first-order theories under the stable model semantics. Such programs inherit from the general language the ability to handle nonmonotonic reasoning under the stable model semantics even in the absence of the unique name and the domain closure assumptions that are built into the grounding-based answer set semantics. On the other hand, the restricted syntax of an extended program yields more succinct loop formulas. Consider the following program Π_1 which represents a simple

[1] Vladimir Lifschitz, personal communication.

insurance policy example:

$$\begin{aligned} HasWife(x) &\leftarrow \exists y\ \mathrm{Spouse}(x, y) \\ \mathrm{HasWife}(x) &\leftarrow \mathrm{Man}(x),\ \mathrm{Married}(x) \\ \mathrm{Married}(x) &\leftarrow \mathrm{Man}(x),\ \mathrm{HasWife}(x) \\ \exists w\ \mathrm{Discount}(x, w) &\leftarrow \mathrm{Married}(x),\ \mathrm{not}\ \exists z\ \mathrm{Accident}(x, z). \end{aligned} \tag{3.1}$$

The second and the third rules express that $\mathrm{Married}(x)$ and $\mathrm{HasWife}(x)$ are synonymous to each other when x is a Man. The last rule states that x is eligible for some discount plan (with the name unknown) if x is married and has no record of accident. The quantifier in the first rule can be dropped without affecting the meaning, but the other quantifiers cannot. We will say that a program Π *entails* a query F (under the stable model semantics) if every stable model of Π satisfies F. For example,

- Π_1 conjoined with $\Pi_2 = \{\mathrm{Man}(\mathrm{John})\}$ entails each of $\neg\exists x\ \mathrm{Married}(x)$ and $\neg\exists xy\ \mathrm{Discount}(x, y)$.
- $\Pi_1 \cup \Pi_2$ conjoined with $\Pi_3 = \{\exists y\ \mathrm{Spouse}(\mathrm{John}, y)\}$ entails neither $\neg\exists x\ \mathrm{Married}(x)$ nor $\neg\exists xy\ \mathrm{Discount}(x, y)$, but entails each of $\exists x\ \mathrm{Married}(x)$, $\exists xy\mathrm{Discount}(x, y)$, and $\forall xy(\mathrm{Discount}(x, y) \rightarrow x = \mathrm{John})$.
- $\Pi_1 \cup \Pi_2 \cup \Pi_3$ conjoined with $\Pi_4 = \{\exists z\ \mathrm{Accident}(\mathrm{John}, z)\}$ does not entail $\forall xy(\mathrm{Discount}(x, y) \rightarrow x = \mathrm{John})$, but entails $\neg\exists w\ \mathrm{Discount}(\mathrm{John}, w)$.

In order to provide such non-monotonic reasoning, non-Herbrand models need to be considered. This is because the names (or identifiers) of discount plans, spouses and accident records may be unknown. On the other hand, the traditional answer set semantics is limited to Herbrand models because of grounding. The reasoning process in the example can be automated using a first-order theorem prover by turning the program into first-order loop formulas.

This chapter is organized as follows. Section 3.1 reviews the theorem on first-order loop formulas by Chen et al. (2006) and extends it to disjunctive programs and to arbitrary first-order sentences, limiting attention to Herbrand stable models. Section 3.2 extends

these results to allow non-Herbrand stable models as well (possibly allowing functions) under a certain semantic condition, and compare the first-order stable model semantics with loop formulas by reformulating the former in terms of the latter. In Section 3.3, we present a series of syntactic conditions that imply the semantic condition in Section 3.2. Section 3.4 provides an extension of logic programs that contain explicit quantifiers and shows how query answering for such extended programs can sometimes be reduced to entailment checking in first-order logic via loop formulas. In Section 3.5, the results are further extended to distinguish between intensional and non-intensional predicates. Related work is described in Section 3.6, and long proofs are given in Section 3.8.

3.1 First-Order Loop Formulas and Herbrand Models

We extend the definition of a first-order loop formula for a nondisjunctive program (Chen et al., 2006) to a disjunctive program and to an arbitrary first-order sentence. We first review Chen et al.'s work in the following.

Loop Formula for Non-Disjunctive Programs

We call a formula *negative* if every occurrence of every predicate constant in it belongs to the antecedent of an implication. For instance, any formula of the form $\neg F$ is negative because this expression is shorthand for $F \to \bot$. An equality $t_1 = t_2$ is also negative because it contains no predicate constants.

A *nondisjunctive* program is a finite set of rules of the form

$$A \leftarrow B, N, \tag{3.2}$$

where A is an atom, B is a set of atoms, and N is a negative formula. The rules may contain function constants of positive arity.[2]

We will say that a nondisjunctive program is in *normal form* if, for all rules (3.2) in it, A is of the form $p(\boldsymbol{x})$ where $\boldsymbol{x}$ is a list of distinct variables. It is clear that every program can be turned into normal form using equality in the body. For instance, $p(a, b) \leftarrow q(a)$ can be rewritten as $p(x, y) \leftarrow q(a), x = a, y = b$.

[2] The original definition by Chen et al. (2006) does not allow function constants of positive arity.

Let Π be a nondisjunctive program and let $\mathrm{Norm}(\Pi)$ be a normal form of Π. By $\sigma(\Pi)$ we denote the signature consisting of function and predicate constants occurring in Π. Given a finite set Y of atoms constructed from symbols in $\sigma(\Pi)$ and object variables, we assume that $\mathrm{Norm}(\Pi)$ does not contain variables in Y, by renaming the variables in $\mathrm{Norm}(\Pi)$. The *(first-order) external support formula* of Y for Π, denoted by $\mathrm{ES}_\Pi(Y)$, is the disjunction of

$$\bigvee_{\theta: A\theta \in Y} \exists \boldsymbol{z} \Bigg(B\theta \wedge N\theta \wedge \bigwedge_{\substack{p(\boldsymbol{t}) \in B\theta \\ p(\boldsymbol{t}') \in Y}} (\boldsymbol{t} \neq \boldsymbol{t}') \Bigg) \tag{3.3}$$

for all rules (3.2) in $\mathrm{Norm}(\Pi)$,[3] where θ is a substitution that maps variables in A to terms occurring in Y, and $\boldsymbol{z}$ is the list of all variables that occur in

$$A\theta \leftarrow B\theta, N\theta$$

but not in Y.

The *(first-order) loop formula* of Y for Π, denoted by $\mathrm{LF}_\Pi(Y)$, is the universal closure of

$$\bigwedge Y \rightarrow \mathrm{ES}_\Pi(Y). \tag{3.4}$$

(The expression $\bigwedge Y$ in the antecedent stands for the conjunction of all elements of Y.) When Π is a propositional program, $\mathrm{LF}_\Pi(Y)$ is equivalent to a conjunctive loop formula as defined by Ferraris et al. (2006).

The definition of a first-order dependency graph and the definition of a first-order loop are as follows. We say that an atom $p(\boldsymbol{t})$ *depends on* an atom $q(\boldsymbol{t}')$ in a rule (3.2) if $p(\boldsymbol{t})$ is A and $q(\boldsymbol{t}')$ is in B. The *(first-order) dependency graph* of Π is an infinite directed graph (V, E) such that

- V is the set of atoms of signature $\sigma(\Pi)$;[4]
- $(p(\boldsymbol{t})\theta, q(\boldsymbol{t}')\theta)$ is in E if $p(\boldsymbol{t})$ depends on $q(\boldsymbol{t}')$ in a rule of Π and θ is a substitution that maps variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$ to terms (including variables) of $\sigma(\Pi)$.

[3] For any lists of terms $\boldsymbol{t} = (t_1, \ldots, t_n)$ and $\boldsymbol{t}' = (t'_1, \ldots, t'_n)$ of the same length, $\boldsymbol{t} = \boldsymbol{t}'$ stands for $(t_1 = t'_1) \wedge \cdots \wedge (t_n = t'_n)$.

[4] Note that V is infinite since infinitely many object variables can be used to form atoms.

A nonempty subset L of V is called a *(first-order) loop* of Π if the subgraph of the first-order dependency graph of Π induced by L is strongly connected.

Example 3 *Let Π be the following program:*

$$\begin{aligned} p(x) &\leftarrow q(x) \\ q(y) &\leftarrow p(y) \\ p(z) &\leftarrow not\ r(z). \end{aligned} \tag{3.5}$$

The following sets of atoms are first-order loops (among many others): $Y_1 = \{p(u)\}$, $Y_2 = \{q(u)\}$, $Y_3 = \{r(u)\}$, $Y_4 = \{p(u), q(u)\}$. Their loop formulas are

$$\begin{aligned} LF_\Pi(Y_1) &= \forall u(p(u) \rightarrow (q(u) \vee \neg r(u))), \\ LF_\Pi(Y_2) &= \forall u(q(u) \rightarrow p(u)), \\ LF_\Pi(Y_3) &= \forall u(r(u) \rightarrow \bot), \\ LF_\Pi(Y_4) &= \forall u(p(u) \wedge q(u) \rightarrow (q(u) \wedge u \neq u) \vee (p(u) \wedge u \neq u) \vee \neg r(u)). \end{aligned}$$

Example 4 *Let Π be the one-rule program*

$$p(x) \leftarrow p(y). \tag{3.6}$$

Its finite first-order loops are $Y_k = \{p(x_1), \ldots, p(x_k)\}$ where $k > 0$. Formula $LF_\Pi(Y_k)$ is

$$\forall x_1 \ldots x_k \Big(p(x_1) \wedge \ldots \wedge p(x_k) \rightarrow \exists y (p(y) \wedge (y \neq x_1) \wedge \ldots \wedge (y \neq x_k)) \Big). \tag{3.7}$$

The following is a reformulation of Theorem 1 from the work of Chen et al. (2006).

Theorem 1 *Let Π be a nondisjunctive program that has no function constants of positive arity, and let I be an Herbrand interpretation of $\sigma(\Pi)$ that satisfies Π.*[5] *The following conditions are equivalent to each other:*

(a) I is a stable model of Π;

(b) for every nonempty finite set Y of atoms of $\sigma(\Pi)$, I satisfies $LF_\Pi(Y)$;[6]

[5] We say that I satisfies Π if I satisfies the FOL-representation of Π.

[6] Note that Y may contain variables.

(c) for every finite first-order loop Y of Π, I satisfies $LF_\Pi(Y)$.

The sets of first-order loop formulas considered in conditions (b) and (c) above have obvious redundancies. For instance, the loop formula of $\{p(x)\}$ is equivalent to the loop formula of $\{p(y)\}$; the loop formula of $\{p(x), p(y)\}$ entails the loop formula of $\{p(z)\}$. Following the definition by Chen et al. (2006), given two sets of atoms Y_1 and Y_2, we say that Y_1 *subsumes* Y_2 if there is a substitution θ that maps variables in Y_1 to terms so that $Y_1\theta = Y_2$.

Proposition 1 *(Chen et al., 2006, Proposition 7) For any nondisjunctive program Π and any loops Y_1 and Y_2 of Π, if Y_1 subsumes Y_2, then $LF_\Pi(Y_1)$ entails $LF_\Pi(Y_2)$.*

Therefore in condition (c) from Theorem 1, it is sufficient to consider a set Γ of loops such that, for every loop L of Π, there is a loop L' in Γ that subsumes L. Chen et al. (2006) called such Γ a *complete* set of loops. In Example 3, set $\{Y_1, Y_2, Y_3, Y_4\}$ is a finite complete set of loops of program (3.5). Program (3.6) in Example 4 has no finite complete set of loops.

Loop Formula for Disjunctive Programs

A *disjunctive* program is a finite set of rules of the form

$$A \leftarrow B, N, \tag{3.8}$$

where A and B are sets of atoms, and N is a negative formula. Similar to a nondisjunctive program, we say that a disjunctive program is in *normal form* if, for all rules (3.8) in it, all atoms in A are of the form $p(\boldsymbol{x})$ where $\boldsymbol{x}$ is a list of distinct variables.

Let Π be a disjunctive program and let $\mathrm{Norm}(\Pi)$ be a normal form of Π. Given a finite set Y of atoms of $\sigma(\Pi)$, we first rename variables in $\mathrm{Norm}(\Pi)$ so that no variables in $\mathrm{Norm}(\Pi)$ occur in Y. The *(first-order) external support formula* of Y for Π, denoted by $\mathrm{ES}_\Pi(Y)$, is the disjunction of

$$\bigvee_{\theta: A\theta \cap Y \neq \emptyset} \exists \boldsymbol{z} \Big(B\theta \wedge N\theta \wedge \bigwedge_{\substack{p(\boldsymbol{t}) \in B\theta \\ p(\boldsymbol{t}') \in Y}} (\boldsymbol{t} \neq \boldsymbol{t}') \wedge \neg \big(\bigvee_{p(\boldsymbol{t}) \in A\theta} \big(p(\boldsymbol{t}) \wedge \bigwedge_{p(\boldsymbol{t}') \in Y} \boldsymbol{t} \neq \boldsymbol{t}' \big) \big) \Big) \tag{3.9}$$

for all rules (3.8) in $\mathrm{Norm}(\Pi)$, where θ is a substitution that maps variables in A to terms occurring in Y or to themselves, and $\boldsymbol{z}$ is the list of all variables that occur in

$$A\theta \leftarrow B\theta, N\theta$$

but not in Y. The *(first-order) loop formula* of Y for Π, denoted by $\mathrm{LF}_\Pi(Y)$, is the universal closure of

$$\bigwedge Y \rightarrow \mathrm{ES}_\Pi(Y).$$

Clearly, (3.9) is equivalent to (3.3) when Π is nondisjunctive. When Π and Y are propositional, $\mathrm{LF}_\Pi(Y)$ is equivalent to the conjunctive loop formula for a disjunctive program as defined by Ferraris et al. (2006).

Example 5 *Let Π be the program*

$$p(x,y)\ ;\ p(y,z)\ \leftarrow\ q(x)$$

and let $Y = \{p(u,v)\}$. Formula $LF_\Pi(Y)$ is the universal closure of

$$\begin{aligned} p(u,v) \rightarrow\ & \exists z(q(u) \wedge \neg(p(v,z) \wedge ((v,z) \neq (u,v)))) \\ & \vee \exists x(q(x) \wedge \neg(p(x,u) \wedge ((x,u) \neq (u,v)))). \end{aligned}$$

Similar to the nondisjunctive case, we say that $p(\boldsymbol{t})$ *depends on* $q(\boldsymbol{t}')$ in Π if there is a rule (3.8) in Π such that $p(\boldsymbol{t})$ is in A and $q(\boldsymbol{t}')$ is in B. The definitions of a first-order dependency graph and a first-order loop are extended to disjunctive programs in a straightforward way. Using these extended notions, the following theorem extends Theorem 1 to a disjunctive program. It is also a generalization of the main theorem by Ferraris et al. (2006) which was restricted to a propositional disjunctive program.

Theorem 1 [d] *Let Π be a disjunctive program that has no function constants of positive arity, and let I be an Herbrand interpretation of $\sigma(\Pi)$ that satisfies Π. The following conditions are equivalent to each other:*

(a) I is a stable model of Π;

(b) for every nonempty finite set Y *of atoms of* $\sigma(\Pi)$*,* I *satisfies* $LF_\Pi(Y)$*;*

(c) for every finite first-order loop Y *of* Π*,* I *satisfies* $LF_\Pi(Y)$*.*

Extension to Arbitrary Sentences

In this section we extend the definition of a first-order loop formula to an arbitrary first-order sentence.

As with a propositional loop formula defined for an arbitrary propositional theory (Ferraris et al., 2006), it is convenient to introduce a formula whose *negation* is close to ES. We define formula $\mathrm{NES}_F(Y)$ (*"Negation of (First-order) External Support Formula"*), where F is a first-order formula and Y is a finite set of atoms, as follows. As before we assume that no variables in Y occur in F, by renaming variables.

- $\mathrm{NES}_{p_i(\boldsymbol{t})}(Y) = p_i(\boldsymbol{t}) \wedge \bigwedge_{p_i(\boldsymbol{t}')\in Y} \boldsymbol{t} \neq \boldsymbol{t}'$;
- $\mathrm{NES}_{t_1=t_2}(Y) = (t_1 = t_2)$;
- $\mathrm{NES}_{\bot}(Y) = \bot$;
- $\mathrm{NES}_{F\wedge G}(Y) = \mathrm{NES}_F(Y) \wedge \mathrm{NES}_G(Y)$;
- $\mathrm{NES}_{F\vee G}(Y) = \mathrm{NES}_F(Y) \vee \mathrm{NES}_G(Y)$;
- $\mathrm{NES}_{F\to G}(Y) = (\mathrm{NES}_F(Y) \to \mathrm{NES}_G(Y)) \wedge (F \to G)$;
- $\mathrm{NES}_{\forall x G}(Y) = \forall x \mathrm{NES}_G(Y)$;
- $\mathrm{NES}_{\exists x G}(Y) = \exists x \mathrm{NES}_G(Y)$.

The *(first-order) loop formula* of Y for F, denoted by $\mathrm{LF}_F(Y)$, is the universal closure of

$$\bigwedge Y \to \neg \mathrm{NES}_F(Y). \tag{3.10}$$

Note that the definition of NES looks similar to the definition of F^* given in Section 2.5. When F and Y are propositional, $\mathrm{LF}_F(Y)$ is equivalent to a conjunctive loop

formula for a propositional formula that is defined by Ferraris et al. (2006). The following lemma tells us that the notion of a loop formula in this section generalizes the notion of a loop formula for a disjunctive program in the previous section.

Lemma 1 *Let Π be a disjunctive program in normal form, F an FOL-representation of Π, and Y a finite set of atoms. Formula $NES_F(Y)$ is equivalent to $\neg ES_\Pi(Y)$ under the assumption F.*

In order to extend the first-order dependency graph to an arbitrary formula, we introduce a few notions. We say that an occurrence of a subformula G in a formula F is *positive* if the number of implications in F containing that occurrence in the antecedent is even; it is *strictly positive* if that number is 0. A *rule* of a first-order formula F is an implication that occurs strictly positively in F. We will say that a formula is *rectified* if it has no variables that are both bound and free, and if all quantifiers in the formula refer to different variables. Any formula can be easily rewritten into a rectified formula by renaming bound variables.

We say that an atom $p(\boldsymbol{t})$ *depends on* an atom $q(\boldsymbol{t}')$ in an implication $G \to H$ if

- $p(\boldsymbol{t})$ has a strictly positive occurrence in H, and
- $q(\boldsymbol{t}')$ has a positive occurrence in G that does not belong to any negative subformula of G.[7]

The definition of a first-order dependency graph is extended to formulas as follows. The *(first-order) dependency graph* of a rectified formula F is the infinite directed graph (V, E) such that

- V is the set of atoms of signature $\sigma(F)$;
- $(p(\boldsymbol{t})\theta, q(\boldsymbol{t}')\theta)$ is in E if $p(\boldsymbol{t})$ depends on $q(\boldsymbol{t}')$ in a rule of F and θ is a substitution that maps variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$ to terms of $\sigma(F)$.

[7] Recall the definition of a negative formula in Section 2.2.

Note that the rectified formula assumption is needed in order to distinguish between dependency graphs of formulas such as

$$\forall x(p(x) \to q(x))$$

and

$$\forall x\, p(x) \to \forall x\, q(x).$$

Once the definition of a dependency graph is given, a loop of a first-order formula is defined in the same way as with a disjunctive program. Theorem 1 can be extended to first-order sentences using these extended notions.

Theorem 1 [f] *Let F be a rectified sentence that has no function constants of positive arity, and let I be an Herbrand interpretation of $\sigma(F)$ that satisfies F. The following conditions are equivalent to each other:*

(a) I is a stable model of F (i.e., I satisfies $SM[F]$);

(b) for every nonempty finite set Y of atoms of $\sigma(F)$, I satisfies $LF_F(Y)$;

(c) for every finite first-order loop Y of F, I satisfies $LF_F(Y)$.

Example 3 (continued) *Consider the FOL-representation F of the program in Example 3, for which $\{Y_1, Y_2, Y_3, Y_4\}$ is a complete set of loops. Under the assumption F,*

- *$LF_F(Y_1)$ is equivalent to the universal closure of*

$$p(u) \to \neg\Big(\forall x(q(x) \to p(x) \wedge x \neq u) \wedge \forall y(p(y) \wedge y \neq u \to q(y)) \\ \wedge \forall z(\neg r(z) \to p(z) \wedge z \neq u)\Big);$$

- *$LF_F(Y_2)$ is equivalent to the universal closure of*

$$q(u) \to \neg\Big(\forall x(q(x) \wedge x \neq u \to p(x)) \wedge \forall y(p(y) \to q(y) \wedge y \neq u)\Big);$$

- $LF_F(Y_3)$ *is equivalent to the universal closure of*

$$r(u) \to \bot;$$

- $LF_F(Y_4)$ *is equivalent to the universal closure of*

$$p(u) \wedge q(u) \to \neg\Big(\forall x(q(x) \wedge x \neq u \to p(x) \wedge x \neq u) \\ \wedge \forall y(p(y) \wedge y \neq u \to q(y) \wedge y \neq u) \wedge \forall z(\neg r(z) \to p(z) \wedge z \neq u)\Big).$$

Proposition 1 can be straightforwardly extended to arbitrary sentences even without restricting the attention to loops.

Proposition 1^f *For any sentence F and any nonempty finite sets of atoms Y_1 and Y_2 of $\sigma(F)$, if Y_1 subsumes Y_2, then $LF_F(Y_1)$ entails $LF_F(Y_2)$.*

Proof. Note that $\mathrm{LF}_F(Y_1)$ is

$$\forall \boldsymbol{z}\big(\bigwedge Y_1 \to \neg \mathrm{NES}_F(Y_1)\big), \tag{3.11}$$

where $\boldsymbol{z}$ is the set of all variables in Y_1. If Y_1 subsumes Y_2, by definition, there is a substitution θ from variables in Y_1 to terms in Y_2 such that $Y_1\theta = Y_2$. It is clear that (3.11) entails

$$\forall \boldsymbol{z}'\big(\bigwedge Y_1\theta \to \neg \mathrm{NES}_F(Y_1\theta)\big), \tag{3.12}$$

where $\boldsymbol{z}'$ is the set of all variables in $Y_1\theta$. (3.12) is exactly $\mathrm{LF}_F(Y_2)$. ∎

Theorem 2 from the work of Ferraris et al. (2006) is a special case of Theorem 1^f when F is restricted to a propositional formula.

Corollary 1 *(Ferraris et al., 2006, Theorem 2) For any propositional formula F, the following formulas are equivalent to each other under the assumption F.*

(a) $SM[F]$;

(b) the conjunction of $LF_F(Y)$ for all nonempty sets Y of atoms occurring in F;

(c) the conjunction of $LF_F(Y)$ for all (ground) loops Y of F.

3.2 Comparing First-Order Stable Model Semantics and First-Order Loop Formulas

The theorems in the previous section were restricted to Herbrand stable models. This section extends the results to allow non-Herbrand stable models as well, and compare the idea of loop formulas with SM by reformulating the latter in the style of loop formulas.

Loop Formulas Relative to an Interpretation

Recall that Theorem 1 and its extensions do not allow function constants of positive arity and are limited to Herbrand models of the particular signature obtained from the given theory. Indeed, the statements become wrong if these conditions are dropped.

Example 6 *The following program contains a unary function constant f.*

$$p(a)$$
$$p(x) \leftarrow p(f(x)).$$

The loops of this program are all singleton sets of atoms, and their loop formulas are satisfied by the Herbrand model $\{p(a), p(f(a)), p(f(f(a))), \dots\}$ of the program, but this model is not stable.

Example 4 (continued) *The mismatch can happen even in the absence of function constants of positive arity. Consider the program in Example 4 and an interpretation I such that the universe is the set of all integers, and p^I contains all integers. Interpretation I satisfies all first-order loop formulas (3.7), but it is not a stable model.*

These examples suggest that the mismatch between the first-order stable model semantics and the first-order loop formulas is related to the presence of infinite paths in the dependency graph that visits infinitely many vertices. In the following we will make this idea more precise, and extend Theorem 1^f to allow non-Herbrand interpretations under a certain condition.

First, we define a dependency graph *relative to an interpretation*. Let F be a rectified formula whose signature is σ and let I be an interpretation of σ. For each element ξ of the

universe $|I|$ of I, we introduce a new symbol $\xi^\diamond$, called an *object name*. By σ^I we denote the signature obtained from σ by adding all object names $\xi^\diamond$ as additional object constants. We will identify an interpretation I of signature σ with its extension to σ^I defined by $I(\xi^\diamond) = \xi$ (For details, see the work of Lifschitz, Morgenstern, and Plaisted, (2008)).

The *dependency graph of F w.r.t. I* is the directed graph (V, E) where

- V is the set of all atoms of the form $p_i(\boldsymbol{\xi}^\diamond)$ where p_i belongs to $\sigma(F)$ and $\boldsymbol{\xi}^\diamond$ is a list of object names for $|I|$, and
- $(p_i(\boldsymbol{\xi}^\diamond), p_j(\boldsymbol{\eta}^\diamond))$ is in E if there are atoms $p_i(\boldsymbol{t})$, $p_j(\boldsymbol{t}')$ such that $p_i(\boldsymbol{t})$ depends on $p_j(\boldsymbol{t}')$ in a rule of F and there is a substitution θ that maps variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$ to object names such that $(\boldsymbol{t}\theta)^I = \boldsymbol{\xi}$ and $(\boldsymbol{t}'\theta)^I = \boldsymbol{\eta}$.

We call a nonempty subset L of V a *loop of F w.r.t. I* if the subgraph of the dependency graph of F w.r.t. I that is induced by L is strongly connected. We say that F is *bounded w.r.t. I* if every infinite path in the dependency graph of F w.r.t. I whose vertices are satisfied by I visits only finitely many vertices. If F is bounded w.r.t. I, then, clearly, every loop L of F w.r.t. I such that $I \models L$ is finite. The definition is extended to a non-rectified formula by first rewriting it as a rectified formula. It also applies to the program syntax by referring to its FOL-representation.

Theorem 2 *Let F be a rectified sentence of signature σ (possibly containing function constants of positive arity), and let I be an interpretation of σ that satisfies F. If F is bounded w.r.t. I, then the following conditions are equivalent to each other:*

(a) $I \models SM[F]$;

(b) for every nonempty finite set Y of atoms formed from predicate constants in $\sigma(F)$ and object names for $|I|$, I satisfies $LF_F(Y)$;

(c) for every finite loop Y of F w.r.t. I, I satisfies $LF_F(Y)$.

The condition that F is bounded w.r.t. I is sufficient for ensuring the equivalence among (a), (b), and (c), but it is not a necessary condition. For instance, consider F to be

$$\forall x p(x) \wedge \forall xy(p(x) \to p(y))$$

and I to be a model of F whose universe is infinite. Formula F is not bounded w.r.t. I, but I satisfies every loop formula, as well as $\mathrm{SM}[F]$.

When I is an Herbrand model of $\sigma(F)$, the dependency graph of F w.r.t. I is isomorphic to the subgraph of the first-order dependency graph of F that is induced by vertices containing ground atoms. A set of ground atoms of $\sigma(F)$ is a loop of F iff it is a loop of F w.r.t. I. Hence Theorem 2 is essentially a generalization of Theorem 1^f.

Note that the programs considered in Examples 4 and 6 are not bounded w.r.t. the interpretations considered there.

Clearly, if the universe of I is finite, then F is bounded w.r.t. I. This fact leads to the following corollary.

Corollary 2 *For any rectified sentence F and any model I of F whose universe is finite, conditions (a), (b), and (c) of Theorem 2 are equivalent to each other.*

In view of Proposition 1^f and Corollary 2, if the size of the universe is known to be a finite number n, it is sufficient to consider at most $2^{|\boldsymbol{p}|} - 1$ loop formulas, where $\boldsymbol{p}$ is the set of all predicate constants occurring in the sentence. Each loop formula is to check the external support of $\bigcup_{p \in K}\{p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_{n^r})\}$ for each K where

- K is a nonempty subset of $\boldsymbol{p}$;
- r is the arity of p and each $\boldsymbol{x}_i$ is a list of variables of the length r such that all variables in $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n^r}$ are pairwise distinct.

For instance, consider program (3.6). If the size of the universe is known to be 3, it is sufficient to consider only one loop formula (3.7) where $k = 3$.

Theorem 1^f essentially follows from Corollary 2 as the Herbrand universe of $\sigma(F)$ is finite when F contains no function constants of positive arity.

Another corollary to Theorem 2 is acquired when F has only "trivial" loops. We say that a formula F is *atomic-tight* w.r.t. I if every path in the dependency graph of F w.r.t. I whose vertices are satisfied by I is finite. Clearly, this is a special case of boundedness condition, and every loop L of an atomic-tight formula F w.r.t. I such that $I \models L$ is a singleton. The following is a corollary to Theorem 2, which tells us the condition under which stable models can be characterized by loop formulas of singleton loops only. By $\mathrm{SLF}[F]$ ("loop formulas of singletons") we denote the set of loop formulas

$$\{\mathrm{LF}_F(\{p(\boldsymbol{x})\}) \mid p \text{ is a predicate constant in } \sigma(F), \text{ and } \boldsymbol{x} \text{ is a list of distinct object variables whose length is the same as the arity of } p\}. \tag{3.13}$$

Corollary 3 *Let F be a rectified sentence (possibly containing function constants of positive arity), and let I be a model of F. If F is atomic-tight w.r.t. I, then I satisfies $SM[F]$ iff I satisfies $SLF[F]$.*

$\mathrm{SLF}[F]$ is similar to Clark's completion. In the propositional case, the relationship between the loop formulas of singletons and completion formulas is studied by Lee (2005). Below we describe their relationship in the first-order case. A sentence is in *Clark normal form* if it is a conjunction of formulas of the form

$$\forall \boldsymbol{x}(G \to p(\boldsymbol{x})), \tag{3.14}$$

one for each predicate constant p occurring in F, where $\boldsymbol{x}$ is a list of distinct variables, and G has no free variables other than $\boldsymbol{x}$. The *completion* of a sentence F in Clark normal form, denoted by $\mathrm{Comp}[F]$, is obtained from it by replacing each conjunctive term (5.19) with

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

Any nondisjunctive program can be turned into Clark normal form (Ferraris et al., 2011a, Section 6.1).

Corollary 4 *Let F be the FOL-representation of a nondisjunctive program Π, and let F' be the Clark normal form of F as obtained by the process described in the work of (Ferraris et al., 2011a, Section 6.1). If F is atomic-tight w.r.t. an interpretation I, then $I \models SM[F]$ iff $I \models Comp[F']$.*

Proof. Since F is atomic-tight w.r.t. I, by Corollary 3, $I \models \mathrm{SM}[F]$ iff $I \models \mathrm{SLF}[F]$. It is sufficient to show that, for each predicate constant p occurring in F, under the assumption that F is atomic-tight w.r.t. I,

$$I \models \forall \boldsymbol{x}\Big(p(\boldsymbol{x}) \rightarrow \bigvee_{p(\boldsymbol{t}')\leftarrow B, N\in\Pi} \exists \boldsymbol{z}\big(B \wedge N \wedge (\boldsymbol{x} = \boldsymbol{t}') \wedge \bigwedge_{p(\boldsymbol{t})\in B} (\boldsymbol{t} \neq \boldsymbol{x})\big)\Big) \tag{3.15}$$

iff

$$I \models \forall \boldsymbol{x}\Big(p(\boldsymbol{x}) \rightarrow \bigvee_{p(\boldsymbol{t}')\leftarrow B, N\in\Pi} \exists \boldsymbol{z}\big(B \wedge N \wedge (\boldsymbol{x} = \boldsymbol{t}')\big)\Big), \tag{3.16}$$

where $\boldsymbol{z}$ is the list of all variables in $p(\boldsymbol{t}') \leftarrow B, N$ in that are not in $\boldsymbol{x}$.

Note that (3.15) is equivalent to saying that

$$I \models \forall \boldsymbol{x}\Big(p(\boldsymbol{x}) \rightarrow \bigvee_{p(\boldsymbol{t}')\leftarrow B, N\in\Pi} \exists \boldsymbol{z}\big(B \wedge N \wedge (\boldsymbol{x} = \boldsymbol{t}') \wedge \bigwedge_{p(\boldsymbol{t})\in B} (\boldsymbol{t} \neq \boldsymbol{t}')\big)\Big). \tag{3.17}$$

From the assumption that F is atomic-tight w.r.t. I, it follows that, for any rule $p(\boldsymbol{t}') \leftarrow B, N$ in Π and any atom of $p(\boldsymbol{t})$ in B, $I \models \forall \boldsymbol{y}(\boldsymbol{t} \neq \boldsymbol{t}')$, where $\boldsymbol{y}$ is the list of all variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$ (otherwise we find a singleton loop with a self-cycle, which contradicts that F is atomic tight w.r.t. I). Consequently, (3.17) is equivalent to (3.16). ∎

For example, let F be the FOL-representation of the program

$$p(b) \leftarrow p(a). \tag{3.18}$$

$\mathrm{SLF}[F]$ is $\forall x(p(x) \rightarrow x = b \wedge p(a) \wedge x \neq a)$, while $\mathrm{Comp}[F]$ is $\forall x(p(x) \leftrightarrow x = b \wedge p(a))$. The additional conjunctive term $x \neq a$ can be dropped when we consider any interpretation I such that $a^I \neq b^I$ (Such requirement can be syntactically stated if we add a constraint $\leftarrow a = b$ to the program). The proof of this fact is given in Section 3.5.

Corollary 4 generalizes Theorem 11 from the work of Ferraris et al. (2011a) by referring to atomic-tightness in place of tightness. The program (3.18) is not tight, but is atomic-tight w.r.t. any model of the program.

Theorem 2 tells us that one of the limitations of first-order loop formulas is that, even if infinitely many first-order loop formulas are considered, they cannot ensure the external support of a certain infinite set that forms an infinite path in the dependency graph of F w.r.t. I. In the next section, by reformulating $\mathrm{SM}[F]$, we show that the definition of $\mathrm{SM}[F]$ essentially encompasses loop formulas, ensuring the external support of any sets of atoms, including those "difficult" infinite sets.

A Reformulation of SM

As before, let F be a first-order formula of signature σ, let $\boldsymbol{p} = (p_1, \ldots, p_n)$ be the list of all predicate constants occurring in F, and let $\boldsymbol{u}$ and $\boldsymbol{v}$ be lists of predicate variables of the same length as $\boldsymbol{p}$. We define $\mathrm{NSES}_F(\boldsymbol{u})$ ("*Negation of Second-Order External Support Formula*") recursively as follows.

- $\mathrm{NSES}_{p_i(\boldsymbol{t})}(\boldsymbol{u}) = p_i(\boldsymbol{t}) \wedge \neg u_i(\boldsymbol{t})$;
- $\mathrm{NSES}_{t_1=t_2}(\boldsymbol{u}) = (t_1 = t_2)$;
- $\mathrm{NSES}_{\bot}(\boldsymbol{u}) = \bot$;
- $\mathrm{NSES}_{F\wedge G}(\boldsymbol{u}) = \mathrm{NSES}_F(\boldsymbol{u}) \wedge \mathrm{NSES}_G(\boldsymbol{u})$;
- $\mathrm{NSES}_{F\vee G}(\boldsymbol{u}) = \mathrm{NSES}_F(\boldsymbol{u}) \vee \mathrm{NSES}_G(\boldsymbol{u})$;
- $\mathrm{NSES}_{F\to G}(\boldsymbol{u}) = (\mathrm{NSES}_F(\boldsymbol{u}) \to \mathrm{NSES}_G(\boldsymbol{u})) \wedge (F \to G)$;
- $\mathrm{NSES}_{\forall xF}(\boldsymbol{u}) = \forall x \mathrm{NSES}_F(\boldsymbol{u})$;
- $\mathrm{NSES}_{\exists xF}(\boldsymbol{u}) = \exists x \mathrm{NSES}_F(\boldsymbol{u})$.

Lemma 2 *Let F be a rectified sentence of signature σ, I an interpretation of σ, $\boldsymbol{p}$ the list of predicate constants occurring in F, $\boldsymbol{q}$ a list of predicate names* [8] *of the same length as $\boldsymbol{p}$ and Y a set of atoms formed from predicate constants from $\sigma(F)$ and object names such that*

$$p_i(\boldsymbol{\xi}^{\diamond}) \in Y \text{ iff } I \models q_i(\boldsymbol{\xi}^{\diamond}),$$

[8] Like object names, for every $n > 0$, each subset of $|I|^n$ has a name, which is an n-ary predicate constant not from the underlying signature.

where $\boldsymbol{\xi}^\diamond$ is a list of object names. If Y is finite, then

$$I \models NSES_F(\boldsymbol{q}) \;\; \textit{iff} \;\; I \models NES_F(Y).$$

Proof. By induction on F. We only list the case when F is an atom. The other cases are straightforward. Let F be an atom $p_i(\boldsymbol{\xi}^\diamond)$.

$$\begin{array}{ll} & I \models \mathrm{NSES}_F(\boldsymbol{q}) \\ \text{iff} & I \models p_i(\boldsymbol{\xi}^\diamond) \wedge \neg q_i(\boldsymbol{\xi}^\diamond) \\ \text{iff} & I \models p_i(\boldsymbol{\xi}^\diamond) \text{ and } p_i(\boldsymbol{\xi}^\diamond) \notin Y \\ \text{iff} & I \models p_i(\boldsymbol{\xi}^\diamond) \text{ and for all } \boldsymbol{\eta}^\diamond \text{ such that } p_i(\boldsymbol{\eta}^\diamond) \in Y, \text{ it holds that } \boldsymbol{\xi}^\diamond \neq \boldsymbol{\eta}^\diamond \\ \text{iff} & I \models p_i(\boldsymbol{\xi}^\diamond) \wedge \bigwedge_{p_i(\boldsymbol{\eta}^\diamond) \in Y} \boldsymbol{\xi}^\diamond \neq \boldsymbol{\eta}^\diamond \\ \text{iff} & I \models \mathrm{NES}_F(Y). \end{array}$$

∎

$\mathrm{SM}[F]$ can be written in terms of NSES as follows. By $\mathrm{Nonempty}(\boldsymbol{u})$ we denote the formula

$$\exists \boldsymbol{x}^1 u_1(\boldsymbol{x}^1) \vee \cdots \vee \exists \boldsymbol{x}^n u_n(\boldsymbol{x}^n),$$

where each $\boldsymbol{x}^i$ is a list of distinct variables whose length is the same as the arity of p_i.

Proposition 2 *For any sentence F, $SM[F]$ is equivalent to*

$$F \wedge \forall \boldsymbol{u}((\boldsymbol{u} \leq \boldsymbol{p}) \wedge Nonempty(\boldsymbol{u}) \rightarrow \neg NSES_F(\boldsymbol{u})). \tag{3.19}$$

Now we represent the notion of a loop by a second-order formula. Given a rectified formula F, by $E_F(\boldsymbol{v}, \boldsymbol{u})$ we denote

$$\bigvee_{\substack{(p_i(\boldsymbol{t}), p_j(\boldsymbol{t}')) : \\ p_i(\boldsymbol{t}) \text{ depends on } p_j(\boldsymbol{t}') \text{ in a rule of } F}} \exists \boldsymbol{z}(v_i(\boldsymbol{t}) \wedge u_j(\boldsymbol{t}') \wedge \neg v_j(\boldsymbol{t}')),$$

where $\boldsymbol{z}$ is the list of all object variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$. By $\mathrm{Loop}_F(\boldsymbol{u})$ we denote the second-order formula

$$\mathrm{Nonempty}(\boldsymbol{u}) \wedge \forall \boldsymbol{v}((\boldsymbol{v} < \boldsymbol{u}) \wedge \mathrm{Nonempty}(\boldsymbol{v}) \rightarrow E_F(\boldsymbol{v}, \boldsymbol{u})). \tag{3.20}$$

Formula (3.20) represents the concept of a loop without referring to the notion of a dependency graph explicitly. This is based on the following observation. Consider a finite propositional program Π. A nonempty set U of atoms that occur in Π is a loop of Π iff, for every nonempty proper subset V of U, there is an edge from an atom in V to an atom in $U \setminus V$ in the dependency graph of Π (Gebser et al., 2006).

Recall the definition of a dependency graph relative to an interpretation. Let F be a rectified sentence of signature σ, and let I be an interpretation of σ. The following proposition describes the relationship between formula (3.20) and a loop of F w.r.t. I.

Proposition 3 *Let $\boldsymbol{q}$ be a list of predicate names corresponding to $\boldsymbol{p}$, and let Y be a set of atoms in the dependency graph of F w.r.t. I such that*

$$p_i(\boldsymbol{\xi}^\diamond) \in Y \text{ iff } I \models q_i(\boldsymbol{\xi}^\diamond),$$

where $\boldsymbol{\xi}^\diamond$ is a list of object names. Then $I \models Loop_F(\boldsymbol{q})$ iff Y is a loop of F w.r.t. I.

One might expect that, similar to the equivalence between conditions (a) and (c) from Theorem 2, formula $\mathrm{SM}[F]$ is equivalent to the following formula:

$$F \wedge \forall \boldsymbol{u}((\boldsymbol{u} \leq \boldsymbol{p}) \wedge \mathrm{Loop}_F(\boldsymbol{u}) \rightarrow \neg \mathrm{NSES}_F(\boldsymbol{u})). \tag{3.21}$$

However, the equivalence does not hold in general, as the following example illustrates.

Example 7 *Consider the FOL-representation F of the following program*

$$p(x, y) \leftarrow q(x, z)$$
$$q(x, z) \leftarrow p(y, z),$$

and an interpretation I whose universe is the set of all nonnegative integers such that

$$p^I = \{(m, m) \mid m \text{ is a nonnegative integer}\},$$
$$q^I = \{(m, m+1) \mid m \text{ is a nonnegative integer}\}.$$

Formula F is not bounded w.r.t. I since the dependency graph of F w.r.t. I contains an infinite path such as

$$\langle p(0^\diamond, 0^\diamond), q(0^\diamond, 1^\diamond), p(1^\diamond, 1^\diamond), q(1^\diamond, 2^\diamond), \ldots \rangle. \tag{3.22}$$

The interpretation I satisfies every loop formula of every finite loop of F w.r.t. I, but it is not a stable model.

In the example, what distinguishes the set

$$\{p(0^\diamond, 0^\diamond), q(0^\diamond, 1^\diamond), p(1^\diamond, 1^\diamond), q(1^\diamond, 2^\diamond), \dots\} \tag{3.23}$$

from a loop is that, for every loop contained in (3.23), there is an outgoing edge in the dependency graph. This is an instance of what we call "unbounded set." Given a dependency graph of F w.r.t. I, we say that a nonempty set Y of vertices is *unbounded* w.r.t. I if, for every subset Z of Y that is a loop, there is an edge from a vertex in Z to a vertex in $Y \setminus Z$.

The following proposition tells us how an unbounded set can be characterized by a second-order formula.

Proposition 4 *Let $\boldsymbol{q}$ be a list of predicate names corresponding to $\boldsymbol{p}$, and let Y be a set of atoms in the dependency graph of F w.r.t. I such that*

$$p_i(\boldsymbol{\xi}^\diamond) \in Y \text{ iff } I \models q_i(\boldsymbol{\xi}^\diamond),$$

where $\boldsymbol{\xi}^\diamond$ is a list of object names. Then

$$I \models \mathit{Nonempty}(\boldsymbol{q}) \wedge \forall \boldsymbol{v}((\boldsymbol{v} \leq \boldsymbol{q}) \wedge \mathit{Loop}_F(\boldsymbol{v}) \rightarrow E_F(\boldsymbol{v}, \boldsymbol{q}))$$

iff Y is an unbounded set of F w.r.t. I.

In order to check the stability of a model, we need to check the external support of every loop and every unbounded set. An *extended loop* of F w.r.t. I is a loop or an unbounded set of F w.r.t. I. We define $\mathrm{Ext\text{-}Loop}_F(\boldsymbol{u})$ as

$$\mathrm{Loop}_F(\boldsymbol{u}) \vee (\mathrm{Nonempty}(\boldsymbol{u}) \wedge\ \forall \boldsymbol{v}((\boldsymbol{v} \leq \boldsymbol{u}) \wedge \mathrm{Loop}_F(\boldsymbol{v}) \rightarrow E_F(\boldsymbol{v}, \boldsymbol{u}))). \tag{3.24}$$

From Propositions 3 and 4, it follows that $I \models \mathrm{Ext\text{-}Loop}_F(\boldsymbol{q})$ iff Y is an extended loop of F w.r.t. I.

If we replace $\mathrm{Loop}_F(\boldsymbol{u})$ with $\mathrm{Ext\text{-}Loop}_F(\boldsymbol{u})$ in (3.21), the formula is equivalent to $\mathrm{SM}[F]$, as the following theorem states.

Theorem 3 *For any rectified sentence F, the following sentences are equivalent to each other:*

(a) $SM[F]$;

(b) $F \wedge \forall \boldsymbol{u}((\boldsymbol{u} \leq \boldsymbol{p}) \wedge Nonempty(\boldsymbol{u}) \rightarrow \neg NSES_F(\boldsymbol{u}))$;

(c) $F \wedge \forall \boldsymbol{u}((\boldsymbol{u} \leq \boldsymbol{p}) \wedge Ext\text{-}Loop_F(\boldsymbol{u}) \rightarrow \neg NSES_F(\boldsymbol{u}))$.

In the following example we use the following fact to simplify the formulas.

Proposition 5 *For any negative formula F, formula*

$$NSES_F(\boldsymbol{u}) \leftrightarrow F$$

is logically valid.

Example 2 (continued) Consider program (3.5) from Example 3:

$$p(x) \leftarrow q(x)$$
$$q(y) \leftarrow p(y)$$
$$p(z) \leftarrow \text{not } r(z).$$

Let F be the FOL-representation of the program:

$$\forall x\big(q(x) \rightarrow p(x)\big) \wedge \forall y\big(p(y) \rightarrow q(y)\big) \wedge \forall z\big(\neg r(z) \rightarrow p(z)\big).$$

1. $\mathbf{SM}[F]$ is equivalent to

$$F \wedge \neg\exists u_1u_2u_3((u_1, u_2, u_3) < (p, q, r)) \wedge$$
$$\forall x(u_2(x) \rightarrow u_1(x)) \wedge \forall y(u_1(y) \rightarrow u_2(y)) \wedge \forall z(\neg r(z) \rightarrow u_1(z))).$$

2. Formula in Theorem 3 (b):

$$F \wedge \forall \boldsymbol{u}(\boldsymbol{u} \leq \boldsymbol{p} \wedge \text{Nonempty}(\boldsymbol{u}) \rightarrow \neg \text{NSES}_F(\boldsymbol{u}))$$

is equivalent to

$$\begin{aligned} F \wedge \forall u_1 u_2 u_3((u_1, u_2, u_3) \leq (p, q, r) \wedge (\exists x\, u_1(x) \vee \exists x\, u_2(x) \vee \exists x\, u_3(x)) \\ \rightarrow \neg(\forall x[q(x) \wedge \neg u_2(x) \rightarrow p(x) \wedge \neg u_1(x)] \\ \wedge \forall y[p(y) \wedge \neg u_1(y) \rightarrow q(y) \wedge \neg u_2(y)] \\ \wedge \forall z[\neg r(z) \rightarrow p(z) \wedge \neg u_1(z)])). \end{aligned} \tag{3.25}$$

3. Formula in Theorem 3 (c): Similar to (3.25) except that

$$\exists x\, u_1(x) \vee \exists x\, u_2(x) \vee \exists x\, u_3(x)$$

in (3.25) is replaced with $\mathrm{Ext\text{-}Loop}_F(\boldsymbol{u})$, which is

$$\begin{aligned} &\mathrm{Loop}_F(\boldsymbol{u}) \vee [(\exists x\, u_1(x) \vee \exists x\, u_2(x) \vee \exists x\, u_3(x)) \\ &\wedge \forall v_1 v_2 v_3(((v_1, v_2, v_3) \leq (u_1, u_2, u_3)) \wedge \mathrm{Loop}_F(\boldsymbol{v}) \\ &\qquad \rightarrow (\exists x(v_1(x) \wedge u_2(x) \wedge \neg v_2(x)) \vee \exists y(v_2(y) \wedge u_1(y) \wedge \neg v_1(y))))], \end{aligned}$$

where $\mathrm{Loop}_F(\boldsymbol{u})$ is

$$\begin{aligned} &(\exists x\, u_1(x) \vee \exists x\, u_2(x) \vee \exists x\, u_3(x)) \\ &\wedge \forall v_1 v_2 v_3(((\exists x\, v_1(x) \vee \exists x\, v_2(x) \vee \exists x\, v_3(x)) \wedge (v_1, v_2, v_3) < (u_1, u_2, u_3)) \\ &\quad \rightarrow (\exists x(v_1(x) \wedge u_2(x) \wedge \neg v_2(x)) \vee \exists y(v_2(y) \wedge u_1(y) \wedge \neg v_1(y)))). \end{aligned}$$

The proof of Theorem 2 follows from Theorem 3 using the following lemma.

Lemma 3 *Let F be a rectified sentence of signature σ (possibly containing function constants of positive arity), and let I be an interpretation of σ that satisfies F. If F is bounded w.r.t. I,*

$$I \models \exists \boldsymbol{u}(\boldsymbol{u} \leq \boldsymbol{p} \wedge \mathit{Ext\text{-}Loop}_F(\boldsymbol{u}) \wedge \mathit{NSES}_F(\boldsymbol{u}))$$

iff there is a finite loop Y of F w.r.t. I such that

$$I \models \Big(\bigwedge Y \wedge \mathit{NES}_F(Y) \Big).$$

3.3 Representing First-Order Stable Model Semantics by First-Order Loop Formulas

We noted in the previous section that if a sentence is bounded w.r.t. a model, then loop formulas can be used to check the stability of the model. In this section, we provide a few syntactic counterparts of the boundedness condition.

Bounded Formulas

We say that a rectified formula F is *bounded* if every infinite path in the first-order dependency graph of F visits only finitely many vertices. If F is bounded, then, clearly, every loop of F is finite. Again, the definition is extended to a non-rectified formula by first rewriting it as a rectified formula. It also applies to a program by referring to its FOL-representation.

One might wonder if the syntactic notion of boundedness ensures the semantic notion of boundedness: that is, if a formula is bounded, then it is bounded w.r.t. any interpretation. However, the following example tells us that this is not the case in general.

Example 8 *Consider the FOL-representation F of the following program*

$$\begin{aligned} p(a) &\leftarrow q(x) \\ q(x) &\leftarrow p(b), \end{aligned} \tag{3.26}$$

and an interpretation I whose universe $|I|$ is the set of all nonnegative integers, $a^I = b^I = 0$, $p^I = \{0\}$ and $q^I = |I|$. Formula (3.26) is bounded according to the above definition, but not bounded w.r.t. I: the dependency graph of F w.r.t. I contains an infinite path such as

$$\langle p(0^\diamond), q(1^\diamond), p(0^\diamond), q(2^\diamond), \dots \rangle.$$

Bounded Formulas and Clark's Equational Theory

On the other hand, such a relationship holds if the interpretation satisfies Clark's equational theory (1978). Clark's equational theory of a signature σ, denoted by CET_σ, is the union of the universal closures of the following formulas

$$f(x_1, \dots, x_m) \neq g(y_1, \dots, y_n), \tag{3.27}$$

for all pairs of distinct function constants f, g,

$$f(x_1, \dots, x_n) = f(y_1, \dots, y_n) \rightarrow (x_1 = y_1 \wedge \dots \wedge x_n = y_n), \tag{3.28}$$

for all function constants f of arity > 0, and

$$t \neq x, \tag{3.29}$$

where t is any term which contains the variable x.

Proposition 6 *If a rectified formula F of signature σ is bounded, then F is bounded w.r.t. any interpretation of σ that satisfies CET_σ.*

The following lemma relates loops and loop formulas of different notions of dependency graphs.

Proposition 7 *For any rectified sentence F of signature σ and for any interpretation I of σ that satisfies CET_σ, I is a model of*

$$\{LF_F(Y) \mid Y \text{ is a finite first-order loop of } F\}$$

iff I is a model of

$$\{LF_F(Y) \mid Y \text{ is a finite loop of } F \text{ w.r.t. } I\}.$$

The following theorem follows from Theorem 2, Proposition 6 and Proposition 7.

Theorem 4 *Let F be a rectified sentence of signature σ (possibly containing function constants of positive arity), and let I be an interpretation of σ that satisfies F and CET_σ. If F is bounded, then the following conditions are equivalent to each other:*

(a) $I \models SM[F]$;

(b) for every nonempty finite set Y of atoms of $\sigma(F)$, I satisfies $LF_F(Y)$;

(c) for every finite first-order loop Y of F, I satisfies $LF_F(Y)$.

Proof. By Proposition 6, if F is bounded then F is bounded w.r.t. any interpretation that satisfies CET_σ. Then the equivalence between (a) and (b) follows from the equivalence between (a) and (b) of Theorem 2. The equivalence between (a) and (c) follows from the equivalence between (a) and (c) of Theorem 2 and by Proposition 7. ∎

As every Herbrand interpretation of σ satisfies CET_σ, Theorem 4 applies to Herbrand interpretations as a special case.

The theorem also applies to logic programs, since they can be viewed as a special case of formulas. For example, consider the following program, which is bounded.

$$\begin{aligned} &p(f(x)) \leftarrow q(x) \\ &q(x) \leftarrow p(x), r(x) \\ &p(a) \\ &r(a) \\ &r(f(a)). \end{aligned} \tag{3.30}$$

The set $\{p(a), p(f(a)), p(f(f(a))), q(a), q(f(a)), r(a), r(f(a))\}$ is an answer set of (3.30). In accordance with Theorem 4, it is also the Herbrand interpretation of the signature obtained from the program that satisfies the FOL-representation of (3.30) and the loop formulas, which are the universal closures of

$$\begin{aligned} &p(z) \rightarrow (q(x) \wedge z = f(x)) \vee z = a \\ &q(z) \rightarrow p(z) \wedge r(z) \\ &r(z) \rightarrow z = a \vee z = f(a). \end{aligned}$$

Consider another example program by Bonatti (2004), where $a, \ldots, z, nil$ are object constants.

$$\begin{aligned} &\mathrm{letter}(a) \\ &\ldots \\ &\mathrm{letter}(z) \\ &\mathrm{atomic}([x]) \leftarrow \mathrm{letter}(x) \\ &\mathrm{atomic}([x|y]) \leftarrow \mathrm{letter}(x), \mathrm{atomic}(y). \end{aligned} \tag{3.31}$$

The expression $[x|y]$ is a list whose head is x and whose tail is y, which stands for a function $\mathrm{cons}(x, y)$. The expression $[x]$ stands for $\mathrm{cons}(x, nil)$ where nil is a special symbol for the empty list. This program is bounded. The only answer set of the program is the only Herbrand interpretation of the FOL-representation of (3.31) and the universal closures of

$$\begin{aligned} &\mathrm{letter}(u) \rightarrow u = a \vee \ldots \vee u = z \\ &\mathrm{atomic}(u) \rightarrow \exists v\ (\mathrm{letter}(v) \wedge u = \mathrm{cons}(v, nil)) \\ &\qquad\qquad \vee \exists xy\ (\mathrm{letter}(x) \wedge \mathrm{atomic}(y) \wedge y \neq u \wedge u = \mathrm{cons}(x, y)). \end{aligned}$$

In fact, the definitions of standard list processing predicates, such as $member$, $append$, and $reverse$ (Bonatti, 2004, Figure 1) are bounded, so they can be represented by first-order formulas on Herbrand interpretations.[9]

We say that a formula F is *atomic-tight* if the first-order dependency graph of F has no infinite paths. Similar to Proposition 6, if F is atomic-tight, then F is atomic-tight w.r.t. any interpretation that satisfies CET_σ, so that the following statement follows from Corollary 3.

Corollary 5 *Let F be a rectified sentence of signature σ (possibly containing function constants of positive arity), and let I be an interpretation of σ that satisfies CET_σ. If F is atomic-tight, then I satisfies $SM[F]$ iff I satisfies $SLF[F]$.*

Corollary 5 is an enhancement of Theorem 11 from the work of Ferraris et al. (2011a), which essentially says that, for any tight sentence F, $\mathrm{SM}[F]$ is equivalent to the set of formulas in (3.13). (Tight sentences are defined in a similar way (Ferraris et al., 2011a), but in terms of a predicate dependency graph, whose vertices are predicate constants instead of atoms.) Note that every tight sentence is atomic-tight, but the converse is not true. For example, the FOL-representations of programs (3.30) and (3.31) are atomic-tight, but are not tight.

The statement of Corollary 5 is restricted to interpretations that satisfy CET_σ. Indeed, the statement becomes wrong if this restriction is dropped. For example, program (3.26) in Example 8 is atomic-tight, but the non-stable model considered there satisfies all loop formulas, including those of singleton loops.

Bounded Formulas and Normal Form

Normal form is another syntactic condition that the syntactic notion of boundedness ensures the semantic notion of boundedness. We say that a formula is in *normal form* if every strictly positive occurrence of an atom is of the form $p(\boldsymbol{x})$, where $\boldsymbol{x}$ is a list of distinct variables. It is clear that every formula can be turned into normal form using equality.

[9] They actually satisfy a stronger condition called "finitely recursive" (Bonatti, 2004). See Section 3.6 for more details.

Proposition 8 *If a rectified formula F in normal form is bounded, then F is bounded w.r.t. any interpretation.*

Proposition 9 *If a rectified sentence F in normal form is bounded, then for any interpretation I, I is a model of*

$$\{LF_F(Y) \mid Y \text{ is a finite first-order loop of } F\}$$

iff I is a model of

$$\{LF_F(Y) \mid Y \text{ is a finite loop of } F \text{ w.r.t. } I\}.$$

The following theorem follows from Theorem 2, Proposition 8 and Proposition 9.

Theorem 5 *Let F be a rectified sentence in normal form (possibly containing function constants of positive arity). If F is bounded, then the following formulas are equivalent to each other:*

(a) $SM[F]$*;*

(b) $\{F\} \cup \{LF_F(Y) \mid Y$ *is a nonempty finite set of atoms of* $\sigma(F)\}$*;*

(c) $\{F\} \cup \{LF_F(Y) \mid Y$ *is a finite first-order loop of* $F\}$*.*

Proof. By Proposition 8, if F is bounded then F is bounded w.r.t. any interpretation I. Then the equivalence between (a) and (b) follows from the equivalence between (a) and (b) of Theorem 2. The equivalence between (a) and (c) follows from the equivalence between (a) and (c) of Theorem 2 and by Proposition 9. ∎

Consider a program in normal form

$$\begin{aligned} &p(x) \leftarrow x = a,\ q(a) \\ &q(y) \leftarrow p(b) \end{aligned} \tag{3.32}$$

and an interpretation I such that $|I| = \{1\}$, $a^I = b^I = 1$ and $p^I = q^I = \{1\}$. This interpretation does not satisfy Clark's equational theory, and is not a stable model. In

accordance with Theorem 5, I does not satisfy the loop formula of the loop $\{p(b), q(a)\}$, which is

$$p(b) \wedge q(a) \rightarrow (b = a \wedge q(a) \wedge a \neq a) \vee (p(b) \wedge b \neq b).$$

On the other hand, consider another program in *non-normal* form that has the same stable models as (3.32):

$$\begin{aligned} p(a) &\leftarrow q(a) \\ q(y) &\leftarrow p(b) \end{aligned} \tag{3.33}$$

Program (3.33) has a finite complete set of loops, $\{\{p(z)\}, \{q(z)\}\}$; their loop formulas are the universal closures of

$$\begin{aligned} p(z) &\rightarrow z = a \wedge q(a) \\ q(z) &\rightarrow p(b) \end{aligned}$$

and I satisfies all loop formulas. This example illustrates the role of normal form assumption in Theorem 5 (in place of Clark's equational theory in Theorem 4).

Note that a normal form conversion may turn a bounded sentence into a non-bounded sentence. For instance, a normal form of the bounded program (3.30) is

$$\begin{aligned} p(y) &\leftarrow y = f(x), q(x) \\ q(x) &\leftarrow p(x), r(x) \\ p(x) &\leftarrow x = a \\ r(x) &\leftarrow x = a \\ r(x) &\leftarrow x = f(a), \end{aligned} \tag{3.34}$$

which is not bounded.

Unlike in Corollary 5, if a program is in normal form, atomic-tightness is not more general than tightness. It is not difficult to check that a program in normal form is atomic-tight iff it is tight.

Decidability of Boundedness and Finite Complete Set of Loops

In general, checking whether F is bounded is not decidable, but it becomes decidable if F contains no function constants of positive arity. The same is the case for checking whether F is atomic-tight.

Proposition 10 *For any rectified sentence F (allowing function constants of positive arity),*

(a) checking whether F is bounded is not decidable;

(b) checking whether F is atomic-tight is not decidable.

If F contains no function constants of positive arity,

(c) checking whether F is bounded is decidable;

(d) checking whether F is atomic-tight is decidable.

The proof of Proposition 10 (c) is based on the following fact and the straightforward extension of Theorem 2 by Chen et al. (2006) to first-order formulas, which asserts that checking if F has a finite complete set of loops is decidable.

Proposition 11 *For any rectified formula F that contains no function constants of positive arity, F is bounded iff F has a finite complete set of loops.*

Note that Proposition 11 does not hold if F is allowed to contain function constants of positive arity. For instance,

$$p(x) \leftarrow p(f(x))$$

is not bounded, but has a finite complete set of loops $\{\{p(x)\}\}$.

The following corollary follows from Theorem 4 and Proposition 11.

Corollary 6 *Let F be a rectified sentence of signature σ that has no function constants of positive arity, and let I be an interpretation of σ that satisfies F and CET_σ. If F has a finite complete set of loops, then conditions (a), (b), and (c) of Theorem 4 are equivalent to each other.*

The following corollary follows from Theorem 5 and Proposition 11.

Corollary 7 *Let F be a rectified sentence in normal form that has no function constants of positive arity. If F has a finite complete set of loops, formulas in (a), (b), and (c) of Theorem 5 are equivalent to each other.*

Semi-Safe Formulas

Semi-safety is another decidable syntactic condition that ensures that $\mathrm{SM}[F]$ can be expressed by first-order sentences.

We assume that there are no function constants of positive arity. According to Lee, Lifschitz, and Palla (2009), a semi-safe sentence has the "small predicate property": the relation represented by any of its predicate constants p can hold for a tuple of arguments only if each member of the tuple is represented by an object constant occurring in F. We will show that any semi-safe sentence under the stable model semantics can be turned into a sentence in first-order logic.

First, we review the notion of semi-safety by Lee et al. (2009).[10] As a preliminary step, we assign to every rectified formula F a set $\mathrm{RV}(F)$ of its *restricted variables* as follows:

- 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;
- $\mathrm{RV}(G \wedge H) = \mathrm{RV}(G) \cup \mathrm{RV}(H)$;
- $\mathrm{RV}(G \vee H) = \mathrm{RV}(G) \cap \mathrm{RV}(H)$;
- $\mathrm{RV}(G \to H) = \emptyset$;
- $\mathrm{RV}(QvG) = \mathrm{RV}(G) \setminus \{v\}$ where $Q \in \{\forall, \exists\}$.

We say that a variable x is *restricted* in a quantifier-free formula F if $x \in \mathrm{RV}(F)$. F is *semi-safe* if every strictly positive occurrence of every variable x belongs to a subformula $G \to H$ where x is restricted in G.

[10] The definition here is slightly more general in that it does not refer to prenex form. Instead we require a formula to be rectified.

If a sentence has no strictly positive occurrence of a variable, then it is obviously semi-safe. The FOL-representation of a disjunctive program is semi-safe if, for each rule (3.8) of the program, every variable occurring in the head of the rule occurs in B as well.

Example 9 *The FOL-representation of (3.6) is not semi-safe. Formula*

$$p(a) \wedge q(b) \wedge \forall xy((p(x) \vee q(y)) \rightarrow p(y))$$

is not semi-safe, while

$$p(a) \wedge q(b) \wedge \forall xy((p(x) \wedge q(y)) \rightarrow p(y)) \tag{3.35}$$

is semi-safe.

For any finite set $\boldsymbol{c}$ of object constants, $\mathrm{in}_{\boldsymbol{c}}(x)$ stands for the formula

$$\bigvee_{c \in \boldsymbol{c}} x = c.$$

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

$$\forall v_1, \ldots, v_n \Big(p(v_1, \ldots, v_n) \rightarrow \bigwedge_{i=1,\ldots,n} in_{\boldsymbol{c}}(v_i) \Big)$$

for all predicate constants p occurring in F, where $v_1, \ldots, v_n$ are distinct variables. We denote this conjunction of the sentences by $\mathrm{SPP}_{\boldsymbol{c}}$. By $\boldsymbol{c}(F)$ we denote the set of all object constants occurring in F.

Proposition 12 *(Lee et al., 2009) For any semi-safe sentence F, formula $SM[F]$ entails $SPP_{\boldsymbol{c}(F)}$.*

For example, for the semi-safe sentence (3.35), $\mathrm{SM}[(3.35)]$ entails

$$\forall x \Big(p(x) \rightarrow (x = a \vee x = b)) \wedge \forall x (q(x) \rightarrow (x = a \vee x = b) \Big). \tag{3.36}$$

The following proposition tells us that for a semi-safe sentence F, formula $\mathrm{SM}[F]$ can be equivalently rewritten as a first-order sentence.

Theorem 6 *Let F be a rectified sentence that has no function constants of positive arity. If F is semi-safe, then $SM[F]$ is equivalent to the conjunction of F, $SPP_{\boldsymbol{c}(F)}$ and a finite number of first-order loop formulas.*

Proof. If F is semi-safe, then $\mathrm{SM}[F]$ entails $\mathrm{SPP}_{\boldsymbol{c}(F)}$. So it is sufficient to prove that under the assumption $\mathrm{SPP}_{\boldsymbol{c}(F)}$, $\mathrm{SM}[F]$ is equivalent to the conjunction of F and a finite number of first-order loop formulas. It follows from $I \models \mathrm{SPP}_{\boldsymbol{c}(F)}$ that F is bounded w.r.t. I. Since every finite loop of F w.r.t. I can be represented by a finite set of atoms whose terms are object variables, it follows from Theorem 2 that I satisfies $\mathrm{SM}[F]$ iff I satisfies the loop formulas of those sets. ∎

For example, $\mathrm{SM}[(3.35)]$ is equivalent to the conjunction of F, (3.36) and the universal closures of

$$p(z) \ \rightarrow \ z = a \ \vee \ (p(x) \wedge q(z) \wedge z \neq x)$$
$$q(z) \ \rightarrow \ z = b$$

Note that the condition on a finite complete set of loops in Corollaries 6 and 7, and the condition on semi-safety in Theorem 6 do not entail each other. For instance, formula (3.35) is semi-safe, but has no finite complete set of first-order loops, while $\exists x\, p(x)$ has a finite complete set of loops $\{\{p(x)\}\}$, but it is not semi-safe. Also program (3.1) has a finite complete set of loops, but it is not semi-safe due to w in the fourth rule.

3.4 Programs with Explicit Quantifiers

In the following we extend the syntax of a logic program by allowing explicit quantifiers. A *rule with quantifiers* is of the form

$$H \leftarrow G, \tag{3.37}$$

where G and H are first-order formulas such that every occurrence of every implication in G and H belongs to a negative formula. A *program with quantifiers* is a finite set of rules with quantifiers. The semantics of such a program is defined by identifying the program with its FOL-representation under the stable model semantics. By restricting the syntax of a program like the one above, in comparison with the syntax of an arbitrary formula, we are able to write a more succinct loop formulas, as we show below.

Let F be a formula and Y a finite set of atoms. By F_Y we denote the formula obtained from F by replacing every occurrence of every atom $p(\boldsymbol{t})$ in F that does not belong to a negative formula with $p(\boldsymbol{t}) \wedge \bigwedge_{p(\boldsymbol{t}')\in Y} \boldsymbol{t} \neq \boldsymbol{t}'$. Let Π be a program with quantifiers. Given a finite set Y of atoms of $\sigma(\Pi)$, we first rename variables in Π so that no variables in Π occur in Y. We define the formula $\mathrm{QES}_\Pi(Y)$ ("*External Support Formula for Programs with Quantifiers*") to be the disjunction of

$$\exists \boldsymbol{z}(G_Y \wedge \neg H_Y) \tag{3.38}$$

for every rule (3.37) such that H contains a strictly positive occurrence of a predicate constant that occurs in Y, and $\boldsymbol{z}$ is the list of all free variables in the rule that do not occur in Y.

The loop formula of Y for Π is the universal closure of

$$\bigwedge Y \to \mathrm{QES}_\Pi(Y). \tag{3.39}$$

The following proposition tells us that (3.39) is equivalent to (3.10) when the notions are applied to a program with explicit quantifiers. It also shows that (3.39) is a generalization of the definition of a loop formula for a disjunctive program.

Proposition 13 *Let Π be a program with quantifiers, F the FOL-representation of Π, and Y a finite set of atoms. Under the assumption Π, formula $QES_\Pi(Y)$ is equivalent to $\neg NES_F(Y)$. If Π is a disjunctive program in normal form, then $QES_\Pi(Y)$ is also equivalent to $ES_\Pi(Y)$ under the assumption Π.*

Note that the size of (3.39) for each Y is polynomial to the size of the given program. This is not the case when we apply (3.10) to the FOL-representation of the program, due to the expansion of NES for nested implications. On the other hand, the syntactic condition imposed on the rule with quantifiers avoids such an exponential blow up, as the following lemma tells us.

Lemma 4 *Let F be a formula such that every occurrence of an implication in F belongs to a negative formula and let Y be a set of atoms. $NES_F(Y)$ is equivalent to F_Y.*

Proof. By induction on F. ∎

Example 3 (continued) *Under the assumption* Π*,*

- $LF_{\Pi}(Y_1)$ *is equivalent to the universal closure of*

$$p(u) \to (\exists x(q(x) \wedge \neg(p(x) \wedge x \neq u)) \vee \exists z(\neg r(z) \wedge \neg(p(z) \wedge z \neq u))).$$

- $LF_{\Pi}(Y_2)$ *is equivalent to the universal closure of*

$$q(u) \to \exists y(p(y) \wedge \neg(q(y) \wedge y \neq u)).$$

- $LF_{\Pi}(Y_3)$ *is equivalent to the universal closure of*

$$r(u) \to \bot.$$

- $LF_{\Pi}(Y_4)$ *is equivalent to the universal closure of*

$$\begin{aligned}(p(u) \wedge q(u)) \to (&\exists x((q(x) \wedge x \neq u) \wedge \neg(p(x) \wedge x \neq u)) \\ &\vee \exists y((p(y) \wedge y \neq u) \wedge \neg(q(y) \wedge y \neq u)) \\ &\vee \exists z(\neg r(z) \wedge \neg(p(z) \wedge z \neq u))).\end{aligned}$$

A finite set Γ of sentences *entails* a sentence F under the stable model semantics (symbolically, $\Gamma \models_{\mathrm{SM}} F$), if every stable model of Γ satisfies F.

If $\mathrm{SM}[F]$ can be reduced to a first-order sentence, as described in Theorem 5 and Theorem 6, then

$$\Gamma \models_{\mathrm{SM}} F \text{ iff } \Gamma \cup \Delta \models F,$$

where Δ is the set of first-order loop formulas required (and possibly including $\mathrm{SPP}_{\boldsymbol{c}(F)}$ when Theorem 6 is applied). This fact allows us to use first-order theorem provers to reason about query entailment under the stable model semantics.

Example 10 *Consider program (3.1), which has the following finite complete set of loops:*

$\{Man(u)\}, \{Spouse(u,v)\}, \{HasWife(u)\}, \{Married(u)\}, \{Accident(u,v)\}, \{Discount(u,v)\},$

and $\{HasWife(u), Married(u)\}$. *Their loop formulas for* $\Pi_1 \cup \Pi_2 \cup \Pi_3$ *are equivalent to the universal closure of*

$$Man(u) \to \neg\big(Man(John) \wedge John \neq u\big);$$

$$Spouse(u,v) \to \neg\big(\exists y\big(Spouse(John,y) \wedge (John,y) \neq (u,v)\big)\big);$$

$$HasWife(u) \to \exists x\big(\exists y\, Spouse(x,y) \wedge \neg(HasWife(x) \wedge x \neq u)\big)$$
$$\vee\, \exists x\big(Man(x) \wedge Married(x) \wedge \neg(HasWife(x) \wedge x \neq u)\big);$$

$$Married(u) \to \exists x\big(Man(x) \wedge HasWife(x) \wedge \neg(Married(x) \wedge x \neq u)\big);$$

$$Accident(u,v) \to \bot;$$

$$Discount(u,v) \to$$
$$\exists x\big(Married(x) \wedge \neg\exists z\, Accident(x,z) \wedge \neg(\exists w(Discount(x,w) \wedge (x,w) \neq (u,v)))\big);$$

$$Married(u) \wedge HasWife(u) \to$$
$$\exists x\big(\exists y\, Spouse(x,y) \wedge \neg(HasWife(x) \wedge (x \neq u))\big)$$
$$\vee\, \exists x\big(Man(x) \wedge Married(x) \wedge x \neq u \wedge \neg(HasWife(x) \wedge x \neq u)\big)$$
$$\vee\, \exists x\big(Man(x) \wedge HasWife(x) \wedge x \neq u \wedge \neg(Married(x) \wedge x \neq u)\big).$$

These loop formulas, conjoined with the FOL-representation of $\Pi_1 \cup \Pi_2 \cup \Pi_3$*, entail under first-order logic each of* $\exists x\ Married(x)$ *and* $\forall xy(Discount(x,y) \to x = John)$. *We verified the answers using a first-order theorem prover Vampire* [11].

3.5 Extension to Allow Extensional Predicates

The results in the earlier sections can be extended to distinguish between intensional and non-intensional (a.k.a. extensional) predicates in view of Proposition 14 below, which characterizes $\mathrm{SM}[F;\boldsymbol{p}]$ in terms of $\mathrm{SM}[F]$. By $\mathrm{pr}(F)$ we denote the list of all predicate constants occurring in F; by $\mathrm{Choice}(\boldsymbol{p})$ we denote the conjunction of "choice formulas"

[11] http://www.vampire.fm .

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

Proposition 14 *For any list $\boldsymbol{p}$ of predicate constants, formula $SM[F; \boldsymbol{p}]$ is equivalent to*

$$SM[F \wedge Choice(pr(F) \backslash \boldsymbol{p}) \wedge False(\boldsymbol{p} \backslash pr(F))] \tag{3.40}$$

and to

$$SM[F^{\neg\neg} \wedge Choice(pr(F) \backslash \boldsymbol{p}) \wedge False(\boldsymbol{p} \backslash pr(F))], \tag{3.41}$$

where $F^{\neg\neg}$ is obtained from F by replacing every atom of the form $q(\boldsymbol{t})$ in F such that q does not belong to $\boldsymbol{p}$ by $\neg\neg q(\boldsymbol{t})$.

This proposition allows us to extend the results established for $\mathrm{SM}[F]$ to $\mathrm{SM}[F; \boldsymbol{p}]$. For instance, Theorem 3 can be extended to $\mathrm{SM}[F; \boldsymbol{p}]$ by first rewriting it into the form $\mathrm{SM}[G]$, where G is

$$F^{\neg\neg} \wedge \mathrm{Choice}(\mathrm{pr}(F) \backslash \boldsymbol{p}) \wedge \mathrm{False}(\boldsymbol{p} \backslash \mathrm{pr}(F)). \tag{3.42}$$

In the next three corollaries, σ is a signature, F is a rectified sentence of σ (possibly containing function constants of positive arity), $\boldsymbol{p}$ is any finite list of predicate constants from σ, and G is (3.42).

The first corollary follows from Theorem 2 and Proposition 14.

Corollary 8 *For any interpretation I of σ that satisfies F, if G is bounded w.r.t. I, then the following conditions are equivalent to each other:*

(a) $I \models SM[F; \boldsymbol{p}]$;

(b) for every nonempty finite set Y of atoms formed from predicate constants in $\boldsymbol{p}$ and object names for $|I|$, I satisfies $LF_F(Y)$;

(c) for every finite loop Y of G w.r.t. I whose predicate constants are contained in $\boldsymbol{p}$, I satisfies $LF_F(Y)$.

The next corollary follows from Theorem 4 and Proposition 14.

Corollary 9 *If G is bounded, then, for any interpretation I of σ that satisfies F and CET_σ, the following conditions are equivalent to each other:*

(a) $I \models SM[F;\boldsymbol{p}]$;

(b) for every nonempty finite set Y of atoms of $\sigma(G)$ whose predicate constants are contained in $\boldsymbol{p}$, I satisfies $LF_F(Y)$;

(c) for every finite first-order loop Y of G whose predicate constants are contained in $\boldsymbol{p}$, I satisfies $LF_F(Y)$.

The last corollary follows from Theorem 5 and Proposition 14.

Corollary 10 *If G is in normal form and is bounded, then the following formulas are equivalent to each other:*

(a) $SM[F;\boldsymbol{p}]$;

(b) $\{F\}\cup\{LF_F(Y) \mid Y$ *is a nonempty finite set of atoms of* $\sigma(G)$ *whose predicate constants are contained in* $\boldsymbol{p}$

(c) $\{F\}\cup\{LF_F(Y) \mid Y$ *is a finite first-order loop of* G *whose predicate constants are contained in* $\boldsymbol{p}\}$.

Example 11 *Consider Example 10 again, assuming that Man and $Spouse$ are extensional. Let F be the FOL-presentation of $\Pi_1 \cup \Pi_2 \cup \Pi_3$ and let G be the formula (3.42). The loops of G are the same as the loops of F. The loop formulas remain the same as before except for the following loop formulas of $Man(u)$ and $Spouse(u,v)$:*

$$Man(u) \to \neg\big(Man(John) \wedge John \neq u\big) \vee \exists x \neg\big((Man(x) \wedge x \neq u) \vee \neg Man(x)\big);$$

$$Spouse(u,v) \to \neg\big(\exists y\big(Spouse(John,y) \wedge (John,y) \neq (u,v)\big)\big) \vee \exists xy \neg\big((Spouse(x,y) \wedge (x,y) \neq (u,v)) \vee \neg Spouse(x,y)\big).$$

These two formulas are tautologies. As a result, the loop formulas of all loops, conjoined with G, entail $\exists xy Discount(x,y)$, but no longer entail $\forall xy\ (Discount(x,y) \to x = John)$.

In general, there are no loops of G that contain both intensional and extensional predicates. Also every loop of G that contains an extensional predicate is a singleton, and the loop formula of such a loop is a tautology.

Corollary 3 is extended to allow extensional predicates as in the following. By $\mathrm{SLF}[F;\boldsymbol{p}]$, we mean the conjunction of F with all loop formulas in

$\{\mathrm{LF}_F(\{p(\boldsymbol{x})\}) \mid p$ is a predicate constant in $\boldsymbol{p}$, and $\boldsymbol{x}$ is a list of distinct object variables whose length is the same as the arity of $p\}$.

We say that a formula F is *$\boldsymbol{p}$-atomic-tight* w.r.t. I if every infinite path in the dependency graph of F w.r.t. I whose vertices are satisfied by I contains an atom whose predicate constant is not in $\boldsymbol{p}$.

Corollary 11 *Let F be a rectified sentence (possibly containing function constants of positive arity), and let I be a model of F. If F is $\boldsymbol{p}$-atomic-tight w.r.t. I, then I satisfies $SM[F;\boldsymbol{p}]$ iff I satisfies $SLF[F;\boldsymbol{p}]$.*

The definition of semi-safety is extended to distinguish between intensional and non-intensional predicates as follows. Let F be a formula that has no function constants of positive arity. To every first-order formula F we assign a set $\mathrm{RV}_{\boldsymbol{p}}(F)$ 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 $\boldsymbol{p}$, then $\mathrm{RV}_{\boldsymbol{p}}(F) = \emptyset$;
  - otherwise, $\mathrm{RV}_{\boldsymbol{p}}(F)$ is the set of all variables occurring in F;
- $\mathrm{RV}_{\boldsymbol{p}}(G \wedge H) = \mathrm{RV}_{\boldsymbol{p}}(G) \cup \mathrm{RV}_{\boldsymbol{p}}(H)$;
- $\mathrm{RV}_{\boldsymbol{p}}(G \vee H) = \mathrm{RV}_{\boldsymbol{p}}(G) \cap RV_{\boldsymbol{p}}(H)$;
- $\mathrm{RV}_{\boldsymbol{p}}(G \to H) = \emptyset$.
- $\mathrm{RV}_{\boldsymbol{p}}(QvG) = \mathrm{RV}_{\boldsymbol{p}}(G) \setminus \{v\}$ where $Q \in \{\forall, \exists\}$.

We say that a variable x is $\boldsymbol{p}$*-restricted* in a first-order formula F if $x \in \mathrm{RV}_{\boldsymbol{p}}(F)$. We say that F is *semi-safe relative to* $\boldsymbol{p}$ if every strictly positive occurrence of every variable x belongs to a subformula $G \to H$, where x is $\boldsymbol{p}$-restricted in G.

The small predicate property is generalized as follows. Formula $\mathrm{SPP}^{\boldsymbol{p}}_{\boldsymbol{c}}$ is the conjunction of the sentences

$$\forall v_1, \ldots, v_n \Big(p(v_1, \ldots, v_n) \to \bigwedge_{i=1,\ldots,n} in_{\boldsymbol{c}}(v_i) \Big)$$

for all predicate constants p in $\boldsymbol{p}$, where $v_1, \ldots, v_n$ are distinct variables.

Proposition 15 *(Lee et al., 2009) For any semi-safe sentence F relative to $\boldsymbol{p}$, formula $SM[F; \boldsymbol{p}]$ entails $SPP^{\boldsymbol{p}}_{c(F)}$.*

The following proposition tells us that for a semi-safe sentence F, formula $\mathrm{SM}[F; \boldsymbol{p}]$ can be equivalently rewritten as a first-order sentence.

Theorem 7 *Let F be a rectified sentence that has no function constants of positive arity. If F is semi-safe relative to $\boldsymbol{p}$, then $SM[F; \boldsymbol{p}]$ is equivalent to the conjunction of F, $SPP^{\boldsymbol{p}}_{\boldsymbol{c}(F)}$ and a finite number of first-order loop formulas.*

Proof. Let F be a sentence of the signature σ. If F is semi-safe relative to $\boldsymbol{p}$, then $\mathrm{SM}[F; \boldsymbol{p}]$ entails $\mathrm{SPP}^{\boldsymbol{p}}_{\boldsymbol{c}(F)}$, so it is sufficient to prove that under the assumption $\mathrm{SPP}^{\boldsymbol{p}}_{\boldsymbol{c}(F)}$, $\mathrm{SM}[F; \boldsymbol{p}]$ is equivalent to the conjunction of F and a finite number of first-order loop formulas. By Proposition 14, $\mathrm{SM}[F; \boldsymbol{p}]$ is equivalent to $\mathrm{SM}[G]$, where G is (3.42). Consider any interpretation I of σ that satisfies G and $\mathrm{SPP}^{\boldsymbol{p}}_{\boldsymbol{c}(F)}$. Note that the dependency graph of G w.r.t. I contains no outgoing edges from a vertex whose predicate constant does not belong to $\boldsymbol{p}$. Together with the fact that $I \models \mathrm{SPP}^{\boldsymbol{p}}_{\boldsymbol{c}(F)}$, we conclude that each path in the dependency graph whose vertices are satisfied by I visits only finitely many vertices. Consequently, G is bounded w.r.t. I. Since every finite loop of G w.r.t. I can be represented by a finite set of atoms whose terms are object variables, it follows from Theorem 2 that I satisfies $\mathrm{SM}[G]$ iff I satisfies the loop formulas of those sets. ∎

3.6 Related Work

The notion of a bounded program is related to the notion of a *finitely recursive* program studied by Bonatti (2004), where a different definition of a dependency graph was considered. The *atom dependency graph* of a nondisjunctive ground program defined in (Bonatti, 2004) is a directed graph such that the vertices are the set of ground atoms, and the edges go from the atom in the head to atoms in the body of every rule, including those in the negative body. A program is called *finitely recursive* if, for every atom, there are only finitely many atoms reachable from it in the atom dependency graph. It is clear that every finitely recursive program is bounded, but the converse does not hold. For instance, the program

$$p(x) \leftarrow \text{not } p(f(x))$$

is bounded, but is not finitely recursive because there are infinite paths that involve negative edges. Also the program

$$p(a) \leftarrow q(f(x))$$

is bounded, but is not finitely recursive because infinitely many atoms $q(f(a)), q(f(f(a))), \ldots$ can be reached from $p(a)$ in the atom dependency graph. Like bounded programs, checking finitely recursive programs is undecidable in the presence of function constants of positive arity.

Lin and Wang (2008) extended answer set semantics with functions by extending the definition of a reduct, and also provided loop formulas for such programs. We can provide an alternative account of their results by considering the notions there as special cases of our definitions. For simplicity, we assume non-sorted languages.[12] Essentially, they restricted attention to a special case of non-Herbrand interpretations such that object constants form the universe, and ground terms other than object constants are mapped to object constants. According to Lin and Wang, an *LW-program* P consists of *type definitions* and a set of rules. Type definitions introduce the domains for a many-sorted signature consisting of some object constants, and includes the evaluation of each function symbol of positive arity that maps a list of object constants to an object constant. Since we assume

[12] Lin and Wang (2008) considers essentially many-sorted languages. The result of this section can be extended to that case by considering many-sorted SM (Kim, Lee, & Palla, 2009).

non-sorted languages, we consider only a single domain (universe). We say that an interpretation I is a *P-interpretation* if the universe is the set of object constants specified by P, object constants are evaluated to itself, and ground terms other than object constants are evaluated conforming to the type definitions of P.

Proposition 16 *Let P be an LW-program and let F be the FOL-representation of the set of rules in P. The following conditions are equivalent to each other:*

(a) I is an answer set of P according to (Lin & Wang, 2008);

(b) I is a P-interpretation that satisfies $SM[F]$;

(c) I is a P-interpretation that satisfies F and the loop formulas of Y for all loops Y of F w.r.t. I.

The equivalence between (b) and (c) follows from Proposition 2 since the universe is finite. The equivalence between (a) and (c) follows from the fact that LW answer sets can be characterized by loop formulas that are defined by Lin and Wang (2008) and that these loop formulas are essentially the same as the loop formulas in (c).

Since the proposal of the first-order stable model semantics, there have been some papers about first-order definability of $\mathrm{SM}[F]$. Zhang and Zhou (2010) show that, for a nondisjunctive program Π that has no function constants of positive arity, its first-order stable model semantics can be reformulated by a progression based semantics. They also showed that the programs whose answer sets can be found by a finite progression are exactly those that can be represented by first-order formulas. Some researchers have paid special attention to first-order definability of $\mathrm{SM}[F]$ on finite structures. Chen, Zhang, and Zhou (2010) show a game-theoretic characterization for the first-order indefinability of first-order answer set programs on finite structures. Asuncion, Lin, Zhang, and Zhou (2010) show first-order definability on finite structures by turning programs into modified completion using new predicates to record levels. Chen, Lin, Zhang, and Zhou (2011) present a condition called "loop-separable," which is more refined than finite complete set of loops under which the finite answer sets of a program can be captured by first-order sentences.

However, like the condition of finite complete set of loops, this condition is disjoint with semi-safety. The following program is semi-safe but not loop-separable:

$$p(x) \leftarrow p(y), q(x, y).$$

However, all this work is limited to nondisjunctive programs that contain no function constants of positive arity. Our work is not limited to finite structures, and considers function constants of positive arity as well. Nonetheless the above papers on first-order definability are closely related to our work and more insights would be gained from the relationship between them.

The use of first-order theorem provers for the stable model semantics was already investigated by Sabuncu and Alpaslan (2007), but their results are limited in several ways. They considered nondisjunctive logic programs with "trivial" loops only, in which case the stable model semantics is equivalent to the completion semantics. They also restricted attention to Herbrand models.

3.7 Conclusion

This chapter relates first-order logic and first-order stable model semantics via first-order loop formulas. We identify three decidable conditions under which the formulas under the first-order stable model semantics can be represented by formulas in first-order logic. The discovered relationship provides useful insights into first-order reasoning with stable models. This allows us to compute non-Herbrand stable models using first-order theorem provers.

3.8 Proofs

The proofs are presented in the order of dependencies. Theorem 3 is the main theorem. The proof of Theorem 2 uses Theorem 3. The proofs of Theorems 4 and 5 follow from Theorem 2. The proof of Lemma 1 follows from Proposition 13.

In the following, unless otherwise noted, F is a rectified first-order sentence, $\boldsymbol{p}$ is the list of distinct predicate constants $p_1, \ldots, p_n$ occurring in F, symbols $\boldsymbol{u}$, $\boldsymbol{v}$ are lists of distinct predicate variables of the same length as $\boldsymbol{p}$, and symbols $\boldsymbol{q}$, $\boldsymbol{r}$ are lists of distinct predicate names of the same length as $\boldsymbol{p}$.

Proof of Theorem 3

Theorem 3 *For any rectified sentence F, the following sentences are equivalent to each other:*

(a) $SM[F]$;

(b) $F \wedge \forall \boldsymbol{u}((\boldsymbol{u} \leq \boldsymbol{p}) \wedge Nonempty(\boldsymbol{u}) \rightarrow \neg NSES_F(\boldsymbol{u}))$;

(c) $F \wedge \forall \boldsymbol{u}((\boldsymbol{u} \leq \boldsymbol{p}) \wedge Ext\text{-}Loop_F(\boldsymbol{u}) \rightarrow \neg NSES_F(\boldsymbol{u}))$.

The notation that we use in the proof involves *predicate expressions* (Lifschitz, 1994, Section 3.1) of the form

$$\lambda \boldsymbol{x} F(\boldsymbol{x}), \tag{3.43}$$

where $F(\boldsymbol{x})$ is a formula. If e is (3.43) and $G(p)$ is a formula containing a predicate constant p of the same arity as the length of $\boldsymbol{x}$ then $G(e)$ stands for the result of replacing each atomic part of the form $p(\boldsymbol{t})$ in $G(p)$ with $F(\boldsymbol{t})$, 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 $\boldsymbol{e}$ of predicate expressions for a tuple $\boldsymbol{p}$ of predicate constants is defined in a similar way.

Lemma 5 *Let $\boldsymbol{v}$ be the list of $\lambda \boldsymbol{y}^i(p_i(\boldsymbol{y}^i) \wedge \neg u_i(\boldsymbol{y}^i))$. The following formulas are logically valid:*

- $\boldsymbol{u} \leq \boldsymbol{p} \rightarrow (F^*(\boldsymbol{u}) \leftrightarrow NSES_F(\boldsymbol{v}))$;
- $\boldsymbol{u} \leq \boldsymbol{p} \rightarrow (F^*(\boldsymbol{v}) \leftrightarrow NSES_F(\boldsymbol{u}))$.

Proof. By induction.

Proof of Equivalence between (a) and (b) of Theorem 3

It is sufficient to show that

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

is equivalent to

$$\exists \boldsymbol{v}(\boldsymbol{v} \leq \boldsymbol{p} \wedge \mathrm{Nonempty}(\boldsymbol{v}) \wedge \mathrm{NSES}_F(\boldsymbol{v})).$$

From left to right: Take $\boldsymbol{u}$ such that $\boldsymbol{u} < \boldsymbol{p} \wedge F^*(\boldsymbol{u})$. Let $\boldsymbol{v}$ be the list of $\lambda \boldsymbol{y}^i(p_i(\boldsymbol{y}^i) \wedge \neg u_i(\boldsymbol{y}^i))$.

- Clearly, $\boldsymbol{v} \leq \boldsymbol{p}$ holds.
- From $\boldsymbol{u} < \boldsymbol{p}$, it follows that there are $\boldsymbol{x}$ and i such that $p_i(\boldsymbol{x}) \wedge \neg u_i(\boldsymbol{x})$, from which $\bigvee_i \exists \boldsymbol{x}^i v_i(\boldsymbol{x}^i)$ follows, so that $\mathrm{Nonempty}(\boldsymbol{v})$ follows.
- By Lemma 5, $\mathrm{NSES}_F(\boldsymbol{v})$ follows from $\boldsymbol{u} < \boldsymbol{p}$ and $F^*(\boldsymbol{u})$.

From right to left: Take $\boldsymbol{v}$ such that $\boldsymbol{v} \leq \boldsymbol{p} \wedge \mathrm{Nonempty}(\boldsymbol{v}) \wedge \mathrm{NSES}_F(\boldsymbol{v})$. Let $\boldsymbol{u}$ be the list of $\lambda \boldsymbol{y}^i(p_i(\boldsymbol{y}^i) \wedge \neg v_i(\boldsymbol{y}^i))$.

- Clearly, $\boldsymbol{u} \leq \boldsymbol{p}$ holds. Moreover $\neg(\boldsymbol{u} = \boldsymbol{p})$ holds. Indeed, if $\boldsymbol{u} = \boldsymbol{p}$, then $\forall \boldsymbol{x}^i \neg v_i(\boldsymbol{x}^i)$ follows, which contradicts the assumption $\mathrm{Nonempty}(\boldsymbol{v})$. Consequently, $\boldsymbol{u} < \boldsymbol{p}$ follows.
- By Lemma 5, $F^*(\boldsymbol{u})$ follows from $\boldsymbol{v} \leq \boldsymbol{p}$ and $\mathrm{NSES}_F(\boldsymbol{v})$.

∎

Proof of Proposition 3

Lemma 6 *Let I be an interpretation of σ that contains $\sigma(F)$, and let $\boldsymbol{q}$, $\boldsymbol{r}$ be lists of predicate names corresponding to $\boldsymbol{p}$. Let Z and Y be sets of atoms in the dependency graph of F w.r.t. I such that*

$$p_i(\boldsymbol{\xi}^\diamond) \in Y \text{ iff } I \models q_i(\boldsymbol{\xi}^\diamond)$$

and

$$p_i(\boldsymbol{\xi}^\diamond) \in Z \text{ iff } I \models r_i(\boldsymbol{\xi}^\diamond),$$

where $\boldsymbol{\xi}^\diamond$ is a list of object names. Then

$$I \models \boldsymbol{r} \leq \boldsymbol{q} \wedge E_F(\boldsymbol{r}, \boldsymbol{q})$$

iff Z is a subset of Y and there is an edge from an atom in Z to an atom in $Y \setminus Z$ in the dependency graph of F w.r.t. I.

Proof. *From left to right:* Assume $I \models \boldsymbol{r} \leq \boldsymbol{q} \wedge E_F(\boldsymbol{r}, \boldsymbol{q})$. The fact that Z is a subset of Y follows from the assumption that $I \models \boldsymbol{r} \leq \boldsymbol{q}$ and the construction of Z and Y. Since

$$I \models \bigvee_{\substack{(p_i(\boldsymbol{t}), p_j(\boldsymbol{t}')) \,:\, p_i(\boldsymbol{t}) \text{ depends on } p_j(\boldsymbol{t}') \\ \text{in a rule of } F}} \exists \boldsymbol{z}(r_i(\boldsymbol{t}) \wedge q_j(\boldsymbol{t}') \wedge \neg r_j(\boldsymbol{t}')),$$

where $\boldsymbol{z}$ is the list of all object variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$, there is a substitution θ that maps object variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$ to object names such that

$$I \models \bigvee_{\substack{(p_i(\boldsymbol{t}), p_j(\boldsymbol{t}')) \,:\, p_i(\boldsymbol{t}) \text{ depends on } p_j(\boldsymbol{t}') \\ \text{in a rule of } F}} r_i(\boldsymbol{t}\theta) \wedge q_j(\boldsymbol{t}'\theta) \wedge \neg r_j(\boldsymbol{t}'\theta).$$

Consequently, there are atoms $p_i(\boldsymbol{t})$, $p_j(\boldsymbol{t}')$ such that $p_i(\boldsymbol{t})$ depends on $p_j(\boldsymbol{t}')$ in a rule of F and $I \models r_i(\boldsymbol{t}\theta) \wedge q_j(\boldsymbol{t}'\theta) \wedge \neg r_j(\boldsymbol{t}'\theta)$. From $I \models r_i(\boldsymbol{t}\theta)$ and the construction of Z, it follows that $p_i(((\boldsymbol{t}\theta)^I)^\diamond)$ belongs to Z. Also from $I \models q_j(\boldsymbol{t}'\theta) \wedge \neg r_j(\boldsymbol{t}'\theta)$, it follows that that $p_j(((\boldsymbol{t}'\theta)^I)^\diamond)$ belongs to $Y \setminus Z$. Therefore, there is an edge from an atom in Z to an atom in $Y \setminus Z$ in the dependency graph of F w.r.t. I.

From right to left: Assume that Z is a subset of Y and there is an edge from an atom $p_i(\boldsymbol{\xi}^\diamond)$ in Z to an atom $p_j(\boldsymbol{\eta}^\diamond)$ in $Y \setminus Z$ in the dependency graph of F w.r.t. I. Clearly, $I \models \boldsymbol{r} \leq \boldsymbol{q}$.

From the assumption that $p_i(\boldsymbol{\xi}^\diamond) \in Z$, $p_j(\boldsymbol{\eta}^\diamond) \in Y \setminus Z$ and the construction of Y and Z, it follows that $I \models r_i(\boldsymbol{\xi}^\diamond) \wedge q_j(\boldsymbol{\eta}^\diamond) \wedge \neg r_j(\boldsymbol{\eta}^\diamond)$. From the definition of the dependency graph w.r.t. I, it follows that there are $p_i(\boldsymbol{t})$, $p_j(\boldsymbol{t}')$ such that $p_i(\boldsymbol{t})$ depends on $p_j(\boldsymbol{t}')$ in a rule of F with a substitution θ that maps object variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$ to object names such that $(\boldsymbol{t}\theta)^I = \boldsymbol{\xi}$ and $(\boldsymbol{t}'\theta)^I = \boldsymbol{\eta}$.

Consequently,

$$I \models \bigvee_{\substack{(p_i(\boldsymbol{t}),p_j(\boldsymbol{t}')) \,:\, p_i(\boldsymbol{t}) \text{ depends on } p_j(\boldsymbol{t}') \\ \text{in a rule of } F}} r_i(\boldsymbol{t}\theta) \wedge q_j(\boldsymbol{t}'\theta) \wedge \neg r_j(\boldsymbol{t}'\theta),$$

which is equivalent to saying that

$$I \models \bigvee_{\substack{(p_i(\boldsymbol{t}),p_j(\boldsymbol{t}')) \,:\, p_i(\boldsymbol{t}) \text{ depends on } p_j(\boldsymbol{t}') \\ \text{in a rule of } F}} \exists \boldsymbol{z}(r_i(\boldsymbol{t}) \wedge q_j(\boldsymbol{t}') \wedge \neg r_j(\boldsymbol{t}')),$$

where $\boldsymbol{z}$ is the list of all variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$. ∎

Lemma 7 *A graph (V, E) is strongly connected iff, for any nonempty proper subset U of V, there is an edge from U to $V \setminus U$.*

Proof. Follows from the definition of a strongly connected graph. ∎

Proposition 3 *Let $\boldsymbol{q}$ be a list of predicate names corresponding to $\boldsymbol{p}$, and let Y be a set of atoms in the dependency graph of F w.r.t. I such that*

$$p_i(\boldsymbol{\xi}^\diamond) \in Y \textit{ iff } I \models q_i(\boldsymbol{\xi}^\diamond),$$

where $\boldsymbol{\xi}^\diamond$ is a list of object names. Then $I \models Loop_F(\boldsymbol{q})$ iff Y is a loop of F w.r.t. I.

Proof. *From left to right:* Assume that $I \models \mathrm{Loop}_F(\boldsymbol{q})$. From $I \models \mathrm{Nonempty}(\boldsymbol{q})$, it follows that Y is nonempty.

Take any nonempty proper subset Z of Y. Let $\boldsymbol{r}$ be the list of predicate names such that

$$I \models r_i(\boldsymbol{\xi}^\diamond) \text{ iff } p_i(\boldsymbol{\xi}^\diamond) \in Z.$$

It is clear that

$$I \models \mathrm{Nonempty}(\boldsymbol{r}) \wedge \boldsymbol{r} < \boldsymbol{q}.$$

Consequently, from $I \models \mathrm{Loop}_F(\boldsymbol{q})$, it follows that $I \models \mathrm{E}_F(\boldsymbol{r}, \boldsymbol{q})$. By Lemma 6, there is an edge from an atom in Z to an atom in $Y \setminus Z$. Consequently, by Lemma 7, Y induces a strongly connected subgraph and thus a loop of F w.r.t. I.

From right to left: Let Y be loop of F w.r.t. I and $\boldsymbol{q}$ a list of predicate names such that

$$I \models q_i(\boldsymbol{\xi}^\diamond) \text{ iff } p_i(\boldsymbol{\xi}^\diamond) \in Y.$$

Since Y is nonempty, $I \models \mathrm{Nonempty}(\boldsymbol{q})$.

Consider any list of predicate names $\boldsymbol{r}$ such that

$$I \models \mathrm{Nonempty}(\boldsymbol{r}) \wedge \boldsymbol{r} < \boldsymbol{q}.$$

Let Z be a set of vertices in the dependency graph of F w.r.t. I such that

$$p_i(\boldsymbol{\xi}^\diamond) \in Z \text{ iff } I \models r_i(\boldsymbol{\xi}^\diamond).$$

Clearly, Z is a nonempty proper subset of Y. Since Y induces a strongly connected subgraph, by Lemma 7, there is an edge from an atom in Z to an atom in $Y \setminus Z$. Consequently by Lemma 6, $I \models E_F(\boldsymbol{r}, \boldsymbol{q})$. ∎

Proof of Proposition 4

Proposition 4 *Let $\boldsymbol{q}$ be a list of predicate names corresponding to $\boldsymbol{p}$, and let Y be a set of atoms in the dependency graph of F w.r.t. I such that*

$$p_i(\boldsymbol{\xi}^\diamond) \in Y \textit{ iff } I \models q_i(\boldsymbol{\xi}^\diamond),$$

where $\boldsymbol{\xi}^\diamond$ is a list of object names. Then

$$I \models \mathit{Nonempty}(\boldsymbol{q}) \wedge \forall \boldsymbol{v}((\boldsymbol{v} \leq \boldsymbol{q}) \wedge \mathit{Loop}_F(\boldsymbol{v}) \rightarrow E_F(\boldsymbol{v}, \boldsymbol{q}))$$

iff Y is an unbounded set of F w.r.t. I.

Proof. *From left to right:* Assume

$$I \models \mathrm{Nonempty}(\boldsymbol{q}) \wedge \forall \boldsymbol{v}(\boldsymbol{v} \leq \boldsymbol{q} \wedge \mathrm{Loop}_F(\boldsymbol{v}) \rightarrow E_F(\boldsymbol{v}, \boldsymbol{q})). \tag{3.44}$$

Since $I \models \mathrm{Nonempty}(\boldsymbol{q})$, it is clear that Y is nonempty.

Take any subset Z of Y that is a loop of F w.r.t. I. Let $\boldsymbol{r}$ be a list of predicate names such that

$$I \models r_i(\boldsymbol{\xi}^{\diamond}) \text{ iff } p_i(\boldsymbol{\xi}^{\diamond}) \in Z.$$

Since Z is a subset of Y, it is clear that $I \models \boldsymbol{r} \leq \boldsymbol{q}$. Since Z is a loop of F w.r.t. I, by Proposition 3, $I \models \mathrm{Loop}_F(\boldsymbol{r})$. Consequently, from (3.44) it follows that $I \models \mathrm{E}_F(\boldsymbol{r}, \boldsymbol{q})$. By Lemma 6, there is an edge from an atom in Z to an atom in $Y \setminus Z$. Therefore, Y is an unbounded set of F w.r.t. I.

From right to left: Let Y be an unbounded set of F w.r.t. I. Since Y is nonempty, it is clear that $I \models \mathrm{Nonempty}(\boldsymbol{q})$.

Take any list of predicate names $\boldsymbol{r}$ such that $I \models \boldsymbol{r} \leq \boldsymbol{q} \wedge \mathrm{Loop}_F(\boldsymbol{r})$. Let Z be a set of vertices in the dependency graph of F w.r.t. I such that

$$p_i(\boldsymbol{\xi}^{\diamond}) \in Z \text{ iff } I \models r_i(\boldsymbol{\xi}^{\diamond}).$$

By Proposition 3, Z is a loop of F w.r.t. I. It is clear that Z is a subset of Y. Since Y is an unbounded set of F w.r.t. I, there is an edge from Z to $Y \setminus Z$. Consequently by Lemma 6, $I \models E_F(\boldsymbol{r}, \boldsymbol{q})$. ∎

Proof of Proposition 5

Proposition 5 *For any negative formula F, formula*

$$NSES_F(\boldsymbol{u}) \leftrightarrow F$$

is logically valid.

Proof. The proof follows immediately from the following two lemmas, which can be proved by induction. ∎

Lemma 8 *For any formula F,*

$$NSES_F(\boldsymbol{u}) \rightarrow F$$

is logically valid.

Lemma 9 *Let F be a formula, and let S_F be the set of $p_i(\boldsymbol{t})$ that has a strictly positive occurrence in F. Formula*

$$F \wedge \bigwedge_{p_i(\boldsymbol{t}) \in S_F} \forall \boldsymbol{z} \neg v_i(\boldsymbol{t}) \rightarrow NSES_F(\boldsymbol{v}) \tag{3.45}$$

is logically valid, where $\boldsymbol{z}$ is the tuple of variables in $\boldsymbol{t}$ that are not free in F.

Proof of Equivalence between (b) and (c) of Theorem 3

Lemma 10 *Let F be a rectified formula, let S_F^+ be the set of all atoms $p_i(\boldsymbol{t})$ that have a positive occurrence in F that does not belong to a negative formula, and let S_F^- be the set of all atoms $p_i(\boldsymbol{t})$ that have a negative occurrence in F that does not belong to a negative formula.*[13] *The following formulas are logically valid, where $\boldsymbol{z}$ is the list of all variables in $\boldsymbol{t}$ that are not free in F.*

(a) $(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \bigwedge_{p_i(\boldsymbol{t}) \in S_F^+} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t})) \wedge NSES_F(\boldsymbol{v}) \rightarrow NSES_F(\boldsymbol{u})$;

(b) $(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \bigwedge_{p_i(\boldsymbol{t}) \in S_F^-} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t})) \wedge NSES_F(\boldsymbol{u}) \rightarrow NSES_F(\boldsymbol{v})$.

Proof. Both parts are proved simultaneously by induction on F.

Case 1: F is an atom $p_i(\boldsymbol{t})$.

Part (a): $\mathrm{NSES}_F(\boldsymbol{v})$ entails $\mathrm{NSES}_F(\boldsymbol{u})$ under the assumption

$$\bigwedge_{p_i(\boldsymbol{t}) \in S_F^+} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t})).$$

Part (b): $\mathrm{NSES}_F(\boldsymbol{u})$ entails $\mathrm{NSES}_F(\boldsymbol{v})$ under the assumption $\boldsymbol{v} \leq \boldsymbol{u}$.

Case 2: F is $\bot$ or an equality. It is clear since $\mathrm{NSES}_F(\boldsymbol{v})$ and $\mathrm{NSES}_F(\boldsymbol{u})$ are the same as F.

Case 3: F is $G \wedge H$ or $G \vee H$. Follows from I.H.

Case 4: F is $G \rightarrow H$.

[13] Note that we distinguish between formula being negative and an occurrence being negative. See at the end of Section 2.5.

Part (a): Assume

$$(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \bigwedge_{p_i(\boldsymbol{t}) \in S_F^+} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t})). \tag{3.46}$$

We need to show that

$$(\mathrm{NSES}_G(\boldsymbol{v}) \rightarrow \mathrm{NSES}_H(\boldsymbol{v})) \wedge (G \rightarrow H)$$

entails

$$(\mathrm{NSES}_G(\boldsymbol{u}) \rightarrow \mathrm{NSES}_H(\boldsymbol{u})) \wedge (G \rightarrow H).$$

Note that

$$\bigwedge_{p_i(\boldsymbol{t}) \in S_G^-} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t}))$$

and

$$\bigwedge_{p_i(\boldsymbol{t}) \in S_H^+} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t}))$$

are entailed by formula (3.46). By I.H., $\mathrm{NSES}_G(\boldsymbol{u})$ entails $\mathrm{NSES}_G(\boldsymbol{v})$ and $\mathrm{NSES}_H(\boldsymbol{v})$ entails $\mathrm{NSES}_H(\boldsymbol{u})$.

Part (b): Similar to Part (a).

Case 5: F is $\forall x\, G$

Part (a): Assume

$$(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \bigwedge_{p_i(\boldsymbol{t}) \in S_F^+} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t})) \wedge \forall x \mathrm{NSES}_G(\boldsymbol{v}).$$

From the assumption $\mathrm{NSES}_G(\boldsymbol{v})$, G follows by Lemma 8. Also

$$\bigwedge_{p_i(\boldsymbol{t}) \in S_G^+} \forall \boldsymbol{z}'(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t}))$$

follows, where $\boldsymbol{z}'$ is the list of all variables in $\boldsymbol{t}$ that are not free in G, so that by I.H. on G, $\mathrm{NSES}_G(\boldsymbol{u})$ holds from the assumption. Since x is not free in the assumption, $\forall x \mathrm{NSES}_G(\boldsymbol{u})$ holds as well.

Part (b): Similar to Part (a).

Case 6: F is $\exists x\, G$.

Part (a): Assume

$$(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \bigwedge_{p_i(\boldsymbol{t}) \in S_F^+} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t})) \wedge \exists x \mathrm{NSES}_G(\boldsymbol{v}). \tag{3.47}$$

Take x such that

$$(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \bigwedge_{p_i(\boldsymbol{t}) \in S_F^+} \forall \boldsymbol{z}(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t})) \wedge \mathrm{NSES}_G(\boldsymbol{v}). \tag{3.48}$$

From $\mathrm{NSES}_G(\boldsymbol{v})$, by Lemma 8, G follows. Also

$$\bigwedge_{p_i(\boldsymbol{t}) \in S_G^+} \forall \boldsymbol{z}'(u_i(\boldsymbol{t}) \rightarrow v_i(\boldsymbol{t}))$$

follows, where $\boldsymbol{z}'$ is the list of all variables in $\boldsymbol{t}$ that are not free in G. By I.H. on G, $\mathrm{NSES}_G(\boldsymbol{u})$ holds under the assumption (3.48). Consequently, $\exists x \mathrm{NSES}_G(\boldsymbol{u})$ holds from the same assumption. Since x is not free in (3.47), we conclude that $\exists x \mathrm{NSES}_G(\boldsymbol{u})$ holds from the assumption (3.47).

Part (b): Similar to Part (a). ∎

Lemma 11 *For any rectified formula F,*

$$(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \neg E_F(\boldsymbol{v}, \boldsymbol{u}) \wedge \mathit{NSES}_F(\boldsymbol{u}) \rightarrow \mathit{NSES}_F(\boldsymbol{v})$$

is logically valid.

Proof. By induction on F.

Case 1: F is an atom $p_i(\boldsymbol{t})$. $\mathrm{NSES}_F(\boldsymbol{u})$ entails $\mathrm{NSES}_F(\boldsymbol{v})$ under the assumption $\boldsymbol{v} \leq \boldsymbol{u}$.

Case 2: F is $\bot$ or equality. It is clear since $\mathrm{NSES}_F(\boldsymbol{v})$ and $\mathrm{NSES}_F(\boldsymbol{u})$ are the same as F.

Case 3: F is $G \wedge H$ or $G \vee H$. Follows from I.H.

Case 4: F is $G \rightarrow H$. Assume

$$(\boldsymbol{v} \leq \boldsymbol{u}) \wedge \neg E_F(\boldsymbol{v}, \boldsymbol{u}) \wedge \mathrm{NSES}_F(\boldsymbol{u})$$

and $\mathrm{NSES}_G(\boldsymbol{v})$. From $\mathrm{NSES}_F(\boldsymbol{u})$, by Lemma 8, we conclude $G \to H$. From $\mathrm{NSES}_G(\boldsymbol{v})$, by Lemma 8, G follows, and consequently H.

Assume $\neg\mathrm{NSES}_H(\boldsymbol{v})$ for the sake of contradiction. By Lemma 9, from H and $\neg\mathrm{NSES}_H(\boldsymbol{v})$, it follows that

$$\bigvee_{p_i(\boldsymbol{t}) \,:\, p_i(\boldsymbol{t}) \text{ occurs strictly positively in } H} \exists \boldsymbol{x} v_i(\boldsymbol{t}) \tag{3.49}$$

, where $\boldsymbol{x}$ is the list of variables in $\boldsymbol{t}$ that are not free in H.

Since F is rectified, the variables in F can be partitioned into three sets: the list of variables $\boldsymbol{x}$ that are not free in H, the list of variables $\boldsymbol{y}$ that are not free in G, and the rest. Note that $\neg E_F(\boldsymbol{v}, \boldsymbol{u})$ entails

$$\bigwedge_{\substack{(p_i(\boldsymbol{t}), p_j(\boldsymbol{t}')) \,:\, p_i(\boldsymbol{t}) \text{ depends on } p_j(\boldsymbol{t}') \text{ in a rule } G \to H \text{ in } F \\ p_i(\boldsymbol{t}) \text{ occurs in } H, p_j(\boldsymbol{t}') \text{ occurs in } G}} \Big(\exists \boldsymbol{x} v_i(\boldsymbol{t}) \to \forall \boldsymbol{y} (u_j(\boldsymbol{t}') \to v_j(\boldsymbol{t}')) \Big), \tag{3.50}$$

where $\boldsymbol{x}$ is the list of all variables in $\boldsymbol{t}$ that are not free in H, and $\boldsymbol{y}$ is the list of all variables in $\boldsymbol{t}'$ that are not free in G. From (3.49) and (3.50), we conclude

$$\bigwedge_{p_j(\boldsymbol{t}') \,:\, p_j(\boldsymbol{t}') \text{ occurs positively and not in a negative subformula of } G} \forall \boldsymbol{y} (u_j(\boldsymbol{t}') \to v_j(\boldsymbol{t}')).$$

From this, together with the assumption $(\boldsymbol{v} \leq \boldsymbol{u})$ and $\mathrm{NSES}_G(\boldsymbol{v})$, by Lemma 10 (a), $\mathrm{NSES}_G(\boldsymbol{u})$ follows. Thus $\mathrm{NSES}_H(\boldsymbol{u})$ follows from $\mathrm{NSES}_F(\boldsymbol{u})$ and $\mathrm{NSES}_G(\boldsymbol{u})$. Since $\neg\mathrm{E}_F(\boldsymbol{v}, \boldsymbol{u})$ entails $\neg\mathrm{E}_H(\boldsymbol{v}, \boldsymbol{u})$, by I.H. on H, $\mathrm{NSES}_H(\boldsymbol{v})$ follows, which contradicts the assumption.

Case 5: F is $\forall x G$ or $\exists x G$. Follows from I.H. ∎

Lemma 12

$$\mathit{Nonempty}(\boldsymbol{u}) \to \exists \boldsymbol{v} (\boldsymbol{v} \leq \boldsymbol{u} \wedge \mathit{Ext\text{-}Loop}_F(\boldsymbol{v}) \wedge \neg E_F(\boldsymbol{v}, \boldsymbol{u}))$$

is logically valid.

Proof. Take any list $\boldsymbol{q}$ of predicate names, and any interpretation I that satisfies $\mathrm{Nonempty}(\boldsymbol{q})$. Let Y be a set of vertices in the dependency graph of F w.r.t. I such that

$$p_i(\boldsymbol{\xi}^{\diamond}) \in Y \text{ iff } I \models q_i(\boldsymbol{\xi}^{\diamond}).$$

Consider the subgraph G of the dependency graph of F w.r.t. I that is induced by Y. If Y is an unbounded set w.r.t. I, by Proposition 4, $I \models \mathrm{Ext\text{-}Loop}_F(\boldsymbol{q})$. So

$$I \models \boldsymbol{q} \leq \boldsymbol{q} \wedge \mathrm{Ext\text{-}Loop}_F(\boldsymbol{q}) \wedge \neg E_F(\boldsymbol{q}, \boldsymbol{q}).$$

Otherwise, consider the graph G' that is obtained from G by collapsing strongly connected components of G, i.e., the vertices of G' are the strongly connected components of G and G' has an edge from V to V' if G has an edge from a vertex in V to a vertex in V'. Since we assumed that Y is not an unbounded set w.r.t. I, there exists a vertex Z in G' that has no outgoing edges. Consider the list of predicate names $\boldsymbol{r}$ such that

$$I \models r_i(\boldsymbol{\xi}^{\diamond}) \text{ iff } p_i(\boldsymbol{\xi}^{\diamond}) \in Z.$$

It is clear that $I \models \boldsymbol{r} \leq \boldsymbol{q}$. By Proposition 3, $I \models \mathrm{Loop}_F(\boldsymbol{r})$ thus $I \models \mathrm{Ext\text{-}Loop}_F(\boldsymbol{r})$. Since there is no edge from Z to $Y \setminus Z$, by Lemma 6, $I \models \neg E_F(\boldsymbol{r}, \boldsymbol{q})$. Consequently, the claim follows. ∎

Proof of Equivalence Between (b) and (c) of Theorem 3

From (b) to (c): Clear from that the formula $\mathrm{Ext\text{-}Loop}_F(\boldsymbol{u}) \rightarrow \mathrm{Nonempty}(\boldsymbol{u})$ is logically valid.

From (c) to (b): Assume

$$F \wedge \forall \boldsymbol{v}(\boldsymbol{v} \leq \boldsymbol{p} \wedge \mathrm{Ext\text{-}Loop}_F(\boldsymbol{v}) \rightarrow \neg \mathrm{NSES}_F(\boldsymbol{v})).$$

Take any $\boldsymbol{u}$ such that $\boldsymbol{u} \leq \boldsymbol{p} \wedge \mathrm{Nonempty}(\boldsymbol{u})$. By Lemma 12, it follows from $\mathrm{Nonempty}(\boldsymbol{u})$ that there exists $\boldsymbol{v}$ such that $\boldsymbol{v} \leq \boldsymbol{u} \wedge \mathrm{Ext\text{-}Loop}_F(\boldsymbol{v}) \wedge \neg E_F(\boldsymbol{v}, \boldsymbol{u})$. It is clear that $\boldsymbol{v} \leq \boldsymbol{p}$ follows from $\boldsymbol{v} \leq \boldsymbol{u}$ and $\boldsymbol{u} \leq \boldsymbol{p}$. It follows from the assumption that $\neg \mathrm{NSES}_F(\boldsymbol{v})$. Then by Lemma 11, $\neg \mathrm{NSES}_F(\boldsymbol{u})$ follows from $\boldsymbol{v} \leq \boldsymbol{u}$ and $\neg E_F(\boldsymbol{v}, \boldsymbol{u})$. ∎

Proof of Theorem 2

Lemma 3 *Let F be a rectified sentence of signature σ (possibly containing function constants of positive arity), and let I be an interpretation of σ that satisfies F. If F is bounded w.r.t. I,*

$$I \models \exists \boldsymbol{u}(\boldsymbol{u} \leq \boldsymbol{p} \wedge \mathit{Ext\text{-}Loop}_F(\boldsymbol{u}) \wedge \mathit{NSES}_F(\boldsymbol{u}))$$

iff there is a finite loop Y of F w.r.t. I such that

$$I \models \Big(\bigwedge Y \wedge \mathit{NES}_F(Y) \Big).$$

Proof. *From left to right:* Assume

$$I \models \boldsymbol{q} \leq \boldsymbol{p} \wedge \text{Ext-Loop}_F(\boldsymbol{q}) \wedge \text{NSES}_F(\boldsymbol{q})$$

for some list of predicate names $\boldsymbol{q}$. Consider Y to be the set of vertices in the dependency graph of F w.r.t. I such that

$$p_i(\boldsymbol{\xi}^\diamond) \in Y \text{ iff } I \models q_i(\boldsymbol{\xi}^\diamond).$$

Since $I \models \text{Ext-Loop}_F(\boldsymbol{q})$, by Proposition 3 and Proposition 4, it follows that Y is an extended loop of F w.r.t. I. Since $I \models q_i(\boldsymbol{\xi}^\diamond)$ for all $p_i(\boldsymbol{\xi}^\diamond) \in Y$ and $I \models \boldsymbol{q} \leq \boldsymbol{p}$, it follows that I satisfies every atom in Y. Together with the assumption that F is bounded w.r.t. I, this implies that set Y is finite. Since $I \models \text{NSES}_F(\boldsymbol{q})$ and Y is finite, by Lemma 2, it follows that $I \models \text{NES}_F(Y)$.

From right to left: Consider any finite loop Y of F w.r.t. I. Assume

$$I \models \bigwedge Y \wedge \text{NES}_F(Y).$$

Let $\boldsymbol{q}$ be a list of predicate names such that

$$I \models q_i(\boldsymbol{\xi}^\diamond) \text{ iff } p_i(\boldsymbol{\xi}^\diamond) \in Y.$$

- $I \models \boldsymbol{q} \leq \boldsymbol{p}$ follows from the construction of $\boldsymbol{q}$ and $I \models \bigwedge Y$.
- Since Y is a loop of F w.r.t. I, by Proposition 3, $I \models \text{Loop}_F(\boldsymbol{q})$, and consequently, $I \models \text{Ext-Loop}_F(\boldsymbol{q})$.
- From $I \models \text{NES}_F(Y)$, by Lemma 2, $I \models \text{NSES}_F(\boldsymbol{q})$.

Consequently, $I \models \exists \boldsymbol{u}(\boldsymbol{u} \leq \boldsymbol{p} \wedge \text{Ext-Loop}_F(\boldsymbol{u}) \wedge \text{NSES}_F(\boldsymbol{u}))$. ∎

Theorem 2 *Let F be a rectified sentence of signature σ (possibly containing function constants of positive arity), and let I be an interpretation of σ that satisfies F. If F is bounded w.r.t. I, then the following conditions are equivalent to each other:*

(a) I satisfies $SM[F]$;

(b) for every nonempty finite set Y of atoms formed from predicate constants in $\sigma(F)$ and object names for $|I|$, I satisfies $LF_F(Y)$;

(c) for every finite loop Y of F w.r.t. I, I satisfies $LF_F(Y)$.

Proof. *Between (a) and (c):* By Theorem 3 and Lemma 3.

Between (b) and (c):

- *From (b) to (c):* Clear.
- *From (c) to (b):* Assume that I satisfies $\mathrm{LF}_F(L)$ for every finite loop L of F w.r.t I. Consider any nonempty finite set Y of atoms formed from predicate constants in $\sigma(F)$ and object names such that $I \models \bigwedge Y$. Let $\boldsymbol{q}$ be a list of predicate names such that

  $$I \models q_i(\boldsymbol{\xi}^\diamond) \text{ iff } p_i(\boldsymbol{\xi}^\diamond) \in Y.$$

  Since Y is nonempty, it is clear that $\mathrm{Nonempty}(\boldsymbol{q})$ follows. In view of Lemma 12, there is a list of predicate names $\boldsymbol{r}$ such that

  $$I \models \boldsymbol{r} \leq \boldsymbol{q} \wedge \text{Ext-Loop}_F(\boldsymbol{r}) \wedge \neg E_F(\boldsymbol{r}, \boldsymbol{q}). \tag{3.51}$$

  Consider Z to be the set of vertices in the dependency graph of F w.r.t. I such that

  $$p_i(\boldsymbol{\xi}^\diamond) \in Z \text{ iff } I \models r_i(\boldsymbol{\xi}^\diamond).$$

  Since $I \models \text{Ext-Loop}_F(\boldsymbol{r})$, by Proposition 3 and Proposition 4, Z is an extended loop of F w.r.t. I. Clearly, $I \models \bigwedge Z$ since $Z \subseteq Y$ and $I \models \bigwedge Y$. Since F is bounded w.r.t. I, and Z is satisfied by I, it follows that Z is a finite loop of F w.r.t. I. Since $I \models \boldsymbol{r} \leq \boldsymbol{q} \wedge \neg E_F(\boldsymbol{r}, \boldsymbol{q})$, Z is a subset of Y and, by Lemma 6, there is no edge

from Z to $Y \setminus Z$ in the dependency graph of F w.r.t. I. Since $I \models \mathrm{LF}_F(Z)$, we conclude that $I \models \neg\mathrm{NES}_F(Z)$, and by Lemma 2, $I \models \neg\mathrm{NSES}_F(\boldsymbol{r})$. From (3.51) and that $I \models \neg\mathrm{NSES}_F(\boldsymbol{r})$, by Lemma 11, we have $I \models \neg\mathrm{NSES}_F(\boldsymbol{q})$. By Lemma 2 again, $I \models \neg\mathrm{NES}_F(Y)$. Consequently, $I \models \mathrm{LF}_F(Y)$.

∎

Proof of Proposition 6

Proposition 6 *If a rectified formula F of signature σ is bounded, then F is bounded w.r.t. any interpretation of σ that satisfies CET_σ.*

Lemma 13 *For any terms t_1 and t_2 of signature σ, any interpretation I that satisfies CET_σ, and any substitution θ from object variables in t_1 and t_2 to object names such that $(t_1\theta)^I = (t_2\theta)^I$, Robinson's unification algorithm (Robinson, 1965), when applied to t_1 and t_2, returns a most general unifier (mgu) γ of t_1 and t_2 such that*

(a) $t_1\gamma = t_2\gamma$, and

(b) for every variable x in t_1 or t_2, $(x\gamma\theta)^I = (x\theta)^I$.

Proof. From the assumptions, by Lemma 5.1 from (Kunen, 1987), t_1 and t_2 are unifiable, in which case Robinson's algorithm returns a mgu for t_1 and t_2 that maps variables occurring in t_1 and t_2 into terms. Given this, part (b) can be proven by induction. ∎

The proof of Proposition 6 follows from the following lemma.

Lemma 14 *Let F be a rectified sentence of signature σ, and let I be an interpretation of σ that satisfies CET_σ. For any path*

$$\langle p_1(\boldsymbol{\xi}_1^\diamond), p_2(\boldsymbol{\xi}_2^\diamond), \ldots, p_k(\boldsymbol{\xi}_k^\diamond), p_{k+1}(\boldsymbol{\xi}_{k+1}^\diamond)\rangle \tag{3.52}$$

in the dependency graph of F w.r.t I, there is a path

$$\langle p_1(\boldsymbol{u}_1), p_2(\boldsymbol{u}_2), \ldots, p_k(\boldsymbol{u}_k), p_{k+1}(\boldsymbol{u}_{k+1})\rangle$$

in the first-order dependency graph of F with a substitution θ that maps object variables in $\boldsymbol{u}_i$ to object names such that $(\boldsymbol{u}_i\theta)^I = \boldsymbol{\xi}_i$ for all i.

Proof. Each edge $(p_i(\boldsymbol{\xi}_i^\diamond), p_{i+1}(\boldsymbol{\xi}_{i+1}^\diamond))$ in (3.52) is obtained from a pair of atoms $(p_i(\boldsymbol{t}_i), p_{i+1}(\boldsymbol{t}'_i))$ and a substitution θ_i such that $p_i(\boldsymbol{t}_i)$ depends on $p_{i+1}(\boldsymbol{t}'_i)$ in a rule of F, and

$$(\boldsymbol{t}_1\theta_1)^I = \boldsymbol{\xi}_1,\ \ (\boldsymbol{t}'_i\theta_i)^I = (\boldsymbol{t}_{i+1}\theta_{i+1})^I = \boldsymbol{\xi}_{i+1}(1 \leq i < k),\ \ (\boldsymbol{t}'_k\theta_k)^I = \boldsymbol{\xi}_{k+1}. \tag{3.53}$$

For simplicity we assume that each pair $(p_i(\boldsymbol{t}_i), p_{i+1}(\boldsymbol{t}'_i))$ considered above has no common variables with another pair by first renaming variables. This allows us to use one substitution $\theta = \theta_1 \ldots \theta_k$ in place of individual θ_i in the rest of the proof.

We will show by induction that, for each j where $j \in \{1 \ldots k\}$, there are substitutions σ_i^j $(1 \leq i \leq j)$ from variables in $\boldsymbol{t}_i$ and $\boldsymbol{t}'_i$ to terms such that

(a) $\langle p_1(\boldsymbol{t}_1)\sigma_1^j, p_2(\boldsymbol{t}_2)\sigma_2^j, \ldots, p_j(\boldsymbol{t}_j)\sigma_j^j, p_{j+1}(\boldsymbol{t}'_j)\sigma_j^j\rangle$ is a path in the first-order dependency graph of F, and

(b) $(\boldsymbol{t}_i\sigma_i^j\theta)^I = \boldsymbol{\xi}_i$ for all $1 \leq i \leq j$, and $(\boldsymbol{t}'_j\sigma_j^j\theta)^I = \boldsymbol{\xi}_{j+1}$.

When $j = 1$, we take σ_i^j to be an identity substitution. Clearly, conditions (a) and (b) are satisfied.

Otherwise, by I.H. we assume that, for some j in $\{1, \ldots, k-1\}$, there are substitutions $\sigma_1^j, \ldots, \sigma_j^j$ such that conditions (a) and (b) are satisfied. We will prove that there are substitutions σ_i^{j+1} $(1 \leq i \leq j+1)$ from variables in $\boldsymbol{t}_i$ and $\boldsymbol{t}'_i$ to terms such that

(a') $\langle p_1(\boldsymbol{t}_1)\sigma_1^{j+1}, p_2(\boldsymbol{t}_2)\sigma_2^{j+1}, \ldots, p_{j+1}(\boldsymbol{t}_{j+1})\sigma_{j+1}^{j+1}, p_{j+2}(\boldsymbol{t}'_{j+1})\sigma_{j+1}^{j+1}\rangle$ is a path in the first-order dependency graph of F, and

(b') $(\boldsymbol{t}_i\sigma_i^{j+1}\theta)^I = \boldsymbol{\xi}_i$ for all $1 \leq i \leq j+1$, and $(\boldsymbol{t}'_{j+1}\sigma_{j+1}^{j+1}\theta)^I = \boldsymbol{\xi}_{j+2}$.

From I.H., we have $(\boldsymbol{t}'_j\sigma_j^j\theta)^I = \boldsymbol{\xi}_{j+1}$ and from (3.53) we have $(\boldsymbol{t}_{j+1}\theta)^I = \boldsymbol{\xi}_{j+1}$. By Lemma 13 there is a substitution γ from variables in $\boldsymbol{t}'_j\sigma_j^j$ or $\boldsymbol{t}_{j+1}$ to terms such that $\boldsymbol{t}'_j\sigma_j^j\gamma = \boldsymbol{t}_{j+1}\gamma$ and for any variable x in $\boldsymbol{t}'_j\sigma_j^j$ or $\boldsymbol{t}_{j+1}$,

$$(x\gamma\theta)^I = (x\theta)^I. \tag{3.54}$$

We define σ_i^{j+1} as

- $\sigma_i^j\gamma$ when $1 \leq i \leq j$ and
- γ when $i = j+1$.

It is easy to check that condition (a') is satisfied. To check that condition (b') is satisfied, consider any variable x in the set

$$\{\boldsymbol{t}_1\sigma_1^j, \boldsymbol{t}_2\sigma_2^j, \ldots, \boldsymbol{t}_j\sigma_j^j, \boldsymbol{t}'_j\sigma_j^j, \boldsymbol{t}_{j+1}, \boldsymbol{t}'_{j+1}\}. \tag{3.55}$$

If x is in $\boldsymbol{t}'_j\sigma_j^j$ or $\boldsymbol{t}_{j+1}$, by (3.54), $(x\gamma\theta)^I = (x\theta)^I$. Otherwise, since γ does not change the variables that are not in $\boldsymbol{t}'_j\sigma_j^j$ or $\boldsymbol{t}_{j+1}$, $(x\gamma\theta)^I = (x\theta)^I$. Consequently, for any variable x in (3.55), we get $(x\gamma\theta)^I = (x\theta)^I$. It remains to check the following.

- For $1 \leq i \leq j$, $(\boldsymbol{t}_i\sigma_i^{j+1}\theta)^I = (\boldsymbol{t}_i\sigma_i^j\gamma\theta)^I = (\boldsymbol{t}_i\sigma_i^j\theta)^I$. The last one is equal to $\boldsymbol{\xi}_i$ by I.H.
- $(\boldsymbol{t}_{j+1}\sigma_{j+1}^{j+1}\theta)^I = (\boldsymbol{t}_{j+1}\gamma\theta)^I = (\boldsymbol{t}_{j+1}\theta)^I$. The last one is equal to $\boldsymbol{\xi}_{j+1}$ by (3.53).
- $(\boldsymbol{t}'_{j+1}\sigma_{j+1}^{j+1}\theta)^I = (\boldsymbol{t}'_{j+1}\gamma\theta)^I = (\boldsymbol{t}'_{j+1}\theta)^I$. The last one is equal to $\boldsymbol{\xi}_{j+2}$ by (3.53).

∎

Proof of Proposition 7

Proposition 7 *For any rectified sentence F of signature σ and for any interpretation I of σ that satisfies CET_σ, I is a model of*

$$\{LF_F(Y) \mid Y \textit{ is a finite first-order loop of } F\}$$

iff I is a model of

$$\{LF_F(Y) \mid Y \textit{ is a finite loop of } F \textit{ w.r.t. } I\}.$$

The proof follows immediately from the following fact and Lemma 15.

Fact 1 *Let F be a rectified sentence of signature σ, and let I be an interpretation of σ. For any first-order loop Y of F and any substitution θ that maps variables in Y to object names, $Y' = \{p_i(\boldsymbol{\xi}^\diamond) \mid p_i(\boldsymbol{t}) \in Y\theta,\ \boldsymbol{t}^I = \boldsymbol{\xi}\}$ is a loop of F w.r.t. I.*

Lemma 15 *Let F be a rectified sentence of signature σ, and let I be an interpretation of σ. If I satisfies CET_σ, then, for any finite loop Y' of F w.r.t. I, there is a finite loop Y of F with a substitution θ that maps variables in Y to object names such that $Y' = \{p_i(\boldsymbol{\xi}^\diamond) \mid p_i(\boldsymbol{t}) \in Y,\ (\boldsymbol{t}\theta)^I = \boldsymbol{\xi}\}$.*

Proof. Without loss of generality, consider a path

$$\langle p_1(\boldsymbol{\xi}_1^\diamond), p_2(\boldsymbol{\xi}_2^\diamond), \ldots, p_k(\boldsymbol{\xi}_k^\diamond), p_1(\boldsymbol{\xi}_1^\diamond)\rangle$$

($k \geq 1$) in the dependency graph of F w.r.t. I that consists of the vertices in Y'. Since $I \models CET_\sigma$, by Lemma 14, there is a path

$$\langle p_1(\boldsymbol{u}_1), p_2(\boldsymbol{u}_2), \ldots, p_k(\boldsymbol{u}_k), p_1(\boldsymbol{u}_{k+1})\rangle$$

in the first-order dependency graph of F with a substitution θ that maps variables in $\boldsymbol{u}_i$ to object names such that $(\boldsymbol{u}_i\theta)^I = \boldsymbol{\xi}_i$ for all $1 \leq i \leq k$, and $(\boldsymbol{u}_{k+1}\theta)^I = \boldsymbol{\xi}_1$. Since $(\boldsymbol{u}_{k+1}\theta)^I = (\boldsymbol{u}_1\theta)^I$, by Lemma 13, there is a unifier γ for $\boldsymbol{u}_{k+1}$ and $\boldsymbol{u}_1$ such that, for any variable x in $\boldsymbol{u}_{k+1}$ or $\boldsymbol{u}_1$, $(x\gamma\theta)^I = (x\theta)^I$. Consequently,

$$\{p_1(\boldsymbol{u}_1\gamma), p_2(\boldsymbol{u}_2\gamma), \ldots, p_k(\boldsymbol{u}_k\gamma)\}$$

induces a finite strongly connected subgraph such that $(\boldsymbol{u}_i\gamma\theta)^I = (\boldsymbol{u}_i\theta)^I = \boldsymbol{\xi}_i$. ∎

Proof of Proposition 8

Proposition 8 *If a rectified formula F in normal form is bounded, then F is bounded w.r.t. any interpretation.*

The proof follows from the following lemma.

Lemma 16 *Let F be a rectified sentence of signature σ in normal form, and let I be an interpretation of σ. For any path*

$$\langle p_1(\boldsymbol{\xi}_1^\diamond), p_2(\boldsymbol{\xi}_2^\diamond), \ldots, p_k(\boldsymbol{\xi}_k^\diamond), p_{k+1}(\boldsymbol{\xi}_{k+1}^\diamond)\rangle$$

in the dependency graph of F w.r.t I, there exists a path

$$\langle p_1(\boldsymbol{u}_1), p_2(\boldsymbol{u}_2), \ldots, p_k(\boldsymbol{u}_k), p_{k+1}(\boldsymbol{u}_{k+1})\rangle$$

in the first-order dependency graph of F with a substitution θ that maps object variables in $\boldsymbol{u}_i$ to object names such that $(\boldsymbol{u}_i\theta)^I = \boldsymbol{\xi}_i$ for all i, and $\boldsymbol{u}_1$ is a list of object variables.

Proof. The proof is similar to the proof of Lemma 14 except that we do not require that I satisfy CET_σ. Instead, the existence of a unifier γ for $\boldsymbol{t}'_j\sigma^j_j$ and $\boldsymbol{t}_{j+1}$ is ensured by the assumption on normal form that $\boldsymbol{t}_{j+1}$ is a list of variables and the assumption that $\boldsymbol{t}'_j\sigma^j_j$ contains none of those variables (due to variable renaming).

Proof of Proposition 9

Proposition 9 *If a rectified sentence F in normal form is bounded, then for any interpretation I, I is a model of*

$$\{LF_F(Y) \mid Y \textit{ is a finite first-order loop of } F\}$$

iff I is a model of

$$\{LF_F(Y) \mid Y \textit{ is a finite loop of } F \textit{ w.r.t. } I\}.$$

The proof follows from Fact 1 and the following lemma.

Lemma 17 *If a rectified sentence F in normal form is bounded, then for any finite loop Y' of F w.r.t. I, there is a finite loop Y of F with a substitution θ that maps variables in Y to object names such that $Y' = \{p_i(\boldsymbol{\xi}^\diamond) \mid p_i(\boldsymbol{t}) \in Y,\ (\boldsymbol{t}\theta)^I = \boldsymbol{\xi}\}$.*

Proof. Let Y' be a finite loop of F w.r.t. I. Without loss of generality, there is a path

$$\langle p_1(\boldsymbol{\xi}^\diamond_1), p_2(\boldsymbol{\xi}^\diamond_2), \ldots, p_k(\boldsymbol{\xi}^\diamond_k), p_1(\boldsymbol{\xi}^\diamond_1)\rangle$$

($k \geq 1$) in the dependency graph of F w.r.t. I that consists of the vertices in Y'. Since F is in normal form, by Lemma 16, there are a path

$$\langle p_1(\boldsymbol{u}_1), p_2(\boldsymbol{u}_2), \ldots, p_k(\boldsymbol{u}_k), p_1(\boldsymbol{u}_{k+1})\rangle \tag{3.56}$$

in the first-order dependency graph of F, where $\boldsymbol{u}_1$ consists of object variables only, and a substitution θ that maps variables in $\boldsymbol{u}_i$ to object names such that $(\boldsymbol{u}_i\theta)^I = \boldsymbol{\xi}_i$ for all $1 \leq i \leq k$, and $(\boldsymbol{u}_{k+1}\theta)^I = \boldsymbol{\xi}_1$. There are two cases to consider.

- *Case 1:* There is a unifier γ for $\boldsymbol{u}_1$ and $\boldsymbol{u}_{k+1}$ that maps variables in $\boldsymbol{u}_1$ to terms in $\boldsymbol{u}_{k+1}$ so that $\boldsymbol{u}_1\gamma = \boldsymbol{u}_{k+1}$. It follows that, for any variable x in $\boldsymbol{u}_{k+1}$ or $\boldsymbol{u}_1$, $(x\gamma\theta)^I = (x\theta)^I$. Consequently,

  $$\{p_1(\boldsymbol{u}_1\gamma), p_2(\boldsymbol{u}_2\gamma), \ldots, p_k(\boldsymbol{u}_k\gamma)\}$$

  induces a finite strongly connected subgraph such that $(\boldsymbol{u}_i\gamma\theta)^I = (\boldsymbol{u}_i\theta)^I = \boldsymbol{\xi}_i$.

- *Case 2:* There is no such unifier γ.

  Consider another path

  $$\langle p_1(\boldsymbol{v}_1), p_2(\boldsymbol{v}_2), \ldots, p_k(\boldsymbol{v}_k), p_1(\boldsymbol{v}_{k+1})\rangle$$

  that is obtained similar to (3.56) except that the variables in the path are disjoint from the variables in (3.56). Clearly, there is a unifier γ' for $\boldsymbol{u}_{k+1}$ and $\boldsymbol{v}_1$ that maps the variables $\boldsymbol{v}_1$ to terms, so that

  $$\langle p_1(\boldsymbol{u}_1), p_2(\boldsymbol{u}_2), \ldots, p_k(\boldsymbol{u}_k), p_1(\boldsymbol{v}_1\gamma'), p_2(\boldsymbol{v}_2\gamma'), \ldots, p_k(\boldsymbol{v}_k\gamma')\rangle$$

  is another path in the first-order dependency graph of F. It is clear that using the same construction repeatedly, we can form an infinite path that visits infinitely many vertices in the first-order dependency graph. But this contradicts the assumption that F is bounded.

∎

Proof of Proposition 11

We will use the following lemma in this section and the next section, which extends Theorem 2 of (Chen et al., 2006) that provides a few equivalent conditions for a program to have a finite complete set of loops to a disjunctive program and a sentence.

Lemma 18 *(Chen et al., 2006, Theorem 2) For any formula F that contains no function constants of positive arity, the following conditions are equivalent:*

(a) F has a finite complete set of loops.

(b) There is a nonnegative integer N such that for every loop L of F, the number of variables in L is bounded by N.

(c) For any loop L of F and any atom A_1 and A_2 in L, the variables occurring in A_1 are identical to the variables occurring in A_2.

(d) For any loop L of $Ground_{\sigma(F)\cup\{c_1,c_2\}}(F)$ where c_1, c_2 are two new object constants, there are no two atoms A_1 and A_2 in L such that A_1 mentions c_1 but A_2 does not or A_1 mentions c_2 but A_2 does not.

Proposition 11 *For any rectified formula F that contains no function constants of positive arity, F is bounded iff F has a finite complete set of loops.*

Proof. *From left to right:* Assume that F is bounded. Then every loop of F is finite. It follows that there exists a nonnegative integer N such that the number of variables in any loop is bounded by N. By Lemma 18 (b), F has a finite complete set of loops.

From right to left: Assume that F has a finite complete set of loops and, for the sake of contradiction, assume that it is not bounded. Without loss of generality, there is an infinite path

$$\langle p_1(\boldsymbol{t}_1)\sigma_1, p_2(\boldsymbol{t}_2)\sigma_2, \ldots\rangle \tag{3.57}$$

in the first-order dependency graph of F that visits infinitely many vertices, where $p_i(\boldsymbol{t}_i)$ are atoms occurring in F and σ_i are substitutions.

Since F is a finite string, it contains finitely many atoms. It follows that there is an atom $p_i(\boldsymbol{t}_i)$ occurring in F with arbitrarily many substitutions σ such that atoms $p_i(\boldsymbol{t}_i)\sigma$ are contained in (3.57). Without loss of generality, consider the path

$$\langle p_i(\boldsymbol{t}_i)\sigma_i, p_{i+1}(\boldsymbol{t}_{i+1})\sigma_{i+1}, \ldots, p_i(\boldsymbol{t}_i)\sigma_k\rangle$$

that is contained in (3.57), where σ_k and σ_i agree on substituting object constants for variables in $\boldsymbol{t}_i$. Since $\boldsymbol{t}_i\sigma_i$ and $\boldsymbol{t}_i\sigma_k$ contain no function constant, there exists a substitution σ_0 that maps variables in $\boldsymbol{t}_i\sigma_k$ to terms in $\boldsymbol{t}_i\sigma_i$ so that $\boldsymbol{t}_i\sigma_k\sigma_0 = \boldsymbol{t}_i\sigma_i$. Consequently,

$$\{p_i(\boldsymbol{x}_i)\sigma_i\sigma_0, p_{i+1}(\boldsymbol{x}_{i+1})\sigma_{i+1}\sigma_0, \ldots, p_i(\boldsymbol{x}_i)\sigma_k\sigma_0\}$$

is a loop of F. Since the length of the path is arbitrarily large, there are arbitrarily many variables occurring in the loop. By Lemma 18 (b), it follows that F has no finite complete set of loops. ∎

Proof of Proposition 10

Proposition 10 *For any rectified sentence F (allowing function constants of positive arity),*

(a) checking whether F is bounded is not decidable;

(b) checking whether F is atomic-tight is not decidable.

If F contains no function constants of positive arity,

(c) checking whether F is bounded is decidable;

(d) checking whether F is atomic-tight is decidable.

Proof of Part (a) and (b): We show the proof of Part (a) first. The proof repeats, with minor modifications, the argument from the proof of Theorem 26 from (Bonatti, 2004), which considers the following program $\Pi_{\mathcal{M}}$ to simulate deterministic Turing machines $\mathcal{M}$.

$t(s, L, v, [V \mid R], C) \leftarrow t(s', [v' \mid L], V, R, C{+}1)$	for all instr.$\langle s, v, v', s',$ right$\rangle$
$t(s, L, v, [\,], C) \leftarrow t(s', [v' \mid L], b, [\,], C{+}1)$	for all instr.$\langle s, v, v', s',$ right$\rangle$
$t(s, [V \mid L], v, R, C) \leftarrow t(s', L, V, [v' \mid R], C{+}1)$	for all instr.$\langle s, v, v', s',$ left$\rangle$
$t(s, [\,], v, R, C) \leftarrow t(s', [\,], b, [v' \mid R], C{+}1)$	for all instr.$\langle s, v, v', s',$ left$\rangle$
$t(s, L, v, R, C)$	for all final states s.

The Halting problem can be reduced to the problem of checking bounded formulas. More precisely, we show that $\Pi_{\mathcal{M}}$ is bounded iff $\mathcal{M}$ terminates from every configuration.

We first establish the following facts:

(i) for every non-terminating computation of $\mathcal{M}$ on input x, there is a corresponding infinite path in the first-order dependency graph of $\Pi_{\mathcal{M}}$ that visits infinitely many vertices;

(ii) if there is an infinite path in the first-order dependency graph of $\Pi_{\mathcal{M}}$, then there is an infinite path starting with a legal encoding of an input and corresponds to a non-terminating computation of $\mathcal{M}$.

Fact (i) is immediate from the definition of $\Pi_{\mathcal{M}}$: Note that the step counter (the last argument of t) ensures that the dependency graph is acyclic. Then, whenever $\mathcal{M}$ falls into a cycle, the dependency graph contains an infinite acyclic path that visits infinitely many vertices and hence the program is not bounded.

Fact (ii) can be proven as follows. Assume that there is an infinite path in the dependency graph. We observe that the first argument of every vertex in the path must be a legal state and the third argument of every vertex must be a legal tape value. Otherwise, there is no outgoing edge from the vertices in the dependency graph of $\Pi_{\mathcal{M}}$. So only the second, fourth and fifth arguments can contain variables or illegal values which were obtained from substitutions from the variables L, R, V and C. In this case, we can easily find substitutions from these variables or illegal values to legal values and apply them uniformly along the path, so that we obtain another infinite path starting from the vertex that correctly encodes a configuration of $\mathcal{M}$ and thus $\mathcal{M}$ has a corresponding non-terminating computation.

The claim follows immediately from the two facts: if $\mathcal{M}$ does not terminate on some computation, then by (i), $\Pi_{\mathcal{M}}$ is unbounded. If $\Pi_{\mathcal{M}}$ is unbounded, then by (ii), $\mathcal{M}$ does not terminate.

The same proof works for Part (b) as well. This is because the step counter (the last argument of t) ensures that the dependency graph is acyclic. Consequently, every infinite path in the dependency graph visits infinitely many vertices, so that $\Pi_{\mathcal{M}}$ is atomic-tight iff $\Pi_{\mathcal{M}}$ is bounded. ∎

Proof of Part (c): In view of the equivalence between (a) and (d) in Lemma 18, checking

whether a formula F containing no function constants of positive arity has a finite complete set of loops can be done by examining a finite number of loops from a finite dependency graph, which is decidable. By Proposition 11, it follows that checking whether F is bounded is decidable. ∎

Proof of Part (d): For any sentence F that has no function constants of positive arity and any finite set $\boldsymbol{c}$ of object constants, $\mathrm{Ground}_{\boldsymbol{c}}(F)$ is defined recursively. If F is an atomic formula then$\mathrm{Ground}_{\boldsymbol{c}}(F)$ is F. The function $\mathrm{Ground}_{\boldsymbol{c}}$ commutes with all propositional connectives; quantifiers turn into finite conjunctions and disjunctions over all object constants occurring in $\boldsymbol{c}$.

Lemma 19 *Let $\boldsymbol{c}$ be the set consisting of all object constants occurring in F, and possibly a new object constant if F contains no object constants. F has a non-trivial loop iff $Ground_{\boldsymbol{c}}(F)$ has a non-trivial loop.*

Proof. In order to check whether F is atomic-tight, we first check whether F is bounded, which is decidable. If F is not bounded, then F is not atomic-tight. Otherwise, in view of Lemma 19, checking whether F is atomic-tight is the same as checking whether $\mathrm{Ground}_{\boldsymbol{c}}(F)$ is atomic-tight. Since F contains no function constants of positive arity, the dependency graph of $\mathrm{Ground}_{\boldsymbol{c}}(F)$ is finite. So it is decidable to check whether the dependency graph of $\mathrm{Ground}_{\boldsymbol{c}}(F)$ has a non-trivial loop. ∎

Proof of Proposition 13

Lemma 20 *Let F be a formula and Y a set of atoms. If no predicate constant occurring in Y occurs strictly positively in F, then $NES_F(Y)$ is equivalent to F.*

Proof. By induction. ∎

Proposition 13 *Let Π be a program with quantifiers, F the FOL-representation of Π, and Y a finite set of atoms. Under the assumption Π, formula $QES_\Pi(Y)$ is equivalent to*

$\neg NES_F(Y)$. *If* Π *is a disjunctive program in normal form, then* $QES_\Pi(Y)$ *is also equivalent to* $ES_\Pi(Y)$ *under the assumption* Π.

Proof. *Between* $QES_\Pi(Y)$ *and* $\neg NES_F(Y)$*:* $\neg\mathrm{NES}_F(Y)$ is

$$\neg \bigwedge_{H\leftarrow G\in\Pi} \forall \boldsymbol{x}[(G\to H)\wedge(\mathrm{NES}_G(Y)\to \mathrm{NES}_H(Y))]. \tag{3.58}$$

Under the assumption F, formula (3.58) is equivalent to

$$\bigvee_{H\leftarrow G\in\Pi} \exists \boldsymbol{x}(\mathrm{NES}_G(Y)\wedge\neg\mathrm{NES}_H(Y)). \tag{3.59}$$

In view of Lemma 20, if H does not contain any strictly positive occurrence of a predicate constant that belongs to Y, $\mathrm{NES}_H(Y)$ is equivalent to H. Also, it follows from Lemma 2 and Lemma 8 that $\mathrm{NES}_G(Y)$ implies G. So $\mathrm{NES}_G(Y)\wedge\neg\mathrm{NES}_H(Y)$ conflicts the assumption $G\to H$ when H does not contain any strictly positive occurrence of a predicate constant that belongs to Y. As a result, under the assumption F, formula (3.59) is equivalent to the disjunction of

$$\exists \boldsymbol{x}(\mathrm{NES}_G(Y)\wedge\neg\mathrm{NES}_H(Y)) \tag{3.60}$$

for all rules $H\leftarrow G$, where H contains a strictly positive occurrence of a predicate constant that belongs to Y. Note that G and H are formulas such that every occurrence of an implication in G and H belongs to a negative formula. By Lemma 4, (3.60) is equivalent to $\mathrm{QES}_\Pi(Y)$.

Between $QES_\Pi(Y)$ *and* $ES_\Pi(Y)$*:* When Π is a disjunctive program, $\mathrm{QES}_\Pi(Y)$ is the disjunction of

$$\exists \boldsymbol{z}\Bigg(B\wedge N\wedge \bigwedge_{\substack{p(\boldsymbol{t})\in B\\ p(\boldsymbol{t}')\in Y}} (\boldsymbol{t}\neq\boldsymbol{t}')\wedge\neg\Big(\bigvee_{p(\boldsymbol{t})\in A}(p(\boldsymbol{t})\wedge\bigwedge_{p(\boldsymbol{t}')\in Y}\boldsymbol{t}\neq\boldsymbol{t}')\Big)\Bigg) \tag{3.61}$$

over all rules (3.8) such that A contains a predicate constant that occurs in Y, where $\boldsymbol{z}$ is a list of variables in (3.8) but not in Y. On the other hand, $\mathrm{ES}_\Pi(Y)$ is the disjunction of

$$\begin{array}{l}\exists \boldsymbol{z}'\Bigg(B\sigma\wedge N\sigma\wedge\bigwedge_{\substack{p(\boldsymbol{t})\in B\sigma\\ p(\boldsymbol{t}')\in Y}}(\boldsymbol{t}\neq\boldsymbol{t}')\\ \wedge\neg\big(\bigvee_{p(\boldsymbol{t})\in A\sigma}(p(\boldsymbol{t})\wedge\bigwedge_{p(\boldsymbol{t}')\in Y}\boldsymbol{t}\neq\boldsymbol{t}')\big)\Bigg)\end{array} \tag{3.62}$$

over all rules (3.8) such that A contains a predicate constant that occurs in Y and $A\sigma \cap Y \neq \emptyset$, where $\boldsymbol{z}'$ is a list of variables in $A\sigma \leftarrow B\sigma, N\sigma$ but not in Y.

It is clear that (3.62) implies (3.61). To prove that (3.61) implies (3.62), assume

$$B \wedge N \wedge \bigwedge_{\substack{p(\boldsymbol{t})\in B \\ p(\boldsymbol{t}')\in Y}} (\boldsymbol{t} \neq \boldsymbol{t}') \wedge \neg\Big(\bigvee_{p(\boldsymbol{t})\in A} (p(\boldsymbol{t}) \wedge \bigwedge_{p(\boldsymbol{t}')\in Y} \boldsymbol{t} \neq \boldsymbol{t}')\Big) \tag{3.63}$$

and consider two cases.

If $\bigwedge_{p(\boldsymbol{t}')\in Y} \boldsymbol{t} \neq \boldsymbol{t}'$ for all $p(\boldsymbol{t}) \in A$, then (3.63) is equivalent to

$$B \wedge N \wedge \bigwedge_{\substack{p(\boldsymbol{t})\in B \\ p(\boldsymbol{t}')\in Y}} (\boldsymbol{t} \neq \boldsymbol{t}') \wedge \neg \bigvee_{p(\boldsymbol{t})\in A} p(\boldsymbol{t})$$

which contradicts the assumption Π.

Otherwise, there exists $p(\boldsymbol{t}) \in A$ and $p(\boldsymbol{t}') \in Y$ such that $\boldsymbol{t} = \boldsymbol{t}'$. Since Π is in normal form, there exists σ that maps $\boldsymbol{t}$ to $\boldsymbol{t}'$, so that $A\sigma \cap Y \neq \emptyset$. Consequently, (3.63) is equivalent to

$$B\sigma \wedge N\sigma \wedge \bigwedge_{\substack{p(\boldsymbol{t})\in B\sigma \\ p(\boldsymbol{t}')\in Y}} (\boldsymbol{t} \neq \boldsymbol{t}') \wedge \neg\Big(\bigvee_{p(\boldsymbol{t})\in A\sigma} (p(\boldsymbol{t}) \wedge \bigwedge_{p(\boldsymbol{t}')\in Y} \boldsymbol{t} \neq \boldsymbol{t}')\Big).$$

Thus the claim follows. ∎

Proof of Proposition 16

Proposition 16 *Let P be an LW-program and let F be the FOL-representation of the set of rules in P. The following conditions are equivalent to each other:*

(a) I is an answer set of P in the sense of (Lin & Wang, 2008);

(b) I is a P-interpretation that satisfies $SM[F]$;

(c) I is a P-interpretation that satisfies F and the loop formulas of Y for all loops Y of F w.r.t. I.

Given a program Π, $\mathrm{Norm}(\Pi)$ is a normal form of Π and $\mathrm{Ground}(\Pi)$ is a ground program obtained from Π as described in (Lin & Wang, 2008). The proof of Proposition 16 follows from the following lemma. We refer readers to (Lin & Wang, 2008) for the definition of $\mathrm{ES}(\cdot,\cdot,\cdot)$ defined there.

Lemma 21 *For any program* Π *and any set* Y *of ground atoms,* $ES_{Norm(\Pi)}(Y)$ *is equivalent to* $\bigvee_{p(\boldsymbol{c})\in Y} ES(p(\boldsymbol{c}), Y, Ground(\Pi))$.

Proof. By definition, $\mathrm{ES}_{Norm(\Pi)}(Y)$ is

$$\bigvee_{\substack{p(\boldsymbol{x})\leftarrow B,N,\boldsymbol{x}=\boldsymbol{t} \text{ is in } Norm(\Pi)\\ \theta:p(\boldsymbol{x})\theta\in Y}} \exists \boldsymbol{z}\Big(B\theta \wedge N\theta \wedge \boldsymbol{x}\theta = \boldsymbol{t}\theta \wedge \bigwedge_{\substack{q(\boldsymbol{t})\in B\theta\\ q(\boldsymbol{t}')\in Y}} (\boldsymbol{t} \neq \boldsymbol{t}')\Big), \tag{3.64}$$

where $\boldsymbol{x}$ is a list of distinct object variables, θ is a substitution that maps variables in $\boldsymbol{x}$ to object constants occurring in Y, and $\boldsymbol{z}$ is the list of all variables that occur in the rule $p(\boldsymbol{x})\theta \leftarrow B\theta, N\theta, \boldsymbol{x}\theta = \boldsymbol{t}\theta$. (3.64) is equivalent to

$$\bigvee_{\substack{p(\boldsymbol{t})\leftarrow B,N\in\Pi\\ p(\boldsymbol{c})\in Y}} \exists \boldsymbol{z}'\Big(B \wedge N \wedge \boldsymbol{t} = \boldsymbol{c} \wedge \bigwedge_{\substack{q(\boldsymbol{t})\in B\\ q(\boldsymbol{t}')\in Y}} (\boldsymbol{t} \neq \boldsymbol{t}')\Big), \tag{3.65}$$

where $\boldsymbol{z}'$ is the list of all variables that occur in the rule $p(\boldsymbol{t}) \leftarrow B, N$. In turn, (3.65) is equivalent to

$$\bigvee_{\substack{p(\boldsymbol{d})\leftarrow B',N'\in Ground(\Pi)\\ p(\boldsymbol{c})\in Y}} \Big(B' \wedge N' \wedge \boldsymbol{d} = \boldsymbol{c} \wedge \bigwedge_{\substack{q(\boldsymbol{t}_g)\in B'\\ q(\boldsymbol{t}')\in Y}} (\boldsymbol{t}_g \neq \boldsymbol{t}')\Big). \tag{3.66}$$

Note that when $\boldsymbol{d}$ does not cover $\boldsymbol{c}$, there exists $d_i \in \boldsymbol{d}$ such that d_i mentions only constants and pre-interpreted functions and d_i can not be evaluated to c_i independent of interpretations. In that case, $\boldsymbol{d} = \boldsymbol{c}$ is equivalent to $\bot$. Thus (3.66) is equivalent to

$$\bigvee_{p(\boldsymbol{c})\in Y} \bigvee_{\substack{p(\boldsymbol{d})\leftarrow B',N'\in Ground(\Pi)\\ p(\boldsymbol{d}) \text{ can cover } p(\boldsymbol{c})}} \Big(B' \wedge N' \wedge \boldsymbol{d} = \boldsymbol{c} \wedge \bigwedge_{\substack{q(\boldsymbol{t}_g)\in B'\\ q(\boldsymbol{t}')\in Y}} (\boldsymbol{t}_g \neq \boldsymbol{t}')\Big), \tag{3.67}$$

which is essentially $\bigvee_{p(\boldsymbol{c})\in Y} \mathrm{ES}(p(\boldsymbol{c}), Y, \mathrm{Ground}(\Pi))$. ∎

Chapter 4

ON SEMANTICS OF AGGREGATES

In this chapter we study and reformulate different semantics of aggregates by viewing them as short-hands for propositional formulas in the stable model semantics. For the FLP semantics, a novel reductive reformulation in terms of propositional formulas is presented. For the PDB-SPT semantics and the Ferraris semantics, which already have propositional formula characterization of aggregates, we show that each semantics can be reformulated according to the other's approach. Such uniform characterization has some merits. First, it provides new insights into each of the semantics and helps us compare and relate them. Second, all the important results already established for the propositional stable model semantics, such as the strong equivalence (Lifschitz et al., 2001; Ferraris, 2005) and loop formulas (Ferraris et al., 2006), can be immediately applied to programs with aggregates by first representing them as the corresponding propositional formulas. Third, while it does not appear immediate to extend to disjunctive programs, the non-reductive approaches for the PDB-SPT semantics, such as the T_p operator based one with conditional satisfaction, or Φ_p^{aggr} operator based one, the extension is straightforward in the reductive approach.

The reductive approach also guides us to define loop formulas in the form of "aggregate formulas," in which aggregates are treated like usual formulas. In (Liu & Truszczynski, 2006; You & Liu, 2008), the authors show that such loop formulas can be encoded as Pseudo-Boolean (PB) constraints and can be effectively computed by PB solvers. However, the result of (Liu & Truszczynski, 2006) is limited to monotone and convex constraints, and the PDB-SPT semantics. Here we present aggregate loop formulas for the Ferraris and the FLP semantics guided by the reductive approach. This part is also a further extension of the idea of unfounded sets for the FLP semantics studied in (Faber, 2005), where the author provided a model-theoretic account of loop formulas but left the definition of loops and loop formulas as future work.

We also present generalizations of the Ferraris semantics and FLP semantics. This is done by extension and modification of the stable model operator SM given in (Ferraris, Lee, & Lifschitz, 2011b) and by adopting the notion of satisfaction extended to aggregates

as in the FLP semantics. The generalizations avoid grounding and fixpoints and allows non-Herbrand models to be considered. This allows us to show how the FLP semantics is related to the first-order stable model semantics. When we consider Herbrand models, the two generalized semantics agree with their propositional counter-part. We study the precise relationship which properly generalizes the results in the propositional case.

In the next section we first review the syntax and three representative semantics of programs with aggregates. In Section 4.2, we show our reformulations and compare the three semantics based on the reformulations. In Section 4.3 we extended the theorem on loop formulas to allow aggregates for the two semantics. Section 4.4 introduces the syntax and semantics of aggregate formulas. In Section 4.5 and Section 4.6, we provide the first-order semantics of aggregates for the Ferraris semantics and the FLP semantics that applies to aggregate formulas. Section 4.7 compares and relates the two semantics.

4.1 Syntax and Existing Semantics of Programs with Aggregates

Syntax of Programs with Aggregates

Following (Lee & Meng, 2009; Ferraris & Lifschitz, 2010), by a *number* we understand an element of some fixed set **Num**. For example, **Num** is $\boldsymbol{Z} \cup \{+\infty, -\infty\}$, where $\boldsymbol{Z}$ is the set of integers. An *aggregate function* is a partial function from the class of multisets to **Num**. The domain of an aggregate function is defined as usual. For instance, COUNT is defined for any multisets; SUM, TIMES, MIN and MAX are defined for multisets of numbers; SUM is undefined for multisets containing infinitely many positive integers and infinitely many negative integers.

An *aggregate* is of the form

$$\mathrm{OP}\langle \boldsymbol{x}.F(\boldsymbol{x})\rangle \succeq b \tag{4.1}$$

where

- OP is a symbol for an *aggregate function* op;
- $\boldsymbol{x}$ is a nonempty list of distinct object variables;
- $F(\boldsymbol{x})$ is an arbitrary quantifier-free formula;

- $\succeq$ is a symbol for a binary relation over integers, such as $\leq$, $\geq$, $<$, $>$, $=$, $\neq$;
- b is an integer constant.

A more general definition of an aggregate is presented later.

A *rule (with aggregates)* is an expression of the form

$$A_1; \ldots; A_l \leftarrow E_1, \ldots, E_m, \text{not } E_{m+1}, \ldots, \text{not } E_n \tag{4.2}$$

($l \geq 0$; $n \geq m \geq 0$), where each A_i is an atomic formula and each E_i is an atomic formula or an aggregate. A *program (with aggregates)* is a finite set of rules.

Example 2 continued In the following, we will revisit the program Π_1 from Section 2.3.

$$\begin{aligned} p(2) &\leftarrow \text{not SUM}\langle x.p(x)\rangle < 2 \\ p(-1) &\leftarrow \text{SUM}\langle x.p(x)\rangle \geq 0 \\ p(1) &\leftarrow p(-1). \end{aligned}$$

FLP Semantics

The FLP semantics (Faber et al., 2004) is based on a modified definition of traditional reduct and the notion of satisfaction extended to aggregates.[1]

Notation: Given a multiset of object constants $\{\!\{c_1, \ldots, c_n\}\!\}$ and an integer constant b,

$$\text{OP}\langle\{\!\{c_1, \ldots, c_n\}\!\}\rangle \succeq b$$

if

- multiset $\{\!\{c_1, \ldots, c_n\}\!\}$ is in the domain of the aggregate function op (corresponding to OP), and
- $op(\{\!\{c_1, \ldots, c_n\}\!\}) \succeq b$.[2]

[1] The syntax from (Faber et al., 2004) is more restrictive than what we consider here: There $F(\boldsymbol{x})$ in (4.1) is required to be a conjunction of atoms.

[2] Here, we abuse the notation. We identify an integer constant with the corresponding integer, and $\succeq$ with the corresponding relation.

OP$\langle \{\!\{c_1, \ldots, c_n\}\!\} \rangle \not\succeq b$ if it is not the case that OP$\langle \{\!\{c_1, \ldots, c_n\}\!\} \rangle \succeq b$.

For any list of object constants $\boldsymbol{c}$, by $\boldsymbol{c}[1]$ we denote the first element of $\boldsymbol{c}$. Consider any aggregate (4.1) occurring in Π and any Herbrand interpretation I of $\sigma(\Pi)$. Let S_I be the multiset consisting of all $\boldsymbol{c}[1]$ such that

- $\boldsymbol{c}$ is a list of object constants of $\sigma(\Pi)$ whose length is the same as the length of $\boldsymbol{x}$, and
- I satisfies $F(\boldsymbol{c})$.

We say that I *satisfies* the aggregate expression (4.1) if OP$\langle S_I \rangle \succeq b$. For instance, an Herbrand interpretation $\{p(a)\}$ satisfies COUNT$\langle x.p(x) \rangle > 0$ but does not satisfy SUM$\langle x.p(x) \rangle > 0$ because $\{\!\{a\}\!\}$ is not in the domain of SUM.

The definition of the FLP reduct and FLP answer set is the same as in Section 2.4. the FLP reduct of Π relative to I is obtained from Π by removing every rule whose body is not satisfied by I. Set I is an FLP answer set of Π if it is minimal among the sets of atoms that satisfy the FLP reduct of Π relative to I. For example, in program Π_1, the FLP reduct of Π_1 relative to $\{p(-1), p(1)\}$ contains the last two rules only. Set $\{p(-1), p(1)\}$ is minimal among the sets of atoms that satisfy the reduct, and thus is an FLP answer set of Π_1. One can check that this is the only FLP answer set.

Ferraris Semantics

The Ferraris semantics (Ferraris, 2005) is to understand an aggregate as an abbreviation of a propositional formula. The semantics from (Ferraris, 2005) can be extended to allow variables as follows.

Let $E = \text{OP}\langle \boldsymbol{x}.F(\boldsymbol{x}) \rangle \succeq b$ be an aggregate occurring in Π, let $\boldsymbol{O}_\Pi(E)$ be the set of all lists of object constants of $\sigma(\Pi)$ whose length is the same as the length of $\boldsymbol{x}$, and let $C_\Pi(E)$ be the set of all subsets $\boldsymbol{C}$ of $\boldsymbol{O}_\Pi(E)$ such that OP$\langle \{\!\{\boldsymbol{c}[1] : \boldsymbol{c} \in \boldsymbol{C}\}\!\} \rangle \not\succeq b$. For instance, in program Π_1, for $E_1 = \text{SUM}\langle x.p(x) \rangle < 2$, set $\boldsymbol{O}_{\Pi_1}(E_1)$ is $\{-1, 1, 2\}$, and $C_{\Pi_1}(E_1)$ is $\{\{2\}, \{1, 2\}, \{-1, 1, 2\}\}$. Similarly, for $E_2 = \text{SUM}\langle x.p(x) \rangle \geq 0$, set $C_{\Pi_1}(E_2)$ is $\{\{-1\}\}$.

By $\mathrm{Fer}_\Pi(E)$ we denote

$$\bigwedge_{\boldsymbol{C}\in C_\Pi(E)} \Big(\bigwedge_{\boldsymbol{c}\in\boldsymbol{C}} F(\boldsymbol{c}) \to \bigvee_{\boldsymbol{c}\in\boldsymbol{O}_\Pi(E)\setminus\boldsymbol{C}} F(\boldsymbol{c}) \Big). \tag{4.3}$$

For instance, $\mathrm{Fer}_{\Pi_1}(E_1)$ is

$$(p(2) \to p(-1) \vee p(1)) \wedge (p(1) \wedge p(2) \to p(-1)) \wedge (p(-1) \wedge p(1) \wedge p(2) \to \bot).$$

By $\mathrm{Fer}(\Pi)$ we denote the propositional formula obtained from Π by replacing every aggregate E in it by $\mathrm{Fer}_\Pi(E)$.[3] The *Ferraris answer sets* of Π are defined as the answer sets of $\mathrm{Fer}(\Pi)$. For example, the Ferraris answer sets of Π_1 are the answer sets of the following formula $\mathrm{Fer}(\Pi_1)$:[4]

$$\begin{aligned}
&(\neg\underline{((p(2)\to p(-1)\vee p(1)) \wedge (p(1)\wedge p(2)\to p(-1)) \wedge (p(-1)\wedge p(1)\wedge p(2)\to\bot))} \to p(2))\\
&\wedge (\underline{(p(-1)\to p(1)\vee p(2))} \to p(-1))\\
&\wedge (p(-1) \to p(1)).
\end{aligned} \tag{4.4}$$

This formula has two answer sets: $\{p(-1), p(1)\}$ and $\{p(-1), p(1), p(2)\}$.

PDB-SPT Semantics

Under the semantics proposed in (Pelov et al., 2003), an aggregate can be identified with a nested expression in the form of disjunctions over conjunctions, but unlike the naive attempt given in the introduction, it involves the notion of a "(maximal) local power set."[5]

Given a set A of some sets, a pair $\langle B, T\rangle$ where $B, T \in A$ and $B \subseteq T$ is called a *local power set (LPS)* of A if every set S such that $B \subseteq S \subseteq T$ belongs to A as well. A local power set is called *maximal* if there is no other local power set $\langle B', T'\rangle$ of A such that $B' \subseteq B$ and $T \subseteq T'$.

Under the PDB-SPT semantics, a negation in front of an aggregate expression is eliminated by a "classically equivalent" transformation. Program $\mathrm{Pos}(\Pi)$ is obtained from a

[3] Strictly speaking, $\mathrm{Fer}_\Pi(E)$ is a ground formula of a first-order signature. However, since answer sets are Herbrand interpretations, we can view it as a propositional formula under the corresponding propositional signature. In the following, we identify a ground formula with a propositional formula in this sense, unless otherwise noted.

[4] We underline the parts of a formula that correspond to aggregates.

[5] We saw the term first in (You & Liu, 2008). Notice that a *maximal local power set* is not in fact a set.

program Π by replacing $\mathrm{not}\ E_i$ in each rule (4.2) in Π where $E_i = \mathsf{OP}\langle \boldsymbol{x}.F(\boldsymbol{x})\rangle \succeq b$ with $\mathsf{OP}\langle \boldsymbol{x}.F(\boldsymbol{x})\rangle \not\succeq b$ ($\not\succeq$ is the symbol for the relation complementary to $\succeq$). Clearly, $\mathrm{Pos}(\Pi)$ contains no negation in front of any aggregate expression. For instance, the first rule of $\mathrm{Pos}(\Pi_1)$ is

$$p(2) \leftarrow \mathsf{SUM}\langle x.p(x)\rangle \geq 2.$$

Let $E = \mathsf{OP}\langle \boldsymbol{x}.F(\boldsymbol{x})\rangle \succeq b$ be an aggregate occurring in $\mathrm{Pos}(\Pi)$, let HB_Π be the set of all ground atoms that can be constructed from $\sigma(\Pi)$, and let $\mathcal{I}_\Pi(E)$ be the set of all Herbrand interpretations I of $\sigma(\Pi)$ such that I satisfies E (satisfaction as defined in Section 4.1). For instance, for program Π_1, HB_{Π_1} is $\{p(-1), p(1), p(2)\}$, and, for $\overline{E_1} = \mathsf{SUM}\langle x.p(x)\rangle \geq 2$, $\mathcal{I}_{\Pi_1}(\overline{E_1})$ is

$$\{\{p(2)\}, \{p(1), p(2)\}, \{p(-1), p(1), p(2)\}\},$$

and, for $E_2 = \mathsf{SUM}\langle x.p(x)\rangle \geq 0$, $\mathcal{I}_{\Pi_1}(E_2)$ is

$$\{\emptyset, \{p(1)\}, \{p(2)\}, \{p(-1), p(1)\}, \{p(-1), p(2)\}, \{p(1), p(2)\}, \{p(-1), p(1), p(2)\}\}.$$

The maximal local power sets of $\mathcal{I}_{\Pi_1}(\overline{E_1})$ are

$$\langle\{p(2)\}, \{p(1), p(2)\}\rangle, \ \ \langle\{p(1), p(2)\}, \{p(-1), p(1), p(2)\}\rangle,$$

and the maximal local power sets of $\mathcal{I}_{\Pi_1}(E_2)$ are

$$\langle\emptyset, \{p(1), p(2)\}\rangle, \ \ \langle\{p(1)\}, \{p(-1), p(1), p(2)\}\rangle, \ \ \langle\{p(2)\}, \{p(-1), p(1), p(2)\}\rangle.$$

By $\text{PDB-SPT}_\Pi(E)$ we denote

$$\bigvee_{\langle B,T\rangle \text{ is a maximal LPS of } \mathcal{I}_\Pi(E)} \Big(\bigwedge_{A\in B} A \wedge \bigwedge_{A\in HB_\Pi\setminus T} \neg A \Big). \tag{4.5}$$

For instance, $\text{PDB-SPT}_{\Pi_1}(\overline{E_1})$ is $(p(2) \wedge \neg p(-1)) \vee (p(1) \wedge p(2))$.

By $\text{PDB-SPT}(\Pi)$ we denote the propositional formula obtained from $\mathrm{Pos}(\Pi)$ by replacing all aggregates E in it by $\text{PDB-SPT}_\Pi(E)$. The *PDB-SPT answer sets* of Π are defined as the answer sets of $\text{PDB-SPT}(\Pi)$. For example, Π_1 has no PDB-SPT answer sets, and neither does the following formula $\text{PDB-SPT}(\Pi_1)$:

$$\begin{aligned} &(\underline{((p(2) \wedge \neg p(-1)) \vee (p(1) \wedge p(2)))} \rightarrow p(2)) \\ &\wedge (\underline{(\neg p(-1) \vee p(1) \vee p(2))} \rightarrow p(-1)) \\ &\wedge (p(-1) \rightarrow p(1)). \end{aligned} \tag{4.6}$$

Note that the original semantics defined in (Pelov et al., 2003; Son et al., 2007) is limited to programs with nondisjunctive heads. It is not immediate how the nonreductive approaches to defining PDB-SPT semantics (Son et al., 2007; Pelov et al., 2007) can be extended to allow disjunction in the heads, while it is straightforward here.

4.2 Reformulation and Comparison of the Semantics of Aggregates

A Reformulation of Ferraris Semantics

The propositional formula representation of an aggregate according to the Ferraris semantics can be written in a more compact way by considering maximal local power sets as in the PDB-SPT semantics. Let E be an aggregate occurring in Π. By $\mathrm{MLPS\text{-}Fer}_\Pi(E)$ we denote

$$\bigwedge_{\langle B,T\rangle \text{ is a maximal LPS of } C_\Pi(E)} \Big(\bigwedge_{\boldsymbol{c}\in B} F(\boldsymbol{c}) \rightarrow \bigvee_{\boldsymbol{c}\in \boldsymbol{O}_\Pi(E)\setminus T} F(\boldsymbol{c}) \Big). \tag{4.7}$$

Lemma 22 *$MLPS\text{-}Fer_\Pi(E)$ is strongly equivalent to $Fer_\Pi(E)$.*

This fact provides an alternative characterization of the Ferraris semantics. We define the *MLPS-Ferraris answer sets* same as the Ferraris answer sets except that we refer to $\mathrm{MLPS\text{-}Fer}_\Pi(E)$ in place of $\mathrm{Fer}_\Pi(E)$.

Proposition 17 *The MLPS-Ferraris answer sets of Π are precisely the Ferraris answer sets of Π.*

For example, in program Π_1, the maximal local power sets of $C_{\Pi_1}(E_1)$ are $\langle\{2\},\{1,2\}\rangle$, $\langle\{1,2\},\{-1,1,2\}\rangle$. The maximal local power set of $C_{\Pi_1}(E_2)$ is $\langle\{-1\},\{-1\}\rangle$. Formula (4.4) is strongly equivalent to the following shorter formula

$$\begin{aligned}
&(\neg\underline{((p(2)\rightarrow p(-1))\wedge(p(1)\wedge p(2)\rightarrow\bot))}\rightarrow p(2))\\
&\wedge(\underline{(p(-1)\rightarrow p(1)\vee p(2))}\rightarrow p(-1))\\
&\wedge(p(-1)\rightarrow p(1)).
\end{aligned}$$

A Reformulation of FLP Semantics

The FLP semantics can also be defined by reduction to propositional formulas. As with the PDB-SPT semantics, before turning a program to the propositional formula representation

for the FLP semantics, we eliminate the negation in front of an aggregate using the classically equivalent transformation $\mathrm{Pos}(\Pi)$ (Section 4.1). Let E be an aggregate occurring in $\mathrm{Pos}(\Pi)$. By $I_\Pi(E)$ we denote the set of all Herbrand interpretations I of $\sigma(\Pi)$ such that I does not satisfy E (as defined in Section 4.1). Clearly $I_\Pi(E)$ and $\mathcal{I}_\Pi(E)$ partition HB_Π. By $\mathrm{FLP}_\Pi(E)$ we denote

$$\bigwedge_{I \in I_\Pi(E)} \Big(\bigwedge_{A \in I} A \to \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A \Big). \tag{4.8}$$

By $\mathrm{FLP}(\Pi)$ we denote the propositional formula obtained from $\mathrm{Pos}(\Pi)$ by replacing all aggregates E in it by $\mathrm{FLP}_\Pi(E)$. For example, for program Π_1, set $I_{\Pi_1}(\overline{E_1})$ is $\{\emptyset, \{p(-1)\}, \{p(1)\}, \{p(-1), p(1)\}, \{p($ and $I_{\Pi_1}(E_2)$ is $\{\{p(-1)\}\}$, so that $\mathrm{FLP}(\Pi_1)$ is

$$\begin{aligned}
&(\underline{((p(-1) \vee p(1) \vee p(2)) \wedge (p(-1) \to p(1) \vee p(2)) \wedge (p(1) \to p(-1) \vee p(2))} \\
&\qquad \underline{\wedge (p(-1) \wedge p(1) \to p(2)) \wedge (p(-1) \wedge p(2) \to p(1)))} \to p(2)) \\
&\wedge (\underline{(p(-1) \to p(1) \vee p(2))} \to p(-1)) \\
&\wedge (p(-1) \to p(1)).
\end{aligned}$$

The following proposition tells us that the FLP semantics can be characterized by propositional formulas.

Proposition 18 *The answer sets of $FLP(\Pi)$ are precisely the FLP answer sets of Π (as defined in Section 4.1)*

Similar to (4.7), formula $\mathrm{FLP}_\Pi(E)$ can be rewritten using the notion of maximal local power sets, which provides yet another characterization of the FLP semantics. This is due to the following lemma. By $\mathrm{MLPS\text{-}FLP}_\Pi(E)$ we denote

$$\bigwedge_{\langle B,T\rangle \text{ is a maximal LPS of } I_\Pi(E)} \Big(\bigwedge_{A \in B} A \to \bigvee_{A \in \mathrm{HB}_\Pi \setminus T} A \Big). \tag{4.9}$$

Lemma 23 *$FLP_\Pi(E)$ is strongly equivalent to $MLPS\text{-}FLP_\Pi(E)$.*

Formula $\mathrm{MLPS\text{-}FLP}(\Pi)$ is defined the same as $\mathrm{FLP}(\Pi)$ except that we refer to $\mathrm{MLPS\text{-}FLP}_\Pi(E)$ in place of $\mathrm{FLP}_\Pi(E)$. For example, Lemma 23 tells us that $\mathrm{FLP}(\Pi_1)$ has the same answer

sets as the following formula $\mathrm{MLPS\text{-}FLP}(\Pi_1)$:

$$\begin{aligned}&((\underline{p(2) \wedge (p(-1) \rightarrow p(1))}) \rightarrow p(2))\\ &\wedge ((\underline{p(-1) \rightarrow p(1) \vee p(2)}) \rightarrow p(-1))\\ &\wedge (p(-1) \rightarrow p(1)).\end{aligned} \tag{4.10}$$

While the Ferraris semantics and the FLP semantics can be characterized either with or without involving the concept of local power sets, this is not the case with the PDB-SPT semantics. In other words, in the definition of a PDB-SPT answer set, if we replace (4.5) with the formula

$$\bigvee_{I \in \mathcal{I}_\Pi(E)} \Big(\bigwedge_{A \in I} A \wedge \bigwedge_{A \in HB \setminus I} \neg A \Big), \tag{4.11}$$

then the resulting definition is no longer equivalent. Note that (4.5) and (4.11) are classically equivalent but are not strongly equivalent.

A Reformulation of PDB-SPT Semantics

Consider the following formula modified from $\mathrm{FLP}_\Pi(E)$ by simply eliminating implications in favor of negations and disjunctions as in classical logic. By $\mathrm{Mod\text{-}FLP}_\Pi(E)$ we denote

$$\bigwedge_{I \in I_\Pi(E)} \Big(\bigvee_{A \in I} \neg A \vee \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A \Big). \tag{4.12}$$

$\mathrm{Mod\text{-}FLP}_\Pi(E)$ is classically equivalent to $\mathrm{FLP}_\Pi(E)$, but is not strongly equivalent. However, interestingly, the following holds.

Lemma 24 *$Mod\text{-}FLP_\Pi(E)$ is strongly equivalent to $PDB\text{-}SPT_\Pi(E)$.*

This fact provides a simple reformulation of the PDB-SPT semantics, without involving the notion of local power sets. We define $\mathrm{Mod\text{-}FLP}(\Pi)$ in the same way as $\mathrm{PDB\text{-}SPT}(\Pi)$ except that we refer to $\mathrm{Mod\text{-}FLP}_\Pi(E)$ in place of $\mathrm{PDB\text{-}SPT}_\Pi(E)$.

Proposition 19 *The answer sets of $Mod\text{-}FLP(\Pi)$ are precisely the PDB-SPT answer sets of Π.*

For instance, the PDB-SPT answer sets of Π_1 are the same as the answer sets of the following formula:

$$\begin{aligned}&(((p(-1)\vee p(1)\vee p(2))\wedge(\neg p(-1)\vee p(1)\vee p(2))\wedge(\neg p(1)\vee p(-1)\vee p(2))\\&\qquad\wedge(\neg p(-1)\vee\neg p(1)\vee p(2))\wedge(\neg p(-1)\vee\neg p(2)\vee p(1)))\rightarrow p(2))\\&\wedge((\neg p(-1)\vee p(1)\vee p(2))\rightarrow p(-1))\\&\wedge(p(-1)\rightarrow p(1)).\end{aligned}\tag{4.13}$$

Similar to the Ferraris and the FLP semantics, maximal local power sets can be used instead, as the following lemma states.

Lemma 25 *Mod-$FLP_{\Pi}(E)$ is strongly equivalent to*

$$\bigwedge_{\langle B,T\rangle \text{ is a maximal LPS of } I_{\Pi}(E)} \Big(\bigvee_{A\in B}\neg A\vee\bigvee_{A\in HB_{\Pi}\setminus T}A\Big).\tag{4.14}$$

Again note the similarity between (4.9) and (4.14). They are classically equivalent to each other, but are not strongly equivalent.

Relationship among the Semantics

The following proposition shows how one formula is stronger than another formula under the stable model semantics. As before $\boldsymbol{p}$ is the list of distinct predicate constants occurring in F or G, and $\boldsymbol{q}$ is a list of new, distinct predicate constants of the same length as $\boldsymbol{p}$.

Proposition 20 *If formulas $F\leftrightarrow G$ and*

$$\boldsymbol{q}<\boldsymbol{p}\rightarrow(F^*(\boldsymbol{q})\rightarrow G^*(\boldsymbol{q}))$$

are logically valid, then $SM[G;\boldsymbol{p}]\rightarrow SM[F;\boldsymbol{p}]$ is logically valid.

The proposition implies that, for the two formulas F and G that satisfies the condition, every answer set of G is an answer set of F.

The characterizations of each semantics in terms of the uniform framework of propositional formulas give new insights into their relationships. Note that the propositional

formula representations of an aggregate according to each semantics are classically equivalent (under the Herbrand models of $\sigma(\Pi)$), but not strongly equivalent, which accounts for the difference in the semantics.

Proposition 21 *For any program Π and any aggregate E occurring in Π and any set X of ground atoms of $\sigma(\Pi)$, $X \models E$ iff $X \models PDB\text{-}SPT_{\Pi}(E)$ iff $X \models FLP_{\Pi}(E)$ iff $X \models Fer_{\Pi}(E)$.*

The relationship between the PDB-SPT semantics and the FLP semantics is known as follows.

Proposition 22 *(Son & Pontelli, 2007, Theorem 2) Every PDB-SPT answer set of Π is an FLP answer set of Π.*

The converse does not hold as illustrated by program Π_1. An alternative proof of Proposition 22 follows from Propositions 20, 21 and the following lemma.

Lemma 26

$$(\boldsymbol{q} \leq \boldsymbol{p}) \rightarrow (PDB\text{-}SPT_{\Pi}(E)^*(\boldsymbol{q}) \rightarrow FLP_{\Pi}(E)^*(\boldsymbol{q}))$$

is logically valid.

For program Π_1, its only FLP answer set is a Ferraris answer set. Indeed, such relationship holds if the program is "semi-positive." We call a program *semi-positive* if, for every aggregate (4.1) occurring in it, $F(\boldsymbol{x})$ is a quantifier-free formula that contains no implications (this, in particular, means that there are no negations since we treat $\neg G$ as shorthand for $G \rightarrow \bot$). For example, Π_1 is semi-positive.

Proposition 23 *For any semi-positive program Π, every FLP answer set of Π is a Ferraris answer set of Π.*

However, the relationship does not hold for arbitrary programs. For instance, the following non-semi-positive program

$$p(a) \leftarrow \text{COUNT}\langle x.\neg\neg p(x) \vee q(x)\rangle \neq 1$$
$$q(b) \leftarrow p(a)$$
$$p(a) \leftarrow q(b)$$

has no Ferraris answer sets while it has only one FLP answer set $\{p(a), q(b)\}$.

The following proposition is a slight extension of Theorem 3 from (Ferraris, 2005), which describes a class of programs whose FLP answer sets coincide with Ferraris answer sets.

Proposition 24 *For any semi-positive program Π, the FLP answer sets of Π are precisely the Ferraris answer sets of $Pos(\Pi)$.*

4.3 Loop Formulas for Programs with Aggregates

Loop formulas in the previous chapter do not consider aggregates. You and Liu (2008) define loop formulas for programs with aggregates according to the PDB-SPT semantics. Their loop formulas contain aggregates, but the result can be explained by a reductive approach in the following simple way. A set of ground atoms is a loop of Π according to their definition iff it is a loop of $\text{PDB-SPT}(\Pi)$ according to (Ferraris et al., 2006). If we replace aggregates in their loop formulas with their propositional formula representations, the resulting formulas are essentially identical to loop formulas of $\text{PDB-SPT}(\Pi)$ according to (Ferraris et al., 2006). As shown in (Liu & Truszczynski, 2006; You & Liu, 2008), an advantage of loop formulas containing aggregates is that such loop formulas can be compactly encoded in pseudo-Boolean constraints, which allows pseudo-Boolean solvers to be used for computing answer sets.

In this section we define loop formulas for programs with aggregates for the two other semantics.

We assume that Π contains no function constants of positive arity and no free variables. Let us identify rule (4.2) with

$$A \leftarrow B, C, N \tag{4.15}$$

where $A = \{A_1, \ldots, A_l\}$, B is the set of all atoms E_i where $1 \leq i \leq m$, and C is the set of all aggregates E_j where $1 \leq j \leq m$, and N is the set of all remaining expressions $\mathrm{not}\ E_k$ ($m{+}1 \leq k \leq n$) in the body.

Remind that loop formulas for arbitrary first-order formulas are defined using the notion of NES, which can be extended to cover aggregates in a straightforward way. For any aggregate $E = \mathsf{OP}\langle \boldsymbol{x}.F \rangle \succeq b$ and any finite set Y of ground atoms, formula $\mathrm{NES}_E(Y)$ is defined recursively as

$$\mathsf{OP}\langle \boldsymbol{x}.\mathrm{NES}_F(Y) \rangle \succeq b \wedge E.$$

For instance, in program Π_1, formula $\mathrm{NES}_{E_2}(\{p(-1), p(1)\})$ is

$$\mathsf{SUM}\langle x.p(x) \wedge x \neq -1 \wedge x \neq 1 \rangle \geq 0 \ \wedge \ \mathsf{SUM}\langle x.p(x) \rangle \geq 0.$$

For any finite set Z of expressions, by $Z^\wedge$ and $Z^\vee$ we denote the conjunction and, respectively, disjunction of the elements of Z. We define the *external support formula* of a finite set Y of ground atoms for Π, denoted by $\mathrm{ES}_\Pi(Y)$, as the disjunction of

$$B^\wedge \wedge \bigwedge_{E \in C} \mathrm{NES}_E(Y) \wedge N^\wedge \wedge \ \neg (A \setminus Y)^\vee$$

for all rules (4.2) in Π such that $A \cap Y \neq \emptyset$ and $B \cap Y = \emptyset$. For instance, if Y is $\{p(-1), p(1)\}$, the external support formula of Y for Π_1 is

$$p(-1) \wedge p(1) \rightarrow (\mathsf{SUM}\langle x.p(x) \wedge x \neq -1 \wedge x \neq 1 \rangle \geq 0) \wedge \mathsf{SUM}\langle x.p(x) \rangle \geq 0.$$

The *aggregate loop formula of* Y *for* Π, denoted by $\mathrm{LF}_\Pi(Y)$, is

$$Y^\wedge \rightarrow \mathrm{ES}_\Pi(Y).$$

This definition extends the definition of a loop formula given in (Liu & Truszczynski, 2006), which is limited to programs with monotone aggregates.

Loops	Aggregate loop formulas
$\{p(-1)\}$	$p(-1) \to \mathsf{SUM}\langle x.p(x) \wedge x \neq -1\rangle \geq 0 \wedge \mathsf{SUM}\langle x.p(x)\rangle \geq 0$
$\{p(1)\}$	$p(1) \to p(-1)$
$\{p(2)\}$	$p(2) \to \neg\mathsf{SUM}\langle\{x.p(x)\}\rangle < 2$
$\{p(-1), p(1)\}$	$p(-1) \wedge p(1) \to \mathsf{SUM}\langle x.p(x) \wedge x \neq -1 \wedge x \neq 1\rangle \geq 0$ $\wedge\mathsf{SUM}\langle x.p(x)\rangle \geq 0$

Figure 4.1: Loops and aggregate loop formulas for Π_1

The *Ferraris dependency graph* of Π is the directed graph such that

- its vertices are the ground atoms of $\sigma(\Pi)$;
- for every rule (4.15) in Π, an edge goes from each element of A to $p(\boldsymbol{t})$ if
  - $p(\boldsymbol{t})$ is an element of B, or
  - if there are an monotone or non-monotone aggregate $E = \mathsf{OP}\langle \boldsymbol{x}.F\rangle \succeq b$ occurring in C and a substitution θ such that $p(\boldsymbol{t}')$ has a positive occurrence in F_i that does not belong to any negative subformula of F_i for some $1 \leq i \leq n$ and $p(\boldsymbol{t}')\theta = p(\boldsymbol{t})$.

For example, Π_1 has four loops: $\{p(-1)\}$, $\{p(1)\}$, $\{p(2)\}$, $\{p(-1), p(1)\}$. Figure 4.1 shows the aggregate loop formulas of all loops.

Proposition 25 *For any set X of ground atoms of $\sigma(\Pi)$ that satisfies Π, the following conditions are equivalent to each other.*

(a) X is a Ferraris answer set of Π;

(b) for every nonempty set Y of ground atoms of $\sigma(\Pi)$, X satisfies the aggregate loop formula of Y for Π;

(c) for every loop Y of Π, X satisfies the aggregate loop formula of Y for Π;

Loop Formulas for FLP semantics

In view of Proposition 24, Proposition 25 can be applied to FLP semantics as well. We assume that, for every aggregate (4.1) occurring in the program, $F(\boldsymbol{x})$ is a conjunction of atoms. Notice that, under this assumption, $\mathrm{Pos}(\Pi)$ is a semi-positive program.

Proposition 26 *For any set X of ground atoms of $\sigma(\Pi)$ that satisfies Π, the following conditions are equivalent to each other.*

(a) X is an FLP answer set of Π;

(b) for every nonempty set Y of ground atoms of $\sigma(\Pi)$, X satisfies the aggregate loop formula of Y for $Pos(\Pi)$;

(c) for every loop Y of $Pos(\Pi)$, X satisfies the aggregate loop formula of Y for $Pos(\Pi)$.

Figure 4.2 shows the aggregate loop formulas of all loops for $\mathrm{Pos}(\Pi_1)$. Notice that the loops of $\mathrm{Pos}(\Pi)$ may be different from the loops of Π.

Loops	Aggregate loop formulas
$\{p(-1)\}$	$p(-1) \rightarrow \mathsf{SUM}\langle x.p(x) \wedge x \neq -1\rangle \geq 0 \wedge \mathsf{SUM}\langle x.p(x)\rangle \geq 0$
$\{p(1)\}$	$p(1) \rightarrow p(-1)$
$\{p(2)\}$	$p(2) \rightarrow \neg\mathsf{SUM}\langle x.p(x) \wedge x \neq 2\rangle < 2 \wedge \neg\mathsf{SUM}\langle x.p(x)\rangle < 2$
$\{p(-1), p(1)\}$	$p(-1) \wedge p(1) \rightarrow \mathsf{SUM}\langle x.p(x) \wedge x \neq -1 \wedge x \neq 1\rangle \geq 0$ $\wedge \mathsf{SUM}\langle x.p(x)\rangle \geq 0$
$\{p(-1), p(2)\}$	$p(-1) \wedge p(2) \rightarrow (\neg\mathsf{SUM}\langle x.p(x) \wedge x \neq -1 \wedge x \neq 2\rangle < 2$ $\wedge \neg\mathsf{SUM}\langle x.p(x)\rangle < 2)$ $\vee(\mathsf{SUM}\langle x.p(x) \wedge x \neq -1 \wedge x \neq 2\rangle \geq 0$ $\wedge \mathsf{SUM}\langle x.p(x)\rangle \geq 0)$

Figure 4.2: Loops and aggregate loop formulas for $\mathrm{Pos}(\Pi_1)$

Faber (2005) defined the notion of *external support* for the FLP semantics as follows.[6] Given sets X and Y of ground atoms, Y is called *FLP externally supported* by Π w.r.t. X if there is a rule (4.2) in Π such that

- $A \cap Y \neq \emptyset$,
- $(A \setminus Y) \cap X = \emptyset$,
- $X \setminus Y \models E_1 \wedge \cdots \wedge E_m$,
- $X \models E_1 \wedge \cdots \wedge E_m \wedge \neg E_{m+1} \wedge \cdots \wedge \neg E_n$.

Proposition 27 *Y is FLP externally supported by Π w.r.t. X iff X satisfies $ES_{Pos(\Pi)}(Y)$.*

[6] More precisely, that paper defined the negation of this concept, unfoundedness.

4.4 Syntax and Semantics of Aggregate Formulas

Below we lift up the syntax and semantics of aggregate programs from Section 4.1 to more general cases. We will introduce a slightly more general definition of an aggregate and define aggregate formulas as an extension of first-order formulas by treating aggregates as a base case in addition to (standard) atomic formulas (including equality) and $\bot$ (falsity).

We assume that the signature σ contains symbols for all numbers, and some collection of *comparison operators* that stands for binary relations over numbers. We assume that symbols for aggregate functions are not part of the signature.

Following (Lee & Meng, 2009; Ferraris & Lifschitz, 2010), we define an *aggregate formula* as an extension of a first-order formula by adding the following clause.

- $$\mathrm{OP}\langle \boldsymbol{x}^1.F_1, \dots, \boldsymbol{x}^n.F_n\rangle \succeq b \tag{4.16}$$

  is *first-order formula with aggregates* where

  - OP is an *aggregate function*;
  - $\boldsymbol{x}^1, \dots, \boldsymbol{x}^n$ are nonempty lists of distinct object variables;
  - $F_1, \dots, F_n$ are arbitrary *first-order formulas with aggregates*;
  - $\succeq$ is a comparison operator;
  - b is a term.

For instance,

$$(\mathrm{SUM}\langle x.p(x)\rangle \geq 1 \ \vee \ \exists y\, q(y)) \to r(x)$$

is an aggregate formula.

We say that an occurrence of a variable v in an aggregate formula H is *bound* if the occurrence is in a part of H of the form (4.16) where v is in at least one of $\boldsymbol{x}^i$, or in a part of H of the form QvG. Otherwise it is *free*. We say that v is *free* in H if H contains a free occurrence of v. An aggregate sentence is an aggregate formula with no free variables.

The definition of an interpretation is the same as in first-order logic. We consider only the interpretations such that each number and each comparison operator is interpreted as itself. The definition of satisfaction in first-order logic is extended to aggregate sentences as follows. For any interpretation I of the underlying signature, satisfaction is extended to cover aggregate (4.16) that contains no free variables as follows. For any set X of n-tuples ($n \geq 1$), let $msp(X)$ be the multiset consisting of all ξ_1 such that $(\xi_1, \ldots, \xi_n) \in X$ for at least one (n-1)-tuple $(\xi_2, \ldots, \xi_n)$, with the multiplicity equal to the number of such (n-1)-tuples (and to $+\infty$ if there are infinitely many of them). Using this notation, we define:

- $(x_1 \ldots x_n.F(x_1, \ldots, x_n))^I$ is

$$msp(\{(\xi_1, \ldots, \xi_n) \in |I|^n : F(\xi_1^*, \ldots, \xi_n^*)^I = \mathrm{TRUE}\});$$

- (OP$\langle \boldsymbol{x}^1.F_1, \ldots, \boldsymbol{x}^n.F_n \rangle \succeq b)^I$ equals TRUE if the join α of the multisets $(\boldsymbol{x}^1.F_1)^I, \ldots, (\boldsymbol{x}^n.F_n)^I$ belongs to the domain of OP and satisfies the condition OP$(\alpha) \succeq b^I$.

With this extension, the recursive definition of satisfaction for an aggregate sentence is given in the same way as in first-order logic. We say that an aggregate sentence F is *logically valid* if every interpretation satisfies it. For instance, an Herbrand interpretation $\{p(a)\}$ satisfies COUNT$\langle x.p(x) \rangle > 0$ but does not satisfy SUM$\langle x.p(x) \rangle > 0$ because multiset $\{\!\{a\}\!\}$ is not in the domain of SUM. Consider the aggregate

$$\mathsf{SUM}\langle x.p(x) \rangle \geq 0$$

and an Herbrand interpretation $I = \{p(-1), p(1)\}$. S_I is $\{\!\{-1, 1\}\!\}$ and SUM$(S_I) = 0 \geq 0$, so I satisfies SUM$\langle x.p(x) \rangle \geq 0$.

4.5 Stable Model Semantics of First-Order Aggregate Formulas

In this section we provide a general definition of a stable model that applies to arbitrary "aggregate formulas" in the style of the definition in Section 2.5, by extending the notion F^* to aggregates in a way similar to other connectives.

For any aggregate sentence F and any list of predicate constants $\boldsymbol{p}$, expression $\mathrm{SM}[F; \boldsymbol{p}]$ stands for (2.6) where $F^*(\boldsymbol{u})$ is extended to aggregates as

- $$(\mathrm{OP}\langle \boldsymbol{x}^1.F_1, \ldots, \boldsymbol{x}^n.F_n\rangle \succeq b)^* =$$
$$(\mathrm{OP}\langle \boldsymbol{x}^1.F_1^*, \ldots, \boldsymbol{x}^n.F_n^*\rangle \succeq b) \wedge (\mathrm{OP}\langle \boldsymbol{x}^1.F_1, \ldots, \boldsymbol{x}^n.F_n\rangle \succeq b).$$

By a $\boldsymbol{p}$-*stable model* of F we mean a model of $\mathrm{SM}[F;\boldsymbol{p}]$ (under the extended notion of satisfaction). As before, we use $\mathrm{SM}[F]$ in place of $\mathrm{SM}[F;\boldsymbol{p}]$ when $\boldsymbol{p}$ is the list of all predicate constants occurring in F.

The Theorem on strong equivalence (Lifschitz et al., 2001; Ferraris, 2005) can be extended to aggregate formulas in a straightforward way using the extended notion of satisfaction. In the statement of the proposition below, $\boldsymbol{p}$ stands for the list of all predicate constants that occur in F or G, and $\boldsymbol{q}$ is a list of new, distinct predicate constants, of the same length as $\boldsymbol{p}$.

Proposition 28 *Aggregate formulas F and G are strongly equivalent to each other iff the formula*

$$(\boldsymbol{q} \leq \boldsymbol{p}) \rightarrow (F^*(\boldsymbol{q}) \leftrightarrow G^*(\boldsymbol{q})) \tag{4.17}$$

is logically valid.

Programs with Aggregates as a Special Case

The *AF-representation* ("Aggregate Formula representation") of (4.2) is the universal closure of the aggregate formula

$$E_1 \wedge \cdots \wedge E_m \wedge \neg E_{m+1} \wedge \cdots \wedge \neg E_n \rightarrow A_1 \vee \cdots \vee A_l. \tag{4.18}$$

The *AF-representation* of Π is the conjunction of the AF-representation of its rules.

The *stable models* of Π are defined as the models of $\mathrm{SM}[F]$, where F is an AF-representation of Π. The following proposition shows that this definition is a proper generalization of the Ferraris semantics.

Proposition 29 *Let Π be a program that contains no function constants of positive arity and let F be its AF-representation. The Herbrand models of $SM[F]$ whose signature is $\sigma(\Pi)$ are precisely the Ferraris answer sets of Π.*

4.6 FLP Semantics of First-Order Aggregate Formulas

In this section, we will extend the FLP semantics to a *general program with aggregates*, which contains rules of the form

$$H \leftarrow B \tag{4.19}$$

where H and B are aggregate formulas.

The *AF-representation* ("Aggregate Formula representation") of a finite general program Π with aggregates is the conjunction of the universal closures of the aggregate formulas

$$B \rightarrow H$$

for all rules $H \leftarrow B$ in Π.

Let Π be a finite program whose rules have the form (4.19). The *FOL-representation* Π^{FOL} of Π is the conjunction of the universal closures of $B \rightarrow H$ for all rules (4.19) in Π. By $\mathrm{FLP}[\Pi; \boldsymbol{p}]$ we denote the second-order formula

$$\Pi^{FOL} \wedge \neg\exists \boldsymbol{u}(\boldsymbol{u} < \boldsymbol{p} \wedge \Pi^{\triangle}(\boldsymbol{u})) \tag{4.20}$$

where $\Pi^{\triangle}(\boldsymbol{u})$ is defined as the conjunction of

$$\forall \boldsymbol{x}(B \wedge B(\boldsymbol{u}) \rightarrow H(\boldsymbol{u})) \tag{4.21}$$

for all rules $H \leftarrow B$ in Π, where $\boldsymbol{x}$ is the list of all (free) variables in $H \leftarrow B$.[7]

We will often simply write $\mathrm{FLP}[\Pi]$ instead of $\mathrm{FLP}[\Pi; \boldsymbol{p}]$ when $\boldsymbol{p}$ is the list of all predicate constants occurring in Π, and call a model of $\mathrm{FLP}[\Pi]$ an *FLP-stable* model of Π.

Example 2 continued *The AF-representation of* Π_1*, denoted by* F*, is the following:*

$$\begin{aligned} &(\neg(\mathit{SUM}\langle x.p(x)\rangle < 2) \rightarrow p(2)) \\ &\wedge\ (\mathit{SUM}\langle x.p(x)\rangle \geq 0 \rightarrow p(-1)) \\ &\wedge\ (p(-1) \rightarrow p(1))\ . \end{aligned} \tag{4.22}$$

The FLP-stable models of Π_1 *are the models of*

$$F \wedge \neg\exists u(u < p \wedge \Pi^{\triangle}(u)) \tag{4.23}$$

[7] Note that we assume that Π is finite in order to avoid infinite conjunctions in the FOL representation.

where $\Pi^{\triangle}(u)$ *is*

$$\begin{aligned}&(\neg(\mathit{SUM}\langle x.p(x)\rangle < 2) \wedge \neg(\mathit{SUM}\langle x.u(x)\rangle < 2) \rightarrow u(2))\\ &\wedge\ (\mathit{SUM}\langle x.p(x)\rangle \geq 0 \wedge \mathit{SUM}\langle x.u(x)\rangle \geq 0 \rightarrow u(-1))\\ &\wedge\ (p(-1) \wedge u(-1) \rightarrow u(1))\ .\end{aligned} \tag{4.24}$$

The following theorem tells us that our semantics is a proper generalization of the semantics from (Faber et al., 2011).

Theorem 8 *Let* Π *be a finite disjunctive program with aggregates that contains at least one object constant. The FLP-answer sets of* Π *in the sense of (Faber et al., 2011) are precisely the Herbrand models of* $FLP[\Pi]$ *whose signature is* $\sigma(\Pi)$.

It is known that the FLP semantics from (Faber et al., 2011) has the anti-chain property: no FLP-answer set is a proper subset of another FLP-answer set. This property is still preserved in our generalized semantics.

Proposition 30 *For any finite general program* Π*, if* I *is an Herbrand interpretation of* $\sigma(\Pi)$ *that satisfies* $FLP[\Pi]$*, then* I *is a minimal model of* Π.

4.7 Comparing FLP and the First-Order Stable Model Semantics

Disregarding aggregates, the main difference among FLP and SM has to do with the treatment of an implication. It is known that they coincide for programs whose rules have the form (4.2) (Faber et al., 2011, Theorem 3.6), (Truszczyński, 2010, Theorem 3). However, this is not the case for more general classes of programs (having rules of the form (4.18)), or for arbitrary formulas. In fact, none of them is stronger than the other, as the following example shows.

Example 12 *For propositional signature* $\{p\}$ *and program* $\mathcal{P}_1 = \{p \leftarrow p \vee \neg p\}$*, whose FOL-representation is* $F_1 = p \vee \neg p \rightarrow p$*, each of* $FLP[\mathcal{P}_1]$ *has* $\{p\}$ *as the only model, and* $SM[F_1]$ *has no models.*

Formula F_1 is strongly equivalent (in the sense of (Ferraris et al., 2011a)) to $F_2 = (p \to p) \wedge (\neg p \to p)$. Again, $SM[F_2]$ has no models. Neither does $FLP[\mathcal{P}_2]$, where $\mathcal{P}_2$ is the program corresponding to F_2.

For program $\mathcal{P}_3 = \{p \vee \neg p \leftarrow \top\}$, whose FOL-representation is $F_3 = \top \to p \vee \neg p$, $SM[F_3]$ has two models: $\emptyset$ and $\{p\}$, while $FLP[\mathcal{P}_3]$ has only one model: $\emptyset$.

Formula F_3 is strongly equivalent to $F_4 = \neg\neg p \to p$. $FLP[\mathcal{P}_4]$ ($\mathcal{P}_4$ is the program corresponding to F_4) has only one model: $\emptyset$, while $SM[F_4]$ has the same models as $SM[F_3]$.

The following theorem presents a class of programs for which the FLP semantics and the stable model semantics coincide. Following (Ferraris & Lifschitz, 2010), we say that an aggregate function OP is *monotone* w.r.t. $\succeq$ if for any multisets α, β such that $\alpha \subseteq \beta$,

- if OP(α) is defined then so is OP(β), and
- for any $n \in$**Num**, if OP$(\alpha) \succeq n$ then OP$(\beta) \succeq n$.

We consider two numbers, k and m, for an occurrence of a predicate constant or any other subexpression in a formula F,

- k: the number of implications in F that contain that occurrence in the antecedent;
- m: the number of aggregates (4.16) containing that occurrence such that OP is not monotone w.r.t. $\succeq$.

We call an occurrence of a subexpression in F *strictly positive* if $k + m$ for that occurrence in F is 0. For example, in formula $(p \to q) \to p$, the second occurrence of p is strictly positive. In $\neg$(SUM$\langle x.p(x)\rangle < 2$), the occurrence of p is not strictly positive (for that occurrence, $k = m = 1$).

We first define two notions. We call an aggregate formula *semi-positive relative to* $\boldsymbol{p}$ if, for every aggregate OP$\langle \boldsymbol{x}^1.F_1, \ldots, \boldsymbol{x}^n.F_n\rangle \succeq b$ in it, every occurrence of every predicate

p from $\boldsymbol{p}$ is strictly positive in each F_i $(1 \leq i \leq n)$. We say that an aggregate formula F is *canonical* relative to a list $\boldsymbol{p}$ of predicate constants if

- F is semi-positive relative to $\boldsymbol{p}$;
- for every occurrence of every predicate constant p from $\boldsymbol{p}$ in F, we have that $k+m \leq 1$;
- if a predicate constant p from $\boldsymbol{p}$ occurs in the scope of a strictly positive occurrence of $\exists$ or $\vee$ in F, then the occurrence of p is strictly positive in F.

For any canonical formula, the following result holds:

Proposition 31 *For any aggregate formula F, if F is canonical relative to $\boldsymbol{p}$, then formula*

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

is logically valid.

In Proposition 31, when F is a first-order formula, it is easy to see that $\mathrm{CIRC}[F;\boldsymbol{p}]$ and $\mathrm{SM}[F;\boldsymbol{p}]$ are equivalent to each other. This fact is used in (Kim et al., 2009; Lee & Palla, 2010) to compute the circumscriptive action formalisms using ASP solvers. Here we generalize the result to formulas containing aggregates and use it to relate the FLP to the SM operator.

Theorem 9 *Let Π be a finite general program with aggregates and let F be the AF-representation of Π. For every rule (4.18) in Π, if B is canonical relative to $\boldsymbol{p}$ and every occurrence of p from $\boldsymbol{p}$ in H is strictly positive in H, then $FLP[\Pi;\boldsymbol{p}]$ is equivalent to $SM[F;\boldsymbol{p}]$.*

Among the programs in Example 12, only $\mathcal{P}_2$ satisfies the condition of Theorem 9. For another example, in program Π_1, $\neg(\mathrm{sum}\langle x.p(x)\rangle < 2)$ is not canonical relative to $\{p\}$. In fact, $\{p(-1), p(1), p(2)\}$ is an Herbrand interpretation that satisfies $\mathrm{SM}[(4.22)]$, but it does not satisfy $\mathrm{FLP}[\Pi_1]$.

Theorem 9 is a generalization of Proposition 24. When is syntax of rule in $\mathrm{Pos}(\Pi)$ is limited to the form

$$A_1; \ldots; A_l \leftarrow E_1, \ldots, E_n,$$

it is clear that (1) for the body $E_1, \ldots, E_n$, the number $k + m \leq 1$; (2) every occurrence of predicate constant in the head is strictly positive in it. If Π is semi-positive, then it follows that the condition of Theorem 9 is satisfied as thus $\mathrm{FLP}[\Pi; \boldsymbol{p}]$ is equivalent to $\mathrm{SM}[F; \boldsymbol{p}]$.

4.8 Conclusion

The chapter investigated the semantics of aggregates in both propositional level and first-order level.

We presented a reductive approach to understand the existing semantics of aggregates in terms of propositional formulas and related different semantics in terms of the underlying general language. The reformulations give us insights into each of the semantics. The reduction to propositional formulas allows the established results for the propositional formula under the stable model semantics, such as the strong equivalence and loop formulas (Ferraris et al., 2006), to be applied to the semantics of aggregates, which saves efforts of extending such results to each semantics. Guided by the reduction, we defined the loop formulas of a program that contain aggregates.

The reductive approach led us to the general semantics of aggregates presented in Section 4.5 and Section 4.6. which extends the definition of a stable model of a first-order formula to an aggregate formula, using a notion of satisfaction extended from the one used in the FLP semantics. The semantics do not refer to grounding and allow one to reason about non-Herbrand models. They appear to be natural in defining arbitrary recursive and nested aggregates. We also present the precise relationship between the two generalized semantics.

4.9 Proofs

We omit the proof of Proposition 28 since the proof requires minor rewriting from the proof of Theorem 9 from (Ferraris et al., 2011b). The proof of Proposition 29 is immediate from Lemma 30. Proposition 25 is a special case of Theorem 15 in Section 5.3. The proof of Proposition 26 is immediate from Propositions 24 and 25. Theorem 9 is a special case of Proposition 41 in Section 5.4. Below we provide the remaining proofs.

In the following, Π is a program that contains no free variables, $E = \mathsf{OP}\langle \boldsymbol{x}.F(\boldsymbol{p},\boldsymbol{x})\rangle \succeq b$ is an aggregate expression occurring in Π, $\boldsymbol{p}$ is the list of predicate constants occurring in Π and $\boldsymbol{q}$ is a list of new, distinct predicate constants of the same length as $\boldsymbol{p}$, X and Y are sets of ground atoms of $\sigma(\Pi)$ such that $Y \subseteq X$. By $Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ we denote the set of ground atoms obtained from Y by substituting the members of new predicate constants $\boldsymbol{q}$ for the corresponding members of $\boldsymbol{p}$.

We will often use the following lemmas from (Ferraris et al., 2011b), each of which can be proven by induction.

Lemma 27 *Formula*

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

is logically valid.

Lemma 28 *Formula*

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

is logically valid.

Lemma 29 *Let $E = \mathsf{OP}\langle \boldsymbol{x} : F(\boldsymbol{p},\boldsymbol{x})\rangle \succeq b$ be an aggregate expression, and let $\boldsymbol{S}$ be the set of all lists $\boldsymbol{c}$ of object constants of $\sigma(\Pi)$ whose length is $|\boldsymbol{x}|$ such that*

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models F^*(\boldsymbol{q},\boldsymbol{c}).$$

Then

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathsf{OP}\langle \boldsymbol{x} : F^*(\boldsymbol{q},\boldsymbol{x})\rangle \succeq b$$

iff

$$\mathsf{OP}\langle\{\!\{\boldsymbol{c}[1] : \boldsymbol{c} \in S\}\!\}\rangle \succeq b.$$

Proof. Clear from the definitions. ∎

Lemma 30 $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models E^*(\boldsymbol{q})$ *iff* $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models Fer_{\Pi}(E)^*(\boldsymbol{q})$.

Proof. It is sufficient to show that

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \mathsf{OP}\langle \boldsymbol{x} : F^*(\boldsymbol{q}, \boldsymbol{x})\rangle \succeq b$$

iff

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \bigwedge_{\boldsymbol{C} \in C_{\Pi}(E)} \Big(\bigwedge_{\boldsymbol{c} \in \boldsymbol{C}} F^*(\boldsymbol{q}, \boldsymbol{c}) \rightarrow \bigvee_{\boldsymbol{c} \in \boldsymbol{O}_{\Pi}(E) \setminus \boldsymbol{C}} F^*(\boldsymbol{q}, \boldsymbol{c}) \Big).$$

From left to right: Assume $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \mathsf{OP}\langle \boldsymbol{x} : F^*(\boldsymbol{q}, \boldsymbol{x})\rangle \succeq b$. Let $\boldsymbol{S}$ be the set of all lists $\boldsymbol{c}$ of object constants of $\sigma(\Pi)$ whose length is $|\boldsymbol{x}|$ such that

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models F^*(\boldsymbol{q}, \boldsymbol{c}).$$

By Lemma 29, it follows from $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \mathsf{OP}\langle \boldsymbol{x} : F^*(\boldsymbol{q}, \boldsymbol{x})\rangle \succeq b$ that $\mathsf{OP}\langle\{\!\{\boldsymbol{c}[1] : \boldsymbol{c} \in \boldsymbol{S}\}\!\}\rangle \succeq b$. Consequently, $\boldsymbol{S}$ is not in $C_{\Pi}(E)$.

Consider any $\boldsymbol{C}$ in $C_{\Pi}(E)$ such that

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \bigwedge_{\boldsymbol{c} \in \boldsymbol{C}} F^*(\boldsymbol{q}, \boldsymbol{c}).$$

Clearly, $\boldsymbol{C}$ is a subset of $\boldsymbol{S}$. Furthermore $\boldsymbol{C}$ is a strict subset since $\boldsymbol{S}$ is not in $C_{\Pi}(E)$. Consequently,

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \bigvee_{\boldsymbol{c} \in \boldsymbol{O}_{\Pi}(E) \setminus \boldsymbol{C}} F^*(\boldsymbol{q}, \boldsymbol{c}).$$

From right to left: Assume

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \bigwedge_{\boldsymbol{C} \in C_{\Pi}(E)} \Big(\bigwedge_{\boldsymbol{c} \in \boldsymbol{C}} F^*(\boldsymbol{q}, \boldsymbol{c}) \rightarrow \bigvee_{\boldsymbol{c} \in \boldsymbol{O}_{\Pi}(E) \setminus \boldsymbol{C}} F^*(\boldsymbol{q}, \boldsymbol{c}) \Big).$$

Again let $\boldsymbol{S}$ be the set of all lists $\boldsymbol{c}$ of object constants of $\sigma(\Pi)$ of length $|\boldsymbol{x}|$ such that $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models F^*(\boldsymbol{q}, \boldsymbol{c})$. Clearly, $\boldsymbol{S}$ is not in $C_\Pi(E)$. Since

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \not\models \bigwedge_{\boldsymbol{c} \in \boldsymbol{S}} F^*(\boldsymbol{q}, \boldsymbol{c}) \rightarrow \bigvee_{\boldsymbol{c} \in \boldsymbol{O}_\Pi(E) \setminus \boldsymbol{S}} F^*(\boldsymbol{q}, \boldsymbol{c}).$$

Consequently,

$$\mathsf{OP}\langle \{\!\{ \boldsymbol{c}[1] : \boldsymbol{c} \in \boldsymbol{S} \}\!\} \rangle \succeq b.$$

By Lemma 29, we conclude that

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models \mathsf{OP}\langle \boldsymbol{x} : F^*(\boldsymbol{q}, \boldsymbol{x}) \rangle \succeq b.$$

∎

Lemma 31 *Let F be a ground formula that contains no implications. We have that $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models F^*(\boldsymbol{q})$ iff $Y \models F$.*

Proof. By induction on F. ∎

Lemma 32 *Let F be a ground formula such that, for every subformula of the form $G \rightarrow H$, neither G nor H contains implications. Then $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models F^*(\boldsymbol{q})$ iff $X \models F$ and $Y \models F$.*

Proof. By induction on F. We consider when F is $G \rightarrow H$. It is sufficient to show that under the assumption that $X \models F$,

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models G^*(\boldsymbol{q}) \rightarrow H^*(\boldsymbol{q}) \text{ iff } Y \models G \rightarrow H.$$

Since neither G nor H contains an implication, by Lemma 31,

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models G^*(\boldsymbol{q}) \text{ iff } Y \models G$$

and

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models H^*(\boldsymbol{q}) \text{ iff } Y \models H.$$

∎

Lemma 33

$$\boldsymbol{q} \leq \boldsymbol{p} \rightarrow \Bigg(\Big(\bigvee_{\langle B,T\rangle \text{ is an LPS of } \mathcal{I}_\Pi(E)} \Big(\bigwedge_{A\in B} A \wedge \bigwedge_{A\in HB_\Pi\setminus T} \neg A \Big) \Big)^* (\boldsymbol{q}) \leftrightarrow PDB\text{-}SPT_\Pi(E)^*(\boldsymbol{q}) \Bigg)$$

is logically valid.

Proof. In view of Lemma 28, it is sufficient to show that for any maximal LPS $\langle B, T\rangle$ of $\mathcal{I}_\Pi(E)$, the formula

$$\bigvee_{\substack{B',T':\\ B\subseteq B'\subseteq T'\subseteq T}} \Big(\bigwedge_{A\in B'} A^*(\boldsymbol{q}) \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus T'} \neg A \Big) \tag{4.25}$$

is equivalent to

$$\bigwedge_{A\in B} A^*(\boldsymbol{q}) \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus T} \neg A.$$

From right to left: Clear.

From left to right: Assume (4.25). There is a tuple $\langle B', T'\rangle$ such that $B \subseteq B' \subseteq T' \subseteq T$ and

$$\bigwedge_{A\in B'} A^*(\boldsymbol{q}) \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus T'} \neg A.$$

Since $B \subseteq B'$ and $HB_\Pi \setminus T \subseteq HB_\Pi \setminus T'$,

$$\bigwedge_{A\in B} A^*(\boldsymbol{q}) \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus T} \neg A.$$

follows. ∎

Lemmas 22, 23, 24, 25 can be jointly reformulated as follows. We call formula (4.14) $\mathrm{MLPS\text{-}Mod\text{-}FLP}_\Pi(E)$.

The following formulas are logically valid.

(a) $(MLPS\text{-}\mathrm{Fer}_\Pi(E)^*(\boldsymbol{q}) \leftrightarrow \mathrm{Fer}_\Pi(E)^*(\boldsymbol{q}))$*;*

(b) $(MLPS\text{-}FLP_\Pi(E)^*(\boldsymbol{q}) \leftrightarrow FLP_\Pi(E)^*(\boldsymbol{q}))$*;*

(c) $\boldsymbol{q} \leq \boldsymbol{p} \rightarrow (Mod\text{-}FLP_\Pi(E)^*(\boldsymbol{q}) \leftrightarrow PDB\text{-}SPT_\Pi(E)^*(\boldsymbol{q}))$*;*

(d) $\boldsymbol{q} \leq \boldsymbol{p} \rightarrow (MLPS\text{-}Mod\text{-}FLP_{\Pi}(E)^*(\boldsymbol{q}) \leftrightarrow Mod\text{-}FLP_{\Pi}(E)^*(\boldsymbol{q}))$.

The proof of Proposition 17 is immediate from (a); the proof of Proposition 19 is immediate from (c).

Logical validity of (a): It is sufficient to show that for any two subsets B, T of $\boldsymbol{O}_{\Pi}(E)$ such that B is a subset of T, formula

$$\bigwedge_{B \subseteq S \subseteq T} \Big(\bigwedge_{\boldsymbol{c} \in S} F^*(\boldsymbol{q}, \boldsymbol{c}) \rightarrow \bigvee_{\boldsymbol{c} \in \boldsymbol{O}_{\Pi}(E) \setminus S} F^*(\boldsymbol{q}, \boldsymbol{c}) \Big) \tag{4.26}$$

is equivalent to

$$\bigwedge_{\boldsymbol{c} \in B} F^*(\boldsymbol{q}, \boldsymbol{c}) \rightarrow \bigvee_{\boldsymbol{c} \in \boldsymbol{O}_{\Pi}(E) \setminus T} F^*(\boldsymbol{q}, \boldsymbol{c}).$$

From right to left: Clear from the facts that $B \subseteq S$ and $\boldsymbol{O}_{\Pi}(E) \setminus T \subseteq \boldsymbol{O}_{\Pi}(E) \setminus S$.

From left to right: Assume (4.26) and $\bigwedge_{\boldsymbol{c} \in B} F^*(\boldsymbol{q}, \boldsymbol{c})$. Consider several cases, each of which corresponds to a set S such that $B \subseteq S \subseteq T$ and

$$\bigwedge_{\boldsymbol{c} \in S} F^*(\boldsymbol{q}, \boldsymbol{c}) \wedge \bigwedge_{\boldsymbol{c} \in T \setminus S} \neg F^*(\boldsymbol{q}, \boldsymbol{c}). \tag{4.27}$$

It follows from (4.26) and the first conjunctive term of (4.27) that

$$\bigvee_{\boldsymbol{c} \in \boldsymbol{O}_{\Pi}(E) \setminus S} F^*(\boldsymbol{q}, \boldsymbol{c}).$$

From this and the second conjunctive term of (4.27), we conclude that

$$\bigvee_{\boldsymbol{c} \in \boldsymbol{O}_{\Pi}(E) \setminus T} F^*(\boldsymbol{q}, \boldsymbol{c}).$$

∎

Logical validity of (b). Similar to (a).

Logical validity of (c). In view of Lemma 28 and Lemma 33, it is sufficient to prove that, under the assumption $\boldsymbol{q} \leq \boldsymbol{p}$,

$$\bigwedge_{I \in I_{\Pi}(E)} \Big(\bigvee_{A \in I} \neg A \vee \bigvee_{A \in \mathrm{HB}_{\Pi} \setminus I} A^*(\boldsymbol{q}) \Big) \tag{4.28}$$

is equivalent to

$$\bigvee_{\langle B,T\rangle \text{ is an LPS of } \mathcal{I}_\Pi(E)} \Big(\bigwedge_{A\in B} A^*(\boldsymbol{q}) \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus T} \neg A\Big). \tag{4.29}$$

From right to left: Assume (4.29). There exists an LPS $\langle B, T\rangle$ of $\mathcal{I}_\Pi(E)$ such that

$$\Big(\bigwedge_{A\in B} A^*(\boldsymbol{q}) \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus T} \neg A\Big). \tag{4.30}$$

Consider any I in $I_\Pi(E)$. Since $\langle B, T\rangle$ is an LPS of $\mathcal{I}_\Pi(E)$, it is not the case that $B \subseteq I \subseteq T$. Therefore two cases are possible.

Case 1: There is an atom A that belongs to $B \cap (HB_\Pi \setminus I)$. From (4.30), $A^*(\boldsymbol{q})$ follows, so that

$$\Big(\bigvee_{A\in I} \neg A \vee \bigvee_{A\in \mathrm{HB}_\Pi\setminus I} A^*(\boldsymbol{q})\Big) \tag{4.31}$$

follows.

Case 2: There is an atom A that belongs to $I \cap (HB_\Pi \setminus T)$. From (4.30), $\neg A$ follows, so that (4.31) follows.

From left to right: Assume (4.28). Consider several cases, each of which corresponds to sets Y, X of atoms that belong to HB_Π. The assumption characterizing each case is that

$$\Big(\bigwedge_{A\in Y} A^*(\boldsymbol{q}) \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus Y} \neg A^*(\boldsymbol{q})\Big) \tag{4.32}$$

and

$$\Big(\bigwedge_{A\in X} A \wedge \bigwedge_{A\in \mathrm{HB}_\Pi\setminus X} \neg A\Big). \tag{4.33}$$

By Lemma 27, it follows that Y is a subset of X. Take any set S of ground atoms such that $Y \subseteq S \subseteq X$. Clearly,

$$\bigwedge_{A\in S} A$$

follows from (4.33) and

$$\bigwedge_{A\in \mathrm{HB}_\Pi\setminus S} \neg A^*(\boldsymbol{q}),$$

follows from (4.32), so that we get

$$\Big(\bigwedge_{A \in S} A \wedge \bigwedge_{A \in \mathrm{HB}_\Pi \setminus S} \neg A^*(\boldsymbol{q}) \Big).$$

In view of (4.28), it follows that S is not in $I_\Pi(E)$, that is, S is in $\mathcal{I}_\Pi(E)$. Since this holds for every S such that $Y \subseteq S \subseteq X$, it follows that $\langle Y, X \rangle$ is an LPS of $\mathcal{I}_\Pi(E)$.

Logical validity of (d). It is sufficient to show that, under the assumption $\boldsymbol{q} \leq \boldsymbol{p}$, for any two subsets B and T of HB_Π such that B is a subset of T, formula

$$\bigwedge_{B \subseteq S \subseteq T} \Big(\bigvee_{A \in S} \neg A \vee \bigvee_{A \in \mathrm{HB}_\Pi \setminus S} A^*(\boldsymbol{q}) \Big) \tag{4.34}$$

is equivalent to

$$\bigvee_{A \in B} \neg A \vee \bigvee_{A \in \mathrm{HB}_\Pi \setminus T} A^*(\boldsymbol{q}).$$

From right to left: Clear from the facts that $B \subseteq S$ and $\mathrm{HB}_\Pi \setminus T \subseteq \mathrm{HB}_\Pi \setminus S$.

From left to right: Assume (4.34) and $\bigwedge_{A \in B} A$. Consider several cases, each of which corresponds to a set S such that $B \subseteq S \subseteq T$ and

$$\bigwedge_{A \in S} A \wedge \bigwedge_{A \in T \setminus S} \neg A. \tag{4.35}$$

It follows from (4.34) and the first conjunctive term of (4.35) that

$$\bigvee_{A \in \mathrm{HB}_\Pi \setminus S} A^*(\boldsymbol{q}). \tag{4.36}$$

From the second conjunctive term of (4.35), by Lemma 27,

$$\bigwedge_{A \in T \setminus S} \neg A^*(\boldsymbol{q}).$$

From this and (4.36) we conclude that

$$\bigvee_{A \in \mathrm{HB}_\Pi \setminus T} A^*(\boldsymbol{q}).$$

∎

Proof of Proposition 18

Lemma 34 $X \models E$ *iff* $X \models FLP_{\Pi}(E)$.

Proof. *From left to right:* Assume that $X \models E$. Consider any I in $I_{\Pi}(E)$ such that $X \models \bigwedge_{A \in I} A$. Clearly, $I \subseteq X$. From the facts that $X \models E$ and $I \in I_{\Pi}(E)$, we conclude that $X \neq I$. Consequently, $I \subset X$. So there is an atom A such that $A \in X$ and $A \notin I$. In other words, $X \models \bigvee_{A \in \mathrm{HB}_{\Pi} \setminus I} A$.

From right to left: Assume $X \models \mathrm{FLP}_{\Pi}(E)$, that is,

$$X \models \bigwedge_{I \in I_{\Pi}(E)} \Big(\bigwedge_{A \in I} A \rightarrow \bigvee_{A \in \mathrm{HB}_{\Pi} \setminus I} A \Big).$$

It follows that $X \notin I_{\Pi}(E)$. Indeed, it is clear that

$$X \not\models \bigwedge_{A \in X} A \rightarrow \bigvee_{A \in \mathrm{HB}_{\Pi} \setminus X} A.$$

Consequently, $X \models E$. ∎

Proposition 18 *The answer sets of* $FLP(\Pi)$ *are precisely the FLP answer sets of* Π *(as defined in Section 4.1).*

Proof. Without loss of generality, let us assume that the program Π contains no negation in front of aggregate expressions. This is because Π and $\mathrm{Pos}(\Pi)$ are equivalent under the FLP semantics.

Let X be a set of ground atoms of $\sigma(\Pi)$. If $X \not\models \Pi$, then X is not an FLP answer set of Π. Also, in view of Lemma 34, X is not an answer set of $\mathrm{FLP}(\Pi)$.

Assume that $X \models \Pi$. Let

$$A_1; \ldots; A_k \leftarrow A_{k+1}, \ldots, A_m, E_1, \ldots, E_l, \mathrm{not}\ A_{m+1}, \ldots, \mathrm{not}\ A_n \tag{4.37}$$

be any rule in Π, where all A_i are standard atoms and all E_i are aggregate expressions.

In view of Lemma 28, it is sufficient to show that, for any subset Y of X, set Y satisfies the FLP-reduct of (4.37) relative to X iff $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ satisfies

$$\begin{aligned} A^*_{k+1}(\boldsymbol{q}) \wedge \ldots \wedge A^*_m(\boldsymbol{q}) \wedge \mathrm{FLP}(E_1)^*(\boldsymbol{q}) \wedge \ldots \wedge \mathrm{FLP}(E_l)^*(\boldsymbol{q}) \\ \wedge \neg A_{m+1} \wedge \ldots \wedge \neg A_n \ \rightarrow \ A^*_1(\boldsymbol{q}) \vee \ldots \vee A^*_k(\boldsymbol{q}). \end{aligned} \tag{4.38}$$

Case 1: X does not satisfy some A_i for $k{+}1 \leq i \leq m$ or satisfies some A_i for $m{+}1 \leq i \leq n$. It is clear that the FLP reduct of this rule is empty and $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ does not satisfy the antecedent of (4.38), in view of Lemma 27.

Case 2: X does not satisfy some E_i for $1 \leq i \leq l$. Again the reduct is empty. By Lemma 34, $X \not\models \mathrm{FLP}_{\Pi}(E_i)$, and consequently, $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \not\models \mathrm{FLP}_{\Pi}(E_i)^*(\boldsymbol{q})$ by Lemma 27.

Case 3: X satisfies the body of rule (4.37). The reduct is (4.37) itself. Under this condition, it is sufficient to prove that Y satisfies

$$A_1; \ldots; A_k \leftarrow A_{k+1}, \ldots, A_m, \ldots, E_1, \ldots, E_l$$

iff $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ satisfies

$$A^*_{k+1}(\boldsymbol{q}) \wedge \ldots A^*_m(\boldsymbol{q}) \wedge \mathrm{FLP}_{\Pi}(E_1)^*(\boldsymbol{q}) \wedge \cdots \wedge \mathrm{FLP}_{\Pi}(E_l)^*(\boldsymbol{q}) \rightarrow A^*_1(\boldsymbol{q}) \vee \ldots \vee A^*_k(\boldsymbol{q}).$$

The claim follows from Lemmas 32 and 34. ∎

Proof of Proposition 20

Proposition 20 *If formulas $F \leftrightarrow G$ and*

$$\boldsymbol{q} < \boldsymbol{p} \rightarrow (F^*(\boldsymbol{q}) \rightarrow G^*(\boldsymbol{q}))$$

are logically valid, then $SM[G; \boldsymbol{p}] \rightarrow SM[F; \boldsymbol{p}]$ is logically valid.

Proof. Since F is equivalent to G,

$$\begin{aligned} \mathrm{SM}[G; \boldsymbol{p}] \quad &= \quad G \wedge \neg \exists \boldsymbol{u}((\boldsymbol{u} < \boldsymbol{p}) \wedge G^*(\boldsymbol{u})) \\ &\Leftrightarrow \quad F \wedge \neg \exists \boldsymbol{u}((\boldsymbol{u} < \boldsymbol{p}) \wedge G^*(\boldsymbol{u})). \end{aligned}$$

Since $\boldsymbol{u} < \boldsymbol{p} \rightarrow (F^*(\boldsymbol{u}) \rightarrow G^*(\boldsymbol{u}))$ is logically valid, the last formula entails $F \wedge \neg \exists \boldsymbol{u}((\boldsymbol{u} < \boldsymbol{p}) \wedge F^*(\boldsymbol{u}))$ which is exactly $\mathrm{SM}[F; \boldsymbol{p}]$. ∎

Proof of Proposition 21

Proposition 21 *For any program* Π *and any aggregate expression* E *occurring in* Π *and any set* X *of ground atoms of* $\sigma(\Pi)$*,* $X \models E$ *iff* $X \models PDB\text{-}SPT_\Pi(E)$ *iff* $X \models FLP_\Pi(E)$ *iff* $X \models Fer_\Pi(E)$*.*

Proof. Clearly, $\mathrm{FLP}_\Pi(E)$ is classically equivalent to $\mathrm{Mod\text{-}FLP}_\Pi(E)$, and, by Lemma 24, to $\mathrm{PDB\text{-}SPT}_\Pi(E)$. According to Lemma 34, $X \models \mathrm{FLP}_\Pi(E)$ iff $X \models E$. Also, as a special case of Lemma 30 when Y=X, we get $X \models E$ iff $X \models \mathrm{Fer}_\Pi(E)$. ∎

Proofs of Lemma 26 and Proposition 22

Lemma 26

$$(\boldsymbol{q} \leq \boldsymbol{p}) \rightarrow (PDB\text{-}SPT_\Pi(E)^*(\boldsymbol{q}) \rightarrow FLP_\Pi(E)^*(\boldsymbol{q}))$$

is logically valid.

Proof. In view of Lemma 24 and Lemma 28, it is sufficient to prove that, under the assumption $\boldsymbol{q} \leq \boldsymbol{p}$,

$$\bigwedge_{I \in I_\Pi(E)} \Big(\bigvee_{A \in I} \neg A \vee \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A^*(\boldsymbol{q}) \Big) \\ \rightarrow \bigwedge_{I \in I_\Pi(E)} \Big((\bigwedge_{A \in I} A \rightarrow \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A) \wedge (\bigwedge_{A \in I} A^*(\boldsymbol{q}) \rightarrow \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A^*(\boldsymbol{q})) \Big)$$

is logically valid. Assume

$$\bigwedge_{I \in I_\Pi(E)} \Big(\bigvee_{A \in I} \neg A \vee \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A^*(\boldsymbol{q}) \Big)$$

and consider any I in $I_\Pi(E)$.

Case 1: $\bigvee_{A \in I} \neg A$. It follows from Lemma 27 that,

$$\bigvee_{A \in I} \neg A \rightarrow \bigvee_{A \in I} \neg A^*(\boldsymbol{q})$$

is logically valid. Thus we get

$$\Big(\bigvee_{A \in I} \neg A \vee \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A \Big) \wedge \Big(\bigvee_{A \in I} \neg A^*(\boldsymbol{q}) \vee \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A^*(\boldsymbol{q}) \Big). \tag{4.39}$$

Case 2: $\bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A^*(\boldsymbol{q})$. By Lemma 27,

$$\bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A^*(\boldsymbol{q}) \rightarrow \bigvee_{A \in \mathrm{HB}_\Pi \setminus I} A$$

Thus we get (4.39). ∎

Proposition 22 *Every PDB-SPT answer set of* Π *is an FLP answer set of* Π*.*

Proof. Clearly, $\mathrm{PDB\text{-}SPT}(\Pi)$ is equivalent to $\mathrm{FLP}(\Pi)$. In view of Lemma 28 it is sufficient to prove that

$$(\boldsymbol{q} \leq \boldsymbol{p}) \rightarrow (\mathrm{PDB\text{-}SPT}_\Pi(E)^*(\boldsymbol{q}) \rightarrow \mathrm{FLP}_\Pi(E)^*(\boldsymbol{q}))$$

which is Lemma 26. ∎

Proofs of Proposition 23 and Proposition 24

Proposition 24 *For any semi-positive program* Π*, the FLP answer sets of* Π *are precisely the Ferraris answer sets of* $Pos(\Pi)$*.*

Proof. It is sufficient to show that, for any aggregate expression E occurring in a semi-positive program $\mathrm{Pos}(\Pi)$,

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{Fer}_\Pi(E)^*(\boldsymbol{q})$$

iff

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{FLP}_\Pi(E)^*(\boldsymbol{q})$$

Since $\mathrm{Fer}_\Pi(E)$ and $\mathrm{FLP}_\Pi(E)$ are conjunctions of implications $G \rightarrow H$ where each of G and H contains no implications, by Lemma 32, the above is equivalent to saying that

$$X \models \mathrm{Fer}_\Pi(E), \;\; Y \models \mathrm{Fer}_\Pi(E)$$

iff

$$X \models \mathrm{FLP}_\Pi(E), \;\; Y \models \mathrm{FLP}_\Pi(E)$$

This claim follows from Proposition 21.

Proposition 23 *For any semi-positive program* Π*, every FLP answer set of* Π *is a Ferraris answer set of* Π*.*

Proof. According to Proposition 21, for every Herbrand model X of $\sigma(\Pi)$, $X \models \mathrm{FLP}_\Pi(E)$ iff $X \models \mathrm{Fer}_\Pi(E)$. Let $\overline{E}$ be OP$\langle \boldsymbol{x} : F(\boldsymbol{x})\rangle \not\geq b$. In view of the proof of Proposition 24, it is sufficient to show that for any aggregate expression E in the negative body of (4.2), if $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ satisfies $\mathrm{FLP}_{\mathrm{Pos}(\Pi)}(\overline{E})^*(\boldsymbol{q})$, then it satisfies $(\neg\mathrm{Fer}_\Pi(E))^*(\boldsymbol{q})$ as well.

By Lemma 27,

$$\text{if } X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{FLP}_{\mathrm{Pos}(\Pi)}(\overline{E})^*(\boldsymbol{q}) \text{ then } X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{FLP}_{\mathrm{Pos}(\Pi)}(\overline{E}). \tag{4.40}$$

Since E contains no implications, in view of the proof of Proposition 24,

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{FLP}_{\mathrm{Pos}(\Pi)}(\overline{E}) \text{ iff } X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{Fer}_\Pi(\overline{E}). \tag{4.41}$$

It is clear that

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{Fer}_\Pi(\overline{E}) \text{ iff } X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \neg\mathrm{Fer}_\Pi(E). \tag{4.42}$$

By Lemma 28,

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \neg\mathrm{Fer}_\Pi(E) \text{ iff } X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models (\neg\mathrm{Fer}_\Pi(E))^*(\boldsymbol{q}).$$

∎

Proof of Proposition 27

Lemma 35 *For any aggregate formula F and any sets of ground atoms X, Y, under the unique name assumption,*

$$X \models NFES_F(Y) \textit{ iff } X \cup (X \setminus Y)^{\boldsymbol{p}}_{\boldsymbol{q}} \models F^*(\boldsymbol{q}).$$

Proof. By induction on F.

Proposition 27 *Y is FLP externally supported by Π w.r.t. X iff X satisfies $ES_{Pos(\Pi)}(Y)$.*

The proof follows immediately from the following lemma.

Lemma 36 *If E contains no implications in it, then under the unique name assumption,*

$$X \models NFES_E(Y) \textit{ iff } X \models E \textit{ and } X \setminus Y \models E.$$

Proof. By Lemma 35,

$$X \models \mathrm{NES}_E(Y) \quad \text{iff} \quad X \cup (X \setminus Y)^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{Fer}_\Pi(E)^*(\boldsymbol{q}).$$

By Lemma 32,

$$X \cup (X \setminus Y)^{\boldsymbol{p}}_{\boldsymbol{q}} \models \mathrm{Fer}_\Pi(E)^*(\boldsymbol{q}) \quad \text{iff} \quad X \models \mathrm{Fer}_\Pi(E) \text{ and } (X \setminus Y) \models \mathrm{Fer}_\Pi(E).$$

By Proposition 21,

$$X \models \mathrm{Fer}_\Pi(E) \text{ and } (X \setminus Y) \models \mathrm{Fer}_\Pi(E) \quad \text{iff} \quad X \models E \text{ and } (X \setminus Y) \models E.$$

∎

Proof of Theorem 8

For any disjunctive program Π with aggregates, we represent a ground disjunctive rule with aggregates in $\mathrm{Ground}(\Pi)$ as

$$H(\boldsymbol{p}) \leftarrow B(\boldsymbol{p}) \tag{4.43}$$

where $\boldsymbol{p}$ is a list of distinct predicate constants in $\sigma(\Pi)$, and $H(\boldsymbol{p})$ is a disjunction of atoms in the head and $B(\boldsymbol{p})$ is a conjunction of literals and aggregate expressions in the body. Let $\boldsymbol{q}$ be a list of new distinct predicate constants of the same length as $\boldsymbol{p}$. By $H(\boldsymbol{q})$ we denote the formula obtained from $H(\boldsymbol{p})$ by replacing every predicate constant p from $\boldsymbol{p}$ by the corresponding predicate constant q in $\boldsymbol{q}$. Similarly we define $B(\boldsymbol{q})$. Given a set Y of ground atoms, we denote by $Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ the set of atoms obtained from Y by substituting the members of $\boldsymbol{q}$ for the corresponding members of $\boldsymbol{p}$.

Lemma 37 *Let Π be a disjunctive program with aggregates. For any Herbrand interpretations X, Y of $\sigma(\Pi)$, and any ground rule*

$$H(\boldsymbol{p}) \leftarrow B(\boldsymbol{p})$$

in $Ground(\Pi)$,

$$Y \models (H(\boldsymbol{p}) \leftarrow B(\boldsymbol{p}))^{\underline{X}}$$

iff

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models B(\boldsymbol{p}) \wedge B(\boldsymbol{q}) \rightarrow H(\boldsymbol{q}). \tag{4.44}$$

Proof. *Case 1:* X satisfies $B(\boldsymbol{p})$. $(H(\boldsymbol{p}) \leftarrow B(\boldsymbol{p}))^{\underline{X}}$ is $H(\boldsymbol{p}) \leftarrow B(\boldsymbol{p})$. On the other hand, (4.44) is equivalent to saying that $Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models B(\boldsymbol{q}) \rightarrow H(\boldsymbol{q})$, which in turn is equivalent to saying that $Y \models B(\boldsymbol{p}) \rightarrow H(\boldsymbol{p})$.

Case 2: X does not satisfy $B(\boldsymbol{p})$. $(H(\boldsymbol{p}) \leftarrow B(\boldsymbol{p}))^{\underline{X}}$ is equivalent to $\top$; it is clear that (4.44) holds. ∎

Lemma 38 *For any finite ground program* Π*,* $X \models \Pi^{FOL}$ *iff* X *satisfies* $\Pi^{\underline{X}}$*.*

Proof. Immediate from the definition of $\Pi^{\underline{X}}$.

Theorem 8 *Let* Π *be a finite disjunctive program with aggregates that contains at least one object constant. The FLP-answer sets of* Π *in the sense of (Faber et al., 2011) are precisely the Herbrand models of* $FLP[\Pi]$ *whose signature is* $\sigma(\Pi)$*.*

Proof. Let $\boldsymbol{x}$ be the list of variables occurring in Π and let $H(\boldsymbol{x}) \leftarrow B(\boldsymbol{x})$ be a rule of Π. In view of Lemma 38, X is an FLP answer set of Π iff

(i) $X \models \Pi^{FOL}$, and

(ii) no proper subset Y of X satisfies $(\mathrm{Ground}(\Pi))^{\underline{X}}$.

On the other hand, X is a Herbrand model of $\mathrm{FLP}[\Pi]$ iff

(i') $X \models \Pi^{FOL}$, and

(ii') X does not satisfy $\exists \boldsymbol{u}(\boldsymbol{u} < \boldsymbol{p} \wedge \Pi^{\triangle}(\boldsymbol{u}))$.

Condition (ii) can be reformulated as: no proper subset Y of X satisfies every rule $(H(\boldsymbol{t}) \leftarrow B(\boldsymbol{t}))^{\underline{X}}$ where $H(\boldsymbol{t}) \rightarrow B(\boldsymbol{t}) \in \mathrm{Ground}(\Pi)$. Condition (ii') can be reformulated as: there is no proper subset Y of X such that, for every rule $H(\boldsymbol{x}) \leftarrow B(\boldsymbol{x})$ in Π and for every tuple $\boldsymbol{t}$ of ground terms, $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}$ satisfies $(H(\boldsymbol{t}) \leftarrow B(\boldsymbol{t}))^{\triangle}(\boldsymbol{q})$. By Lemma 37, it follows that (ii) is equivalent to (ii'). ∎

Proof of Proposition 30

Proposition 30 *For any finite general program* Π *with aggregates, if* X *is an Herbrand interpretation of* $\sigma(\Pi)$ *that satisfies* $FLP[\Pi]$*, then* X *is a minimal model of* Π*.*

Proof. Assume that X is an Herbrand interpretation of $\sigma(\Pi)$ that satisfies $\mathrm{FLP}[\Pi]$ and Y is any proper subset of X. Note that

$$X \models \Pi^{FOL} \wedge \neg\exists \boldsymbol{u}(\boldsymbol{u} < \boldsymbol{p} \wedge \Pi^{\triangle}(\boldsymbol{u}))$$

iff $X \models \Pi^{FOL}$ and, for every proper subset Z of X,

$$X \cup Z^{\boldsymbol{p}}_{\boldsymbol{q}} \not\models \Pi^{\triangle}(\boldsymbol{q}).$$

Consequently, there exists a rule $H \leftarrow B$ in $\mathrm{Ground}(\Pi)$ such that $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models B(\boldsymbol{q}) \wedge B$ and $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \not\models H(\boldsymbol{q})$. Since $B(\boldsymbol{q})$ and $H(\boldsymbol{q})$ do not contain any predicate from $\boldsymbol{p}$, it follows that $X \models B$, $Y \models B$ and $Y \not\models H$. Consequently, Y does not satisfy $\mathrm{Ground}(\Pi)$. ∎

Chapter 5

FIRST-ORDER STABLE MODEL SEMANTICS FOR GENERALIZED QUANTIFIED FORMULA

Applications of answer set programming motivated various recent extensions to the stable model semantics, for instance, to incorporate aggregates (Faber et al., 2011; Ferraris, 2005; Son & Pontelli, 2007) and abstract constraint atoms (Marek & Truszczynski, 2004), and to facilitate interface with external information source, such as ontology descriptions (Eiter et al., 2008a). While the extensions were driven by different motivations and applications, a common underlying issue is to extend the stable model semantics to incorporate "complex atoms," such as aggregates, abstract constraint atoms and dl-atoms. However, a systematic study is still missing. As we showed in the introduction, many mathematical results on the stable model semantics (e.g. splitting theorem, the theorem on completion, the theorem on loop formulas, the theorem on strong equivalence and the theorem on safety) were re-proven for each of the extension.

HEX programs (Eiter et al., 2005) provide an elegant solution to incorporate such different extensions in a uniform framework via "external atoms." The idea is to define the meaning of external atoms in terms of external functions. For example, aggregate COUNT$\langle x.p(x,a)\rangle \geq 3$ is modelled by a binary external function $f_{\#count}$ such that given an Herbrand interpretation I, $f_{\#count}(I,a,3) = 1$ iff the cardinality of the set $\{c \mid c \in |I|, I \models p(c,a)\}$ is ≥ 3. Once the notion of satisfaction is extended to cover external atoms, the stable models of HEX programs are defined as minimal models of the "FLP-reduct" (Faber et al., 2011). The adoption of the FLP reduct instead of the traditional Gelfond-Lifschitz reduct was a key idea to incorporate external atoms in HEX programs. HEX programs are well studied (Eiter et al., 2006a, 2008a, 2011), and was implemented in the system dlvhex.[1] On the other hand, as we discussed in Section 2.4, FLP semantics is limited to the propositional case and it suffers from some unintuitiveness.

So one wonders: is it possible to combine the versatility of HEX programs and the semantic properties of the first-order stable model semantics? How is the new semantics

[1] http://www.kr.tuwien.ac.at/research/systems/dlvhex/

related to the HEX semantics? These are the subjects of this chapter.

It is hinted in (Ferraris & Lifschitz, 2010) that aggregates may be viewed in terms of generalized quantifiers—a generalizations of the standard quantifiers, $\forall$ and $\exists$, introduced by Mostowski (1957). We follow up on that suggestion, and present an alternative approach to HEX programs by understanding external atoms in terms of generalized quantifiers. Our semantics avoids the above issues with the FLP semantics, and allows natural extensions to several important theorems about the first-order stable model semantics from (Ferraris et al., 2011a), such as the splitting theorem, the theorem on completion and the theorem on strong equivalence, to formulas with generalized quantifiers (GQ-formulas), which in turn can be applied to the individual extensions. This saves efforts in re-proving the theorems for these individual cases. It also allows us to combine the individual extensions in a single language as in the following example.

Example 13 *Consider an extension of nonmonotonic dl-programs $(\mathcal{T}, \Pi)$ that allows aggregates. The ontology description $\mathcal{T}$ specifies that every married man has a spouse who is a woman and similarly for married woman:*

$$Man \sqcap Married \sqsubseteq \exists Spouse.Woman$$
$$Woman \sqcap Married \sqsubseteq \exists Spouse.Man.$$

The following program Π counts the number of people who are eligible for an insurance discount:

$$discount(x) \leftarrow not\ accident(x),$$
$$\#dl[Man \uplus mm, Married \uplus mm, Woman \uplus mw, Married \uplus mw; \exists Spouse.\top](x).$$
$$discount(x) \leftarrow discount(y), family(y, x), not\ accident(x).$$
$$numOfDiscount(z) \leftarrow \mathsf{COUNT}\langle x.discount(x)\rangle = z.$$

The first rule describes that everybody who has a spouse and has no accident is eligible for a discount. The second rule describes that everybody who has no accident and has a family member with a discount is eligible for a discount. We will see that our method can provide the semantics of this combination.

Interestingly, our approach allows us to discover two new extensions of the stable model semantics, yet another semantics of logic programs with abstract constraints, and yet another semantics of nonmonotonic dl-programs, both of which are again special cases of GQ-formulas, and, distinct from the previous definitions, are close to the first-order stable model semantics.

To compare with the HEX semantics, we extend the first-order FLP semantics for programs with aggregate further to programs containing generalized quantifiers, which can be viewed as extending HEX programs to the first-order level. We relate the FLP semantics and the first-order stable model semantics in the general context of programs with generalized quantifiers.

This chapter is organized as follows. In Section 5.1, we review the syntax and the semantics of GQ-formulas. We also present two equivalent stable models semantics, one based on the SM operator and the other based on grounding and reduct. Section 5.2 shows that extension of the stable model semantics, such as logic programs with aggregates, constraints, and nonmonotonic dl-atoms, can be viewed as special cases of GQ-formulas. In Section 5.3, we extend important theorems in answer set programming, such as the splitting theorem, the theorem on completion, and the theorem on strong equivalence, theorem on safety and theorem on loop formulas to GQ-formulas. In Section 5.4, we extend the first-order FLP semantics to cover programs with generalized quantifiers and relate it to the first-order stable model semantics. Section 5.5 proposes a new semantics for nonmonotonic dl-programs. Section 5.6 relates the proposed semantics to the existing semantics of logic programs with generalized quantifiers and to the stable model semantics of infinitary formula. Section 5.7 concludes this chapter.

5.1 Stable Models of Formulas with Generalized Quantifiers

Syntax of Formulas with Generalized Quantifiers

We follow the definition of a formula with generalized quantifiers from (Westerståhl, 2008), which allows Lindström quantifiers (Lindström, 1966) without the isomorphism closure condition.

We assume a set Q of symbols for generalized quantifiers. Each symbol in Q is

associated with a tuple of nonnegative integer $\langle n_1, \ldots, n_k \rangle$ ($k \geq 0$, and each n_i is ≥ 0), called the *type*. A *formula (with the set $\boldsymbol{Q}$ of generalized quantifiers)* is defined in a recursive way.

- an atomic formula is a formula;
- if $F_1, \ldots, F_k$ are formulas and Q is a generalized quantifier of type $\langle n_1, \ldots, n_k \rangle$, then

$$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k)) \tag{5.1}$$

  is a formula, where each $\boldsymbol{x}_i$ ($1 \leq i \leq k$) is a list of distinct object variables whose length is n_i.

We say that an occurrence of a variable x in a formula F is *bound* if it belongs to a subformula of F that has the form $Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k))$, where x is in some $\boldsymbol{x}_i$. Otherwise it is *free*. We say that x is *free* in F if F contains a free occurrence of x. A *sentence* is a formula with no free variables.

We assume that $\boldsymbol{Q}$ contains a type $\langle\rangle$ quantifier $Q_\bot$, a type $\langle 0 \rangle$ quantifier $Q_\neg$, type $\langle 0, 0 \rangle$ quantifiers $Q_\wedge, Q_\vee, Q_\rightarrow$, and type $\langle 1 \rangle$ quantifiers $Q_\forall, Q_\exists$. Each of them corresponds to the standard logical connectives and quantifiers, $\bot, \neg, \wedge, \vee, \rightarrow, \forall, \exists$. These generalized quantifiers will often be written in the familiar form. For example, we write $F \wedge G$ in place of $Q_\wedge[][](F, G)$, and write $\forall x F(x)$ in place of $Q_\forall[x](F(x))$.

Semantics of Formulas with Generalized Quantifiers

An interpretation I of a signature σ consists of a nonempty set U, called the *universe* of I, and a mapping c^I for each function or predicate constant c in σ. For each function constant f of σ whose arity is n, f^I is an element of U if n is 0, and is a function from from U^n to U otherwise. For each predicate constant p of σ whose arity is n, p^I is an element of $\{\boldsymbol{t}, \boldsymbol{f}\}$ if n is 0, and is a function from U^n to $\{\boldsymbol{t}, \boldsymbol{f}\}$ otherwise. For each generalized quantifier Q of type $\langle n_1, \ldots, n_k \rangle$, Q^U is a function from $\mathcal{P}(U^{n_1}) \times \cdots \times \mathcal{P}(U^{n_k})$ to $\{\boldsymbol{t}, \boldsymbol{f}\}$, where $\mathcal{P}(U^{n_i})$ denotes the power set of U^{n_i}.

Example 14 *Besides the standard connectives and quantifiers, the following are other examples of generalized quantifiers.*

- *type* $\langle 1 \rangle$ *quantifier* $Q_{\leq 2}$ *such that* $Q^U_{\leq 2}(R) = \boldsymbol{t}$ *iff the cardinality of* R *is* ≤ 2*;* [2]
- *type* $\langle 1 \rangle$ *quantifier* $Q_{majority}$ *such that* $Q^U_{majority}(R) = \boldsymbol{t}$ *iff the cardinality of* R *is greater than the cardinality of* $U \setminus R$*;*
- *type* $\langle 2, 1, 1 \rangle$ *reachability quantifier* Q_{reach} *such that* $Q^U_{reach}(R_1, R_2, R_3) = \boldsymbol{t}$ *iff there are some* $u, v \in U$ *such that* $R_2 = \{u\}$, $R_3 = \{v\}$ *and* (u, v) *belongs to the transitive closure of* R_1.

Consider an interpretation I of a first-order signature σ. σ^I is defined the same as before: signature obtained from σ by adding names for every element in the universe of I. Given a sentence F of σ^I, F^I is defined recursively as follows:

- $p(t_1, \ldots, t_n)^I = p^I(t_1^I, \ldots, t_n^I)$,
- $(t_1 = t_2)^I = (t_1^I = t_2^I)$,
- For a generalized quantifier Q of type $\langle n_1, \ldots, n_k \rangle$,

  $$(Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k)))^I = Q^U((\boldsymbol{x}_1.F_1(\boldsymbol{x}_1))^I, \ldots, (\boldsymbol{x}_k.F_k(\boldsymbol{x}_k))^I),$$

  where $(\boldsymbol{x}_i.F_i(\boldsymbol{x}_i))^I = \{\boldsymbol{\xi} \in U^{n_i} \mid (F_i(\boldsymbol{\xi}^*))^I = \boldsymbol{t}\}$.

We assume that, for the standard logical connectives and quantifiers Q, functions Q^U have the standard meaning:

- $Q^U_\forall(R) = \boldsymbol{t}$ iff $R = U$;
- $Q^U_\exists(R) = \boldsymbol{t}$ iff $R \cap U \neq \emptyset$;
- $Q^U_\wedge(R_1, R_2) = \boldsymbol{t}$ iff $R_1 = R_2 = \{\epsilon\}$;[3]
- $Q^U_\vee(R_1, R_2) = \boldsymbol{t}$ iff $R_1 = \{\epsilon\}$ or $R_2 = \{\epsilon\}$;
- $Q^U_\rightarrow(R_1, R_2) = \boldsymbol{t}$ iff R_1 is $\emptyset$ or R_2 is

[2] It is clear from the type that R is any subset of U. We will skip such explanation.

[3] ϵ denotes the empty tuple. For any interpretation I, $U^0 = \{\epsilon\}$. For I to satisfy $Q_\wedge[][](F, G)$, both $(\epsilon.F)^I$ and $(\epsilon.G)^I$ have to be $\{\epsilon\}$, which means that $F^I = G^I = \boldsymbol{t}$.

$\{\epsilon\}$;

- $Q^U_{\neg}(R) = \boldsymbol{t}$ iff $R = \emptyset$.
- $Q^U_{\bot}() = \boldsymbol{f}$.

We say that an interpretation I satisfies a sentence F, or is a model of F, and write $I \models F$, if $F^I = \boldsymbol{t}$. A sentence F is *logically valid* if every interpretation satisfies F.

Example 15 *Program* Π_1 *in Example 2 is identified with the following GQ-formula* F_1*:*

$$(\neg Q_{(\mathsf{SUM},<)}[x][y](p(x),\ y{=}2) \to p(2))$$
$$\wedge\ (Q_{(\mathsf{SUM},>)}[x][y](p(x),\ y{=}{-1}) \to p(-1))$$
$$\wedge\ (p(-1) \to p(1))\ .$$

Consider two Herbrand interpretations of the universe $U = \{-1, 1, 2\}$*:* $I_1 = \{p(-1), p(1)\}$ *and* $I_2 = \{p(-1), p(1), p(2)\}$*. We have* $(Q_{(\mathsf{SUM},<)}[x][y](p(x),\ y = 2))^{I_1} = \boldsymbol{t}$ *since*

- $(x.p(x))^{I_1} = \{-1, 1\}$ *and* $(y.y{=}2)^{I_1} = \{2\}$*;*
- $Q^U_{(\mathsf{SUM},<)}(\{-1, 1\}, \{2\}) = \boldsymbol{t}$*.*

Similarly, $(Q_{(\mathsf{SUM},>)}[x][y](p(x),\ y{=}{-1}))^{I_2} = \boldsymbol{t}$ *since*

- $(x.p(x))^{I_2} = \{-1, 1, 2\}$ *and* $(y.y{=}{-1})^{I_2} = \{-1\}$*;*
- $Q^U_{(\mathsf{SUM},>)}(\{-1, 1, 2\}, \{-1\}) = \boldsymbol{t}$*.*

Consequently, both I_1 *and* I_2 *satisfy* F_1*.*

We say that a generalized quantifier (5.1) is *monotone in the* i*-th argument position* if the following holds for any interpretation I: if $Q^U(R_1, \ldots, R_k) = \boldsymbol{t}$ and $R_i \subseteq R'_i \subseteq U^{n_i}$, then $Q^U(R_1, \ldots, R_{i-1}, R'_i, R_{i+1}, \ldots, R_k) = \boldsymbol{t}$. Similarly, we say that Q is *anti-monotone in the* i*-th argument position* if the following holds for any interpretation I: if $Q^U(R_1, \ldots, R_k) = \boldsymbol{t}$ and $R'_i \subseteq R_i \subseteq U^{n_i}$, then $Q^U(R_1, \ldots, R_{i-1}, R'_i, R_{i+1}, \ldots, R_k) = \boldsymbol{t}$. We call an argument position of Q *monotone (anti-monotone)* if Q is monotone (anti-monotone) in that argument position.

Let M be a subset of $\{1,\dots,k\}$. We say that Q is *monotone in* M if Q is monotone in the i-th argument position for all i in M. It is easy to check that both $Q_\wedge$ and $Q_\vee$ are monotone in $\{1,2\}$. $Q_\rightarrow$ is anti-monotone in $\{1\}$ and monotone in $\{2\}$; $Q_\neg$ is anti-monotone in $\{1\}$. In Example 14, $Q_{\leq 2}$ is anti-monotone in $\{1\}$ and $Q_{majority}$ is monotone in $\{1\}$. We will see later that (anti-)monotonicity play an important role in the properties of stable models for formulas with generalized quantifiers.

Stable Models of GQ-Formulas

For any first-order formula F and any list of predicates $\boldsymbol{p} = (p_1,\dots,p_n)$, formula $\mathrm{SM}[F;\boldsymbol{p}]$ is defined as

$$F \wedge \neg\exists \boldsymbol{u}((\boldsymbol{u} < \boldsymbol{p}) \wedge F^*(\boldsymbol{u})), \tag{5.2}$$

where $F^*(\boldsymbol{u})$ is defined recursively:

- $p_i(\boldsymbol{t})^* = u_i(\boldsymbol{t})$ for any list $\boldsymbol{t}$ of terms;
- $F^* = F$ for any atomic formula F that does not contain members of $\boldsymbol{p}$;
- $$(Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1),\dots,F_k(\boldsymbol{x}_k)))^* = \\ Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1),\dots,F_k^*(\boldsymbol{x}_k)) \wedge\ Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1),\dots,F_k(\boldsymbol{x}_k)). \tag{5.3}$$

As in Section 2.5, for a sentence F, the models of $\mathrm{SM}[F;\boldsymbol{p}]$ are called the $\boldsymbol{p}$*-stable* models of F and we write $\mathrm{SM}[F]$ in place of $\mathrm{SM}[F;\boldsymbol{p}]$ when $\boldsymbol{p}$ is the list of all predicate constants occurring in F, and call $\boldsymbol{p}$-stable models simply stable models.

Proposition 32 *Let M be a subset of $\{1,\dots,k\}$ and let $Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1),\dots,F_k(\boldsymbol{x}_k))$ be a formula such that no predicate constant from $\boldsymbol{p}$ occurs in F_j for all $j \in \{1,\dots,k\} \setminus M$.*

(a) If Q is monotone in M, then

$$\boldsymbol{u} \leq \boldsymbol{p} \rightarrow ((Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1),\dots,F_k(\boldsymbol{x}_k)))^* \\ \leftrightarrow Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1),\dots,F_k^*(\boldsymbol{x}_k)))$$

is logically valid.

(b) If Q is anti-monotone in M, then

$$\boldsymbol{u} \leq \boldsymbol{p} \rightarrow ((Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1),\ldots,F_k(\boldsymbol{x}_k)))^* \\ \leftrightarrow Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1),\ldots,F_k(\boldsymbol{x}_k)))$$

is logically valid.

Proposition 32 allows us to simplify the formula $F^*(\boldsymbol{u})$ in (5.2) without affecting the models of (5.2). In formula (5.3), if Q is monotone in all argument positions, we can drop the second conjunctive term in view of Proposition 32 (a). Similarly, if Q is anti-monotone in all argument positions, we can drop the first conjunctive term in view of Proposition 32 (b). For instance, recall that each of $Q_\wedge$, $Q_\vee$, $Q_\forall$, $Q_\exists$ is monotone in all its argument positions, and $Q_\neg$ is anti-monotone in $\{1\}$. If F is a standard first-order formula, then (5.3) can be equivalently rewritten as

- $(\neg F)^* = \neg F$;
- $(F \wedge G)^* = F^* \wedge G^*$; $(F \vee G)^* = F^* \vee G^*$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(\forall x F)^* = \forall x F^*$; $(\exists x F)^* = \exists x F^*$.

This is almost the same as the definition of F^* in Section 2.5, except for $(\neg F)^*$.[4] The only propositional connective which is neither monotone nor anti-monotone in all argument positions is $Q_\rightarrow$, for which the simplification does not apply.

Example 15 continued For GQ-sentence F_1 considered earlier, $\mathrm{SM}[F_1]$ is

$$F_1 \wedge \neg\exists u(u < p \wedge F_1^*(u)) \,, \tag{5.4}$$

where $F_1^*(u)$ is equivalent to the conjunction of F_1 and

$$(\neg Q_{(\mathsf{SUM},<)}[x][y](p(x), y{=}2) \rightarrow u(2)) \\ \wedge\, ((Q_{(\mathsf{SUM},>)}[x][y](p(x), y{=}{-}1) \wedge\ Q_{(\mathsf{SUM},>)}[x][y](u(x), y{=}{-}1)) \rightarrow u(-1)) \\ \wedge\, (u(-1) \rightarrow u(1)) \,.$$

I_1 and I_2 considered earlier satisfy (5.4) and thus are stable models of F_1.

[4] $\neg F$ is understood as $F \rightarrow \bot$ in (Ferraris et al., 2011a), but this difference does not affect stable models.

Reduct-Based Definition

We present a reduct-based definition of stable models for GQ-formulas. We also assume a set $\boldsymbol{Q}$ of generalized quantifiers, which contain all propositional connectives and standard quantifiers.

A *ground GQ-formula w.r.t.* I is defined recursively as follows:

- $p(\xi_1^\diamond, \ldots, \xi_n^\diamond)$, where p is a predicate constant of σ and $\xi_i^\diamond$ are object names of σ^I, is a ground GQ-formula w.r.t. I;
- for any $Q \in \boldsymbol{Q}$ of type $\langle n_1, \ldots, n_k \rangle$, if each S_i is a set of pairs of the form $\boldsymbol{\xi}^\diamond.F$ where $\boldsymbol{\xi}^\diamond$ is a list of object names in σ^I whose length is n_i and F is a ground GQ-formula w.r.t. I, then

  $$Q(S_1, \ldots, S_k)$$

  is a ground GQ-formula w.r.t. I.

The following definition of grounding turns any GQ-sentence into a ground GQ-formula w.r.t. an interpretation:

Let F be a GQ-sentence of a signature σ, and let I be an interpretation of σ. By $gr_I[F]$ we denote the ground formula w.r.t. I, which is obtained by the following process:

- $gr_I[p(t_1, \ldots, t_n)] = p((t_1^I)^\diamond, \ldots, (t_n^I)^\diamond)$;
- $gr_I[t_1 = t_2] = \begin{cases} \top & \text{if } t_1^I = t_2^I, \text{ and} \\ \bot & \text{otherwise;} \end{cases}$
- $gr_I[\top] = \top; \quad gr_I[\bot] = \bot;$
- $gr_I[Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k))] = Q(S_1, \ldots, S_k)$

  where $S_i = \{\boldsymbol{\xi}^\diamond.gr_I[F_i(\boldsymbol{\xi}^\diamond)] \mid \boldsymbol{\xi}^\diamond$ is a list of object names of σ^I whose length is the same as $n_i\}$.

For any interpretation I and any ground GQ-formula F w.r.t. I, the satisfaction relation $I \models F$ is defined recursively as follows.

For any interpretation I and any ground formula F w.r.t. I, the *truth value of* F *under* I, denoted by F^I, is defined recursively as follows.

- $p(\xi_1^\diamond, \ldots, \xi_n^\diamond)^I = p^I(\xi_1, \ldots, \xi_n)$;
- $\top^I = \boldsymbol{t}$; $\quad \bot^I = \boldsymbol{f}$;
- $Q(S_1, \ldots, S_k)^I = Q^U(S_1^I, \ldots, S_k^I)$ where $S_i^I = \{\boldsymbol{\xi} \mid \boldsymbol{\xi}^\diamond . F(\boldsymbol{\xi}^\diamond) \in S_i, F(\boldsymbol{\xi}^\diamond)^I = \boldsymbol{t}\}$.

Example 15 continued For Herbrand interpretation $I_1 = \{p(-1), p(1)\}$, formula $gr_{I_1}[F_1]$ is [5]

$$\begin{aligned} &(\neg Q_{(\mathsf{SUM},<)}(\{-1.p(-1), 1.p(1), 2.p(2)\}, \{-1.\bot, 1.\bot, 2.\top\}) \rightarrow p(2)) \\ &\wedge (Q_{(\mathsf{SUM},>)}(\{-1.p(-1), 1.p(1), 2.p(2)\}, \{-1.\top, 1.\bot, 2.\bot\}) \rightarrow p(-1)) \\ &\wedge (p(-1) \rightarrow p(1)) \ . \end{aligned} \tag{5.5}$$

I_1 satisfies $Q_{(\mathsf{SUM},<)}(\{-1.p(-1), 1.p(1), 2.p(2)\}, \{-1.\bot, 1.\bot, 2.\top\})$ because $I_1 \models p(-1)$, $I_1 \models p(1)$, $I_1 \not\models p(2)$, and

$$Q^U_{(\mathsf{SUM},<)}(\{-1, 1\}, \{2\}) = \boldsymbol{t}.$$

I_1 satisfies $Q_{(\mathsf{SUM},>)}(\{-1.p(-1), 1.p(1), 2.p(2)\}, \{-1.\top, 1.\bot, 2.\bot\})$ because

$$Q^U_{(\mathsf{SUM},>)}(\{-1, 1\}, \{-1\}) = \boldsymbol{t}.$$

Consequently, I_1 satisfies (5.5).

Proposition 33 *Let σ be a signature that contains finitely many predicate constants, let $\sigma^{\boldsymbol{p}}$ be the set of predicate constants in σ, let $I = \langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}} \rangle$ be an interpretation of σ, and let F be a GQ-sentence of σ. Then $I \models F$ iff $I^{\boldsymbol{p}} \models gr_I[F]$.*

For any ground formula F w.r.t. I, the *reduct* of F relative to I, denoted by $F^{\underline{I}}$, is obtained by replacing each maximal subformula that is not satisfied by I with $\bot$. It can also be defined recursively as follows.

[5] For simplicity, we write $-1, 1, 2$ instead of their object names $(-1)^\diamond, 1^\diamond, 2^\diamond$.

- $(p(\xi_1^\diamond, \ldots, \xi_n^\diamond))^{\underline{I}} = \begin{cases} p(\xi_1^\diamond, \ldots, \xi_n^\diamond) & \text{if } I \models p(\xi_1^\diamond, \ldots, \xi_n^\diamond), \\ \bot & \text{otherwise;} \end{cases}$
- $\top^{\underline{I}} = \top; \quad \bot^{\underline{I}} = \bot;$
- $(Q(S_1, \ldots, S_k))^{\underline{I}} = \begin{cases} Q(S_1^{\underline{I}}, \ldots, S_k^{\underline{I}}) & \text{if } I \models Q(S_1, \ldots, S_k), \\ \bot & \text{otherwise;} \end{cases}$

  where $S_i^{\underline{I}} = \{\boldsymbol{\xi}^\diamond.(F(\boldsymbol{\xi}^\diamond))^{\underline{I}} \mid \boldsymbol{\xi}^\diamond.F(\boldsymbol{\xi}^\diamond) \in S_i\}$.

Theorem 10 *Let σ be a signature that contains finitely many predicate constants, let $\sigma^{\boldsymbol{p}}$ be the set of predicate constants in σ, let $I = \langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}} \rangle$ be an interpretation of σ, and let F be a GQ-sentence of σ. $I \models SM[F; \sigma^{\boldsymbol{p}}]$ iff $I^{\boldsymbol{p}}$ is a minimal set of atoms that satisfies $(gr_I[F])^{\underline{I}}$.*

Example 15 continued The interpretation I_1 considered before can be identified as the tuple $\langle I^f, \{p(-1), p(1)\} \rangle$ where I^f is empty. The reduct $(gr_{I_1}[F_1])^{\underline{I_1}}$ is

$$(\bot \to \bot)$$
$$\wedge\, (Q_{(\mathsf{SUM},>)}(\{-1 : p(-1), 1 : p(1), 2 : \bot\}, \{-1 : \top, 1 : \bot, 2 : \bot\}) \to p(-1))$$
$$\wedge\, (p(-1) \to p(1))\ .$$

We can check that $\{p(-1), p(1)\}$ is a minimal model of the reduct.

5.2 Extensions of ASP as GQ formulas

In this section, we show that aggregate formulas, programs with abstract or explicit constraints, nonmonotonic dl-programs can be viewed as special cases of GQ formulas.

Aggregates as GQ-Formulas

We identify expression (4.16), which is

$$\mathsf{OP}\langle \boldsymbol{x}^1.F_1, \ldots, \boldsymbol{x}^n.F_n \rangle \succeq b$$

with the GQ-formula

$$Q_{(\mathsf{OP},\succeq)}[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_n][y](F_1(\boldsymbol{x}_1), \ldots, F_n(\boldsymbol{x}_n), y = b)\ , \tag{5.6}$$

where, for any interpretation I, $Q^U_{(\mathsf{OP},\succeq)}$ is a function that maps $\mathcal{P}(U^{|\boldsymbol{x}_1|}) \times \cdots \times \mathcal{P}(U^{|\boldsymbol{x}_n|}) \times \mathcal{P}(U)$ to $\{\boldsymbol{t}, \boldsymbol{f}\}$ such that $Q^U_{(\mathsf{OP},\succeq)}(R_1, \ldots, R_n, R_{n+1}) = \boldsymbol{t}$ iff

- OP(α) is defined, where α is the join of the multisets $msp(R_1), \ldots, msp(R_n)$,
- $R_{n+1} = \{b^I\}$, where $b^I \in$ **Num**, and
- OP$(\alpha) \succeq b^I$;

The following proposition states that this definition is equivalent to the definition from (Ferraris & Lifschitz, 2010).

Proposition 34 *Let F be a first-order sentence with aggregates whose signature is σ, and let $\boldsymbol{p}$ be a list of predicate constants. For any expansion I of σ_{bg} to σ, I is a $\boldsymbol{p}$-stable model of F in the sense of (Ferraris & Lifschitz, 2010) iff I is a $\boldsymbol{p}$-stable model of F in our sense.*

Non-monotonic DL-Programs as GQ Formulas

Let C be a set of object constants, and let $P_{\mathcal{T}}$ and P_{Π} be disjoint sets of predicate constants. A nonmonotonic $dl\text{-}program$ (Eiter et al., 2008a) is a pair $(\mathcal{T}, \Pi)$, where $\mathcal{T}$ is a theory in description logic (DL) of signature $\langle C, P_{\mathcal{T}} \rangle$ and Π is a *generalized* normal logic program of signature $\langle C, P_{\Pi} \rangle$ such that $P_{\mathcal{T}} \cap P_{\Pi} = \emptyset$. We assume that Π contains no variables by applying grounding w.r.t. C. A generalized normal logic program is a set of nondisjunctive rules that can contain queries to $\mathcal{T}$ in the form of "dl-atoms." A *dl-atom* is of the form

$$DL[S_1 op_1 p_1, \ldots, S_k op_k p_k;\ Query](\boldsymbol{t}) \quad (k \geq 0), \tag{5.7}$$

where $S_i \in P_{\mathcal{T}}$, $p_i \in P_{\Pi}$, and $op_i \in \{\uplus, \cup, \cap\}$; $Query(\boldsymbol{t})$ is a *dl-query* as defined in (Eiter et al., 2008a). A $\mathrm{dl\text{-}rule}$ is of the form

$$a \leftarrow b_1, \ldots, b_m, \mathrm{not}\ b_{m+1}, \ldots, \mathrm{not}\ b_n\ , \tag{5.8}$$

where a is an atom and each b_i is either an atom or a dl-atom. We identify rule (5.8) with

$$a \leftarrow B, N\ , \tag{5.9}$$

where B is $b_1, \ldots, b_m$ and N is $\mathrm{not}\ b_{m+1}, \ldots, \mathrm{not}\ b_n$. 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 (5.7) *relative to* $\mathcal{T}$ if $\mathcal{T} \cup \bigcup_{i=1}^{k} A_i(I)$ entails $Query(\boldsymbol{t})$, where $A_i(I)$ is

- $\{S_i(\boldsymbol{e}) \mid\ p_i(\boldsymbol{e}) \in I\}$ if op_i is $\uplus$,
- $\{\neg S_i(\boldsymbol{e}) \mid\ p_i(\boldsymbol{e}) \in I\}$ if op_i is $\uplus$,
- $\{\neg S_i(\boldsymbol{e}) \mid\ p_i(\boldsymbol{e}) \notin I\}$ if op_i is $\cap$.

A ground dl-atom A is *monotonic* relative to $\mathcal{T}$ if, for any two Herbrand interpretations I and I' such that $I \subseteq I'$, $I \models_{\mathcal{T}} A$ implies $I' \models_{\mathcal{T}} A$. Similarly, a ground dl-atom A is *anti-monotonic* relative to $\mathcal{T}$ if, for any two Herbrand interpretations I and I' such that $I \subseteq I'$, $I' \models_{\mathcal{T}} A$ implies $I \models_{\mathcal{T}} A$.

Given a dl-program $(\mathcal{T}, \Pi)$ and an Herbrand interpretation I of $\langle C, P_\Pi \rangle$, the *weak dl-transform* of Π relative to $\mathcal{T}$, denoted by $w\Pi^I_{\mathcal{T}}$, is the set of rules

$$a \leftarrow B' \tag{5.10}$$

where $a \leftarrow B, N$ is in Π, $I \models_{\mathcal{T}} B \wedge N$, and B' is obtained from B by removing all dl-atoms in it. Similarly, the *strong dl-transform* of Π relative to $\mathcal{T}$, denoted by $s\Pi^I_{\mathcal{T}}$, is the set of rules (5.10), where $a \leftarrow B, N$ is in Π, $I \models_{\mathcal{T}} B \wedge N$ and B' is obtained from B by removing all nonmonotonic dl-atoms in it. The only difference between these two definitions is whether monotonic dl-atoms in the positive body remain in the reduct or not.

An Herbrand interpretation I is a *weak (strong, respectively) answer set* of $(\mathcal{T}, \Pi)$ if I is minimal among the sets of atoms that satisfy $w\Pi^I_{\mathcal{T}} (s\Pi^I_{\mathcal{T}}, respectively)$.

Here we understand dl-programs as a special case of GQ formulas. Consider a dl-program $(\mathcal{T}, \Pi)$ such that Π is ground. Under the strong answer set semantics, we identify every dl-atom (5.7) in Π with

$$Q_{(5.7)}[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](p_1(\boldsymbol{x}_1), \dots, p_k(\boldsymbol{x}_k)) \tag{5.11}$$

if it is monotonic relative to $(\mathcal{T}, \Pi)$, and

$$\neg\neg Q_{(5.7)}[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](p_1(\boldsymbol{x}_1), \dots, p_k(\boldsymbol{x}_k)) \tag{5.12}$$

otherwise.

Given an interpretation I, $Q^U_{(5.7)}$ is a function that maps $\mathcal{P}(U^{|\boldsymbol{x}_1|}) \times \cdots \times \mathcal{P}(U^{|\boldsymbol{x}_k|})$ to $\{\boldsymbol{t}, \boldsymbol{f}\}$ such that, $Q^U_{(5.7)}(R_1, \ldots, R_k) = \boldsymbol{t}$ iff $\mathcal{T} \cup \bigcup_{i=1}^{k} A_i(R_i)$ entails $Query(\boldsymbol{t})$, where $A_i(R_i)$ is

- $\{S_i(\boldsymbol{\xi}_i) \mid \boldsymbol{\xi}_i \in R_i\}$ if op_i is $\uplus$,
- $\{\neg S_i(\boldsymbol{\xi}_i) \mid \boldsymbol{\xi}_i \in R_i\}$ if op_i is $\uplus$,
- $\{\neg S_i(\boldsymbol{\xi}_i) \mid \boldsymbol{\xi}_i \in U^{|\boldsymbol{x}_i|} \setminus R_i\}$ if op_i is $\cap$.

We say that I is a *strong answer set* of $(\mathcal{T}, \Pi)$ if I satisfies $\mathrm{SM}[\Pi; P_\Pi]$.

Similarly a weak answer set of $(\mathcal{T}, \Pi)$ is defined by identifying every dl-atom (5.7) in Π with (5.12) regardless whether A is monotonic or not.

Example 13 continued *The dl-atom*

$$\#dl[Man \uplus \mathrm{mm}, Married \uplus \mathrm{mm}, Woman \uplus \mathrm{mw}, Married \uplus \mathrm{mw}; \exists Spouse.\top](alice) \tag{5.13}$$

is identified with the generalized quantified formula

$$Q_{(5.13)}[x_1][x_2][x_3][x_4](\mathrm{mm}(x_1), \mathrm{mm}(x_2), \mathrm{mw}(x_3), \mathrm{mw}(x_4)) \tag{5.14}$$

where, for any interpretation I, $Q^U_{(5.13)}$ is a function that maps $\mathcal{P}(U) \times \mathcal{P}(U) \times \mathcal{P}(U) \times \mathcal{P}(U)$ to $\{\boldsymbol{t}, \boldsymbol{f}\}$ such that $Q^U_{(5.13)}(R_1, R_2, R_3, R_4) = \boldsymbol{t}$ iff $\mathcal{T} \cup \{Man(c) \mid c \in R_1\} \cup \{Woman(c) \mid c \in R_3\} \cup \{Married(c) \mid c \in R_2 \cup R_4\}$ entails $\exists x Spouse(alice, x)$.

Consider an Herbrand interpretation $I = \{mw(alice)\}$, which satisfies (5.13). I also satisfies (5.14) since $(x.mw(x))^I = \{alice\}$ and $\mathcal{T} \cup \{Woman(alice), Married(alice)\}$ entails $\exists x Spouse(alice, x)$.

The following proposition tells us that the definitions of a strong answer set and a weak answer set given here are equivalent to the definitions from (Eiter et al., 2008a).

Proposition 35 *For any dl-program $(\mathcal{T}, \Pi)$, an Herbrand interpretation is a strong (weak, respectively) answer set of $(\mathcal{T}, \Pi)$ in the sense of (Eiter et al., 2008a) iff it is a strong (weak, respectively) answer set of $(\mathcal{T}, \Pi)$ in our sense.*

Abstract Constraint Atoms as GQ-Formulas

Abstract constraint atoms is a generalization of aggregates to represent arbitrary constraints on atoms. Intuitively, a constraint A represents a condition on models of the program containing A. The definition of A includes an explicit description of conditions interpretations have to meet in order to satisfy it.

Formally, let σ be a propositional signature, D be a finite set of atoms of σ and C be a subset of the power set $\mathcal{P}(D)$. An *abstract constraint atom* (or c-atom following (Son et al., 2007)) is of the form (D, C). Intuitively, a c-atom represents a constraint with a finite set C of admissible solutions over a finite domain D. Given a propositional signature σ, for any interpretation I of σ, we say that I satisfies an c-atom (D, C) if $I \cap D \in C$. Consider the c-atom $(\{a, b\}, \{\{\ \}, \{b\}, \{a, b\}\})$. For any interpretation I, I satisfies the c-atom iff $a \notin I$.

Son, Pontelli and Tu (Son et al., 2007) proposed a semantics of program with abstract constraints atoms based on a extended definition of satisfaction, namely "conditional satisfaction." Lemma 6 from (Son & Pontelli, 2007) shows that the definition is equivalent to the semantics of aggregates by Pelov, Denecker, Bruynooghe (Pelov et al., 2007), which translate aggregates into nested expressions. We will just present the reductive semantics which involves the notion of a "(maximal) local power set.".

For any c-atom (D, C), by $\mathrm{SPT}(D, C)$[6] (Son & Pontelli, 2007) we denote the formula

$$\bigvee_{\langle B,T\rangle \text{ is a maximal LPS of } C} \Big(\bigwedge_{p_i \in B} p_i \wedge \bigwedge_{p_i \in D \setminus T} \neg p_i \Big). \tag{5.15}$$

By $\mathrm{SPT}(\Pi)$ we denote the propositional formula obtained from $\mathrm{Ground}(\Pi)$ by replacing all c-atoms (D, C) in it by $\mathrm{SPT}(D, C)$. The SPT answer sets of Π are defined as the answer sets of $\mathrm{SPT}(\Pi)$ in the sense of SM.

Example 16 *The following example is from (Liu, Pontelli, Son, & Truszczynski, 2010). Con-*

[6] Here we understand a c-atom as an aggregate

sider the following program Π.

$$a. \qquad b.$$
$$c \leftarrow ((a,b,c),\{\{a\},\{a,c\},\{a,b,c\}\}).$$

$SPT(\Pi)$ *is the formula*

$$a \wedge b \wedge (((a \wedge \neg b) \vee (a \wedge c)) \rightarrow c).$$

It is easy to check that there is no SPT *answer sets of* Π.

We view c-atoms as a special case of generalized quantifiers containing no variables, and this provides an alternative semantics of c-atoms that is different from (Son et al., 2007). An abstract constraint $\langle D, C\rangle$, where D is $(p_1,\ldots,p_n)$, can be viewed as a GQ-formula

$$Q_C[\,]\ldots[\,]\ D\ , \tag{5.16}$$

where, for any interpretation I of σ, Q_C^U is a function that maps $\mathcal{P}(\{\epsilon\}) \times \cdots \times \mathcal{P}(\{\epsilon\})$ to $\{\boldsymbol{t}, \boldsymbol{f}\}$ such that $Q_C^U(R_1,\ldots,R_n) = \boldsymbol{t}$ iff $\{p_i \mid 1 \leq i \leq n,\ R_i = \{\epsilon\}\} \in C$.

The following lemma follows immediately from the definition:

Lemma 39 *For any c-atom* $\langle D, C\rangle$ *of* σ *and any interpretation* I *of* σ*,* I *satisfies* $\langle D, C\rangle$ *in the sense of (Son et al., 2007) iff* $I \models (5.16)$.

A *propositional formula with c-atoms* extends the standard syntax of a propositional formula by treating c-atoms as a base case in addition to standard atoms. The stable model semantics of such a formula is defined by understanding the formula as shorthand for the corresponding GQ-formula as described in Lemma 50.

Example 16 continued *Consider the formula* F *that corresponds to* Π

$$a \wedge b \wedge ((a,b,c),\{\{a\},\{a,c\},\{a,b,c\}\}) \rightarrow c.$$

For new atoms d*,*e*,*f*, formula* $F^*(d,e,f)$ *is*

$$d \wedge e \wedge ((a,b,c),\{\{a\},\{a,c\},\{a,b,c\}\}) \rightarrow c) \wedge$$
$$((a,b,c),\{\{a\},\{a,c\},\{a,b,c\}\}) \wedge ((d,e,f),\{\{d\},\{d,f\},\{d,e,f\}\}) \rightarrow f).$$

Any subset X of $\{a, b, c\}$ is an answer set of F iff X satisfies F and for any proper subset Y of X, $X \cup Y^{abc}_{def}$ does not satisfy $F^*(d, e, f)$. (Here Y^{abc}_{def} is the set obtained from Y by replacing a, b, c with d, e, f.)

We can check that $\{a, b\}$ is the only answer set of F. Indeed, $\{a, b\}$ satisfies F and each of $\{a, b\}$, $\{a, b, d\}$, $\{a, b, e\}$ does not satisfy $F^*(d, e, f)$.

Given a c-atom (5.16), we define its *propositional formula representation* as

$$\bigwedge_{\overline{C} \in \mathcal{P}(D) \setminus C} \Big(\bigwedge_{p \in \overline{C}} p \to \bigvee_{p \in D \setminus \overline{C}} p \Big). \tag{5.17}$$

For any propositional formula F with c-atoms, by $\mathrm{Fer}(F)$, we denote the usual propositional formula obtained from F by replacing every c-atom (5.16) with (5.17).

The following proposition tells us that c-atoms in a formula can be rewritten as propositional formulas under the stable model semantics from (Ferraris, 2005).

Proposition 36 *For any propositional formula F with c-atoms and any propositional interpretation X, X is an answer set of F iff X is an answer set of $Fer(F)$.*

Example 16 continued *For the formula F above, $\mathrm{Fer}(F)$ is*

$$a \wedge b \wedge (((a \vee b \vee c) \wedge (b \to a \vee c) \wedge (c \to a \vee b) \wedge (a \wedge b \to c) \wedge (b \wedge c \to a)) \to c).$$

We check that $\{a, b\}$ is the only answer set of $\mathrm{Fer}(F)$ in accordance with Proposition 36.

Explicit Constraints as GQ-Formulas

Let σ_{bg} be a *background* signature consisting of function and predicate constants, and let σ be a superset of σ_{bg}. As before, we designate some ground terms that can be constructed from $\sigma \setminus \sigma_{bg}$ as *constraint variables*. An *explicit constraint* is a formula formed from a background signature σ_{bg} allowing elements of V also used as terms. Let C be the set of all explicit constraints. Like SMT theories (Biere, Biere, Heule, van Maaren, & Walsh, 2009), we assume that the evaluation of constants in σ_{bg} is pre-determined. For instance, numbers are evaluated as themselves, built-in functions (e.g., $+$, $-$), and built-in predicates (e.g., $\leq$, $>$) have their intended meanings. Formally, a *background interpretation* I_{bg} is a

fixed interpretation of σ_{bg}. Given a constraint satisfaction problem (V, D, C), we assume that all elements d in $\mathrm{Dom}(v)$, where $v \in V$, are object constants in σ_{bg} such that $d^{I_{bg}} = d$.

Example 17 *Take σ_{bg} to be the signature that contains all integers and arithmetic functions $+$ and sq(uare). Let V be $\{side_1(obj), side_2(obj), side_3(obj)\}$. The domain of every constraint variable is the set of all integers.*

$$sq(side_1(obj)) + sq(side_2(obj)) = sq(side_3(obj))$$
$$\wedge\, triangle(obj) \rightarrow rightTriangle(obj)$$

is a formula with explicit constraints.

For any formula F of σ and any list of intensional predicates $\boldsymbol{p} = (p_1, \ldots, p_n)$ whose members belong to $\sigma \setminus \sigma_{bg}$, formula $\mathrm{SM}[F; \boldsymbol{p}]$ is defined as

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

where $F^*(\boldsymbol{u})$ is defined as before.

An explicit constraint c, where $v_1, \ldots, v_n$ is the list of all constraint variables that occurs in it, can be viewed as a generalized quantified formula

$$Q_c[x_1] \ldots [x_n](x_1 = v_1, \ldots, x_n = v_n), \tag{5.18}$$

where for any interpretation I that conforms to (V, D, C), $Q_c^I(R_1, \ldots, R_n) = \boldsymbol{t}$ iff each R_i is some singleton set $\{d_i\}$ such that $d_i \in \mathrm{Dom}(v_i)$ and $I \models c'$ where c' is obtained from c by replacing every v_i with d_i $(1 \leq i \leq n)$.

Example 17 continued *The explicit constraint c_1*

$$sq(side_1(obj)) + sq(side_2(obj)) = sq(side_3(obj))$$

is understood as the generalized quantified formula

$$Q_{c_1}[x_1][x_2][x_3](x_1 = side_1(obj), x_2 = side_2(obj), x_3 = side_3(obj)),$$

where $Q_{c_1}^I(R_1, R_2, R_3) = \boldsymbol{t}$ iff each R_i is a singleton set $\{d_i\}$ such that d_i is an integer and $sq(d_1) + sq(d_2) = sq(d_3)$.

The *GQ-representation* ("Generalized Quantified Formula representation") of a formula F with constraints (denoted by F^{GQ}) is the formula obtained from F by identifying all constraints c by their corresponding generalized quantified formula (5.18).

Proposition 37 *Let (V, D, C) be a constraint satisfaction problem, let F be a sentence of signature σ with explicit constraints of signature σ_{bg} such that $\sigma_{bg} \subseteq \sigma$, let $\boldsymbol{p}$ be a list of predicates whose members belong to $\sigma \setminus \sigma_{bg}$. For any expansion I of an interpretation of σ_{bg} to σ that conforms to (V, D, C), $I \models SM[F;\ \boldsymbol{p}]$ iff $I \models SM[F^{GQ};\ \boldsymbol{p}]$.*

5.3 Important Theorems

In this section, we extend some important theorems of the programs under the stable models semantics to formulas with generalized quantifiers.

Strong Equivalence

Strong equivalence (Lifschitz et al., 2001) is an important notion that allows us to substitute one subformula for another subformula without affecting the stable models. In the following, we extend the theorem on strong equivalence GQ-formulas.

About GQ-formulas F and G we say that F is *strongly equivalent* to G if, for any formula H, any occurrence of F in H, and any list $\boldsymbol{p}$ of distinct predicate and function constants, $\mathrm{SM}[H; \boldsymbol{p}]$ is equivalent to $\mathrm{SM}[H'; \boldsymbol{p}]$, where H' is obtained from H by replacing the occurrence of F by G. In this definition, H is allowed to contain object, function and predicate constants that do not occur in F, G; Theorem 11 below shows, however, that this is not essential.

Theorem 11 *Let F and G be GQ-formulas, let $\boldsymbol{p}$ be the list of all predicate constants occurring in F or G and let $\boldsymbol{u}$ be a list of distinct predicate variables corresponding to $\boldsymbol{p}$. Formulas F and G are strongly equivalent to each other iff the formula*

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

is logically valid.

Example 18 *The program (2.3) can be identified with the formula* F

$$(\neg q \to p(a)) \wedge \ (\mathsf{COUNT}\langle x.p(x)\rangle < 1 \to q),$$

and is strongly equivalent to the following formula G*:*

$$(\neg \mathsf{COUNT}\langle x.p(x)\rangle < 1 \to p(a)) \wedge \ (\mathsf{COUNT}\langle x.p(x)\rangle < 1 \to q).$$

One can check that $F^*(u,v)$ *and* $G^*(u,v)$ *are equivalent to each other.*

Splitting Theorem

The splitting method(Lifschitz & Turner, 1994; Janhunen & Oikarinen, 2004; Ferraris, Lee, Lifschitz, & Palla, 2009a) can reduce the task of computing the stable models of a logic program to similar tasks for smaller programs. In this section, we extend the splitting theorem from (Ferraris, Lee, Lifschitz, & Palla, 2009b) to GQ-formulas.

Let F be a GQ-formula. We say that an occurrence of p in F is *mixed* if there is some generalized quantifier Q that contains the occurrence in its argument position which is neither monotone nor anti-monotone. Let l be the number of generalized quantifiers Q in F such that the occurrence of p belongs to an anti-monotone argument position of Q. If the occurrence is not mixed then we call it *positive* in F if l is even, and *negative* otherwise. The occurrence is *strictly positive* in F if $l = 0$. We call an occurrence of predicate constant *semi-positive* if it is positive or mixed. Similarly, it is *semi-negative* if it is negative or mixed.

We say that F is *negative on* $\boldsymbol{p}$ if there is no strictly positive occurrence of a predicate constant from $\boldsymbol{p}$ in F. An occurrence of a predicate constant or a subformula of F is $\boldsymbol{p}$*-negated* in F if it is contained in a subformula of F that is negative on $\boldsymbol{p}$.

The predicate dependency graph of F relative to a list $\boldsymbol{p}$ of intensional predicates (denoted by $\mathrm{DG}_{\boldsymbol{p}}[F]$) is a directed graph such that

- the vertices are the members of $\boldsymbol{p}$, and
- there is an edge from p to q if there is a strictly positive occurrence of a subformula $G = Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](F_1,\ldots,F_k)$ such that
  - p has a strictly positive occurrence in G, and

– q has a semi-positive, non-$\boldsymbol{p}$-negated occurrence in a non-monotone argument position of Q.

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

Example 13 continued *Figure 5.1 shows the dependency graph of F relative to $\{discount, family, mm, mw, accident, numOfDiscount\}$.*

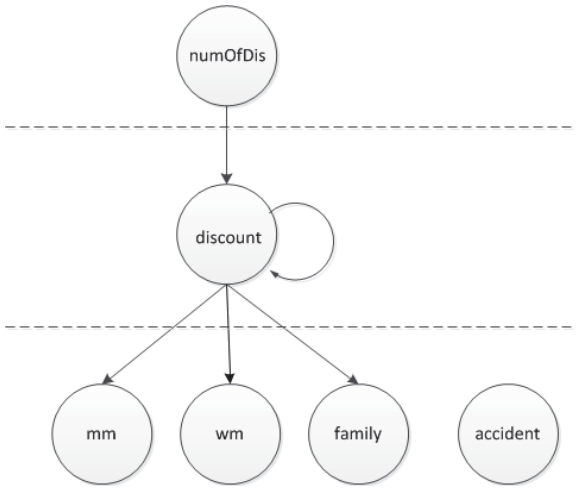

Figure 5.1: The predicate dependency graph of the formula in Example 13

Theorem 12 *Let F be a GQ-sentence, and let $\boldsymbol{p}$ be a tuple of distinct predicate constants. If $\boldsymbol{l}^1, \ldots, \boldsymbol{l}^n$ are all the loops of F relative to $\boldsymbol{p}$ then*

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

The following theorem extends the splitting theorem from (Ferraris et al., 2009b) to GQ-sentences.

Theorem 13 *Let F, G be GQ-sentences, and let $\boldsymbol{p}, \boldsymbol{q}$ be disjoint tuples of distinct predicate constants. If*

- *each strongly connected component of* $DG_{\boldsymbol{pq}}[F \wedge G]$ *is a subset of* $\boldsymbol{p}$ *or a subset of* $\boldsymbol{q}$*,*
- F *is negative on* $\boldsymbol{q}$*, and*
- G *is negative on* $\boldsymbol{p}$

then

$$SM[F \wedge G; \boldsymbol{pq}] \quad \textit{is equivalent to} \quad SM[F; \boldsymbol{p}] \wedge SM[G; \boldsymbol{q}].$$

Example 13 continued $\mathrm{SM}[F;\ \mathrm{discount}, \mathrm{numOfDiscount}]$ is equivalent to

$$\mathrm{SM}[G_1; \mathrm{discount}] \wedge \mathrm{SM}[G_2; \mathrm{numOfDiscount}],$$

where G_1 is the conjunction of the first two implications in F and G_2 is the last implication.

Completion

A GQ-formula F is in *Clark normal form* if it is a conjunction of sentences of the form

$$\forall \boldsymbol{x}(G \to p(\boldsymbol{x})), \tag{5.19}$$

one for each intensional predicate p, where $\boldsymbol{x}$ is a list of distinct object variables, and G has no free variables other than $\boldsymbol{x}$. The *completion* (relative to $\boldsymbol{p}$) of a GQ-formula F in Clark normal form, denoted by $\mathrm{Comp}[F]$, is obtained by replacing each conjunctive term (5.19) with

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

We say that a GQ-formula is *tight* on $\boldsymbol{p}$ if its (predicate) dependency graph relative to $\boldsymbol{p}$ is acyclic.

Theorem 14 *For any GQ-formula* F *in Clark normal form that is tight on* $\boldsymbol{p}$*,* $SM[F; \boldsymbol{p}]$ *is equivalent to the completion of* F *relative to* $\boldsymbol{p}$*.*

Example 13 continued *Let* F' *be the formula obtained from* F *by dropping the second implication. The Clark normal form of* F' *is tight on* $\{\mathit{discount}, \mathit{numOfDiscount}\}$*. So*

$SM[F_3; discount, numOfDiscount]$ *is equivalent to*

$$\forall x(discount(x) \leftrightarrow \neg accident(x) \wedge$$
$$\#dl[Man \uplus \mathrm{mm}, Married \uplus \mathrm{mm}, Woman \uplus \mathrm{mw}, Married \uplus \mathrm{mw}; \exists Spouse.\top](x))$$
$$\wedge \forall y(numOfDiscount(y) \leftrightarrow \mathsf{COUNT}\langle x.discount(x)\rangle = y).$$

Safety for First-order Formulas with Generalized Quantifiers

The condition of safety ensures that answer sets computations are not affected by the choice of domain. This in turn justifies domain-independent reasoning using the grounding mechanisms.

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. Eiter, Ianni, Schindlauer, and Tompits (2006b) extended the result of safety to programs with external atoms. According to the paper, external atoms can be used to restrict variables in very limited cases. For example, the following program is not safe if we view aggregate expressions as external atoms

$$p(y) \leftarrow COUNT\langle x.q(x,y)\rangle \geq 2.$$

The problem is that, when defining safety, external atoms are treated as black boxes: the properties of each different external atoms are not considered.

On the other hand, Lee, Lifschitz, and Palla (2008b) define the meaning of counting and choice by reducing these constructs to first-order formulas and defines the concept of a safe program based on the reduction. According to (Lee et al., 2008b), the above program is safe. However, the approach does not apply to arbitrary aggregates.

Example 19 *For instance, the following program* Π *should be safe but is not covered by the approach in (Lee et al., 2008b):*

$$q(2,1).$$
$$p(y) \leftarrow SUM\langle x.q(x,y)\rangle \geq 2.$$

The answer set of the program is $\{q(2,1), p(1)\}$ *no matter which domain we choose.*

In this section, we show that how the theorem on Safety can be extended to cover GQ-formulas. We consider only the first order formulas with generalized quantifiers which contains no function constant of positive arity.

Semi-Safety

We say that a generalized quantifier Q of type $\langle n_1, \ldots, n_k \rangle$ is *conjunctive* w.r.t. an argument set M if for every interpretation I, any subsets $R_1 \subseteq \mathcal{P}(|I|^{|\boldsymbol{x}_1|}), \ldots, R_k \subseteq \mathcal{P}(|I|^{|\boldsymbol{x}_k|})$, $Q^I(R_1, \ldots, R_k) = \boldsymbol{t}$ implies that $R_i \neq \emptyset$ for every $i \in M$. Similarly, Q is *disjunctive* w.r.t. an argument set M if for every interpretation I, any subsets $R_1 \subseteq \mathcal{P}(|I|^{|\boldsymbol{x}_1|}), \ldots, R_k \subseteq \mathcal{P}(|I|^{|\boldsymbol{x}_k|})$, $Q^I(R_1, \ldots, R_k) = \boldsymbol{t}$ implies that $R_i \neq \emptyset$ for some $i \in M$.

Given a GQ-formula F, we first rename all variables such that all bound and free variables are disjoint. Then we extend the definition of *restricted variables* (Lee et al., 2008a) to cover GQ-formula as follows.

- 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.
- For any GQ-formula F of the form $Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k))$,

$$\mathrm{RV}(F) = \bigcup_{M \subseteq \{1,\ldots,k\}} \mathrm{RV}_M(F)$$

  where $\mathrm{RV}_M(F)$ is defined as follows:
  - If Q is conjunctive w.r.t. M, then

$$\mathrm{RV}_M(F) = \bigcup_{i \in M} (\mathrm{RV}(G_i) \setminus \boldsymbol{x}_i);$$

  - else, if Q is disjunctive w.r.t. M, then

$$\mathrm{RV}_M(F) = \bigcap_{i \in M} (\mathrm{RV}(G_i) \setminus \boldsymbol{x}_i);$$

  - otherwise, $\mathrm{RV}_M(F) = \emptyset$.

A formula F is semi-safe if every strictly positive occurrence of every variable x that does not follow a generalized quantifier is contained in a subformula F in the form of $G \to H$, such that $x \in \mathrm{RV}(G)$

Example 19 continued The program Π

$$q(2,1).$$
$$p(y) \leftarrow SUM\langle x.q(x,y)\rangle \geq 2.$$

is sami-safe. This is because it can be viewed as the generalized quantified formula F

$$q(2,1) \wedge \forall y(p(y) \leftarrow Q_{(SUM,\geq 2)}[x]\ (q(x,y)))$$

where for any interpretation I, $Q^U_{(SUM,\geq 2)}$ is a function that maps $\mathcal{P}(U)$ to $\{\boldsymbol{t}, \boldsymbol{f}\}$ such that $Q^U_{(SUM,\geq 2)}(R) = \boldsymbol{t}$ iff

- $SUM(R)$ is defined, and
- $SUM(R) \geq 2$.

Since $Q_{(SUM,\geq 2)}$ is conjunctive w.r.t. $\{1\}$. As a result,

$$\mathrm{RV}(Q_{(SUM,\geq 2)}[x]\ (q(x,y))) = \{y\}.$$

Note that the representation of an aggregate is more restricted the one we presented in Section 5.2 in that the bound, 2, of the aggregate is now within a generalized quantifier instead of occurring as a term in the generalized quantified formula. This understanding representation disallows variable bounds. On the other hand, if we understand $SUM\langle x.q(x,y)\rangle \geq 2$ as $Q_{(SUM,\geq)}[x][z]\ (q(x,y), z = 2)$ as in Section 5.2, $Q_{(SUM,\geq)}$ is not conjunctive w.r.t. $\{1\}$. We can not conclude that the program is safe according to the representation.

For any finite set $\boldsymbol{c}$ of object constants, $\mathrm{in}_{\boldsymbol{c}}(x_1, \ldots, x_m)$ stands for the formula

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

By $\mathrm{SPP}_{\boldsymbol{c}}$ we denote the conjunction of the sentences

$$\forall \boldsymbol{x}\Big(p_i(\boldsymbol{x}) \rightarrow \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{x})\Big)$$

for all predicate constants p_i occurring in F, where $\boldsymbol{x}$ is a list of distinct object variables whose length is the same as the arity of $\boldsymbol{p}$.

For any set $\boldsymbol{c}$ of object constants, $\boldsymbol{e}_{\boldsymbol{c}}$ denotes the list of predicate expressions consisting of

$$\lambda \boldsymbol{x}\Big(p_i(\boldsymbol{x}) \wedge \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{x})\Big)$$

for all p_i in F.

For any formula F, let $c(F)$ be the set of all object constants in F and $\sigma(F)$ be the set of all object and predicate constants in F.

Proposition 38 *For semi-safe sentence F, $SM[F] \models SPP_{c(F)}$.*

Example 19 continued $\mathrm{SPP}_{\{1,2\}}$ is

$$\forall xy(q(x,y) \rightarrow ((x = 1 \vee x = 2) \wedge (y = 1 \vee y = 2)))$$

According to the above proposition, $\mathrm{SM}[F] \models \mathrm{SPP}_{\{1,2\}}$.

Grounding

Let $F = Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))$ be a generalized quantified formula and $\boldsymbol{c}$ a set of object constants containing $\sigma(F)$. By $\boldsymbol{O}^i_{\boldsymbol{c}}$, we denote the set of all lists of object constants of $\boldsymbol{c}$ whose length is the same as the length of $\boldsymbol{x}^i$. $\overline{\boldsymbol{C}}_{\boldsymbol{c}}(F)$ is the set of all tuples of the form $(R_1,\ldots,R_k)$ such that

- $R_i \subseteq \boldsymbol{O}^i_{\boldsymbol{c}}$, and
- $Q^{\boldsymbol{c}}(R_1,\ldots,R_k) = \boldsymbol{f}$.

For any sentence F and any nonempty finite set $\boldsymbol{c}$ of object constants, the variable-free formula $\mathrm{Ground}_{\boldsymbol{c}}[F]$ are defined as follows.

- If F is an atomic formula, $\mathrm{Ground}_{\boldsymbol{c}}[F] = F$;

- If F is $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))$, $\mathrm{Ground}_{\boldsymbol{c}}[F]$ is

$$\bigwedge_{(R_1,\ldots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(F)}\left(\bigwedge_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in R_i}}\mathrm{Ground}_{\boldsymbol{c}}[G_i(\boldsymbol{d}_i)]\rightarrow\bigvee_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in O^i_{\boldsymbol{c}}\setminus R_i}}\mathrm{Ground}_{\boldsymbol{c}}[G_i(\boldsymbol{d}_i)]\right).$$

Proposition 39 *For any GQ sentence F, any signature σ such that $\sigma(F)\subseteq\sigma$ and any Herbrand interpretation X of σ, let $\boldsymbol{c}$ be the set of all object constants in σ, if $\boldsymbol{c}$ is finite, then $X\models SM[F]$ iff $X\models SM[Ground_{\boldsymbol{c}}[F]]$.*

Example 20 *Consider the generalized quantified formula $E_1 = Q_{\wedge}(G_1, G_2)$. For any finite set of object constants $\boldsymbol{c}$, $\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E_1)$ is $\{(\emptyset,\emptyset),(\epsilon,\emptyset),(\emptyset,\epsilon)\}$. $Ground_{\boldsymbol{c}}[E_1]$ is*

$$\begin{aligned}&(Ground_{\boldsymbol{c}}[G_1]\vee Ground_{\boldsymbol{c}}[G_2])\\ &\wedge(Ground_{\boldsymbol{c}}[G_1]\rightarrow Ground_{\boldsymbol{c}}[G_2])\\ &\wedge(Ground_{\boldsymbol{c}}[G_2]\rightarrow Ground_{\boldsymbol{c}}[G_1])\end{aligned}$$

which is strongly equivalent to

$$Ground_{\boldsymbol{c}}[G_1]\wedge Ground_{\boldsymbol{c}}[G_2].$$

For another example, consider the generalized quantified formula

$$E_2 = Q_{(SUM,\geq 2)}[x]\ (q(x)).$$

Let the set of object constants $\boldsymbol{c}=\{1,2\}$. $\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E_2)$ is $\{\emptyset,\{1\}\}$. $Ground_{\boldsymbol{c}}[E_2]$ is

$$(q(1)\vee q(2))\wedge(q(1)\rightarrow q(2)).$$

Note that the later is exactly the formula representation of the aggregate SUM$\langle x.q(x)\rangle\geq 2$ according to (Ferraris, 2005).

Consider the generalized quantified formula

$$E_3 = Q_{major}[x]\ (p(x))$$

such that for any interpretation I *and any* $R \subseteq |I|$, $Q^I_{major}(R) = \boldsymbol{t}$ *iff* $|R| \geq |I|/2$. *Let the set of object constants* $\boldsymbol{c} = \{a, b, c\}$. $\boldsymbol{O}^1_{\boldsymbol{c}} = \{a, b, c\}$. $\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E_3)$ *is* $\{(\emptyset), (\{a\}), (\{b\}), (\{c\})\}$. $Ground_{\boldsymbol{c}}[E_3]$ *is*

$$(p(a) \vee p(b) \vee p(c)) \wedge (p(a) \rightarrow p(b) \vee p(c))$$
$$\wedge\, (p(b) \rightarrow p(a) \vee p(c)) \wedge (p(c) \rightarrow p(a) \vee p(b)).$$

Safety

Given a GQ-formula, we first rename all variables such that all bound variables are disjoint. The following transformation is a refomulation of the transformation defined in [Cabalar *et al.*, 2009].

- $\neg\bot \mapsto \top,\ \neg\top \mapsto \bot,$
- $\bot \wedge F \mapsto \bot,\ F \wedge \bot \mapsto \bot,\ \top \wedge F \mapsto F,\ F \wedge \top \mapsto F,$
- $\bot \vee F \mapsto F,\ F \vee \bot \mapsto F,\ \top \vee F \mapsto \top,\ F \vee \top \mapsto \top,$
- $\bot \rightarrow F \mapsto \top,\ F \rightarrow \top \mapsto \top,\ \top \rightarrow F \mapsto F,$
- $Qx.\top \mapsto \top, Qx.\bot \mapsto \bot$ for $Q \in \{\forall, \exists\}$.

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

- first replacing every atomic formula containing x by $\bot$,
- then apply the transformation above

the result is $\top$; x is negatively weakly restricted in G if the result is $\bot$.

A semi-safe sentence with aggregates F is safe if, for every (non-strictly positive) occurrence of a variable x in F,

- every occurrence of each of every variable x that is quantified by $\forall$ or $\exists$ in F is contained in a subformula $G \rightarrow H$ that satisfies two conditions

- if x is quantified by $\forall$ that has positive occurrence or $\exists$ that has negative occurrence, then the occurrence is either
  - positively weakly restricted in a formula G that is positive in F, or
  - negatively weakly restricted in a formula G that is negative in F.
- if x is quantified by $\exists$ that has positive occurrence or $\forall$ that has negative occurrence, then the occurrence is either
  - positively weakly restricted in a formula G that is negative in F, or
  - negatively weakly restricted in a formula G that is positive in F.

- for GQ-formula of the form

$$Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\dots,G_k(\boldsymbol{x}_k))$$

such that Q is neither $Q_\forall$ nor $Q_\exists$, every variable in $\boldsymbol{x}^i$ belongs to $\mathrm{RV}(G_i)$.

Proposition 40 *For any safe GQ sentence F and any nonempty finite set $\boldsymbol{c}$ of object constants containing $c(F)$, $SM[F]$ is equivalent to $SM[Ground_{\boldsymbol{c}}[F]]$.*

Example 21 *Consider the formula F_1*

$$p(a)\wedge p(b)\wedge\forall x(p(x)\rightarrow(q(x)\vee\neg q(x)))$$
$$\wedge\ \mathsf{COUNT}\langle y.q(y)\rangle\leq 1\rightarrow r.$$

The formula is semi-safe because only the variable x has strictly positive occurrences (in the second conjunctive terms) and they are contained in the implication where $x\in RV(p(x))$. As a result, the following holds

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

F_1 is also safe because x is quantified by $\forall$ that has positive occurrence and is negatively weakly restricted in $p(x)$ that is negative in F_1. The variable y is not quantified by $\forall$ or $\exists$ and it belongs to $RV(q(y))$.

As a result, F_1 has the same answer sets as $Ground_{\{a,b\}}(F_1)$ which is the following formula

$$p(a) \wedge p(b)$$
$$\wedge(p(a) \rightarrow (q(a) \vee \neg q(a)))$$
$$\wedge(p(b) \rightarrow (q(b) \vee \neg q(b)))$$
$$\wedge((q(a) \wedge q(b)) \rightarrow \bot) \rightarrow r.$$

Loop Formulas for GQ-Formula

In this section we extend the definition of a first-order loop formula to a GQ-sentence.

As with a propositional loop formula defined for an arbitrary propositional theory (Ferraris et al., 2006), it is convenient to introduce a formula whose *negation* is close to ES. We define formula $\mathrm{NES}_F(Y)$ (*"Negation of (First-order) External Support Formula"*), where F is a first-order formula and Y is a finite set of atoms, as follows. We assume that no variables in Y occur in F, by renaming variables.

- $\mathrm{NES}_{p_i(\boldsymbol{t})}(Y) = p_i(\boldsymbol{t}) \wedge \bigwedge_{p_i(\boldsymbol{t}') \in Y} \boldsymbol{t} \neq \boldsymbol{t}'$;
- $\mathrm{NES}_F(Y) = F$ for any atomic formula F that does not contain members of $\boldsymbol{p}$;
- $\mathrm{NES}_{Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))}(Y) =$
  $$Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Y),\ldots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Y)) \cdot$$
  $$\wedge\, Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k)).$$

The *(first-order) loop formula* of Y for F, denoted by $\mathrm{LF}_F(Y)$, is the universal closure of

$$\bigwedge Y \rightarrow \neg \mathrm{NES}_F(Y). \tag{5.20}$$

Let F be a GQ-formula and $\boldsymbol{p}$ be the list of all predicates that occur in F. We say that an occurrence of a predicate constant or subformula in a formula F is *mixed* if there is some generalized quantifier Q that contains the occurrence in its argument position which is neither monotone nor anti-monotone. Let l be the number of generalized quantifiers Q in F such that the occurrence of p belongs to an anti-monotone argument position of Q. If the occurrence is not mixed then we call it *positive* in F if l is even, and *negative* otherwise. The

occurrence is *strictly positive* in F if $l = 0$. We call an occurrence of a predicate constant or a subformula *semi-positive* if it is positive or mixed. Similarly, it is *semi-negative* if it is negative or mixed. We say that F is *negative on* $\boldsymbol{p}$ if there is no strictly positive occurrence of a predicate constant from $\boldsymbol{p}$ in F. An occurrence of a predicate constant or a subformula of F is $\boldsymbol{p}$*-negated* in F if it is contained in a subformula of F that is negative on $\boldsymbol{p}$.

We will say that a GQ-formula is *rectified* if it has no variables that are both bound and free, and if all quantifiers in the formula refer to different variables. Any formula can be easily rewritten into a rectified formula by renaming bound variables.

Let $F = Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](G_1, \dots, G_k)$ be a rectified generalized formula. We say that an atom $p(\boldsymbol{t})$ *depends on* an atom $q(\boldsymbol{t}')$ in F if

- $p(\boldsymbol{t})$ has a strictly positive occurrence in F, and
- $q(\boldsymbol{t}')$ has a semi-positive, non-$\boldsymbol{p}$-negated occurrence in a non-monotone argument position of Q.

The definition of a first-order atomic dependency graph is extended to formulas as follows. The *(first-order) atomic dependency graph* of a rectified GQ-formula F is the infinite directed graph (V, E) such that

- V is the set of atoms of signature $\sigma(F)$;
- $(p(\boldsymbol{t})\theta, q(\boldsymbol{t}')\theta)$ is in E if $p(\boldsymbol{t})$ depends on $q(\boldsymbol{t}')$ in a strictly positive occurrence of a subformula of F and θ is a substitution that maps variables in $\boldsymbol{t}$ and $\boldsymbol{t}'$ to terms of $\sigma(F)$.

A *loop* of F is a nonempty subset L of V such that the subgraph induced by L is strongly connected. It is clear that the strongly connected components can be characterized as the maximal loops of F.

Example 13 continued *The dependency graph of* F *contains the following finite loops* $\{family(u)\}, \{mm(u)\}, \{\mathrm{mw}(u)\}, \{accident(u)\}, \{numOfDiscount(u)\}$ *and* $\{discount(u_1), \dots, discount(u_n)\}$ *for any* $n \geq 1$.

$LF_F(\{discount(u_1), \ldots, discount(u_n)\})$ *is the following formula*

$$\bigwedge_{1\le i\le n} discount(u_i) \rightarrow \Big((discount(y) \wedge \bigwedge_{1\le i\le n} y \neq u_i \wedge family(y,x) \wedge \neg accident(x)) \vee (\neg accident(x) \wedge Q_{(5.13)}[x_1][x_2][x_3][x_4](mm(x_1), mm(x_2), mw(x_3), mw(x_4)))\Big)$$

Theorem 15 *Let F be a rectified GQ-sentence that has no function constants of positive arity, and let X be an Herbrand interpretation of $\sigma(F)$ that satisfies F. The following conditions are equivalent to each other:*

(a) X is a stable model of F (i.e., I satisfies $SM[F]$);

(b) for every nonempty finite set Y of atoms of $\sigma(F)$, X satisfies $LF_F(Y)$;

(c) for every finite first-order loop Y of F, X satisfies $LF_F(Y)$.

5.4 First-Order FLP Semantics for Programs with Generalized Quantifiers

To compare to the HEX semantics, we first extend the first-order FLP semantics for programs with aggregate further to programs containing generalized quantifiers, which can be viewed as extending HEX programs to the first-order level. We then relate the FLP semantics and the first-order stable model semantics in the general context of programs with generalized quantifiers.

A *(general) rule* is of the form

$$H \leftarrow B \tag{5.21}$$

where H and B are arbitrary GQ-formulas. A *(general) program* is a finite set of rules.

The definition of $\mathrm{FLP}[\Pi; \boldsymbol{p}]$ is the same as the one for formulas with aggregates in Section 4.6. Let Π be a finite program whose rules have the form (5.21). The *GQ-representation* Π^{GQ} of Π is the conjunction of the universal closures of $B \rightarrow H$ for all rules (5.21) in Π. By $\mathrm{FLP}[\Pi; \boldsymbol{p}]$ we denote the second-order formula

$$\Pi^{GQ} \wedge \neg\exists \boldsymbol{u}(\boldsymbol{u} < \boldsymbol{p} \wedge \Pi^{\triangle}(\boldsymbol{u}))$$

where $\Pi^{\triangle}(\boldsymbol{u})$ is defined as the conjunction of the universal closures of

$$B \wedge B(\boldsymbol{u}) \rightarrow H(\boldsymbol{u})$$

for all rules $H \leftarrow B$ in Π.

We will often simply write $\mathrm{FLP}[\Pi]$ instead of $\mathrm{FLP}[\Pi; \boldsymbol{p}]$ when $\boldsymbol{p}$ is the list of all predicate constants occurring in Π, and call a model of $\mathrm{FLP}[\Pi]$ an *FLP-stable* model of Π.

Example 15 continued For GQ-formula F_1 considered earlier, $\mathrm{FLP}[F_1]$ is

$$F_1 \wedge \neg \exists u(u < p \wedge F_1^{\triangle}(u)) \ , \tag{5.22}$$

where $F_1^{\triangle}(u)$ is

$$\begin{aligned}
&(\neg Q_{(\mathsf{SUM},<)}[x][y](p(x), y{=}2) \wedge \neg Q_{(\mathsf{SUM},<)}[x][y](u(x), y{=}2) \rightarrow u(2)) \\
&\wedge (Q_{(\mathsf{SUM},>)}[x][y](p(x), y{=}{-}1) \wedge (Q_{(\mathsf{SUM},>)}[x][y](u(x), y{=}{-}1) \rightarrow u(-1)) \\
&\wedge (p(-1) \wedge u(-1) \rightarrow u(1)) \ .
\end{aligned}$$

I_1 considered earlier satisfies (5.22) but I_2 does not.

Consider the following one-rule program Π

$$p(a) \leftarrow \mathrm{not\ not}\ p(a).$$

Both $\emptyset$ and $\{p(a)\}$ satisfy $\mathrm{SM}[\Pi; p]$, but only $\emptyset$ satisfies $\mathrm{FLP}[\Pi; p]$. Even in the absence of double negations, the two semantics may still disagree. For example,

$$p(a) \leftarrow \mathrm{not}\ Q_{\leq 0}[x]\ p(x) \tag{5.23}$$

has the same Herbrand stable models as the above, as $Q_{\leq 0}$ behaves the same as negation.

We show the class of programs for which the FLP semantics and the stable model semantics coincide, which extends the one in the previous chapter. We say that a formula F with generalized quantifiers is *canonical* relative to a list $\boldsymbol{p}$ of predicate constants if

- for every occurrence of every predicate constant p from $\boldsymbol{p}$ in F, the number of non-monotone arguments that contain the occurrence is ≤ 1;

- if a predicate constant p from $\boldsymbol{p}$ occurs in the scope of a strictly positive occurrence of a generalized quantifier Q in F such that $Q \notin \{Q_{\rightarrow}, Q_{\wedge}, Q_{\forall}\}$, then the occurrence of p is strictly positive and not mixed in F.

Proposition 41 *Let Π be a finite general program and let F be the GQ-representation of Π. For every rule (5.21) in Π, if B is canonical relative to $\boldsymbol{p}$ and every occurrence of p from $\boldsymbol{p}$ in H is strictly positive and not mixed in H, then $FLP[\Pi; \boldsymbol{p}]$ is equivalent to $SM[F; \boldsymbol{p}]$.*

For example, program (5.23) does not satisfy the condition in Proposition 41 since $Q_{\leq 0}[x]\ p(x)$ is not monotone in $\{1\}$.

The definition of a formula being canonical relative to a list of predicate constants is simplified in that, instead of the two numbers k and m, we only need to consider the number of nonmonotone arguments. This is because the antecedence of an implication is also a nonmonotone argument of the generalized quantifier $Q_{\rightarrow}$. The generality of GQ-formulas identifies the property of connective that the theorem relies on.

5.5 Revisiting Non-Monotonic DL-programs

Eiter et al. (2008b) note that weak answer set semantics produce counterintuitive answer sets with circular justifications by self-supporting loops. This are caused by the removal of dl-atoms when forming the reduct. To overcome this problem, authors refer to strong answer sets. However, Shen (2011) notes that both strong and weak answer set semantics suffer from circular justifications too.

Example 22 *(Shen, 2011) Consider $(\mathcal{T}, \Pi)$, where $\mathcal{T} = \emptyset$ and Π is the program*

$$p(a) \leftarrow \#dl[c \uplus p, b \cap q;\ c \sqcap \neg b](a)\ , \tag{5.24}$$

in which the dl-atom is neither monotonic nor anti-monotonic. The dl-program has two strong (weak, respectively) answer sets: $\emptyset$ and $\{p(a)\}$. According to (Shen, 2011), the second answer set is circularly justified:

$$p(a) \Leftarrow \#dl[c \uplus p, b \cap q; c \sqcap \neg b](a) \Leftarrow p(a) \wedge \neg q(a).$$

Indeed, $s\Pi_{\mathcal{T}}^{\{p(a)\}}$ ($w\Pi_{\mathcal{T}}^{\{p(a)\}}$, respectively) is $p(a) \leftarrow$, and $\{p(a)\}$ is its minimal model.

From the above example, we observe that the issue is related to the fact that both strong and weak answer set semantics do not distinguish between anti-monotonic and non-anti-monotonic dl-atoms. In view of the Theorem on Loop Formulas, the former does not contribute to loops, but the latter does. As a result, non-anti-monotonic dl-atoms should participate in enforcing minimality of answer sets and should not be removed when forming the reduct. This suggests the following alternative definition of the semantics of dl-programs. Instead of removing every nonmonotonic dl-atoms in forming the reduct under strong answer set semantics, we remove only anti-monotonic dl-atoms from the bodies, but leave non-anti-monotonic dl-atoms. In other words, the *dl-transform* of Π relative to $\mathcal{T}$ and an Herbrand interpretation I of $\langle C, P_\Pi \rangle$, denoted by $\Pi^I_\mathcal{T}$, is the set of rules (5.10), where $a \leftarrow B, N$ is in Π, $I \models_\mathcal{T} B \wedge N$ and B' is obtained from B by removing all anti-monotonic dl-atoms in it. We say that an Herbrand interpretation I is an *answer set* of $(\mathcal{T}, \Pi)$ if I is minimal among the sets of atoms that satisfy $\Pi^I_\mathcal{T}$.

Example 22 continued *$\{p(a)\}$ is not an answer set of $(\mathcal{T}, \Pi)$ according to the new definition. $\Pi^{\{p(a)\}}_\mathcal{T}$ is (5.24) itself, and $\emptyset$, a proper subset of $\{p(a)\}$ satisfies it.*

This new definition can be also characterized in terms of generalized quantifiers. In fact, the characterization is simpler than those for the other two semantics. We simply identify (5.7) with (5.11) regardless of the (anti-)monotonicity of the dl-atom.

Proposition 42 *For any dl-program $(\mathcal{T}, \Pi)$, and any Herbrand interpretation X of $\langle C, P_\Pi \rangle$, X is an answer set of $(\mathcal{T}, \Pi)$ as defined here iff X satisfies $SM[\Pi; \boldsymbol{p}]$ when we identify every dl-atom (5.7) in Π with (5.11).*

In (Shen, 2011), the author proposed the semantics based on conditional satisfaction to resolve the circular justification of the above mentioned semantics. However, the semantics are sometimes too strong and rule out some intuitive answer sets.

Example 23 *Consider the dl-program $(\mathcal{T}, \Pi)$ such that $\mathcal{T}$ is empty, and Π is the following*

program containing a nonmonotonic dl-atom:

$$p(x) \leftarrow q(x).$$
$$q(x) \leftarrow p(x).$$
$$p(a) \leftarrow \ \#dl[B \uplus p, C \uplus q;\ \neg B \sqcup C](a).$$

According to the first two rules, either $p(a)$ *and* $q(a)$ *are both true or both false. In any of the cases, the dl-atom in the third rule is satisfied. As a result,* $p(a)$ *is true. So the answer set should be* $\{p(a), q(a)\}$*. However,* $\{p(a), q(a)\}$ *is not an answer set according to (Shen, 2011).*

Fink and Pearce (2010) proposed another semantics of nonmonotonic dl-programs by viewing them as special cases of HEX programs. Their semantics can be generalized to the first-order case using the FLP operator we introduced before. The following proposition makes this claim clear.

Proposition 43 *For any dl-program* $(\mathcal{T}, \Pi)$*, and any Herbrand interpretation* X *of* $\langle C, P_\Pi \rangle$*,* X *satisfies* $FLP[\Pi^{GQ}; P_\Pi]$ *relative to* $\mathcal{T}$ *iff* X *is an answer set of* $(\mathcal{T}, \Pi)$ *according to Fink and Pearce.*

The following proposition states that the relationship between the two semantics. It follows immediately from Proposition 41.

Proposition 44 *For any dl-program* $(\mathcal{T}, \Pi)$*, and any Herbrand interpretation* X *of* $\langle C, P_\Pi \rangle$*, if every occurrence of nonmonotonic dl-atoms is in the positive body of a rule, then* X *is an answer set of* $(\mathcal{T}, \Pi)$ *in the sense of (Fink & Pearce, 2010) iff* X *is an answer set of* $(\mathcal{T}, \Pi)$ *in our sense.*

The following example shows why the condition in the statement is essential.

Example 24 *Consider the dl-program* $(\mathcal{T}, \Pi)$ *such that* $\mathcal{T}$ *is empty, and* Π *is the following single rule program containing a nonmonotonic dl-atom:*

$$p(a) \leftarrow not\ \#dl[C \uplus p;\ \neg C](a).$$

While $\emptyset$ and $\{p(a)\}$ are answer sets according to us, only $\emptyset$ is the answer set according to (Fink & Pearce, 2010).

As in other FLP-based semantics, the semantics by Fink and Pearce suffers from unintuitive cases.

Example 24 continued Consider a rewriting of the program

$$p(a) \leftarrow q.$$
$$q \leftarrow \text{not } \#dl[C \Cap p;\ \neg C](a)$$

both $\emptyset$ and $\{p(a)\}$ are answer sets according to (Fink & Pearce, 2010).

5.6 Related Work

Relation to GQ-Stable Models by Eiter et al.

In fact, the incorporation of generalized quantifiers in logic programming was considered earlier in (Eiter, Gottlob, & Veith, 1997a, 1997b). The authors extended logic programs by allowing "GQ-atoms," a special class of generalized quantified formula (5.1) that have the form

$$Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](s_1(\boldsymbol{x}_1), \dots, s_{k-1}(\boldsymbol{x}_{k-1}), \boldsymbol{x}_k = \boldsymbol{v}), \tag{5.25}$$

where $s_1, \dots s_{k-1}$ are predicate constants, and $\boldsymbol{v}$ is a list of variables. They consider that rules have the form

$$A \leftarrow B, N, \tag{5.26}$$

where A is a (normal) atom, B is a set of atoms and GQ-atoms and N is a set of atoms and GQ-atoms preceded by "not."

Let Π be such a finite program, and X an Herbrand interpretation of $\sigma(\Pi)$, the signature consisting of object and predicate constants occurring in Π. We assume that Π has no free variables. The *EGV-reduct* of Π relative to X is the set of rules

$$A \leftarrow B'$$

where $A \leftarrow B, N$ in Π, $X \models B \wedge N$, and B' is obtained from B by removing all GQ-atoms in it. X is an *EGV-answer set* of Π if X is minimal among the sets of atoms that satisfy the EGV-reduct of Π relative to X.

Under this semantics, both GQ-atoms and their negations are treated like negative literals. However, this often leads to unintuitive results.

Example 25 *For instance, according to (Eiter et al., 1997a), program*

$$p(a) \leftarrow \forall x\ p(x) \tag{5.27}$$

has two EGV-answer sets, $\emptyset$ *and* $\{p(a)\}$*. The latter is "unfounded." In our semantics,* $\{p(a)\}$ *is not stable, while* $\emptyset$ *is.*

The semantics of programs by Eiter et al. can be easily expressed in terms of our semantics by prepending $\neg\neg$ to generalized quantifiers. Let Π^{GQ} be the formula representation of Π and let $(\Pi^{GQ})^{\neg\neg}$ be the formula obtained from Π^{GQ} by inserting $\neg\neg$ in front of every GQ-atom.

Proposition 45 *For any program* Π *and any Herbrand interpretation* X *of* $\sigma(\Pi)$*,* X *is an EGV-answer set of* Π *iff* $X \models SM[(\Pi^{GQ})^{\neg\neg}]$*.*

The two semantics coincide if Π is "tight." A program is *tight* if its formula representation is tight.

Corollary 12 *For any tight program* Π *and any Herbrand interpretation* X *of* $\sigma(\Pi)$*,* X *is an EGV-answer set of* Π *iff* $X \models SM[\Pi^{GQ}]$*.*

Relation to Infinitary Formulas

The study on infinitary logic can be dated back to the 18th century (Moore, 1997). In (Scott, 1958) and (Tarski, 1958), infinitary propositional and predicate languages were introduced. The completeness theorem for the infinitary languages were proven in (Karp, 1964). Truszczynski (2012) extended reduct proposed by (Ferraris, 2005) to allow infinitary propositional formula for defining first-order stable models. In the following, we review the definitions in (Truszczynski, 2012).

Let A be a propositional signature. The set of infinitary formula is defined as follows.

- $\mathcal{F}_0 = A \cup \{\bot\}$.
- $\mathcal{F}_{i+1}$, where $i \geq 0$, consists of the expression $\mathcal{H}^\wedge$ and $\mathcal{H}^\vee$, for all subsets $\mathcal{H}$ of $\mathcal{F}_0 \cup \ldots, \cup \mathcal{F}_i$, and of expressions $F \to G$, where $F, G \in \mathcal{F}_0 \cup \ldots \cup \mathcal{F}_i$.

We denote $\mathcal{L}_A^{inf} = \bigcup_{i=0}^{\infty} \mathcal{F}_i$ and call elements of $\mathcal{L}_A^{inf}$ *infinitary formulas* (over A).

Let I be a subset of A and an infinitary formula F, we define $I \models F$ by induction:

- $I \not\models \bot$
- For every $p \in A$, $I \models p$ if $p \in I$
- $I \models \mathcal{H}^\vee$ if there is a formula $F \in \mathcal{H}$ such that $I \models F$
- $I \models \mathcal{H}^\wedge$ if for every formula $F \in \mathcal{H}$, $I \models F$
- $I \models G \to H$ if $I \not\models G$ or $I \models H$.

The concept of *reduct* is extended to infinitary formulas.

- $\bot^{\underline{I}} = \bot$
- For every $p \in A$, $p^{\underline{I}} = \bot$ if $I \not\models p$; otherwise $p^{\underline{I}} = p$
- $(\mathcal{H}^\vee)^{\underline{I}} = \bot$ if $I \not\models \mathcal{H}^\vee$; otherwise, $(\mathcal{H}^\vee)^{\underline{I}} = \{G^{\underline{I}} \mid G \in \mathcal{H}\}^\vee$
- $(\mathcal{H}^\wedge)^{\underline{I}} = \bot$ if $I \not\models \mathcal{H}^\wedge$; otherwise, $(\mathcal{H}^\wedge)^{\underline{I}} = \{G^{\underline{I}} \mid G \in \mathcal{H}\}^\wedge$
- $(G \to H)^{\underline{I}} = \bot$ if $I \not\models G \to H$; otherwise $(G \to H)^{\underline{I}} = G^{\underline{I}} \to H^{\underline{I}}$.

Let F be an infinitary formula and I an interpretation. I is a *stable model* of F if I is a minimal model of F^I.

Let F be a first-order sentence. The first-order stable model of F can be defined in terms of grounding into infinitary formula. The Truszczynski grounding of F w.r.t. I, denoted by $gr_I^T[F]$, is similar to the grounding we defined before except that the last bullet is replaced by the following two bullets:

- $gr_I^T[\exists x F(x)] = \{gr_I^T[F(\xi^\diamond)] \mid \xi \in |I|\}^\vee$
- $gr_I^T[\forall x F(x)] = \{gr_I^T[F(\xi^\diamond)] \mid \xi \in |I|\}^\wedge$.

The following corollary is immediate from Theorem 2 and Proposition 3 in (Truszczynski, 2012).

Corollary 13 *Let σ be a signature that contains finitely many predicate constants, let $\sigma^{\boldsymbol{p}}$ be the set of predicate constants in σ, let $I = \langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}} \rangle$ be an interpretation of σ, and let F be a first-order sentence of σ. $I^{\boldsymbol{p}}$ is a minimal set of atoms that satisfies $(gr_I[F])^{\underline{I}}$ iff $I^{\boldsymbol{p}}$ is a minimal set of atoms that satisfies $(gr_I^T[F])^{\underline{I}}$.*

Relation to Ferraris' Semantics

Let $F = Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k))$ be a GQ-formula of σ and I an interpretation of a signature σ. $F' = Q(S_1, \ldots, S_k)$ is a ground GQ-formula of F w.r.t. I. By $\boldsymbol{O}_I^i = \{\boldsymbol{\xi}^\diamond \mid \boldsymbol{\xi}^\diamond . F(\boldsymbol{\xi}^\diamond) \in S_i\}$. $\overline{\boldsymbol{C}}_I(F')$ is the set of all tuples of the form $(R_1, \ldots, R_k)$ such that

- $R_i \subseteq \boldsymbol{O}_I^i$, and
- $Q(R_1, \ldots, R_k)^I = \boldsymbol{f}$.

The infinitary formula representation of F', denoted by $\mathrm{Inf}[F']$ are defined recursively as follows.

- If F' is an atomic formula, $\mathrm{Inf}[F'] = F'$;
- If F' is $Q(S_1, \ldots, S_k)$, $\mathrm{Inf}[F']$ is

$$\{\mathcal{H}_1^\wedge \rightarrow \mathcal{H}_2^\vee \mid (R_1, \ldots, R_k) \in \overline{\boldsymbol{C}}_I(F')\}^\wedge$$

  where $\mathcal{H}_1 = \{\mathrm{Inf}[G_i(\boldsymbol{\xi}^\diamond)] \mid 1 \leq i \leq k, \boldsymbol{\xi}^\diamond \in R_i\}$ and $\mathcal{H}_2 = \{\mathrm{Inf}[G_i(\boldsymbol{\xi}^\diamond)] \mid 1 \leq i \leq k, \boldsymbol{\xi}^\diamond \in \boldsymbol{O}_I^i \setminus R_i\}$.

When I is the Herbrand interpretation of $\sigma(F)$, the Ferraris formula representation of F', denoted by $\mathrm{Fer}[F']$ are defined recursively as follows.

- If F' is an atomic formula, $\mathrm{Fer}[F'] = F'$;
- If F' is $Q(S_1, \ldots, S_k)$, $\mathrm{Fer}[F']$ is

$$\bigwedge_{(R_1,\ldots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_I(F')} \left(\bigwedge_{1\leq i\leq k, \boldsymbol{\xi}^\diamond\in R_i} \mathrm{Fer}[G_i(\boldsymbol{\xi}^\diamond)] \rightarrow \bigvee_{1\leq i\leq k, \boldsymbol{\xi}^\diamond\in \boldsymbol{O}^i_I\setminus R_i} \mathrm{Fer}[G_i(\boldsymbol{\xi}^\diamond)] \right)$$

Example 26 *Consider the generalized quantified formula $F_1 = Q_\wedge(p, q)$. For any interpretation I, F_1' is $Q_\wedge(\{p\}, \{q\})$. $\overline{\boldsymbol{\mathcal{C}}}_I(F_1')$ is $\{(\emptyset, \emptyset), (\{\epsilon\}, \emptyset), (\emptyset, \{\epsilon\})\}$. $Inf[F_1']$ is*

$$\{\{p, q\}^\vee, \{p\}^\wedge \rightarrow \{q\}^\vee, \{q\}^\wedge \rightarrow \{p\}^\vee\}^\wedge.$$

$Fer[F_1']$ is

$$(p \vee q) \wedge (p \rightarrow q) \wedge (q \rightarrow p).$$

Consider the generalized quantified formula

$$F_2 = Q_{major}[x]\ (p(x))$$

such that for any interpretation J and any $R \subseteq |J|$, $Q^J_{major}(R) = \boldsymbol{t}$ iff $|R| \geq |J|/2$. Consider I such that $|I| = \{a, b, c\}$. $F_2' = Q_{major}(\{a^\diamond.p(a^\diamond), b^\diamond.p(b^\diamond), c^\diamond.p(c^\diamond)\})$. $\boldsymbol{O}^1_I = \{a^\diamond, b^\diamond, c^\diamond\}$. $\overline{\boldsymbol{\mathcal{C}}}_I(F_2')$ is $\{\emptyset, \{a^\diamond\}, \{b^\diamond\}, \{c^\diamond\}\}$. $Inf[F_2']$ is

$$\{\{p(a), p(b), p(c)\}^\vee, \{p(a)\}^\wedge \rightarrow \{p(b), p(c)\}^\vee,$$
$$\{p(b)\}^\wedge \rightarrow \{p(a), p(c)\}^\vee, \{p(c)\}^\wedge \rightarrow \{p(a) \vee p(b)\}^\vee\}^\wedge$$

$Fer[F_2']$ is

$$(p(a) \vee p(b) \vee p(c)) \wedge (p(a) \rightarrow p(b) \vee p(c))$$
$$\wedge\ (p(b) \rightarrow p(a) \vee p(c)) \wedge (p(c) \rightarrow p(a) \vee p(b))$$

Proposition 46 *Let σ be a signature that contains finitely many predicate constants, let $I = \langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}}\rangle$ be an interpretation of σ, let F be a first-order sentence of σ, F' be the ground GQ-formula of F w.r.t. I and $J^{\boldsymbol{p}}$ be any subset of $I^{\boldsymbol{p}}$. $J^{\boldsymbol{p}} \models F'$ iff $J^{\boldsymbol{p}} \models Inf[F']$.*

Proposition 47 *Let σ be a signature that contains finitely many predicate constants, let $I = \langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}}\rangle$ be an interpretation of σ, let F be a first-order sentence of σ and let F' be the ground GQ-formula of F w.r.t. I. $I^{\boldsymbol{p}}$ is a minimal set of atoms that satisfies $(F')^{\underline{I}}$. iff $I^{\boldsymbol{p}}$ is a minimal set of atoms that satisfies $(Inf[F'])^{\underline{I}}$.*

Proposition 48 *Let σ be a signature that contains finitely many predicate constants, let F be a first-order sentence of σ, let $I = \langle I^{\mathbf{f}}, I^{\mathbf{p}} \rangle$ be an Herbrand interpretation of $\sigma(F)$, and let F' be the ground GQ-formula of F w.r.t. I. $I^{\mathbf{p}}$ is a minimal set of atoms that satisfies $(Inf[F'])^{\underline{I}}$ iff $I^{\mathbf{p}}$ is a minimal set of atoms that satisfies $(Fer[F'])^{\underline{I}}$.*

5.7 Conclusion

We presented different reformulations of the stable model semantics and FLP semantics for formulas containing generalized quantifiers, and showed that several recent extensions of the stable model semantics can be viewed as special cases of this formalism. The reformulations help us understand the relationship between the FLP semantics and the first-order stable model semantics, and their extensions. For the class of programs where the two semantics coincide, the system dlv-hex can be viewed as an implementation of the stable model semantics of GQ-formulas.

The generality of the formalism is useful in providing a principled way to study and compare the different extensions of the stable model semantics. Indeed, it led us to define yet another semantics of logic programs with abstract constraints, and yet another semantics of nonmonotonic dl-programs, both of which are in the spirit of the first-order stable model semantics. The unifying framework also saves efforts in re-proving the theorems for these individual extension. We extend several important theorems in ASP to formulas with generalized quantifiers, which in turn can be applied to the particular extensions of the stable model semantics.

5.8 Proofs

We omit the proof of Theorem 11 since the proof is a rewriting from the proof of Theorem 9 from (Ferraris et al., 2011b).

Useful Lemmas

Lemma 40 *Let F be a GQ-formula. Formula*

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

is logically valid.

Proof. By induction on F. ∎

Lemma 41 *Let F be a GQ-formula. Formula*

$$\boldsymbol{q} = \boldsymbol{p} \rightarrow (F^*(\boldsymbol{q}) \leftrightarrow F^*(\boldsymbol{p}))$$

is logically valid.

Proof. By induction on F. ∎

To facilitate the proofs, we introduce the following notion. Let Q be a generalized quantifier and let I be an interpretation. We say that Q^U is *monotone in the i-th argument position* if the following holds: if $Q^U(R_1, \ldots, R_k) = \boldsymbol{t}$ and $R_i \subseteq R_i' \subseteq |I|^{|\boldsymbol{x}^i|}$, then $Q^U(R_1, \ldots, R_{i-1}, R_i', R_{i+1}, \ldots, R_k) = \boldsymbol{t}$. Similarly, we say that Q^U is *anti-monotone in the i-th argument position* if the following holds: if $Q^U(R_1, \ldots, R_k) = \boldsymbol{t}$ and $R_i' \subseteq R_i \subseteq |I|^{|\boldsymbol{x}^i|}$, then $Q^U(R_1, \ldots, R_{i-1}, R_i', R_{i+1}, \ldots, R_k) = \boldsymbol{t}$. Clearly, Q is monotone (anti-monotone) in the i-th argument position iff Q^U is monotone (anti-monotone) in the i-th argument position for any interpretation I. Similarly, we define that Q^U is monotone (anti-monotone) in some set of argument positions.

Lemma 42 *Consider GQ sentences*

$$F = Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k)),$$

$$G = Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k)),$$

any subset M *of* $\{1, \ldots, k\}$*, and any interpretation* I*.*

(a) If Q^U *is monotone in* M*, then*

$$I \models \Big(\textstyle\bigwedge_{i \in M} \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \rightarrow G_i(\boldsymbol{x}_i)) \wedge \\ \textstyle\bigwedge_{i \in \{1,\ldots,k\} \setminus M} \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \leftrightarrow G_i(\boldsymbol{x}_i)) \Big) \\ \rightarrow (F \rightarrow G).$$

(b) If Q^U *is anti-monotone in* M*, then*

$$I \models \Big(\textstyle\bigwedge_{i \in M} \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \rightarrow G_i(\boldsymbol{x}_i)) \wedge \\ \textstyle\bigwedge_{i \in \{1,\ldots,k\} \setminus M} \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \leftrightarrow G_i(\boldsymbol{x}_i)) \Big) \\ \rightarrow (G \rightarrow F).$$

Proof.

(a): Assume

$$I \models \bigwedge_{i \in M} \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \rightarrow G_i(\boldsymbol{x}_i)) \wedge \bigwedge_{i \in \{1,\ldots,k\} \setminus M} \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \leftrightarrow G_i(\boldsymbol{x}_i))$$

and $I \models F$. Consider $(\boldsymbol{x}_i.F_i)^I = \{\boldsymbol{\xi} \mid I \models F_i(\boldsymbol{\xi}^*)\}$ and $(\boldsymbol{x}_i.G_i)^I = \{\boldsymbol{\xi} \mid I \models G_i(\boldsymbol{\xi}^*)\}$ for each i in $\{1, \ldots, k\}$.

- If $i \in M$, it follows from $I \models \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \rightarrow G_i(\boldsymbol{x}_i))$ that $(\boldsymbol{x}_i.F_i)^I \subseteq (\boldsymbol{x}_i.G_i)^I$.
- If $i \in \{1, \ldots, k\} \setminus M$, it follows from $I \models \forall \boldsymbol{x}_i (F_i(\boldsymbol{x}_i) \leftrightarrow G_i(\boldsymbol{x}_i))$ that $(\boldsymbol{x}_i.F_i)^I = (\boldsymbol{x}_i.G_i)^I$.

From $I \models F$, by definition, $Q^U((\boldsymbol{x}_1.F_1)^I, \ldots, (\boldsymbol{x}_k.F_k)^I) = \boldsymbol{t}$. Since Q^U is monotone in M, it follows that $Q^U((\boldsymbol{x}_1.G_1)^I, \ldots, (\boldsymbol{x}_k.G_k)^I) = \boldsymbol{t}$. Thus $I \models G$.

(b): Similar to (a). ∎

The following lemma follows immediately from Lemma 42.

Lemma 43 *Let M be a subset of $\{1,\ldots,k\}$ and Q a generalized quantifier. Consider formulas*

$$F(\boldsymbol{x}) = Q[\boldsymbol{x}_1],\ldots,[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1,\boldsymbol{x}),\ldots,F_k(\boldsymbol{x}_k,\boldsymbol{x})),$$

$$G(\boldsymbol{x}) = Q[\boldsymbol{x}_1],\ldots,[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1,\boldsymbol{x}),\ldots,G_k(\boldsymbol{x}_k,\boldsymbol{x})),$$

where $\boldsymbol{x}$ is a list of all free variables in F and G.

(a) If Q is monotone in M, then

$$\Big(\bigwedge_{i\in M}\forall\boldsymbol{x}_i(F_i(\boldsymbol{x}_i,\boldsymbol{x})\to G_i(\boldsymbol{x}_i,\boldsymbol{x}))\wedge \bigwedge_{i\in\{1,\ldots,k\}\setminus M}\forall\boldsymbol{x}_i(F_i(\boldsymbol{x}_i,\boldsymbol{x})\leftrightarrow G_i(\boldsymbol{x}_i,\boldsymbol{x}))\Big)\to(F(\boldsymbol{x})\to G(\boldsymbol{x}))$$

is logically valid.

(b) If Q is anti-monotone in M, then

$$\Big(\bigwedge_{i\in M}\forall\boldsymbol{x}_i(F_i(\boldsymbol{x}_i,\boldsymbol{x})\to G_i(\boldsymbol{x}_i,\boldsymbol{x}))\wedge \bigwedge_{i\in\{1,\ldots,k\}\setminus M}\forall\boldsymbol{x}_i(F_i(\boldsymbol{x}_i,\boldsymbol{x})\leftrightarrow G_i(\boldsymbol{x}_i,\boldsymbol{x}))\Big)\to(G(\boldsymbol{x})\to F(\boldsymbol{x}))$$

is logically valid.

Lemma 44 *If F is negative on $\boldsymbol{p}$ then*

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

is logically valid.

Proof. By induction on F.

Case 1: F is an atomic formula. If F is of the form $p_i(\boldsymbol{t})$ then $p_i\notin\boldsymbol{p}$ since F is negative on $\boldsymbol{p}$. Consequently, $F^*(\boldsymbol{u})$ is the same as F. Otherwise, $F^*(\boldsymbol{u})$ is the same as F by definition.

Case 2: F is of the form (5.1). In view of Lemma 40, it is sufficient to show that

$$(\boldsymbol{u}\le\boldsymbol{p})\to(F\to F^*(\boldsymbol{u})) \tag{5.28}$$

is logically valid. Let Anti be the set of all anti-monotone argument positions of Q.

- Consider any F_i, where $i \in \{1, \ldots, k\} \setminus \mathrm{Anti}$. Since F is negative on $\boldsymbol{p}$, it follows that F_i is negative on $\boldsymbol{p}$. By I.H.,

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

  is logically valid.

- Consider any F_i, where $i \in \mathrm{Anti}$. By Lemma 40,

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

  is logically valid.

From the two bullets, by Lemma 43 (b), we conclude (5.28). ∎

Proof of Proposition 32

An interpretation I of a signature σ can be represented as a pair $\langle J, X \rangle$, where J is the restriction of I to the function constants in σ, and X is the set of the atoms, formed using predicate constants from σ and the names of elements of $|I|$, which are satisfied by I. When I is an Herbrand interpretation, we often omit J and represent I by X.

By σ^+ we denote the signature obtained from σ by adding new predicate constants $\boldsymbol{q}$, one per each member of $\boldsymbol{p}$. About an atomic formula formed using a predicate constant from σ^+ and names of elements of $|I|$ we say that it is a $\boldsymbol{p}$*-atom* if its predicate constant belongs to $\boldsymbol{p}$, and that it is a $\boldsymbol{q}$*-atom* otherwise. For any set X of $\boldsymbol{p}$-atoms we denote by $X^{\boldsymbol{p}}_{\boldsymbol{q}}$ the set of the $\boldsymbol{q}$-atoms that are obtained from the elements of X by replacing their predicate constants by the corresponding predicate constants from $\boldsymbol{q}$.

Lemma 45 *Let M be a subset of $\{1, \ldots, k\}$ and let $Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k))$ be a sentence such that F_j is negative on $\boldsymbol{p}$ for all j in $\{1, \ldots, k\} \setminus M$. Consider any interpretation $I = \langle J, X \rangle$ and any subset Y of X.*

(a) If Q^U is monotone in M, then

$$\begin{aligned}\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models (Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \dots, F_k(\boldsymbol{x}_k)))^*(\boldsymbol{q}) \\ \leftrightarrow Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1), \dots, F_k^*(\boldsymbol{x}_k)).\end{aligned}$$

(b) If Q^U is anti-monotone in M, then

$$\begin{aligned}\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models (Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \dots, F_k(\boldsymbol{x}_k)))^*(\boldsymbol{q}) \\ \leftrightarrow Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \dots, F_k(\boldsymbol{x}_k)).\end{aligned}$$

Proof. (a) It is sufficient to show that

$$\begin{aligned}\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1), \dots, F_k^*(\boldsymbol{x}_k)) \\ \rightarrow Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \dots, F_k(\boldsymbol{x}_k)).\end{aligned} \tag{5.29}$$

- For every $i \in \{1, \dots, k\} \setminus M$, F_i is negative on $\boldsymbol{p}$. By Lemma 44, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models F_i^*(\boldsymbol{q}) \leftrightarrow F_i$.
- For every $i \in M$, by Lemma 40, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models F_i^*(\boldsymbol{q}) \rightarrow F_i$.

From the above two facts, by Lemma 42(a), we conclude (5.29).

(b) It is sufficient to show that

$$\begin{aligned}\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \dots, F_k(\boldsymbol{x}_k)) \\ \rightarrow Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1), \dots, F_k^*(\boldsymbol{x}_k)).\end{aligned} \tag{5.30}$$

- For every $i \in \{1, \dots, k\} \setminus M$, F_i is negative on $\boldsymbol{p}$. By Lemma 44, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models F_i^*(\boldsymbol{q}) \leftrightarrow F_i$.
- For every $i \in M$, by Lemma 40, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models F_i^*(\boldsymbol{q}) \rightarrow F_i$.

From the above two facts, by Lemma 42(b), we conclude (5.30). ∎

We now prove a slightly more general version of Proposition 32.

Proposition 32' *Let M be a subset of $\{1, \dots, k\}$ and let $Q[\boldsymbol{x}_1] \dots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \dots, F_k(\boldsymbol{x}_k))$ be a formula such F_j is negative on $\boldsymbol{p}$ for all $j \in \{1, \dots, k\} \setminus M$.*

(a) If Q is monotone in M, then

$$\boldsymbol{u} \leq \boldsymbol{p} \rightarrow ((Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k)))^* \\ \leftrightarrow Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1), \ldots, F_k^*(\boldsymbol{x}_k)))$$

is logically valid.

(b) If Q is anti-monotone in M, then

$$\boldsymbol{u} \leq \boldsymbol{p} \rightarrow ((Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k)))^* \\ \leftrightarrow Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k)))$$

is logically valid.

Proof. Clear from Lemma 45. ∎

Proof of Proposition 33

Proposition 33 *Let σ be a signature that contains finitely many predicate constants, let $\sigma^{\boldsymbol{p}}$ be the set of predicate constants in σ, let $I = \langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}} \rangle$ be an interpretation of σ, and let F be a GQ-sentence of σ. Then $I \models F$ iff $I^{\boldsymbol{p}} \models gr_I[F]$.*

Proof. By induction on F.

Case 1: F is $p(t_1, \ldots, t_k)$. $gr_I[p(t_1, \ldots, t_k)] = p((t_1^{I^{\boldsymbol{f}}})^*, \ldots, (t_k^{I^{\boldsymbol{f}}})^*)$.

$$I \models F \text{ iff } p((t_1^{I^{\boldsymbol{f}}})^*, \ldots, (t_k^{I^{\boldsymbol{f}}})^*) \in I^{\boldsymbol{p}} \text{ iff } I^{\boldsymbol{p}} \models gr_I[p(t_1, \ldots, t_k)].$$

Case 2: F is $t_1 = t_2$. $I \models F$ iff $t_1^{I^{\boldsymbol{f}}} = t_2^{I^{\boldsymbol{f}}}$ iff $gr_I[F]$ is $\top$ iff $I^{\boldsymbol{p}} \models gr_I[F]$.

Case 3: F is of the form (5.1). By definition, $I \models F$ iff

$$Q^U((\boldsymbol{x}_1.F_1(\boldsymbol{x}_1))^I, \ldots, (\boldsymbol{x}_k.F_k(\boldsymbol{x}_k))^I) = \boldsymbol{t}, \tag{5.31}$$

where $(\boldsymbol{x}_i.F_i(\boldsymbol{x}_i))^I = \{\boldsymbol{\xi} \in U^{n_i} \mid I \models F_i(\boldsymbol{\xi}^*)\}$. By I.H.

$$(\boldsymbol{x}_i.F_i(\boldsymbol{x}_i))^I = \{\boldsymbol{\xi} \in U^{n_i} \mid I^{\boldsymbol{p}} \models gr_I[F_i(\boldsymbol{\xi}^*)]\}.$$

The later is the same as the set $S_i^I = \{\boldsymbol{\xi} \mid I \models F,\ \boldsymbol{\xi}.F \in S_i\}$ where

$$S_i = \{\boldsymbol{x}_i\theta.gr_I[F(\boldsymbol{x}_i)\theta] \mid \theta \text{ is a substitution from variables in } \boldsymbol{x}_i \text{ to names}\}.$$

As a result, (5.31) iff $Q^U(S_1^I, \ldots, S_k^I) = \boldsymbol{t}$. ∎

Proof of Theorem 10

Theorem 10 *Let σ be a signature that contains finitely many predicate constants, let $\sigma^{\boldsymbol{p}}$ be the set of predicate constants in σ, let $I = \langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}} \rangle$ be an interpretation of σ, and let F be a GQ-sentence of σ. $I \models SM[F; \sigma^{\boldsymbol{p}}]$ iff $I^{\boldsymbol{p}}$ is a minimal set of atoms that satisfies $(gr_I[F])^{\underline{I}}$.*

Proof. It is sufficient to show that for any $J^{\boldsymbol{p}} \subseteq I^{\boldsymbol{p}}$,

$$\langle I^{\boldsymbol{f}}, J^{\boldsymbol{q}} \cup I^{\boldsymbol{p}} \rangle \models F^*(\boldsymbol{q})$$

iff

$$J^{\boldsymbol{p}} \models (\mathrm{gr}_I[F])^{\underline{I}}.$$

This is proven by induction on F.

Case 1: F is $p(t_1, \ldots, t_k)$. Clear.

Case 2: F is $t_1 = t_2$. Clear from definition.

Case 3: F is of the form (5.1). F^* is

$$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1), \ldots, F_k^*(\boldsymbol{x}_k)) \wedge Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1(\boldsymbol{x}_1), \ldots, F_k(\boldsymbol{x}_k)).$$

$gr_I[F]$ is

$$Q^U(S_1, \ldots, S_k)$$

where $S_i = \{\boldsymbol{x}_i\theta.gr_I[F(\boldsymbol{x}_i\theta)] \mid \theta$ is a substitution from $\boldsymbol{x}_i$ to names$\}$.

Consider two cases:

Case 3.1: $I^{\boldsymbol{p}} \not\models gr_I[F]$. $(\mathrm{gr}_I[F])^{\underline{I}}$ is $\bot$. From $I^{\boldsymbol{p}} \not\models gr_I[F]$, by Proposition 33, $\langle I^{\boldsymbol{f}}, I^{\boldsymbol{p}} \rangle \not\models F$ follows. By Lemma 40, $\langle I^{\boldsymbol{f}}, J^{\boldsymbol{q}} \cup I^{\boldsymbol{p}} \rangle \not\models F^*(\boldsymbol{q})$.

Case 3.2: $I^{\boldsymbol{p}} \models gr_I[F]$. $(\mathrm{gr}_I[F])^{\underline{I}}$ is $Q^U(S_1', \ldots, S_k')$ where each $S_i' = \{\boldsymbol{x}_i\theta.(gr_I[F_i(\boldsymbol{x}_i\theta)])^{\underline{I}} \mid \theta$ is a substitution from $\boldsymbol{x}_i$ to names$\}$. By I.H. for each $1 \leq i \leq k$,

$$J^{\boldsymbol{p}} \models (gr_I[F_i(\boldsymbol{x}_i\theta)])^{\underline{I}}$$

iff

$$\langle I^{\boldsymbol{f}}, J^{\boldsymbol{q}} \cup I^{\boldsymbol{p}} \rangle \models (F_i(\boldsymbol{x}_i\theta))^*(\boldsymbol{q}).$$

Let

$$S_i'' = \{\boldsymbol{x}_i\theta \mid J^{\boldsymbol{p}} \models (gr_I[F_i(\boldsymbol{x}_i\theta)])^{\underline{I}}, \theta \text{ is a substitution from } \boldsymbol{x}_i \text{ to names}\}$$

and

$$S_i^* = \{\boldsymbol{x}_i\theta \mid \langle I^{\boldsymbol{f}}, J^{\boldsymbol{q}} \cup I^{\boldsymbol{p}} \rangle \models (F_i(\boldsymbol{x}_i\theta))^*(\boldsymbol{q}), \theta \text{ is a substitution from } \boldsymbol{x}_i \text{ to names}\}.$$

It is clear that $S_i'' = S_i^*$. As a result,

$$Q^U(S_1'', \ldots, S_k'') = \boldsymbol{t} \tag{5.32}$$

iff

$$Q^U(S_1^*, \ldots, S_k^*) = \boldsymbol{t}. \tag{5.33}$$

(5.32) is equivalent to saying that $J^{\boldsymbol{p}} \models (\mathrm{gr}_I[F])^{\underline{I}}$ and (5.33) is equivalent to saying that $\langle I^{\boldsymbol{f}}, J^{\boldsymbol{q}} \cup I^{\boldsymbol{p}} \rangle \models F^*(\boldsymbol{q})$. ∎

Proof of Proposition 34

Lemma 46 *let E be an aggregate expression (4.16) that contains no free variables, let E^{GQ} be the GQ-representation of E, and let I be an interpretation. $I \models E$ iff $I \models E^{GQ}$.*

Proof. Let α be the join of the multisets $msp(\boldsymbol{x}_1.F_1)^I, \ldots, msp(\boldsymbol{x}_n.F_n)^I$. By definition, $I \models E$ iff (i) OP(α) is defined, (ii) $b^I \in \boldsymbol{N}um$, and (iii) OP$(\alpha) \succeq b^I$.

The three conditions hold iff $Q^I_{(\mathsf{OP},\succeq)}((\boldsymbol{x}_1.F_1)^I, \ldots, (\boldsymbol{x}_n.F_n)^I, \{b^I\}) = \boldsymbol{t}$, which is the same as saying that $I \models E^{GQ}$. ∎

Proposition 34 *Let F be an aggregate sentence of σ, let F^{GQ} be the GQ-representation of F, and let $\boldsymbol{p}$ be a list of predicate constants. For any expansion I of σ_{bg} to σ, $I \models SM[F; \boldsymbol{p}]$ (according to Ferraris and Lifschitz) iff $I \models SM[F^{GQ}; \boldsymbol{p}]$.*

Proof. In view of Theorem 11, we will prove that for any aggregate formula F, F^* is equivalent to $(F^{GQ})^*$. It is sufficient to prove that, for any aggregate expression

$$\mathsf{OP}\langle \boldsymbol{x}_1.F_1(\boldsymbol{x}_1, \boldsymbol{p}), \ldots, \boldsymbol{x}_n.F_n(\boldsymbol{x}_n, \boldsymbol{p})\rangle \succeq b$$

that contains no free variables, any interpretation $I = \langle J, X\rangle$, and any subset Y of X, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle$ satisfies

$$\begin{aligned}\mathsf{OP}\langle \boldsymbol{x}_1.F_1(\boldsymbol{x}_1,\boldsymbol{p}),\ldots,\boldsymbol{x}_n.F_n(\boldsymbol{x}_n,\boldsymbol{p})\rangle \succeq b \\ \wedge\ \mathsf{OP}\langle \boldsymbol{x}_1.F^*_1(\boldsymbol{x}_1,\boldsymbol{q}),\ldots,\boldsymbol{x}_n.F^*_n(\boldsymbol{x}_n,\boldsymbol{q})\rangle \succeq b\end{aligned} \tag{5.34}$$

iff $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle$ satisfies

$$\begin{aligned}Q_{(\mathsf{OP},\succeq)}\ [\boldsymbol{x}_1]\ldots[\boldsymbol{x}_n][y](F_1(\boldsymbol{x}_1,\boldsymbol{p}),\ldots,F_n(\boldsymbol{x}_n,\boldsymbol{p}),y=b)\wedge \\ Q_{(\mathsf{OP},\succeq)}\ [\boldsymbol{x}_1]\ldots[\boldsymbol{x}_n][y](F^*_1(\boldsymbol{x}_1,\boldsymbol{q}),\ldots,F^*_n(\boldsymbol{x}_n,\boldsymbol{q}),y=b).\end{aligned} \tag{5.35}$$

By Lemma 46, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle$ satisfies the first (second) conjunctive term of (5.34) iff $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle$ satisfies the first (second) conjunctive term of (5.35). ∎

Proof of Proposition 35

Lemma 47 *For any dl-program $(\mathcal{T}, \Pi)$, any dl-atom (5.7) in Π that contains no free variables and any Herbrand interpretation I of $\langle C, P_\Pi\rangle$, $I \models_{\mathcal{T}} (5.7)$ iff $I \models (5.11)$ iff $I \models (5.12)$.*

Proof. It is sufficient to consider a GQ-formula of the form (5.11) since (5.12) is equivalent to (5.11).

By definition, $I \models_{\mathcal{T}} (5.7)$ iff $\mathcal{T} \cup \bigcup_{i=1}^{k} A_i(I)$ entails $Query(\boldsymbol{t})$. Note that $p_i(\boldsymbol{e}) \in I$ iff $\boldsymbol{e} \in \{\boldsymbol{c} \mid I \models p_i(\boldsymbol{c})\}$ and $p_i(\boldsymbol{e}) \notin I$ iff $\boldsymbol{e} \in |I|^{|\boldsymbol{e}|} \setminus \{\boldsymbol{c} \mid I \models p_i(\boldsymbol{c})\}$. Consequently, $A_i(I)$ is the same as $A_i(R_i)$, which is defined as

- $\{S_i(\boldsymbol{e}) \mid \boldsymbol{e} \in R_i\}$ if op_i is $\oplus$,
- $\{\neg S_i(\boldsymbol{e}) \mid \boldsymbol{e} \in R_i\}$ if op_i is $\odot$,
- $\{\neg S_i(\boldsymbol{e}) \mid \boldsymbol{e} \in |I|^{|\boldsymbol{e}|} \setminus R_i\}$ if op_i is $\ominus$,

where $R_i = \{\boldsymbol{c} \mid I \models p_i(\boldsymbol{c})\}$. Clearly, $\mathcal{T} \cup \bigcup_{i=1}^{k} A_i(I)$ entails $Query(\boldsymbol{t})$ iff $\mathcal{T} \cup \bigcup_{i=1}^{k} A_i(R_i)$ entails $Query(\boldsymbol{t})$ iff $I \models (5.11)$. ∎

Given a dl-program $(\mathcal{T}, \Pi)$, we denote $s(\Pi)^X_{\mathcal{T}}$ as the strong reduct of Π relative to $\mathcal{T}$. For a set X of dl-atoms, we denote X^{sGQ} as the set of atoms obtained from X by

identifying each dl-atom as (5.11) if it is monotonic and (5.12) otherwise. Similarly, X^{wGQ} is the set of atoms obtained from X by identifying each dl-atom as (5.12).

Lemma 48 *For any dl-program $(\mathcal{T}, \Pi)$, any Herbrand interpretations X, Y of $\langle C, P_\Pi \rangle$ such that $Y \subseteq X$, and any rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π,*

$$Y \models_{\mathcal{T}} s(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$$

iff

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models_{\mathcal{T}} (B^{sGQ} \wedge N^{sGQ})^*(\boldsymbol{q}) \rightarrow q(\boldsymbol{t}). \tag{5.36}$$

Proof. By Lemma 44, $(N^{sGQ})^*(\boldsymbol{q})$ is equivalent to N^{sGQ}. We partition B into two sets: the set B_1 of all monotonic dl-atoms and the set B_2 of all non-monotonic dl-atoms.

It is clear from (5.12) that B_2^{sGQ} is negative on $\boldsymbol{p}$. By Lemma 44 again, $(B_2^{sGQ})^*(\boldsymbol{q})$ is equivalent to B_2^{sGQ}. Thus (5.36) is equivalent to saying that

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models_{\mathcal{T}} (B_1^{sGQ})^*(\boldsymbol{q}) \wedge B_2^{sGQ} \wedge N^{sGQ} \rightarrow q(\boldsymbol{t}). \tag{5.37}$$

Consider two cases.

Case 1: $X \models_{\mathcal{T}} B_2 \wedge N$. Then $s(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ is $p(\boldsymbol{t}) \leftarrow B_1$. By Lemma 47, $Y \models_{\mathcal{T}} B_1 \rightarrow p(\boldsymbol{t})$ iff

$$Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models_{\mathcal{T}} (B_1^{sGQ})(\boldsymbol{q}) \rightarrow q(\boldsymbol{t}). \tag{5.38}$$

From $Y \subseteq X$ and that all dl-atoms in B_1 are monotonic, it follows that $Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models_{\mathcal{T}} (B_1^{sGQ})(\boldsymbol{q})$ implies $X \models_{\mathcal{T}} B_1^{sGQ}$. So (5.38) is equivalent to

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models_{\mathcal{T}} (B_1^{sGQ})(\boldsymbol{q}) \wedge B_1^{sGQ} \rightarrow q(\boldsymbol{t}),$$

which is also equivalent to (5.37) under the assumption that $X \models_{\mathcal{T}} B_2 \wedge N$.

Case 2: $X \not\models_{\mathcal{T}} B_2 \wedge N$. Then $s(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ is equivalent to $\top$. On the other hand, by Lemma 47, $X \not\models_{\mathcal{T}} B_2^{sGQ} \wedge N^{sGQ}$. So we get (5.37). ∎

Lemma 49 *For any dl-program $(\mathcal{T}, \Pi)$ and any Herbrand interpretation X of $\langle C, P_\Pi \rangle$, $X \models \Pi^{sGQ}$ iff $X \models_{\mathcal{T}} s\Pi^X_{\mathcal{T}}$.*

Proof. Immediate from the definition of $s\Pi^X_{\mathcal{T}}$, $X \models_{\mathcal{T}} s\Pi^X_{\mathcal{T}}$ iff $X \models_{\mathcal{T}} \Pi$. By Lemma 47, $X \models_{\mathcal{T}} \Pi$ iff $X \models \Pi^{sGQ}$. ∎

Proposition 35 *For any dl-program $(\mathcal{T}, \Pi)$, the weak (strong, respectively) answer sets of $(\mathcal{T}, \Pi)$ are precisely the Herbrand interpretations of $\langle C, P_\Pi \rangle$ that satisfy $SM[\mathcal{P}^{wGQ}; P_\Pi]$ ($SM[\mathcal{P}^{sGQ}; P_\Pi]$, respectively) relative to $\mathcal{T}$.*

Proof. We only prove the case for strong answer sets. The proof for weak answer sets is similar.

Let X be an Herbrand interpretation of $\langle C, P_\Pi \rangle$. X is a strong answer set of $(\mathcal{T}, \Pi)$ iff

(i) $X \models_{\mathcal{T}} s\Pi^X_{\mathcal{T}}$, and

(ii) no proper subset Y of X satisfies $s\Pi^X_{\mathcal{T}}$ relative to $\mathcal{T}$.

On the other hand, $X \models \mathrm{SM}[P^{sGQ}; P_\Pi]$ iff

(i′) $X \models \Pi^{sGQ}$, and

(ii′) X does not satisfy $\exists \boldsymbol{u}(\boldsymbol{u} < P_\Pi \wedge (\Pi^{sGQ})^*(\boldsymbol{u}))$.

By Lemma 49, (i) is equivalent to (i′). Assume (i′). Condition (ii) can be reformulated as: no proper subset Y of X satisfies $s(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ relative to $\mathcal{T}$ for every rule $p(\boldsymbol{t}) \leftarrow B, N \in \Pi$. Under the assumption (i′), condition (ii′) can be reformulated as: there is no proper subset Y of X such that $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models_{\mathcal{T}} (B^{sGQ} \wedge N^{sGQ})^*(\boldsymbol{q}) \rightarrow q(\boldsymbol{t})$ for every rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π. By Lemma 48, (ii) is equivalent to (ii′). ∎

Proof of Proposition 36

Lemma 50 *For any c-atom (D, C) of σ, let I be an interpretation of σ. I satisfies (D, C) iff $I \models (5.16)$.*

Proof. $I \models (D, C)$ iff $I \cap D \in C$ iff $R \in C$ where $R = I \cap D$. Consider $R_1, \ldots, R_n$ such that $R_i = \{\epsilon\}$ if $p_i \in R$ and $R_i = \emptyset$ otherwise. $R = I \cap D$ iff $Q^I_C(R_1, \ldots, R_n) = \boldsymbol{t}$. ∎

Lemma 51 *For any c-atom* (D, C)*,*

$$Q_C[\,]\ldots[\,]D \tag{5.39}$$

is equivalent to

$$\bigwedge_{\overline{C} \in \mathcal{P}(D)\setminus C} (\bigwedge_{p \in \overline{C}} p \rightarrow \bigvee_{p \in D\setminus\overline{C}} p). \tag{5.40}$$

Proof. Consider any subset X of $\{p_1, \ldots, p_n\}$. It is sufficient to prove that $X^{\boldsymbol{p}}_{\boldsymbol{q}} \models (5.39)$ iff $X^{\boldsymbol{p}}_{\boldsymbol{q}} \models (5.40)$.

From left to right: Assume $X^{\boldsymbol{p}}_{\boldsymbol{q}} \models (5.39)$. It is follows from Lemma 50 that $X \models (D, C)$. As a result, $X \notin \mathcal{P}(D) \setminus C$. Consider any $\overline{C} \in \mathcal{P}(D) \setminus C$ such that $X^{\boldsymbol{p}}_{\boldsymbol{q}} \models \bigwedge_{p \in \overline{C}} q$. It is clear that $\overline{C} \subseteq X$ and $\overline{C} \neq X$ (since $X \notin \mathcal{P}(D) \setminus C$). Consequently,

$$X^{\boldsymbol{p}}_{\boldsymbol{q}} \models \bigvee_{p \in D\setminus\overline{C}} q.$$

From right to left: Assume $X^{\boldsymbol{p}}_{\boldsymbol{q}} \models (5.40)$. Clearly, $X \notin \mathcal{P}(D) \setminus C$. So $X \models (D, C)$ and, by Lemma 50, $X^{\boldsymbol{p}}_{\boldsymbol{q}} \models (5.39)$. ∎

Lemma 52 *For any c-atoms* (D, C)*,* (5.16) *is strongly equivalent to* (5.17)*.*

Proof. In view of Theorem 11, it is sufficient to prove that for any list $(q_1, \ldots, q_n)$ of new atoms such that $(q_1, \ldots, q_n) \leq (p_1, \ldots, p_n)$,

$$Q_C[\,]\ldots[\,](p_1, \ldots, p_n) \wedge Q_C[\,]\ldots[\,](q_1, \ldots, q_n) \tag{5.41}$$

is equivalent to

$$\begin{array}{l} \bigwedge_{\overline{C} \in \mathcal{P}(D)\setminus C} (\bigwedge_{p \in \overline{C}} p \rightarrow \bigvee_{p \in D\setminus\overline{C}} p) \wedge \\ \qquad \bigwedge_{\overline{C} \in \mathcal{P}(D)\setminus C} (\bigwedge_{p \in \overline{C}} q \rightarrow \bigvee_{p \in D\setminus\overline{C}} q). \end{array} \tag{5.42}$$

Consider any subset X of $\{p_1, \ldots, p_n\}$ and any subset Y of X, we will show that $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models (5.41)$ iff $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models (5.42)$. It follows from Lemma 51 that X satisfies the first conjunctive term of (5.41) iff X satisfies the first conjunctive term of (5.42). Similarly, $Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ satisfies the second conjunctive term of (5.41) iff $Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ satisfies the second conjunctive term of (5.42). ∎

Proposition 36 *For any propositional formula F with c-atoms and any propositional interpretation X, X is an answer set of F iff X is an answer set of $Fer(F)$.*

Proof. Clear from Lemma 52. ∎

Proof of Proposition 37

Lemma 53 *Let (V, D, C) be a constraint satisfaction problem, c an explicit constraint in C, and I an expansion of an interpretation of σ_{bg} to σ that conforms to (V, D, C). I satisfies c iff $I \models (5.18)$.*

Proof. Clear from definition. ∎

Proposition 37 *Let (V, D, C) be a constraint satisfaction problem, let F be a sentence of signature σ with explicit constraints of signature σ_{bg} such that $\sigma_{bg} \subseteq \sigma$, let $\boldsymbol{p}$ be a list of predicates whose members belong to $\sigma \setminus \sigma_{bg}$. For any expansion I of an interpretation of σ_{bg} to σ that conforms to (V, D, C), $I \models SM[F;\ \boldsymbol{p}]$ iff $I \models SM[F^{GQ};\ \boldsymbol{p}]$.*

Proof. Let $I = \langle J, X\rangle$ as defined before. Consider any subset Y of X. We will prove that for any formula F with explicit constraints, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \models F^*(\boldsymbol{q})$ iff $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \models (F^{GQ})^*(\boldsymbol{q})$. It is sufficient to prove that, for any explicit constraint $c \in C$, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \models c^*(\boldsymbol{q})$ iff $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \models (Q_c[x_1]\ldots[x_n](x_1 = v_1, \ldots, x_n = v_n))^*(\boldsymbol{q})$. It is clear that that both c and $Q_c[x_1]\ldots[x_n](x_1 = v_1, \ldots, x_n = v_n)$ are negative on $\boldsymbol{p}$. By Lemma 44, $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \models c^*(\boldsymbol{q})$ is equivalent to $I \models c$ and $\langle J, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \models (Q_c[x_1]\ldots[x_n](x_1 = v_1, \ldots, x_n = v_n))^*(\boldsymbol{q})$ is equivalent to $I \models Q_c[x_1]\ldots[x_n](x_1 = v_1, \ldots, x_n = v_n)$. So the claim follows from Lemma 53. ∎

Lemma 54 *If every occurrence of every predicate constant from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F, then*

$$(\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2) \rightarrow (F^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \leftrightarrow F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)) \tag{5.43}$$

is logically valid.

Proof. By induction on F.

Case 1: F is an atomic formula.

- If F is of the form $p(\boldsymbol{t})$ then $p \notin \boldsymbol{p}_2$ since every occurrence of every predicate constant from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F. Clearly, $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ is the same as $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$.
- Otherwise, it is clear that both $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ and $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ are the same as F.

Case 2: F is of the form (5.1).

- If F is negative on $\boldsymbol{p}$, by Lemma 44, both $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ and $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ are equivalent to F.
- Otherwise, F is not negative on $\boldsymbol{p}$. Consider any F_i where $i \in \{1, \ldots, k\}$. Note that every occurrence of every predicate constant from $\boldsymbol{p}_2$ in F is contained in a subformula of F that is negative on $\boldsymbol{p}$. Since F is not negative on $\boldsymbol{p}$, such subformula can not be F. It follows that every occurrence of every predicate constant from $\boldsymbol{p}_2$ in F_i is $\boldsymbol{p}$-negated in F_i. By I.H.,

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

  is logically valid. Consequently, (5.43) is logically valid.

∎

Lemma 55 *Let $\boldsymbol{p}$ be the list of all intensional predicates and let $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ be a partition of $\boldsymbol{p}$, 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 semi-positive occurrence of every predicate constant from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F, then

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

is logically valid.

(b) If every semi-negative occurrence of every predicate constant from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F, 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 proven simultaneously by induction on F.

Case 1: F is an atomic formula $p_i(\boldsymbol{t})$.

(a) Since every semi-positive occurrence of every predicate constant from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F, predicate constant p_i is not in $\boldsymbol{p}_2$, so $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ is the same as $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$.

(b) Clear from $(\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)$.

Case 2: F is $t_1 = t_2$ or $\bot$. Clear since both $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ and $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ are the same as F.

Case 3: F is of the form (5.1). Without loss of generality, we partition the set of all argument positions of Q into three sets: the set of monotone argument positions Mon, the set of anti-monotone argument positions Anti and the rest of argument positions Mixed.

(a) If F is negative on $\boldsymbol{p}$, by Lemma 44, both $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ and $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ are equivalent to F. Otherwise, assume $(\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)$.

- Consider any F_i, where $i \in \mathrm{Mon}$. Note that every semi-positive occurrence of predicates from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F. Since F is not negative on $\boldsymbol{p}$, such subformula can not be F. It follows that every semi-positive occurrence of predicates from $\boldsymbol{p}_2$ in F_i is $\boldsymbol{p}$-negated in F_i. By I.H. (a),

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \rightarrow F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \tag{5.44}$$

  is logically valid.

- Consider any F_i, where $i \in \mathrm{Mixed}$. Note that every semi-positive occurrence of predicates from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F. Since F is not negative on $\boldsymbol{p}$, such subformula can not be F. It follows that every occurrence of predicates from $\boldsymbol{p}_2$ in F_i is $\boldsymbol{p}$-negated in F_i. By Lemma 54,

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \rightarrow (F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \leftrightarrow F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2)) \tag{5.45}$$

  is logically valid.

- Consider any F_i, where $i \in \mathrm{Anti}$. Note that every semi-positive occurrence of predicates from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F. Since F is not negative on $\boldsymbol{p}$, such subformula can not be F. It follows that every semi-negative occurrence of predicates from $\boldsymbol{p}_2$ in F_i is $\boldsymbol{p}$-negated in F_i. By I.H. (b),

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \rightarrow F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \tag{5.46}$$

  is logically valid.

Since Q is monotone in Mon and anti-monotone in Anti, by Lemma 43 (a) and Lemma 43 (b),

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

follows from (5.44), (5.45) and (5.46).

(b) If F is negative on $\boldsymbol{p}$, by Lemma 44, both $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ and $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ are equivalent to F. Otherwise, assume $(\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)$.

- Consider any F_i, where $i \in \mathrm{Mon}$. Note that every semi-negative occurrence of predicates $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F_i. Since F is not negative on $\boldsymbol{p}$, such subformula can not be F. It follows that every semi-negative occurrence of predicates from $\boldsymbol{p}_2$ in F_i is $\boldsymbol{p}$-negated in F_i. By I.H. (b),

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \rightarrow F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \tag{5.47}$$

  is logically valid.

- Consider any F_i, where $i \in \mathrm{Mixed}$. Note that every semi-negative occurrence of predicates from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F. Since F is not negative on $\boldsymbol{p}$, such subformula can not be F. It follows that every occurrence of predicates from $\boldsymbol{p}_2$ in F_i is $\boldsymbol{p}$-negated in F_i. By Lemma 54,

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \rightarrow (F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \leftrightarrow F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2)) \tag{5.48}$$

  is logically valid.

- Consider any F_i where $i \in \mathrm{Anti}$. Note that every semi-negative occurrence of predicates from $\boldsymbol{p}_2$ in F is $\boldsymbol{p}$-negated in F. Since F is not negative on $\boldsymbol{p}$, such subformula can not be F. It follows that every semi-positive occurrence of predicates from $\boldsymbol{p}_2$ in F_i is $\boldsymbol{p}$-negated in F_i. By I.H. (a),

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \rightarrow F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \tag{5.49}$$

  is logically valid.

Since Q is monotone in Mon and anti-monotone in Anti, by Lemma 43(a) and Lemma 43(b),

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

follows from (5.47), (5.48) and (5.49). ∎

Lemma 56 *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*

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

is logically valid.

Proof. By induction on F.

Case 1: F is an atomic formula.

- If F is of the form $p(\boldsymbol{t})$, where $p \in \boldsymbol{p}_1$, then both $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ and $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ are $u(\boldsymbol{t})$.
- If F is of the form $p(\boldsymbol{t})$ where $p \in \boldsymbol{p}_2$, clear from Lemma 40 and the assumption $\boldsymbol{u}_2 \leq \boldsymbol{p}_2$.
- Otherwise, $F^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ and $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ are the same as F.

Case 2: F is of the form (5.1). Without loss of generality, we partition the set of all argument positions of Q into three sets: the set of monotone argument positions Mon, the set of anti-monotone argument positions Anti, and the rest of argument positions Mixed.

SubCase 2.1: F_i is negative on $\boldsymbol{p}_1$ for each $i \in \mathrm{Mon} \cup \mathrm{Mixed}$. Then F is negative on $\boldsymbol{p}_1$. Assuming

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

by Lemma 40, we get F, or equivalently $F^*(\boldsymbol{p}_1, \boldsymbol{p}_2)$, and by Lemma 44, we get $F^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$.

SubCase 2.2: F_i is not negative on $\boldsymbol{p}_1$ for some $i \in \mathrm{Mon} \cup \mathrm{Mixed}$.

– Consider any F_j where $j \in \mathrm{Anti} \cup \mathrm{Mixed}$. Since $\mathrm{DG}_{\boldsymbol{p}_1\boldsymbol{p}_2}[F]$ has no edges from predicates in $\boldsymbol{p}_1$ to predicates in $\boldsymbol{p}_2$, every semi-positive occurrence of predicates from $\boldsymbol{p}_2$ in F_j is $\boldsymbol{p}$-negated in F_j. By Lemma 55 (a),

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F_j^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \rightarrow F_j^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \tag{5.50}$$

is logically valid.

– Consider any F_j where $j \in \mathrm{Mon} \cup \mathrm{Mixed}$. Since the occurrence of F_j is strictly positive in F, $\mathrm{DG}_{\boldsymbol{p}_1\boldsymbol{p}_2}[F_j]$ is a subgraph of $\mathrm{DG}_{\boldsymbol{p}_1\boldsymbol{p}_2}[F]$. It follows that $\mathrm{DG}_{\boldsymbol{p}_1\boldsymbol{p}_2}[F_j]$ has no edges from predicate constants in $\boldsymbol{p}_1$ to predicate constants in $\boldsymbol{p}_2$. By I.H.,

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \wedge F_j^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \rightarrow F_j^*(\boldsymbol{u}_1, \boldsymbol{p}_2) \tag{5.51}$$

is logically valid.

From (5.50) and (5.51), it follows that

$$((\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)) \rightarrow (F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2) \leftrightarrow F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2)) \tag{5.52}$$

is logically valid for every $i \in \mathrm{Mixed}$.

Assume $(\boldsymbol{u}_1, \boldsymbol{u}_2) \leq (\boldsymbol{p}_1, \boldsymbol{p}_2)$, and

$$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1^*(\boldsymbol{u}_1, \boldsymbol{u}_2), \ldots, F_k^*(\boldsymbol{u}_1, \boldsymbol{u}_2)). \tag{5.53}$$

Let F' be the formula obtained from (5.53) by replacing $F_i^*(\boldsymbol{u}_1, \boldsymbol{u}_2)$ with $F_i^*(\boldsymbol{u}_1, \boldsymbol{p}_2)$ for every $i \in \mathrm{Mon} \cup \mathrm{Mixed}$. Since Q is monotone in Mon, by Lemma 43 (a), formula F' follows from (5.51) and (5.52). Since Q is anti-monotone in Anti, by Lemma 43 (b),

$$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](F_1^*(\boldsymbol{u}_1, \boldsymbol{p}_2), \ldots, F_k^*(\boldsymbol{u}_1, \boldsymbol{p}_2))$$

follows from F' and (5.50).

∎

Lemma 57 *For any GQ 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).

Theorem 12 *Let F be a GQ sentence, and let $\boldsymbol{p}$ be a tuple of distinct predicate constants. If $\boldsymbol{l}^1, \ldots, \boldsymbol{l}^n$ are all the loops of F relative to $\boldsymbol{p}$ then*

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

Proof. It is sufficient to prove the logical validity of the formula

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

where each $\boldsymbol{u}^i$ is the part of $\boldsymbol{u}$ that corresponds to the part $\boldsymbol{l}^i$ of $\boldsymbol{p}$, and $\widetilde{\boldsymbol{u}^i}$ is the list of symbols obtained from $\boldsymbol{p}$ by replacing every intensional predicate p that belongs to $\boldsymbol{l}^i$ with the corresponding predicate variable u.

From right to left: Clear.

From left to right: Assume $\exists \boldsymbol{u}((\boldsymbol{u} < \boldsymbol{p}) \wedge F^*(\boldsymbol{u}))$ and take $\boldsymbol{u}$ such that $(\boldsymbol{u} < \boldsymbol{p}) \wedge F^*(\boldsymbol{u})$. Consider several cases, each corresponding to a nonempty subset Y of $\boldsymbol{p}$. The assumption characterizing each case is that $u < p$ for each member p of $\boldsymbol{p}$ that belongs to Y, and that $u = p$ for each p that does not belong to Y. By Lemma 57, there is a loop $\boldsymbol{l}^i$ of F that is contained in Y such that the dependency graph $\mathrm{DG}_{\boldsymbol{p}}[F]$ has no edges from predicate constants in $\boldsymbol{l}^i$ to predicate constants in $Y \setminus \boldsymbol{l}^i$. Since $\boldsymbol{l}^i$ is contained in Y, from the fact that $u < p$ for each p in Y we can conclude that

$$\boldsymbol{u}^i < \boldsymbol{l}^i. \tag{5.54}$$

Let $\boldsymbol{u}'$ be the list of symbols obtained from $\boldsymbol{p}$ by replacing every member p that belongs to Y with the corresponding variable u. Under the assumption characterizing each case, $\boldsymbol{u} = \boldsymbol{u}'$, so that $F^*(\boldsymbol{u}) \leftrightarrow F^*(\boldsymbol{u}')$. Consequently, we can derive $F^*(\boldsymbol{u}')$. It follows from Lemma 56 that the formula

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

is logically valid, so that we further conclude that $F^*(\widetilde{\boldsymbol{u}^i})$. In view of (5.54), it follows that $\exists \boldsymbol{u}^i((\boldsymbol{u}^i < \boldsymbol{l}^i) \wedge F^*(\widetilde{\boldsymbol{u}^i}))$. ∎

Theorem 13 *Let F, G be GQ sentences, and let $\boldsymbol{p}$, $\boldsymbol{q}$ be disjoint tuples of distinct predicate constants. If*

- *each strongly connected component of $DG_{\boldsymbol{pq}}[F \wedge G]$ is a subset of $\boldsymbol{p}$ or a subset of $\boldsymbol{q}$,*
- *F is negative on $\boldsymbol{q}$, and*
- *G is negative on $\boldsymbol{p}$*

then

$$SM[F \wedge G; \boldsymbol{pq}] \quad \textit{is equivalent to} \quad SM[F; \boldsymbol{p}] \wedge SM[G; \boldsymbol{q}].$$

Proof. Same as the proof in (Ferraris et al., 2009b). ∎

Proof of Theorem 14

Theorem 14 *For any GQ formula F in Clark normal form that is tight on $\boldsymbol{p}$, $SM[F; \boldsymbol{p}]$ is equivalent to the completion of F relative to $\boldsymbol{p}$.*

Proof. Since F is tight on $\boldsymbol{p}$, the loops of F relative to $\boldsymbol{p}$ are singletons only. By Theorem 13, $\mathrm{SM}[F; \boldsymbol{p}]$ is equivalent to the conjunction of $\mathrm{SM}[\forall \boldsymbol{x}_i(G_i(\boldsymbol{x}_i) \rightarrow p_i(\boldsymbol{x}_i)); p_i]$ for each $p_i \in \boldsymbol{p}$, which, under the assumption F, is equivalent to

$$\forall u_i(u_i < p_i \rightarrow \exists \boldsymbol{x}_i(G_i^*(\boldsymbol{x}_i) \wedge \neg u_i(\boldsymbol{x}_i))). \tag{5.55}$$

Since F is tight on $\boldsymbol{p}$, it follows that $G_i(\boldsymbol{x}_i)$ is negative on p_i. By Lemma 44, $G_i^*(\boldsymbol{x}_i)$ is equivalent to $G_i(\boldsymbol{x}_i)$. Consequently, (5.55) is equivalent to

$$\forall u_i(u_i < p_i \rightarrow \exists \boldsymbol{x}_i(G_i(\boldsymbol{x}_i) \wedge \neg u_i(\boldsymbol{x}_i))). \tag{5.56}$$

It is sufficient to prove that, under the assumption

$$\forall \boldsymbol{x}_i(G_i(\boldsymbol{x}_i) \rightarrow p_i(\boldsymbol{x}_i)), \tag{5.57}$$

formula (5.56) is equivalent to $\forall \boldsymbol{x}_i(p_i(\boldsymbol{x}_i) \rightarrow G_i(\boldsymbol{x}_i))$.

From left to right: Assume (5.56) and, for the sake of contradiction, assume that there exists $\boldsymbol{x}$ such that

$$p_i(\boldsymbol{x}) \wedge \neg G_i(\boldsymbol{x}). \tag{5.58}$$

Take u_i such that

$$\forall \boldsymbol{x}_i(u_i(\boldsymbol{x}_i) \leftrightarrow G_i(\boldsymbol{x}_i)). \tag{5.59}$$

From (5.57), (5.58), and (5.59), we conclude $u_i < p_i$. From (5.56), $\exists \boldsymbol{x}_i(G_i(\boldsymbol{x}_i) \wedge \neg u_i(\boldsymbol{x}_i)))$ follows, which contradicts with (5.59).

From right to left: Assume $\forall \boldsymbol{x}_i(G_i(\boldsymbol{x}_i) \leftrightarrow p_i(\boldsymbol{x}_i))$. We further assume that $u_i < p_i$ for some u_i. From $u_i < p_i$, $\exists \boldsymbol{x}_i(p_i(\boldsymbol{x}_i) \wedge \neg u_i(\boldsymbol{x}_i))$ follows. Consequently, $\exists \boldsymbol{x}_i(G_i(\boldsymbol{x}_i) \wedge \neg u_i(\boldsymbol{x}_i))$ follows. ∎

Proof of Proposition 38

Lemma 58 *For any generalized quantifiers Q of the type $\langle n_1, \ldots, n_k \rangle$ and any formula F in the form*

$$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, G_k(\boldsymbol{x}_k, \boldsymbol{y}_k)),$$

(a) If Q is conjunctive w.r.t. M for some $M \subseteq \{1, \ldots, k\}$, then

$$F \rightarrow \bigwedge_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i, \boldsymbol{y}_i)$$

is logically valid;

(b) If Q is disjunctive w.r.t. M for some $M \subseteq \{1, \ldots, k\}$, then

$$F \rightarrow \bigvee_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i, \boldsymbol{y}_i)$$

is logically valid.

Proof. Consider any interpretation I and the lists of object names $\boldsymbol{\gamma}_i$ of the same length as $\boldsymbol{x}_i$ for $i \in M$.

(a): For sake of contradiction, we assume

$$I \models Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1, \boldsymbol{\gamma}_1), \ldots, G_k(\boldsymbol{x}_k, \boldsymbol{\gamma}_k)) \tag{5.60}$$

but

$$I \models \bigvee_{i \in M} \forall \boldsymbol{x}_i \neg G_i(\boldsymbol{x}_i, \boldsymbol{\gamma}_i). \tag{5.61}$$

From (5.61), there is i such that $i \in M$ and $(\boldsymbol{x}_i.G_i(\boldsymbol{x}_i, \boldsymbol{\gamma}_i))^I = \emptyset$. By assumption,

$$I \not\models Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1, \boldsymbol{\gamma}_1), \ldots, G_k(\boldsymbol{x}_k, \boldsymbol{\gamma}_k))$$

which contradicts (5.60).

(b): For sake of contradiction, we assume

$$I \models Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1, \boldsymbol{\gamma}_1), \ldots, G_k(\boldsymbol{x}_k, \boldsymbol{\gamma}_k)) \tag{5.62}$$

but

$$I \models \bigwedge_{i \in M} \forall \boldsymbol{x}_i \neg G_i(\boldsymbol{x}_i, \boldsymbol{\gamma}_i). \tag{5.63}$$

From (5.63), $(\boldsymbol{x}_i.G_i(\boldsymbol{x}_i, \boldsymbol{\gamma}_i))^I = \emptyset$ for every $i \in M$. By assumption,

$$I \not\models Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1, \boldsymbol{\gamma}_1), \ldots, G_k(\boldsymbol{x}_k, \boldsymbol{\gamma}_k))$$

which contradicts (5.62). ∎

Lemma 59 *For any GQ-formula F,*

$$F^*(\boldsymbol{e}_{\boldsymbol{c}}) \rightarrow in_{\boldsymbol{c}}(RV(F))$$

is logically valid, where $\boldsymbol{c}$ is any set of object constants containing $c(F)$.

Proof. By induction on F.

Case 1: F is an atomic formula $p(\boldsymbol{t})$ or $\boldsymbol{t}_1 = \boldsymbol{t}_2$, it is clear from definition of $\boldsymbol{e}_C$.

Case 2: If F is a GQ-formula of the form

$$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k)).$$

- If Q is conjunctive w.r.t. M for some $M \subseteq \{1, \ldots, k\}$, by Lemma 58(a), $F^*(\boldsymbol{e_c})$ implies

$$\bigwedge_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i) \wedge \bigwedge_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i)^*(\boldsymbol{e_c})$$

which in turn implies

$$\bigwedge_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i)^*(\boldsymbol{e_c}).$$

By I.H.

$$\bigwedge_{i \in M} \mathrm{in}_{\boldsymbol{c}}(\mathrm{RV}(G_i(\boldsymbol{x}_i)) \setminus \boldsymbol{x}^i)$$

follows. It follows that

$$\mathrm{in}_{\boldsymbol{c}}(\bigcup_{i \in M} (\mathrm{RV}(G_i(\boldsymbol{x}^i)) \setminus \boldsymbol{x}^i)),$$

which is exactly $\mathrm{in}_{\boldsymbol{c}}(\mathrm{RV}_M(F))$.

- If Q is disjunctive w.r.t. M for some $M \subseteq \{1, \ldots, k\}$, by Lemma 58(b), $F^*(\boldsymbol{e_c})$ implies

$$\bigvee_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i) \wedge \bigvee_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i)^*(\boldsymbol{e_c})$$

which in turn implies

$$\bigvee_{i \in M} \exists \boldsymbol{x}_i G_i(\boldsymbol{x}_i)^*(\boldsymbol{e_c}).$$

By I.H.

$$\bigvee_{i \in M} \mathrm{in}_{\boldsymbol{c}}(\mathrm{RV}(G_i(\boldsymbol{x}_i)) \setminus \boldsymbol{x}^i)$$

follows. It follows that

$$\mathrm{in}_{\boldsymbol{c}}(\bigcap_{i \in M} (\mathrm{RV}(G_i(\boldsymbol{x}^i)) \setminus \boldsymbol{x}^i)),$$

which is exactly $\mathrm{in}_{\boldsymbol{c}}(\mathrm{RV}_M(F))$.

- Otherwise, $\mathrm{RV}_M(F) = \emptyset$.

As a result,

$$\mathrm{in}_{\boldsymbol{c}}(\bigcup_{M \subseteq \{1,\ldots,k\}} \mathrm{RV}_M(F))$$

follows. ∎

A variable x in a GQ-formula F is semi-safe in F if every strictly positive occurrence of x that does not follow any generalized quantifier is contained in a subformula $Q_{\rightarrow}(G_1, G_2)$

such that x is in $\mathrm{RV}(G_1)$. It is clear that a GQ-formula is semi-safe iff all variables occurring in it are semi-safe. By $\mathrm{NS}(F)$ we will denote the set of the variables of F that are not semi-safe.

Lemma 60 *For any GQ-formula $F(\boldsymbol{x})$ such that all unsafe variables $NS(F)$ are free in $F(\boldsymbol{x})$, let $\boldsymbol{x}$ be all the free variables in $F(\boldsymbol{x})$ and $\boldsymbol{c}$ be a finite set of object constants containing $c(F)$*

$$F(\boldsymbol{x}) \wedge in_{\boldsymbol{c}}(NS(F)) \rightarrow F(\boldsymbol{x})^*(\boldsymbol{e_c}) \tag{5.64}$$

is logically valid.

Proof. By induction on F.

Case 1: $F(\boldsymbol{x})$ is an atomic formula. Since all variables in $F(\boldsymbol{x})$ are unsafe, it is trivial.

Case 2: F is a GQ-formula of the form

$$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k)).$$

- If Q is $Q_{\rightarrow}$, assume

$$G_1 \rightarrow G_2 \ , \tag{5.65}$$

$$\mathrm{in}_{\boldsymbol{c}}(\mathrm{NS}(F)). \tag{5.66}$$

Further, we assume that $G_1^*(\boldsymbol{e_c})$. From the assumption $G_1^*(\boldsymbol{e_c})$, by Lemma 40, we get G_1. Consequently, G_2 follows from (5.65).

Also, from the assumption $G_1^*(\boldsymbol{e_c})$, by Lemma 59,

$$\mathrm{in}_{\boldsymbol{c}}(\mathrm{RV}(G_1)). \tag{5.67}$$

Note that $\mathrm{NS}(G_2) \subseteq \mathrm{NS}(G_1 \rightarrow G_2) \cup \mathrm{RV}(G_1)$. Consequently, from (5.66) and (5.67),

$$\mathrm{in}_{\boldsymbol{c}}(\mathrm{NS}(G_2)) \tag{5.68}$$

follows. Then by I.H., $G_2^*(\boldsymbol{e_c})$ follows from G_2 and (5.68).

- Otherwise, since all unsafe variables are free, it follows that $\mathrm{NS}(F) = \bigcup_{1 \leq i \leq k} \mathrm{NS}(G_i)$ because the variables $\boldsymbol{x}_i$ must be safe in G_i. Assume $F \wedge \mathrm{in}_{\boldsymbol{c}}(\mathrm{NS}(F))$. By I.H.

$$G_i(\boldsymbol{x}_i) \wedge \mathrm{in}_{\boldsymbol{c}}(\mathrm{NS}(G_i)) \rightarrow G_i(\boldsymbol{x}_i)^*(\boldsymbol{e}_{\boldsymbol{c}})$$

is logically valid. By Lemma 40,

$$G_i(\boldsymbol{x}_i)^*(\boldsymbol{e}_{\boldsymbol{c}}) \rightarrow G_i(\boldsymbol{x}_i)$$

is logically valid. So under the assumption $\mathrm{in}_{\boldsymbol{c}}(\mathrm{NS}(F))$, both

$$G_i(\boldsymbol{x}_i)^*(\boldsymbol{e}_{\boldsymbol{c}}) \leftrightarrow G_i(\boldsymbol{x}_i)$$

for every $1 \leq i \leq k$ and

$$F(\boldsymbol{x}) \leftrightarrow F(\boldsymbol{x})^*(\boldsymbol{e}_{\boldsymbol{c}})$$

are logically valid.

∎

Proposition 38 *For semi-safe sentence F, $SM[F] \models SPP_{c(F)}$.*

Proof. We will prove a more general claim that, for any semi-safe GQ-sentence F, formula $\mathrm{SM}[F]$ entails $\mathrm{SPP}_{c(F)}$. Clearly, any safe sentence is a special case of such sentence.

We will show that $F \wedge \neg\mathrm{SPP}_{c(F)}$ entails

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

Assume F and $\neg\mathrm{SPP}_{c(F)}$. It is sufficient to show that

$$\boldsymbol{e}_{c(F)} < \boldsymbol{p} \wedge F^*(\boldsymbol{e}_{c(F)}) \tag{5.69}$$

holds.

Note that Formula

$$\bigwedge_{p \in \boldsymbol{p}} \Big(\forall \boldsymbol{x} \Big((p(\boldsymbol{x}) \wedge \mathrm{in}_{c(F)}(\boldsymbol{x})) \rightarrow p(\boldsymbol{x}) \Big) \Big)$$

is logically valid. Assume for the sake of contradiction,

$$\bigwedge_{p\in\boldsymbol{p}} \forall \boldsymbol{x}\Big(p(\boldsymbol{x}) \rightarrow (p(\boldsymbol{x}) \wedge \mathrm{in}_{c(F)}(\boldsymbol{x}))\Big). \tag{5.70}$$

Formula (5.70) entails $\mathrm{SPP}_{c(F)}$, which contradicts the assumption $\neg\mathrm{SPP}_{c(F)}$. Consequently, we conclude $\boldsymbol{e}_{c(F)} < \boldsymbol{p}$. By Lemma 60, the second conjunctive term of (5.69) follows from the assumption about F. ∎

Proof of Proposition 39

Lemma 61 *For any GQ-formula F of the form $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{p},\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{p},\boldsymbol{x}_k))$ that contain no free variables and any finite set $\boldsymbol{c}$ of object constants containing $\sigma(F)$,*

$$Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1^*(\boldsymbol{q},\boldsymbol{x}_1)\wedge in_{\boldsymbol{c}}(\boldsymbol{x}_1),\ldots,G_k^*(\boldsymbol{q},\boldsymbol{x}_k)\wedge in_{\boldsymbol{c}}(\boldsymbol{x}_k)) \tag{5.71}$$

is equivalent to

$$\bigwedge_{(R_1,\ldots,R_k)\in\overline{\boldsymbol{C}}_{\boldsymbol{c}}(E)} \Big(\bigwedge_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in R_i}} G_i^*(\boldsymbol{q},\boldsymbol{d}_i) \rightarrow \bigvee_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in \boldsymbol{O}_{\boldsymbol{c}}^i\setminus R_i}} G_i^*(\boldsymbol{q},\boldsymbol{d}_i)\Big). \tag{5.72}$$

Proof. Consider any interpretation I. We identify I as $\langle I^f, X\rangle$ where I^f is the interpretation of functions and X is a set of ground atoms forms from predicate constants in $\boldsymbol{p}$ and elements in the universe. Let Y be a subset of X. It is sufficient to prove that when F contains no free variable, $\langle I^f, X\cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}\rangle \models (5.71)$ iff $\langle I^f, X\cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}\rangle \models (5.72)$.

From left to right: Assume $\langle I^f, X\cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}\rangle \models (5.71)$. let R_i' be the set of all lists $\boldsymbol{d}_i$ of object constants such that

$$\langle I^f, X\cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}\rangle \models G_i^*(\boldsymbol{q},\boldsymbol{d}_i)\wedge \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{d}_i).$$

Clearly, $\boldsymbol{d}_i \in \boldsymbol{O}_{\boldsymbol{c}}^i$. From (5.71), $Q^{\boldsymbol{c}}(R_1',\ldots,R_k') = \boldsymbol{t}$ follows. Consequently, $(R_1',\ldots,R_k')$ is not in $\overline{\boldsymbol{C}}_{\boldsymbol{c}}(F)$. Consider any $(R_1,\ldots,R_k)$ in $\overline{\boldsymbol{C}}_{\boldsymbol{c}}(F)$ such that

$$\langle I^f, X\cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}\rangle \models \bigwedge_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in R_i}} G_i^*(\boldsymbol{q},\boldsymbol{d}_i).$$

Clearly, each R_i is a subset of R_i'. Furthermore, there is an index j such $1\le j\le k$ and R_j is a strict subset of R_j' since $(R_1',\ldots,R_k')$ is not in $\overline{\boldsymbol{C}}_{\boldsymbol{c}}(F)$. Consequently,

$$\langle I^f, X\cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}\rangle \models \bigvee_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in \boldsymbol{O}_{\boldsymbol{c}}^i\setminus R_i}} G_i^*(\boldsymbol{q},\boldsymbol{d}_i).$$

From right to left: Assume (5.72). Again let R'_i be the set of all lists $\boldsymbol{d}_i$ of object constants such that $\langle I^f, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \models G^*_i(\boldsymbol{q}, \boldsymbol{d}_i)$. Clearly, $(R'_1, \ldots, R'_k)$ is not in $\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(F)$. Since

$$\langle I^f, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}\rangle \not\models \bigwedge_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in R'_i}} G^*_i(\boldsymbol{q}, \boldsymbol{d}_i) \rightarrow \bigvee_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R'_i}} G^*_i(\boldsymbol{q}, \boldsymbol{d}_i).$$

Consequently, $Q^{\boldsymbol{c}}(R'_1, \ldots, R'_k) = \boldsymbol{t}$. As a result, we conclude (5.71). ∎

The following corollary immediately follows.

Corollary 14 *For any GQ-formula F of the form $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{p}, \boldsymbol{x}_1), \ldots, G_k(\boldsymbol{p}, \boldsymbol{x}_k))$ that contain no free variables and any finite set $\boldsymbol{c}$ of object constants containing $\sigma(F)$, let X and Y Herbrand interpretations of $\boldsymbol{c}$ such that $Y \subseteq X$. The following holds*

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G^*_1(\boldsymbol{q}, \boldsymbol{x}_1), \ldots, G^*_k(\boldsymbol{q}, \boldsymbol{x}_k))$$

iff

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \bigwedge_{(R_1,\ldots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E)} \Big(\bigwedge_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in R_i}} G^*_i(\boldsymbol{q}, \boldsymbol{d}_i) \rightarrow \bigvee_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} G^*_i(\boldsymbol{q}, \boldsymbol{d}_i)\Big).$$

Proposition 39 *For any GQ sentence F, any signature σ such that $\sigma(F) \subseteq \sigma$ and any Herbrand interpretation X of σ, let $\boldsymbol{c}$ be the set of all object constants in σ, if $\boldsymbol{c}$ is finite, then $X \models SM[F]$ iff $X \models SM[Ground_{\boldsymbol{c}}[F]]$.*

Proof. It is sufficient to prove that for any subset Y of X, $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models F^*(\boldsymbol{q})$ iff $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models (\mathrm{Ground}_{\boldsymbol{c}}[F])^*(\boldsymbol{q})$. This is proven by induction.

- F is an atomic formula, clear since $\mathrm{Ground}_{\boldsymbol{c}}[F]$ is the same as F.
- F is a GQ-formula of the form $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{p}, \boldsymbol{x}_1), \ldots, G_k(\boldsymbol{p}, \boldsymbol{x}_k))$. By Corollary 14, $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G^*_1(\boldsymbol{q}, \boldsymbol{x}_1), \ldots, G^*_k(\boldsymbol{q}, \boldsymbol{x}_k))$ iff

  $$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \bigwedge_{(R_1,\ldots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E)} \Big(\bigwedge_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in R_i}} G^*_i(\boldsymbol{q}, \boldsymbol{d}_i) \rightarrow \bigvee_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} G^*_i(\boldsymbol{q}, \boldsymbol{d}_i)\Big).$$

  By I.H. the later is the same as saying

  $$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models \bigwedge_{(R_1,\ldots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E)} \Big(\bigwedge_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in R_i}} \mathrm{Ground}_{\boldsymbol{c}}[G^*_i(\boldsymbol{q}, \boldsymbol{d}_i)] \rightarrow \bigvee_{\substack{1\leq i\leq k\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} \mathrm{Ground}_{\boldsymbol{c}}[G^*_i(\boldsymbol{q}, \boldsymbol{d}_i)]\Big).$$

∎

Proof of Proposition 40

Lemma 62 *Let F be a generalized quantified formula of the form $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))$. Formulas*

$$\bigwedge_{(R_1,\ldots,R_k)\in\overline{\mathcal{C}}_{\boldsymbol{c}}(E)} \Big(\bigwedge_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in R_i}} G_i(\boldsymbol{d}_i) \rightarrow \bigvee_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} G_i(\boldsymbol{d}_i)\Big) \tag{5.73}$$

and

$$\bigwedge_{\langle B,T\rangle \text{ is an MLPS of } \overline{\mathcal{C}}_{\boldsymbol{c}}(E)} \Big(\bigwedge_{\substack{R_i\in B\\ \boldsymbol{d}_i\in R_i}} G_i(\boldsymbol{d}_i) \rightarrow \bigvee_{\substack{R_i\in T\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} G_i(\boldsymbol{d}_i)\Big) \tag{5.74}$$

are strongly equivalent.

Proof. It is sufficient to show that for any two subsets B, T of $\boldsymbol{O}^i_{\boldsymbol{c}}$ such that B is a subset of T, formula

$$\bigwedge_{B\subseteq S\subseteq T} \Big(\bigwedge_{\substack{R_i\in S\\ \boldsymbol{d}_i\in R_i}} G_i(\boldsymbol{d}_i) \rightarrow \bigvee_{\substack{R_i\in S\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} G_i(\boldsymbol{d}_i)\Big) \tag{5.75}$$

is equivalent to

$$\bigwedge_{\substack{R_i\in B\\ \boldsymbol{d}_i\in R_i}} G_i(\boldsymbol{d}_i) \rightarrow \bigvee_{\substack{R_i\in T\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} G_i(\boldsymbol{d}_i)$$

in the Logic of Here-and-There.

From right to left: Clear from the facts that $B \subseteq S \subseteq T$.

From left to right: Assume (5.75). Consider several cases, each of which corresponds to S such that $B \subseteq S \subseteq T$. The assumption characterizing each case is that each $R_i \in S$ contains all $\boldsymbol{d}_i$ such that $G_i(\boldsymbol{d}_i)$ holds. Consequently, we have

$$\bigwedge_{\substack{R_i\in S\\ \boldsymbol{d}_i\in R_i}} G_i(\boldsymbol{d}_i) \wedge \bigwedge_{\substack{R_i\in T, R'_i\in S\\ \boldsymbol{d}_i\in R_i\setminus R'_i}} \neg G_i(\boldsymbol{d}_i). \tag{5.76}$$

It follows from (5.75) and (5.76) that

$$\bigvee_{\substack{R_i\in S\\ \boldsymbol{d}_i\in \boldsymbol{O}^i_{\boldsymbol{c}}\setminus R_i}} G_i(\boldsymbol{d}_i).$$

From this and the second conjunctive term of (5.76), we conclude that

$$\bigvee_{\substack{R_i \in T \\ \boldsymbol{d}_i \in \boldsymbol{O}^i_{\boldsymbol{c}} \setminus R_i}} G_i(\boldsymbol{d}_i).$$

∎

Lemma 63 *For any GQ-formula F and list of predicates $\boldsymbol{p}$, if $x \in RV(F)$, then*

$$F \to in_{\boldsymbol{c}}(x)$$

follows from $SPP_{\boldsymbol{c}}$.

Proof. By induction on F.

Case 1: F is an atomic formula $p(\boldsymbol{t})$ or $\boldsymbol{t}_1 = \boldsymbol{t}_2$, it is clear from $\mathrm{SPP}_{\boldsymbol{c}}$.

Case 2: F is a GQ-formula $Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k))$.

- F is $G_1 \wedge G_2$. If $x \in \mathrm{RV}(G \wedge H)$, then $x \in \mathrm{RV}(G)$ or $x \in \mathrm{RV}(H)$. In either case, $\mathrm{in}_{\boldsymbol{c}}(x)$ follows from F and I.H.
- F is $G_1 \vee G_2$. If $x \in \mathrm{RV}(G \vee H)$, then $x \in \mathrm{RV}(G)$ and $x \in \mathrm{RV}(H)$. Thus $\mathrm{in}_{\boldsymbol{c}}(x)$ follows from I.H.
- F is $\forall \boldsymbol{x} G(\boldsymbol{x})$ or $\exists \boldsymbol{x} G(\boldsymbol{x})$. $\mathrm{RV}(F) = \mathrm{RV}(G) \setminus \boldsymbol{x}$. So $\mathrm{in}_{\boldsymbol{c}}(\mathrm{RV}(F))$ follows from $G(\boldsymbol{x})$ and I.H.
- Otherwise, clear since $\mathrm{RV}(F) = \emptyset$.

∎

Lemma 64 *Let F be a GQ-formula.*

- *If a variable x is positively weakly restricted in F, let $F^*(\boldsymbol{q})^A_\perp$ be the formula obtained from $F^*(\boldsymbol{q})$ by replacing every atomic formula A or $A^*(\boldsymbol{q})$ in it such that $x \in RV(A)$ by $\perp$ and apply the transformation, $F^*(\boldsymbol{q})^A_\perp$ is equivalent to $\top$;*

- *If a variable x is negatively weakly restricted in F, let $F^*(\boldsymbol{q})^A_\perp$ be the formula obtained from $F^*(\boldsymbol{q})$ by replacing every atomic formula A or $A^*(\boldsymbol{q})$ in it such that $x \in RV(A)$ by $\perp$ and apply the transformation, $F^*(\boldsymbol{q})^A_\perp$ is equivalent to $\perp$.*

Proof. We will only prove the second bullet. Proof of the first bullet is similar. By induction on F.

Case 1: F is an atomic formula $p(\boldsymbol{t})$. $F^*(\boldsymbol{q})$ is $q(\boldsymbol{t})$. $F^*(\boldsymbol{q})^A_\perp$ is $\perp$.

Case 2: F is an atomic formula $\boldsymbol{t}_1 = \boldsymbol{t}_2$. $F^*(\boldsymbol{q})$ is F. Clear from the definition of negatively weakly restricted.

Case 3: F is a GQ-formula $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))$.

- F is $G_1 \wedge G_2$. $F^*(\boldsymbol{q})$ is $G_1^*(\boldsymbol{q}) \wedge G_2^*(\boldsymbol{q})$. x is negatively weakly restricted in G_1 or G_2. So $F^*(\boldsymbol{q})^A_\perp$ is $\perp$ from I.H.
- F is $G_1 \vee G_2$. $F^*(\boldsymbol{q})$ is $G_1^*(\boldsymbol{q}) \vee G_2^*(\boldsymbol{q})$. x is negatively weakly restricted in both G_1 and G_2. So $F^*(\boldsymbol{q})^A_\perp$ is $\perp$ from I.H.
- F is $G_1 \rightarrow G_2$. $F^*(\boldsymbol{q})$ is $(G_1^*(\boldsymbol{q}) \rightarrow G_2^*(\boldsymbol{q}))\wedge(G_1 \rightarrow G_2)$. Since x is negatively weakly restricted in $G_1 \rightarrow G_2$, it is clear that $(G_1 \rightarrow G_2)^A_\perp$ is $\perp$. As a result, x is negatively weakly restricted in G_2 and positively weakly restricted in G_1. By I.H., $G_1^*(\boldsymbol{q})^A_\perp$ is $\top$ and $G_2^*(\boldsymbol{q})^A_\perp$ is $\perp$. So $F^*(\boldsymbol{q})^A_\perp$ is equivalent to $\perp$.
- F is $\forall \boldsymbol{x} G(\boldsymbol{x})$ or $\exists \boldsymbol{x} G(\boldsymbol{x})$. Clear from I.H.
- Otherwise, there is no way for $F^*(\boldsymbol{q})^A_\perp$ to be $\perp$.

∎

Lemma 65 *For any GQ-formula F where all bound variables are disjoint,*

(a) if x is positively weakly restricted in F, then

$$\boldsymbol{q} \leq \boldsymbol{p} \wedge \neg in_{\boldsymbol{c}}(x) \rightarrow (F^*(\boldsymbol{q}, x) \leftrightarrow \top)$$

is derivable from $SPP_{\boldsymbol{c}}$.

(b) if x is negatively weakly restricted in F, then

$$\boldsymbol{q} \leq \boldsymbol{p} \wedge \neg in_{\boldsymbol{c}}(x) \rightarrow (F^*(\boldsymbol{q}, x) \leftrightarrow \bot)$$

is derivable from $SPP_{\boldsymbol{c}}$.

Proof. We will only prove it for (a), proof of (b) is similar. Since $\boldsymbol{q} \leq \boldsymbol{p}$, by Lemma 63, for any atomic formula A such that $x \in \mathrm{RV}(A)$, $A \leftrightarrow \bot$ follows from $\mathrm{SPP}_{\boldsymbol{c}}$. Also, it follows from Lemma 40 that $A^*(\boldsymbol{q}) \leftrightarrow \bot$ is derivable from $\mathrm{SPP}_{\boldsymbol{c}}$. So $F^*(\boldsymbol{q})$ is equivalent to $F^*(\boldsymbol{q})^A_\bot$, where $F^*(\boldsymbol{q})^A_\bot$ is obtained from $F^*(\boldsymbol{q})$ by replacing every atomic formula A or $A^*(\boldsymbol{q})$ by $\bot$. In view of Lemma 64, since x is positively weakly restricted in F, $F^*(\boldsymbol{q})^A_\bot \leftrightarrow \top$. ∎

Proposition 40 *For any safe GQ sentence F and any nonempty finite set $\boldsymbol{c}$ of object constants containing $c(F)$, $SM[F]$ is equivalent to $SM[Ground_{\boldsymbol{c}}[F]]$.*

Proof of Proposition 40 immediately follows from the following lemma.

Lemma 66 *For any safe GQ-sentence F, and for any nonempty finite set $\boldsymbol{c}$ of object constants containing $c(F)$,*

$$\boldsymbol{q} \leq \boldsymbol{p} \rightarrow \big(F^*(\boldsymbol{q}) \leftrightarrow (Ground_{\boldsymbol{c}}[F])^*(\boldsymbol{q})\big)$$

is logically valid.

Proof. By induction on F.

Case 1: F is an atomic formula or 0-place connectives. Trivial since $\mathrm{Ground}_{\boldsymbol{c}}[F]$ is itself.

Case 2: F is a GQ-formula $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k))$.

- F is $\forall x F(\boldsymbol{p}, x)$. $\forall x F^*(\boldsymbol{q}, x)$ can be rewritten as

$$\forall x(\mathrm{in}_c(x) \rightarrow F^*(\boldsymbol{q}, x)) \wedge \forall x(\neg \mathrm{in}_c(x) \rightarrow F^*(\boldsymbol{q}, x)),$$

and consequently as

$$\bigwedge_{c\in\boldsymbol{c}} F^*(\boldsymbol{q},c)\wedge\forall x(\neg\mathrm{in}_c(x)\to F^*(\boldsymbol{q},x)). \tag{5.77}$$

Note that x is quantified by $\forall$ which has positive occurrence. Consider the maximal positive subformula $G(x)$ of $F(x)$ such that x is positively weakly restricted in $G(x)$. By Lemma 65 (a), for each of these subformulas, the implication

$$\boldsymbol{q}\leq\boldsymbol{p}\wedge\neg\mathrm{in}_{\boldsymbol{c}}(x)\to(G^*(\boldsymbol{q},x)\leftrightarrow\top)$$

is derivable from $\mathrm{SPP}_{\boldsymbol{c}}$. Also, when $\boldsymbol{q}$ is $\boldsymbol{p}$,

$$\neg\mathrm{in}_{\boldsymbol{c}}(x)\to(G(x)\leftrightarrow\top)$$

follows from $\mathrm{SPP}_{\boldsymbol{c}}$. As a result, under the assumption $\mathrm{SPP}_{\boldsymbol{c}}$, (5.77) can be equivalently rewritten as

$$\bigwedge_{c\in\boldsymbol{c}} F^*(\boldsymbol{q},c)\wedge\forall x(\neg\mathrm{in}_c(x)\to S_1^*(\boldsymbol{q})). \tag{5.78}$$

where $S_1^*(\boldsymbol{q})$ is the formula obtained from $F^*(\boldsymbol{q},x)$ by replacing each of these $G^*(\boldsymbol{q},x)$ with $\top$ and then replacing G with $\top$. Now consider the maximal negative subformula $H(x)$ of S_1 such that x is negatively weakly restricted in $H(x)$. By Lemma 65 (b), for each of these subformulas, the implication

$$\boldsymbol{q}\leq\boldsymbol{p}\wedge\neg\mathrm{in}_{\boldsymbol{c}}(x)\to(H^*(\boldsymbol{q},x)\leftrightarrow\bot)$$

is derivable from $\mathrm{SPP}_{\boldsymbol{c}}$. Also, when $\boldsymbol{q}$ is $\boldsymbol{p}$,

$$\neg\mathrm{in}_{\boldsymbol{c}}(x)\to(H(x)\leftrightarrow\bot)$$

follows from $\mathrm{SPP}_{\boldsymbol{c}}$. As a result, under the assumption $\mathrm{SPP}_{\boldsymbol{c}}$, (5.78) can be equivalently rewritten as

$$\bigwedge_{c\in\boldsymbol{c}} F^*(\boldsymbol{q},c)\wedge\forall x(\neg\mathrm{in}_c(x)\to S_2^*(\boldsymbol{q})) \tag{5.79}$$

where $S_2^*(\boldsymbol{q})$ is the formula obtained from $S_1^*(\boldsymbol{q})$ by replacing each of these $H^*(\boldsymbol{q},x)$ with $\bot$ and then replacing H with $\bot$.

We claim that x does not occur in $S_2^*(\boldsymbol{q})$. Indeed, consider any occurrence of x in S_1. Since $\forall xF(x)$ is safe, in view of the fact that if a subformula G is positive (negative) in

F, then $G^*(\boldsymbol{q})$ is also positive (negative) in $F^*(\boldsymbol{q})$, by the construction of $S_1^*(\boldsymbol{q})$, that occurrence is in a negative subformula $H'(x)$ of S_1, which is obtained from a negative subformula $H(x)$ of $F(x)$ in which x is negatively weakly restricted, by replacing some of its subformulas by $\top$. Clearly, x is negatively weakly restricted in $H'(x)$ as well. By the construction of $S_2^*(\boldsymbol{q})$, a formula that contains $H'(x)$ or $H'^*(\boldsymbol{q}, x)$ is replaced by $\bot$.

It follows that $S_2^*(\boldsymbol{q})$ can be obtained from $F^*(\boldsymbol{q}, c)$ in the same way as it was obtained from $F^*(\boldsymbol{q}, x)$, that is by replacing some subformulas that are positive in $F^*(\boldsymbol{q}, c)$ with $\top$ and then replacing some subformulas that are negative in the resulting formula with $\bot$. Consequently, $F^*(\boldsymbol{q}, c) \to S_2^*(\boldsymbol{q})$ follows and so is $F^*(\boldsymbol{q}, c) \to \forall x(\neg \mathrm{in}_{\boldsymbol{c}}(x) \to S_2^*(\boldsymbol{q}))$. Thus the second conjunctive term of (5.79) can be dropped. So (5.79) is equivalent to

$$\bigwedge_{c \in \boldsymbol{c}} F^*(\boldsymbol{q}, c). \tag{5.80}$$

(5.80) can be viewed as

$$\bigwedge_{d \in \boldsymbol{c}} \Big(\bigwedge_{d \in \emptyset} F^*(\boldsymbol{q}, d) \to \bigvee_{d \in \boldsymbol{c} \setminus (\boldsymbol{c} \setminus \{d\})} F^*(\boldsymbol{q}, d) \Big).$$

By Lemma 62, the later is equivalent to

$$\bigwedge_{S \subset \boldsymbol{c}} \Big(\bigwedge_{d \in S} F^*(\boldsymbol{q}, d) \to \bigvee_{d \in \boldsymbol{c} \setminus S} F^*(\boldsymbol{q}, d) \Big). \tag{5.81}$$

By I.H. (5.81) is equivalent to

$$\bigwedge_{S \subset \boldsymbol{c}} \Big(\bigwedge_{d \in S} \mathrm{Ground}_{\boldsymbol{c}}[F^*(\boldsymbol{q}, d)] \to \bigvee_{d \in \boldsymbol{c} \setminus S} \mathrm{Ground}_{\boldsymbol{c}}[F^*(\boldsymbol{q}, d)] \Big)$$

which is exactly $\mathrm{Ground}_{\boldsymbol{c}}[\forall x F(\boldsymbol{p}, x)]$.

- F is $\exists x F(\boldsymbol{p}, x)$. Similar to the first case.

- Otherwise, every variable in $\boldsymbol{x}^i$ belongs to $\mathrm{RV}(G_i)$. By Lemma 63

  $$G_i(\boldsymbol{p}, \boldsymbol{x}_i) \to \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{x}_i)$$

  is logically valid. So F is equivalent to

  $$Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{p}, \boldsymbol{x}_1) \wedge \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{x}_k)), \ldots, G_k(\boldsymbol{p}, \boldsymbol{x}_k) \wedge \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{x}_k))).$$

By Lemma 61,

$$Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](G_1^*(\boldsymbol{q},\boldsymbol{x}_1)\wedge \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{x}_1),\dots,G_k^*(\boldsymbol{q},\boldsymbol{x}_k)\wedge \mathrm{in}_{\boldsymbol{c}}(\boldsymbol{x}_k))$$

is equivalent to

$$\bigwedge_{(R_1,\dots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E)}\Big(\bigwedge_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in R_i}} G_i^*(\boldsymbol{q},\boldsymbol{d}_i)\rightarrow \bigvee_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in \boldsymbol{O}_{\boldsymbol{c}}^i\setminus R_i}} G_i^*(\boldsymbol{q},\boldsymbol{d}_i)\Big).$$

By I.H. the later is equivalent to

$$\bigwedge_{(R_1,\dots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(E)}\Big(\bigwedge_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in R_i}} \mathrm{Ground}_{\boldsymbol{c}}[G_i^*(\boldsymbol{q},\boldsymbol{d}_i)]\rightarrow \bigvee_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in \boldsymbol{O}_{\boldsymbol{c}}^i\setminus R_i}} \mathrm{Ground}_{\boldsymbol{c}}[G_i^*(\boldsymbol{q},\boldsymbol{d}_i)]\Big).$$

∎

Proof of Theorem 15

Proof Between (a) and (b) of Theorem 15:

Let $\boldsymbol{c}$ be the set of all object constants in $\sigma(F)$. By Proposition 39, $X\models \mathrm{SM}[F]$ iff $X\models \mathrm{SM}[\mathrm{Ground}_{\boldsymbol{c}}[F]]$. According to Theorem 1^f in (Lee & Meng, 2011), $X\models \mathrm{SM}[\mathrm{Ground}_{\boldsymbol{c}}[F]]$ iff X satisfies $\mathrm{LF}_{\mathrm{Ground}_{\boldsymbol{c}}[F]}(Y)$ for every nonempty finite set Y of atoms of $\sigma(F)$. As a result, it is sufficient to prove that when F contains no free variables, for any nonempty finite set Y of atoms of $\sigma(F)$, $X\models \mathrm{NES}_F(Y)$ iff $X\models \mathrm{NES}_{\mathrm{Ground}_{\boldsymbol{c}}[F]}(Y)$. We will prove it by induction.

Case 1: F is an atomic formula. Clear since $\mathrm{Ground}_{\boldsymbol{c}}[F]$ is the same as F.

Case 2: F is of the form $Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\dots,G_k(\boldsymbol{x}_k))$. $\mathrm{Ground}_{\boldsymbol{c}}[F]$ is

$$\bigwedge_{(R_1,\dots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(F)}\Big(\bigwedge_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in R_i}} \mathrm{Ground}_{\boldsymbol{c}}[G_i(\boldsymbol{d}_i)]\rightarrow \bigvee_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in \boldsymbol{O}_{\boldsymbol{c}}^i\setminus R_i}} \mathrm{Ground}_{\boldsymbol{c}}[G_i(\boldsymbol{d}_i)]\Big).$$

$\mathrm{NES}_{\mathrm{Ground}_{\boldsymbol{c}}[F]}(Y)$ is the conjunction of $\mathrm{Ground}_{\boldsymbol{c}}[F]$ and

$$\bigwedge_{(R_1,\dots,R_k)\in\overline{\boldsymbol{\mathcal{C}}}_{\boldsymbol{c}}(F)}\Big(\bigwedge_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in R_i}} \mathrm{NES}_{\mathrm{Ground}_{\boldsymbol{c}}[G_i(\boldsymbol{d}_i)]}(Y)\rightarrow \bigvee_{\substack{1\le i\le k\\ \boldsymbol{d}_i\in \boldsymbol{O}_{\boldsymbol{c}}^i\setminus R_i}} \mathrm{NES}_{\mathrm{Ground}_{\boldsymbol{c}}[G_i(\boldsymbol{d}_i)]}(Y)\Big). \tag{5.82}$$

On the other hand, $\mathrm{NES}_F(Y)$ is the conjunction of F and

$$Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Y),\dots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Y)) \tag{5.83}$$

It follows from the proof of Proposition 39 that $X \models \mathrm{Ground}_{\boldsymbol{c}}[F]$ iff $X \models F$. By I.H. $X \models \mathrm{NES}_{G_i(\boldsymbol{d}_i)}(Y)$ iff $X \models \mathrm{NES}_{\mathrm{Ground}_{\boldsymbol{c}}[G_i(\boldsymbol{d}_i)]}(Y)$ for each $1 \leq i \leq k$ and any $\boldsymbol{d}_i \in c(F)^{|\boldsymbol{x}_i|}$. As a result, $X \models (5.83)$ iff

$$X \models Q[\boldsymbol{x}_1]\dots[\boldsymbol{x}_k](\mathrm{NES}_{\mathrm{Ground}_{\boldsymbol{c}}[G_1(\boldsymbol{x}_1)]}(Y),\dots,\mathrm{NES}_{\mathrm{Ground}_{\boldsymbol{c}}[G_k(\boldsymbol{x}_k)]}(Y))$$

which, by Corollary 14, iff $X \models (5.82)$.

Proof Between (ba) and (c) of Theorem 15: By Theorem 2 in (Lee & Meng, 2011), (b) is equivalent to (b′)

(b′) for every nonempty finite set Y of ground atoms of $\sigma(F)$, X satisfies $\mathrm{LF}_F(Y)$;

It is also clear that (c) is equivalent to (c′)

(c′) for every finite ground loop Y of F, X satisfies $\mathrm{LF}_F(Y)$.

We will show that (b′) is equivalent to (c′).

Lemma 67 *For any GQ-formula F and any set Y of atoms, $NES_F(Y)$ implies F.*

Proof. By induction on F. ∎

Lemma 68 *For any GQ-formula F and any set Y of ground atoms, let S_F be the set of all atoms that has strictly positive occurrences in F. If for every substitution θ from variables in S_F to ground terms in Y, $S_F\theta \cap Y = \emptyset$ then $NES_F(Y)$ is equivalent to F.*

Proof. By induction on F.

Case 1: F is $p_i(\boldsymbol{t})$. $\mathrm{NES}_F(Y)$ is

$$p_i(\boldsymbol{t}) \wedge \bigwedge_{p_i(\boldsymbol{d})\in Y} \boldsymbol{t} \neq \boldsymbol{d}. \tag{5.84}$$

Since every substitution θ from variables in S_F to ground terms in Y, $S_F\theta \cap Y = \emptyset$. It follows that $\boldsymbol{t} \neq \boldsymbol{d}$ is logically valid for every $p_i(\boldsymbol{d}) \in Y$. As a result, (5.84) is equivalent to $p_i(\boldsymbol{t})$.

Case 2: F is an atomic formula that does not contain members of $\boldsymbol{p}$. It is clear since $\mathrm{NES}_F(Y) = F$.

Case 3: F is of the form $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))$. $\mathrm{NES}_F(Y)$ is

$$\begin{aligned} &Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Y),\ldots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Y)) \\ &\quad \wedge Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k)). \end{aligned} \tag{5.85}$$

Without loss of generality, we partition the set of all argument positions of Q into three sets: the set of monotone argument positions Mon, the set of antimonotone argument positions Anti and the rest of argument positions Mixed.

(a) Consider any $i \in \mathrm{Mon} \cup \mathrm{Mixed}$. Note that $S_{G_i} \subseteq S_F$. Since for every substitution θ from variables in S_F to ground terms in Y, $S_F\theta \cap Y = \emptyset$, it is immediate that $S_{G_i}\theta \cap Y = \emptyset$. By I.H. $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$ is equivalent to $G_i(\boldsymbol{x}_i)$.

(b) Consider any $i \in \mathrm{Anti}$. By Lemma 67, $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$ implies $G_i(\boldsymbol{x}_i)$.

From (a) and (b), by Lemma 43(b), $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))$ implies

$$Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Y),\ldots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Y)).$$

As a result, (5.85) is equivalent to F. ∎

Lemma 69 *For any GQ-formula F, set Y of ground atoms, and subset Z of Y, let S_F^+ be the set of all atoms that has semi-positive non-$\boldsymbol{p}$-negated occurrences in F and S_F^- be the set of all atoms that has semi-negative non-$\boldsymbol{p}$-negated occurrences in F.*

(a) If for every substitution θ from variables in S_F^+ to ground terms in Y, $S_F^+\theta \cap (Y \setminus Z) = \emptyset$ then $NES_F(Z)$ implies $NES_F(Y)$;

(b) If for every substitution θ from variables in S_F^- to ground terms in Y, $S_F^-\theta \cap (Y \setminus Z) = \emptyset$ then $NES_F(Y)$ implies $NES_F(Z)$.

Proof. Both parts are proved simultaneously by induction on F.

Case 1: F is $p_i(\boldsymbol{t})$.

Part (a): $\mathrm{NES}_F(Z)$ is

$$p_i(\boldsymbol{t}) \wedge \bigwedge_{p_i(\boldsymbol{d}) \in Z} \boldsymbol{t} \neq \boldsymbol{d}. \tag{5.86}$$

Since every substitution θ from variables in S_F^+ to ground terms in Y, $S_F^+\theta \cap (Y \setminus Z) = \emptyset$. It follows that $\boldsymbol{t} \neq \boldsymbol{d}$ is logically valid for every $p_i(\boldsymbol{d}) \in Y \setminus Z$. As a result, (5.86) is equivalent to

$$p_i(\boldsymbol{t}) \wedge \bigwedge_{p_i(\boldsymbol{d}) \in Y} \boldsymbol{t} \neq \boldsymbol{d},$$

which is exactly $\mathrm{NES}_F(Y)$.

Part (b): Clear from $Z \subseteq Y$.

Case 2: F is an atomic formula that does not contain members of $\boldsymbol{p}$. It is clear since $\mathrm{NES}_F(Y) = \mathrm{NES}_F(Z) = F$.

Case 3: F is of the form $Q[\boldsymbol{x}_1] \ldots [\boldsymbol{x}_k](G_1(\boldsymbol{x}_1), \ldots, G_k(\boldsymbol{x}_k))$. Without loss of generality, we partition the set of all argument positions of Q into three sets: the set of monotone argument positions Mon, the set of antimonotone argument positions Anti and the rest of argument positions Mixed.

Part (a): If F is negated on $\boldsymbol{p}$, then by Lemma 68, $\mathrm{NES}_F(Y)$ and $\mathrm{NES}_F(Z)$ are both equivalent to F. Otherwise,

- Consider any $i \in \mathrm{Mon} \cup \mathrm{Mixed}$. Note that $S_{G_i}^+ \subseteq S_F^+$. It follows that $S_{G_i}^+\theta \cap (Y \setminus Z) = \emptyset$. By I.H.(a) $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$ implies $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$.
- Consider any $i \in \mathrm{Anti} \cup \mathrm{Mixed}$. Note that $S_{G_i}^- \subseteq S_F^+$. It follows that $S_{G_i}^-\theta \cap (Y \setminus Z) = \emptyset$. By I.H.(b) $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$ implies $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$.

From the above two bullets, we conclude

- for any $i \in \mathrm{Mon}$, $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$ implies $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$;

- for any $i \in \mathrm{Mixed}$, $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$ is equivalent to $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$;
- for any $i \in \mathrm{Anti}$, $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$ implies $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$.

By Lemma 43, $\mathrm{NES}_F(Z)$, which is

$$Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Z),\ldots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Z)) \wedge\ Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k)),$$

implies $\mathrm{NES}_F(Y)$, which is

$$Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Y),\ldots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Y)) \wedge\ Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k)).$$

Part (b): Similar to (a). ∎

Lemma 70 *For any GQ-formula F and any nonempty set Y of atoms, there exists a subset Z of Y such that*

(a) Z is a loop of F, and

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

Lemma 71 *Let F be a GQ-formula, Y be a set of ground atoms of $\sigma(F)$ and Z a nonempty subset of Y such that the dependency graph of F has no edges from atoms in Z to atoms in $Y \setminus Z$. $NES_F(Y)$ implies $NES_F(Z)$.*

Proof. By induction on F.

Case 1: F is $p_i(\boldsymbol{t})$. Clear from $Z \subseteq Y$.

Case 2: F is an atomic formula that does not contain members of $\boldsymbol{p}$. It is clear since $\mathrm{NES}_F(Y) = F$.

Case 3: F is of the form $Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k))$. Without loss of generality, we partition the set of all argument positions of Q into three sets: the set of monotone argument positions Mon, the set of antimonotone argument positions Anti and the rest of argument positions Mixed.

Consider two subcases.

3.1: Let S_F be the set of all atoms that has strictly positive occurrences in F. If for every substitution θ from variables in F to ground terms in Y, $S_F\theta \cap Y = \emptyset$ then, by Lemma 68, both $\mathrm{NES}_F(Z)$ and $\mathrm{NES}_F(Y)$ are equivalent to F.

3.2: Otherwise, there is a substitution θ from variables in S_F to ground terms in Y such that $S_F\theta \cap Y \neq \emptyset$.

- Consider any $i \in \mathrm{Anti} \cup \mathrm{Mixed}$. Let $S^+_{G_i}$ be the set of all atoms that has semi-positive non-$\boldsymbol{p}$-negated occurrences in G_i. Since there are no edges from Z to $Y \setminus Z$ in the dependency graph of F, it follows that for any substitution θ' from variables in $S^+_{G_i}$ to ground terms in Y $S^+_F\theta\theta' \cap (Y \setminus Z) = \emptyset$. By Lemma 69 (a), $\mathrm{NES}_{F\theta}(Z)$ implies $\mathrm{NES}_{F\theta}(Y)$. Since this holds for any such substitution θ, so we derive $\mathrm{NES}_F(Z)$ implies $\mathrm{NES}_F(Y)$.
- Consider any $i \in \mathrm{Mon} \cup \mathrm{Mixed}$. Note that the dependency graph of G_i is a subgraph of the dependency graph of F. It follows that the dependency graph of G_i has no edges from atoms in Z to atoms in $Y \setminus Z$. By I.H. $\mathrm{NES}_F(Y)$ implies $\mathrm{NES}_F(Z)$.

From the above two bullets, we conclude

- for any $i \in \mathrm{Mon}$, $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$ implies $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$;
- for any $i \in \mathrm{Mixed}$, $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$ is equivalent to $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$;
- for any $i \in \mathrm{Anti}$, $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Z)$ implies $\mathrm{NES}_{G_i(\boldsymbol{x}_i)}(Y)$.

By Lemma 43, $\mathrm{NES}_F(Y)$, which is

$$Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Y),\ldots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Y)) \wedge Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k)),$$

implies $\mathrm{NES}_F(Z)$, which is

$$Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](\mathrm{NES}_{G_1(\boldsymbol{x}_1)}(Z),\ldots,\mathrm{NES}_{G_k(\boldsymbol{x}_k)}(Z)) \wedge Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](G_1(\boldsymbol{x}_1),\ldots,G_k(\boldsymbol{x}_k)).$$

∎

Proof Between (b′) and (c′). It is clear that (b′) implies (c′). To prove the other direction, for the sake of contradiction, assume (c′) and that (b′) does not hold. Let Y be a nonempty subset of X such that

$$X \models Y^{\wedge} \tag{5.87}$$

and

$$X \models \mathrm{NES}_F(Y). \tag{5.88}$$

By Lemma 70, there there exists a subset Z of Y such that Z is a loop of F, and the dependency graph of F has no edges from Z to $Y \setminus Z$. From (5.87), we conclude that $X \models Z^{\wedge}$. By Lemma 71, (5.88) implies that $X \models \mathrm{NES}_F(Z)$. This conflicts with (c'). ∎

Proof of Proposition 41

Lemma 72 *Let F be a formula with generalized quantifiers. If every occurrence of every predicate constant p_i from $\boldsymbol{p}$ in F is strictly positive and not mixed in F, the formula*

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

is logically valid.

Proof. By induction on F.

Case 1: F is an atomic formula.

- If F is an atom $p_i(\boldsymbol{t})$ where $p_i \in \boldsymbol{p}$, $F^*(\boldsymbol{u})$ is exactly $F(\boldsymbol{u})$.
- Otherwise, both $F^*(\boldsymbol{u})$ and $F(\boldsymbol{u})$ are the same as F.

Case 2: F is of the form (5.1). Since every predicate from $\boldsymbol{p}$ in F is strictly positive and not mixed in F, it follows that every predicate from $\boldsymbol{p}$ occurs in a monotone argument position in F. By Proposition 32(a),

$$\begin{aligned}\boldsymbol{u} \leq \boldsymbol{p} \rightarrow ((Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](F_1(\boldsymbol{x}_1),\ldots,F_k(\boldsymbol{x}_k)))^* \\ \leftrightarrow Q[\boldsymbol{x}_1]\ldots[\boldsymbol{x}_k](F_1^*(\boldsymbol{x}_1),\ldots,F_k^*(\boldsymbol{x}_k)))\end{aligned}$$

is logically valid. Consider each F_i where $1 \leq i \leq k$. Note that every occurrence of every predicate constant p from $\boldsymbol{p}$ in F_i is strictly positive and not mixed in F_i. By I.H.

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

is logically valid. So the claim follows. ∎

Lemma 73 *Let $\boldsymbol{p}$ be a list of predicate constants. If F is canonical relative to $\boldsymbol{p}$, then the formula*

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

is logically valid.

Proof. By Lemma 40,

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

is logically valid. It is sufficient to prove that under the assumption of F,

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

is logically valid. This can be proven by induction on F.

Case 1: F is an atomic formula.

- F is an atom $p_i(\boldsymbol{t})$ where $p_i \in \boldsymbol{p}$, $F^*(\boldsymbol{u})$ is exactly $F(\boldsymbol{u})$.
- Otherwise, $F^*(\boldsymbol{u})$ and $F(\boldsymbol{u})$ are the same as F.

Case 2: F is of the form (5.1). Consider the following subcases.

- $Q \notin \{Q_\rightarrow, Q_\wedge, Q_\forall\}$. Consider each F_i where $1 \leq i \leq k$. Since every predicate constant p_i from $\boldsymbol{p}$ in F_i is strictly positive and not mixed in F_i, by Lemma 72,

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

  is logically valid. So under the assumption F, the formula $F^*(\boldsymbol{u})$ is equivalent to $F(\boldsymbol{u})$.

- $Q \in \{Q_\wedge, Q_\forall\}$, clear from I.H.

- Q is $Q_\rightarrow$. Assume $(\boldsymbol{u} \leq \boldsymbol{p}) \wedge (F_1 \rightarrow F_2)$. It is sufficient to show

$$(F_1^*(\boldsymbol{u}) \rightarrow F_2^*(\boldsymbol{u})) \leftrightarrow (F_1(\boldsymbol{u}) \rightarrow F_2(\boldsymbol{u}))). \tag{5.89}$$

  Since $F_1 \rightarrow F_2$ is canonical relative to $\boldsymbol{p}$, every occurrence of every predicate constant p_i from $\boldsymbol{p}$ in F_1 is strictly positive and not mixed in F_1, so that, by Lemma 72, $F_1^*(\boldsymbol{u})$ is equivalent to $F_1(\boldsymbol{u})$.

  - Case 1: $\neg F_1$. By Lemma 40, $\neg F_1^*(\boldsymbol{u})$. The claim follows since $\neg F_1^*(\boldsymbol{u})$ is equivalent to $\neg F_1(\boldsymbol{u})$.
  - Case 2: F_2. By I.H. $F_2^*(\boldsymbol{u})$ is equivalent to $F_2(\boldsymbol{u})$. The claim follows since $F_1^*(\boldsymbol{u})$ is equivalent to $F_1(\boldsymbol{u})$.

∎

Proposition 41 *Let Π be a finite general program and let F be the GQ-representation of Π. For every rule (5.21) in Π, if B is canonical relative to $\boldsymbol{p}$ and every occurrence of p from $\boldsymbol{p}$ in H is strictly positive and not mixed in H, then $FLP[\Pi; \boldsymbol{p}]$ is equivalent to $SM[F; \boldsymbol{p}]$.*

Proof. It is sufficient to show that under the assumption $\boldsymbol{u} < \boldsymbol{p}$ and Π, for each rule (5.21) in Π,

$$B(\boldsymbol{u}) \wedge B \rightarrow H(\boldsymbol{u})$$

is equivalent to

$$(B^*(\boldsymbol{u}) \rightarrow H^*(\boldsymbol{u})) \wedge (B \rightarrow H).$$

From the fact that B is canonical relative to $\boldsymbol{p}$, by Lemma 73,

$$B^*(\boldsymbol{u}) \leftrightarrow (B(\boldsymbol{u}) \wedge B).$$

Also, since every occurrence of p_i from $\boldsymbol{p}$ in H is strictly positive and not mixed in H, by Lemma 72, $H^*(\boldsymbol{u}) \leftrightarrow H(\boldsymbol{u})$ follows. ∎

Proof of Proposition 42

Lemma 74 *For any dl-program $(\mathcal{T}, \Pi)$, any dl-atom A of the form (5.7) in Π that contains no free variables, A is monotonic (anti-monotonic) relative to $\mathcal{T}$ iff Q_A^U is motonone (anti-monotone) in $\{1, \ldots, k\}$ for all Herbrand interpretations I of $\langle C, P_\Pi \rangle$.*

Proof. We will show the case of monotonic dl-atoms. The case of anti-monotonic dl-atoms is similar.

From left to right: Assume that A is monotonic relative to $\mathcal{T}$. We further assume $Q_A^U(R_1, \ldots, R_k) = \boldsymbol{t}$, where $R_j \subseteq |I|^{|\boldsymbol{x}_j|}$ for $1 \leq j \leq k$. Consider any $i \in \{1, \ldots, k\}$ and any $R_i' \subseteq |I|^{|\boldsymbol{x}_i|}$ such that $R_i \subseteq R_i'$, we will show that $Q_A^U(R_1, \ldots, R_{i-1}, R_i', R_{i+1}, \ldots, R_k) = \boldsymbol{t}$.

Let I' be the Herbrand interpretation

$$I \cup \{p_i(\boldsymbol{d}) \mid \boldsymbol{d} \in R_i' \setminus R_i\},$$

whose signature is the same as I. It is clear that $I \subseteq I'$. Also, by Lemma 47, $I \models_\mathcal{T} A$ follows from $Q_A^U(R_1, \ldots, R_i, \ldots, R_k) = \boldsymbol{t}$. Since A is monotonic relative to $\mathcal{T}$, $I' \models_\mathcal{T} A$ and by Lemma 47, $Q_A^U(R_1, \ldots, R_{i-1}, R_i', R_{i+1}, \ldots, R_k) = \boldsymbol{t}$. Since I and I' have the same universe, $Q_A^U(R_1, \ldots, R_{i-1}, R_i', R_{i+1}, \ldots, R_k) = \boldsymbol{t}$ follows.

From right to left: Assume that Q_A^U is monotone in $\{1, \ldots, k\}$ for all Herbrand interpretations I of $\langle C, P_\Pi \rangle$. Consider any Herbrand interpretations J, J' of $\langle C, P_\Pi \rangle$ such that $J \subseteq J'$ and assume that $J \models_\mathcal{T} A$. We will show that $J' \models_\mathcal{T} A$.

Let $R_i = \{\boldsymbol{d} \in |J|^{|\boldsymbol{x}_i|} \mid (p_i(\boldsymbol{d}))^J = \boldsymbol{t}\}$ and $R_i' = \{\boldsymbol{d} \in |J'|^{|\boldsymbol{x}_i|} \mid (p_i(\boldsymbol{d}))^{J'} = \boldsymbol{t}\}$ for each $1 \leq i \leq k$. From $J \models_\mathcal{T} A$, by Lemma 47, $Q_A^U(R_1, \ldots, R_k) = \boldsymbol{t}$. Since $J \subseteq J'$, it follows that $R_i \subseteq R_i'$ for each $1 \leq i \leq k$. From the fact that Q_A^U is monotone in $\{1, \ldots, k\}$, $Q_A^U(R_1', \ldots, R_k') = \boldsymbol{t}$ follows. By Lemma 47, $J' \models_\mathcal{T} A$. ∎

Lemma 75 *For any dl-program $(\mathcal{T}, \Pi)$, any Herbrand interpretations X, Y of $\langle C, P_\Pi \rangle$ such that $Y \subseteq X$, and any rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π,*

$$Y \models_\mathcal{T} (p(\boldsymbol{t}) \leftarrow B, N)_\mathcal{T}^X$$

iff

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models (B^{GQ})^*(\boldsymbol{q}) \wedge (N^{GQ})^*(\boldsymbol{q}) \rightarrow q(\boldsymbol{t}). \tag{5.90}$$

Proof. We partition B into two sets: the set B_2 of all anti-monotonic dl-atoms and the set B_1 of all remaining dl-atoms. In view of Lemma 74, B_2^{GQ} is a conjunction of GQ-formulas (5.11) such that Q^U is anti-monotone in all argument positions. By Lemma 44 and Proposition 32 (b), (5.90) is equivalent to

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models (B_1^{GQ})^*(\boldsymbol{q}) \wedge B_2^{GQ} \wedge N^{GQ} \rightarrow q(\boldsymbol{t}),$$

which is the same as

$$X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models B_1^{GQ}(\boldsymbol{q}) \wedge B_1^{GQ} \wedge B_2^{GQ} \wedge N^{GQ} \rightarrow q(\boldsymbol{t}). \tag{5.91}$$

Consider two cases.

Case 1: $X \models_{\mathcal{T}} B_2 \wedge N$. $(p(\boldsymbol{t}) \leftarrow B, N)_{\mathcal{T}}^X$ is $p(\boldsymbol{t}) \leftarrow B_1$. On the other hand, by Lemma 47, $X \models B_2^{GQ} \wedge N^{GQ}$. Since $Y \subseteq X$, $Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models B_1^{GQ}(\boldsymbol{q})$ implies $X \models B_1^{GQ}$. Thus (5.91) is equivalent to saying that $Y_{\boldsymbol{q}}^{\boldsymbol{p}} \models B_1^{GQ}(\boldsymbol{q}) \rightarrow q(\boldsymbol{t})$, which in turn is equivalent to saying that $Y \models B_1^{GQ} \rightarrow p(\boldsymbol{t})$. By Lemma 47 again, $Y \models B_1^{GQ} \rightarrow p(\boldsymbol{t})$ iff $Y \models_{\mathcal{T}} B_1 \rightarrow p(\boldsymbol{t})$.

Case 2: $X \not\models_{\mathcal{T}} B_2 \wedge N$. $(p(\boldsymbol{t}) \leftarrow B, N)_{\mathcal{T}}^X$ is equivalent to $\top$. By Lemma 47, $X \not\models B_2^{GQ} \wedge N^{GQ}$. Thus (5.91) follows. ∎

Lemma 76 *For any dl-program* $(\mathcal{T}, \Pi)$*,* $X \models \Pi^{GQ}$ *iff* $X \models_{\mathcal{T}} \Pi_{\mathcal{T}}^X$*.*

Proof. Immediate from the definition of $\Pi_{\mathcal{T}}^X$ that $X \models_{\mathcal{T}} \Pi_{\mathcal{T}}^X$ iff $X \models_{\mathcal{T}} \Pi$. It follows from Lemma 47 that $X \models_{\mathcal{T}} \Pi$ iff $X \models \Pi^{GQ}$. ∎

Proposition 42 *For any dl-program* $(\mathcal{T}, \Pi)$*, and any Herbrand interpretation* X *of* $\langle C, P_\Pi \rangle$*,* X *is an answer set of* $(\mathcal{T}, \Pi)$ *iff* X *satisfies* $SM[\Pi^{GQ}; P_\Pi]$ *relative to* $\mathcal{T}$*.*

Proof. X is an answer set of $(\mathcal{T}, \Pi)$ iff

(i) $X \models_{\mathcal{T}} \Pi^X_{\mathcal{T}}$, and

(ii) no proper subset Y of X satisfies $\Pi^X_{\mathcal{T}}$ relative to $\mathcal{T}$.

On the other hand, $X \models \mathrm{SM}[\Pi^{GQ}; P_\Pi]$ iff

(i′) $X \models \Pi^{GQ}$, and

(ii′) X does not satisfy $\exists \boldsymbol{u}(\boldsymbol{u} < P_\Pi \wedge (\Pi^{GQ})^*(\boldsymbol{u}))$.

By Lemma 76, (i) is equivalent to (i′). Assume (i′). Condition (ii) can be reformulated as: no proper subset Y of X satisfies $(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ relative to $\mathcal{T}$ for every rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π. Under the assumption (i′), condition (ii′) can be reformulated as: there is no proper subset Y of X such that, for every rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π, $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ satisfies $(B^{GQ})^*(\boldsymbol{q}) \wedge (N^{GQ})^*(\boldsymbol{q}) \rightarrow q(\boldsymbol{t})$. By Lemma 75, it follows that (ii) is equivalent to (ii′). ∎

Proof of Proposition 43

Let $(\mathcal{T}, \Pi)$ be a dl-program and X an Herbrand interpretation of $\langle C, P_\Pi \rangle$. By $f\Pi^X_{\mathcal{T}}$, we denote the FLP reduct of Π as defined by viewing a dl-program as a Hex program.

Lemma 77 *Let $(\mathcal{T}, \Pi)$ be a dl-program. For any Herbrand interpretations X, Y of $\langle C, P_\Pi \rangle$ such that $Y \subseteq X$ and any rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π,*

$$Y \models_{\mathcal{T}} f(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$$

iff

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models B^{GQ}(\boldsymbol{q}) \wedge N^{GQ}(\boldsymbol{q}) \wedge B^{GQ} \wedge N^{GQ} \rightarrow q(\boldsymbol{t}). \tag{5.92}$$

Proof. *Case 1:* $X \models_{\mathcal{T}} B, N$. $f(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ is $p(\boldsymbol{t}) \leftarrow B, N$. On the other hand, by Lemma 47, $X \models B^{GQ} \wedge N^{GQ}$ follows. (5.92) is equivalent to saying that $Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models B(\boldsymbol{q})^{GQ} \wedge N(\boldsymbol{q})^{GQ} \rightarrow q(\boldsymbol{t})$, which in turn is equivalent to saying that $Y \models B^{GQ} \wedge N^{GQ} \rightarrow p(\boldsymbol{t})$. By Lemma 47 again, $Y \models_{\mathcal{T}} p(\boldsymbol{t}) \leftarrow B, N$ iff $Y \models B^{GQ} \wedge N^{GQ} \rightarrow p(\boldsymbol{t})$.

Case 2: $X \not\models_{\mathcal{T}} B, N$. $f(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ is equivalent to $\top$. By Lemma 47, $X \not\models B^{GQ} \wedge N^{GQ}$. So (5.92) holds. ∎

Lemma 78 *For any dl-program* $(\mathcal{T}, \Pi)$ *such that* Π *is a ground program,* $X \models \Pi^{GQ}$ *iff* $X \models_{\mathcal{T}} f\Pi_{\mathcal{T}}^{X}$.

Proof. Immediate from the definition of $f\Pi_{\mathcal{T}}^{X}$ that $X \models_{\mathcal{T}} f\Pi_{\mathcal{T}}^{X}$ iff $X \models_{\mathcal{T}} \Pi$. It follows from Lemma 47 that $X \models_{\mathcal{T}} \Pi$ iff $X \models \Pi^{GQ}$. ∎

Proposition 43 *For any dl-program* $(\mathcal{T}, \Pi)$*, and any Herbrand interpretation* X *of* $\langle C, P_{\Pi}\rangle$*,* X *satisfies* $FLP[\Pi^{GQ}; P_{\Pi}]$ *relative to* $\mathcal{T}$ *iff* X *is an answer set of* $(\mathcal{T}, \Pi)$ *according to Fink and Pearce.*

Proof. X is an answer set of $(\mathcal{T}, \Pi)$ according to Fink and Pearce iff

(i) $X \models_{\mathcal{T}} f\Pi_{\mathcal{T}}^{X}$, and

(ii) no proper subset Y of X satisfies $f\Pi_{\mathcal{T}}^{X}$ relative to $\mathcal{T}$.

On the other hand, X satisfies $\mathrm{FLP}[\Pi^{GQ}; P_{\Pi}]$ iff

(i') $X \models \Pi^{GQ}$, and

(ii') X does not satisfy $\exists \boldsymbol{u}(\boldsymbol{u} < P_{\Pi} \wedge (\Pi^{GQ})^{\triangle}(\boldsymbol{u}))$.

By Lemma 80, (i) iff (i'). Condition (ii) can be reformulated as: no proper subset Y of X satisfies $f(p(\boldsymbol{t}) \leftarrow B, N)_{\mathcal{T}}^{X}$ relative to $\mathcal{T}$ for every rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π. Condition (ii') can be reformulated as: there is no proper subset Y of X such that, for every rule $p(\boldsymbol{t}) \leftarrow B, N$ in Π, $X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}$ satisfies $((p(\boldsymbol{t}) \leftarrow B, N)^{GQ})^{\triangle}(\boldsymbol{q})$. By Lemma 79, it follows that (ii) is equivalent to (ii'). ∎

Proof of Proposition 44

Let $(\mathcal{T}, \Pi)$ be a dl-program and X an Herbrand interpretation of $\langle C, P_{\Pi}\rangle$. By $f\Pi_{\mathcal{T}}^{X}$, we denote the FLP reduct of Π as defined by viewing a dl-program as a HEX program.

Lemma 79 *For any dl-program* $(\mathcal{T}, \Pi)$ *such that every occurrence of non-monotonic dl-atoms is in the positive body of a rule, any Herbrand interpretations* X, Y *of* $\langle C, P_{\Pi}\rangle$ *such*

that $Y \subseteq X$ *and any rule* $p(\boldsymbol{t}) \leftarrow B, N$ *in* Π*,*

$$Y \models_{\mathcal{T}} f(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$$

iff

$$Y \models_{\mathcal{T}} (p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$$

Proof. We partition B into two sets: the set B_1 of all anti-monotonic dl-atoms and the set B_2 of rest of all dl-atoms.

Consider two cases.

Case 1: $X \models_{\mathcal{T}} B \wedge N$. $f(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ is $p(\boldsymbol{t}) \leftarrow B, N$. On the other hand, $(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ is $p(\boldsymbol{t}) \leftarrow B_2$. It is sufficient to show that $Y \models_{\mathcal{T}} B_1 \wedge B_2 \wedge N$ iff $Y \models_{\mathcal{T}} B_2$. From $X \models B \wedge N$, it follows that $X \models B_1$ and $X \models N$. Since B_1 contains only anti-monotonic dl-atoms, $Y \models B_1$ follows from $X \models B_1$. Since N contains only negation of monotonic dl-atoms, $Y \models N$ follows from $X \models N$.

Case 2: $X \not\models_{\mathcal{T}} B \wedge N$. both $(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ and $f(p(\boldsymbol{t}) \leftarrow B, N)^X_{\mathcal{T}}$ are equivalent to $\top$. ∎

Lemma 80 *For any dl-program* $(\mathcal{T}, \Pi)$ *such that* Π *is a ground program and every occurrence of non-monotonic dl-atoms is in the positive body of a rule,* $X \models \Pi^X_{\mathcal{T}}$ *iff* $X \models_{\mathcal{T}} f\Pi^X_{\mathcal{T}}$*.*

Proof. Immediate from the definition of $f\Pi^X_{\mathcal{T}}$ that $X \models_{\mathcal{T}} f\Pi^X_{\mathcal{T}}$ iff $X \models_{\mathcal{T}} \Pi$. It follows from Lemma 47 that $X \models_{\mathcal{T}} \Pi$ iff $X \models \Pi^{GQ}$. ∎

Proposition 44 *For any dl-program* $(\mathcal{T}, \Pi)$*, and any Herbrand interpretation* X *of* $\langle C, P_\Pi \rangle$*, if every occurrence of non-monotonic dl-atoms is in the positive body of a rule, then* X *is an answer set of* $(\mathcal{T}, \Pi)$ *in the sense of (Fink & Pearce, 2010) iff* X *is an answer set of* $(\mathcal{T}, \Pi)$ *in our sense.*

Proof. X is an answer set of $(\mathcal{T}, \Pi)$ according to Fink and Pearce iff

(i) $X \models_{\mathcal{T}} f\Pi^X_{\mathcal{T}}$, and

(ii) no proper subset Y of X satisfies $f\Pi^X_{\mathcal{T}}$ relative to $\mathcal{T}$.

On the other hand, X satisfies $\mathrm{FLP}[\Pi^{GQ}; P_{\Pi}]$ iff

(i′) $X \models_{\mathcal{T}} \Pi^X_{\mathcal{T}}$, and

(ii′) no proper subset Y of X satisfies $\Pi^X_{\mathcal{T}}$ relative to $\mathcal{T}$.

By Lemma 80, (i) is equivalent to (i′). By Lemma 79, (ii) is equivalent to (ii′). ∎

Proof of Proposition 45

Theorem on Double Negations (Ferraris et al., 2009b) *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 $\boldsymbol{p}$ then $SM[H^-; \boldsymbol{p}]$ is equivalent to $SM[H; \boldsymbol{p}]$.*

The following corollary follows from the Splitting Lemma and the Theorem on Double Negations.

Corollary 15 *Let Π be a tight program with generalized quantifiers. $SM[(\Pi^{GQ})^{\neg\neg}; \boldsymbol{p}]$ is equivalent to $SM[\Pi^{GQ}; \boldsymbol{p}]$.*

Lemma 81 *Let Π be a finite program and X, Y be sets of atoms of $\sigma(\Pi)$ such that $Y \subseteq X$ and $\boldsymbol{p}$ be the list of all predicates in Π. Consider any rule*

$$p(\boldsymbol{t}) \leftarrow B, QB, N$$

in Π where B is a set of normal atoms, QB is a set of GQ-atoms and N is a set of normal and GQ atoms preceded by "not".

$$Y \models (p(\boldsymbol{t}) \leftarrow B, QB, N)^X$$

iff

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models (B \wedge QB^{\neg\neg} \wedge N^{\neg\neg})^*(\boldsymbol{q}) \rightarrow q(\boldsymbol{t}). \tag{5.93}$$

Proof. Since $QB^{\neg\neg}$ and $N^{\neg\neg}$ are negative on $\boldsymbol{p}$, in view of Lemma 44, (5.93) is the same as saying

$$X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models B^*(\boldsymbol{q}) \wedge QB^{\neg\neg} \wedge N^{\neg\neg} \rightarrow q(\boldsymbol{t}). \tag{5.94}$$

Consider two cases:

Case 1: $X \not\models QB, N$. $(p(\boldsymbol{t}) \leftarrow B, QB, N)^X$ is $\top$. Clearly, $X \not\models QB^{\neg\neg} \wedge N^{\neg\neg}$. As a result, (5.94) holds.

Case 2: $X \models QB, N$. $(p(\boldsymbol{t}) \leftarrow B, QB, N)^X$ is $p(\boldsymbol{t}) \leftarrow B$. On the other hand, (5.94) is equivalent to saying that

$$Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models B^*(\boldsymbol{q}) \rightarrow q(\boldsymbol{t}). \tag{5.95}$$

Since B is a conjunction of atoms, it is clear that $Y \models p(\boldsymbol{t}) \leftarrow B$ iff (5.95) holds. ∎

Lemma 82 *Let Π be a finite program and X be a set of atoms of $\sigma(\Pi)$. $X \models \Pi^X$ iff $X \models (\Pi^{GQ})^{\neg\neg}$.*

Proof. Clear from Lemma 81 when $Y = X$ and $\boldsymbol{p} = \boldsymbol{q}$. ∎

Proposition 45 *For any program Π and any Herbrand interpretation X of $\sigma(\Pi)$, X is a EGV-answer set of Π iff $X \models SM[(\Pi^{GQ})^{\neg\neg}]$.*

Proof. X is a EGV-answer set of Π iff

(i) $X \models \Pi^X$, and

(ii) no proper subset Y of X satisfies Π^X.

On the other hand, X satisfies $\mathrm{SM}[(\Pi^{GQ})^{\neg\neg}]$ iff

(i') $X \models (\Pi^{GQ})^{\neg\neg}$, and

(ii') X does not satisfy $\exists \boldsymbol{u}(\boldsymbol{u} < \boldsymbol{p} \wedge ((\Pi^{GQ})^{\neg\neg})^*(\boldsymbol{u}))$.

By Lemma 82, (i) is equivalent to (i'). Let rules in Π are of the form

$$p(\boldsymbol{t}) \leftarrow B, QB, N$$

where B is a set of normal atoms, QB is a set of GQ-atoms and N is a set of normal and GQ atoms preceded by "not". Condition (ii) can be reformulated as: no proper subset Y of X satisfies $(p(\boldsymbol{t}) \leftarrow B, QB, N)^X$ for every rule $p(\boldsymbol{t}) \leftarrow B, QB, N \in \Pi$. Condition (ii') can be reformulated as: there is no proper subset Y of X such that, for every rule $p(\boldsymbol{t}) \leftarrow B, QB, N \in \Pi$, $X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}}$ satisfies $(B \wedge QB^{\neg\neg} \wedge N^{\neg\neg})^*(\boldsymbol{q}) \rightarrow q(\boldsymbol{t})$. By Lemma 81, (ii) is equivalent to (ii'). ∎

Chapter 6

ANSWER SET PROGRAMMING MODULO THEORIES AND REASONING ABOUT CONTINUOUS CHANGES

The availability of efficient Answer Set Programming (ASP) solvers has greatly contribute the success of the programming paradigm. Most ASP solvers utilize (i) intelligent grounding—the process that replaces variables with ground terms—and (ii) efficient search methods that originated from propositional satisfiability solvers (SAT solvers). However, this method is not scalable in handling functional fluents that range over a large numeric domain. For example, consider the inertia rule:

$$\mathrm{Speed}_1(x) \leftarrow \mathrm{Speed}_0(x),\ \mathrm{not}\ \neg\mathrm{Speed}_1(x) \tag{6.1}$$

If the range of x is over a large domain of numbers (e.g. all integers), then grounding it w.r.t. all the elements in the domain becomes very inefficient. Moreover, real numbers are not supported at all because grounding cannot be applied.

An alternative representation using functions, such as replacing $\mathrm{Speed}_1(x)$ with $\mathrm{Speed}_1 = x$, does not work under the ASP semantics because (i) answer sets are defined as Herbrand models (e.g., $\mathrm{Speed}_1 = \mathrm{Speed}_0$ is always false under any Herbrand interpretation), and (ii) the nonmonotonicity of the stable model semantics has to do with minimizing the extents of predicates but has nothing to do with functions.

In order to alleviate the "grounding problem," there have been several recent efforts (Gebser et al., 2009; Balduccini, 2009; Janhunen et al., 2011; Liu et al., 2012) to integrate ASP with Constraint Programming or Satisfiability Modulo Theories (SMT), where functional fluents can be represented by variables in Constraint Satisfaction Problems or uninterpreted constants in SMT. However, like the standard ASP, nonmonotonicity of those extensions has to do with predicates only, and nothing to do with functions. For instance, a natural counterpart of (6.1) in the language of CLINGCON (Gebser et al., 2009),

$$\mathrm{Speed}_1 =^{\$} x \leftarrow \mathrm{Speed}_0 =^{\$} x,\ \mathrm{not}\ \mathrm{Speed}_1 \neq^{\$} x\ ,$$

does not correctly represent inertia.

Monotonic	Nonmonotonic
FOL	Functional SM
SMT	ASP Modulo Theories
SAT	ASP

Figure 6.1: Analogy between SMT and ASPMT

In this chapter, we present a framework of combining answer set programming with satisfiability modulo theories, which we call *Answer Set Programming Modulo Theories (ASPMT)*. Just like that SMT is a generalization of SAT and, at the same time, a special case of first-order logic in which certain predicate and function symbols in background theories (such as difference logic and the theory of reals) have fixed interpretations, ASPMT is a generalization of the standard ASP and, at the same time, a special case of functional stable model semantics (Bartholomew & Lee, 2012) that assumes background theories. Figure 6.1 summarizes our view on the analogy. Like the known relationship between SAT and ASP that *tight* ASP programs can be turned into SAT instances, we show that *tight* ASPMT programs can be turned into SMT instances, which allows SMT solvers for computing ASPMT programs.

This results allow us to enhance action language $\mathcal{C}$+ (Giunchiglia et al., 2004) to handle reasoning about continuous changes. Language $\mathcal{C}$+ is an expressive action description language but its semantics was defined in terms of propositional causal theories, which limits the language to express discrete changes only. By reformulating $\mathcal{C}$+ in terms of ASPMT, we naturally extend the language to overcome the limitation, and use SMT solvers to compute the language. Our experiment shows that this approach outperforms the existing implementations of $\mathcal{C}$+ by several orders of magnitude for some benchmark problems.

This chapter is organized as follows. Section 6.1 provides the theoretical foundation of the formalism. In Section 6.2, we extend the action language $\mathcal{C}$+ to allow arbitrary domains and to reasoning about continuous changes. We illustrate the expressiveness of the extended language using various benchmark examples. In Section 6.3, we show that reachability analysis in Hybrid Automata can be automated using the extended framework. Section 6.4 compare the framework with some related work in the modelling of actions and continuous changes.

6.1 Answer Set Programming Modulo Theories

Syntax

Formally, an SMT instance is a formula in many-sorted first-order logic, where some designated function and predicate constants are constrained by some fixed background interpretation. SMT is the problem of determining whether such a formula has a model that expands the background interpretation.

The syntax of ASPMT is the same as that of SMT. Let σ^{bg} be the (many-sorted) signature of the background theory bg (E.g., the theory of reals). In the theory of reals, we consider the set $\mathcal{R}$ of symbols for all real numbers, the set $\{+, -, /, *\}$ of arithmetic functions over reals and the set $\{<, >, \leq, \geq\}$ of binary predicates over reals. Note that we have a fixed set of functions and predicates, all of which are typed. Also, note that every symbol in σ^{bg} has a fixed interpretation. σ is extended from σ^{bg} with uninterpreted symbols - object constants of some sorts and propositional constants. *Terms* are defined as usual:

- object constants of σ and variables are terms;
- if f is a function constants of sort $(s_1, \ldots, s_n, s_{n+1})$ and $t_1, \ldots, t_n$ are terms of sort $s_1, \ldots, s_n$ then $f(t_1, \ldots, t_n)$ is a term of sort s_{n+1}. (N.B. this means that if f has arity > 0, then f must be a function from the background theory)

Atomic Formula are defined as follows:

- if p is a predicate constant of sort $(s_1, \ldots, s_n)$ and $t_1, \ldots, t_n$ are ground terms of sort $s_1, \ldots, s_n$, then $p(t_1, \ldots, t_n)$ is an atomic formula. (N.B. this means that if p has arity > 0, then p must be a predicate from the background theory)
- if t_1, t_2 are terms of the same sort then $t_1 = t_2$ is an atomic formula.

Formulas are defined the same as in the first-order logic. We call a formula *ground* if it contains no variable.

Semantics

We consider the standard semantics of many-sorted first-order formulas. Interpretation and satisfaction are defined as usual.

A *background interpretation* I^{bg} is an interpretation of the signature σ^{bg}. For instance, in the Theory of Reals,

- for each $r \in \mathcal{R}$, $r^{I^{bg}} = r$;
- for each arithmetics functions $f \in \{+, -, /, *\}$, $f^{I^{bg}} : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$;
- for each comparisons predicates $op \in \{<, >, \leq, \geq, =\}$, $op^{I^{bg}} : \mathcal{R} \times \mathcal{R} \to \{\boldsymbol{t}, \boldsymbol{f}\}$.

Let σ be a signature that is disjoint from σ^{bg}. We say that an interpretation I of σ satisfies F w.r.t. the background theory bg, denoted by $I \models_{bg} F$, if $I \cup I^{bg} \models F$. Let F be an ASPMT instance with background theory σ^{bg}. Interpretation I is a stable model of F relative to $\boldsymbol{c}$ (w.r.t. background theory σ^{bg}) if $I \models_{bg} \mathrm{SM}[F; \boldsymbol{c}]$.

We will often write the implication $F \to G$ in a form familiar in logic programs, $G \leftarrow F$. A finite set of formulas is identified with the conjunction of the formulas in the set.

An Example

Example 27 *The following formula F describes the inertia assumption on the speed of a car and the effect of accelerating. Assume that the background theory is the theory of reals.*

$$\begin{array}{lcl} Speed_1 = x & \leftarrow & Speed_0 = x \wedge \neg\neg(Speed_1 = x) \\ Speed_1 = x & \leftarrow & (x = Speed_0 + 3 \times Duration) \wedge Accelerate. \end{array} \tag{6.2}$$

Here x is a variable of sort $\mathcal{R}_{\geq 0}$; $ßSpeed_0$, $ßSpeed_1$ and $ßDuration$ are function constants of sort $\mathcal{R}_{\geq 0}$ and $ßAccelerate$ is a Boolean function constant. For interpretation I_1 of signature $\{Speed_0, Speed_1, Duration, Accelerate\}$ such that $Accelerate^{I_1} = \boldsymbol{f}$, $Speed_0^{I_1} = 1$, $Speed_1^{I_1} = 1$, $Duration^{I_1} = 1.5$, we have $I_1 \models_{bg} SM[F; Speed_1]$.

Consider another interpretation I_2 of the same signature that agrees with I except that $Accelerate^{I_2} = \boldsymbol{t}$, $Speed_1^{I_2} = 5.5$, we check that $I_2 \models_{bg} SM[F; Speed_1]$.

Another interpretation I_3 that agrees with I_1 except that $(Speed_1)^{I_3} = 5.5$. $I_3 \models_{bg} F$ but $I_3 \not\models_{bg} SM[F; Speed_1]$.

Completion

In this section, we present a generalization of the *theorem on completion* from (Bartholomew & Lee, 2012) by (i) not restricting to "f-plain" theories, and (ii) referring to a weaker notion of "tightness."

Let f be a function constant. A first-order formula is called *f-plain* if each atomic formula in it does not contain f, or is of the form $f(\boldsymbol{t}) = t_1$ where $\boldsymbol{t}$ is a list of terms not containing f, and t_1 is a term not containing f. For any list $\boldsymbol{c}$ of predicate and function constants, we say that F is $\boldsymbol{c}$-plain if F is f-plain for each function constant f in $\boldsymbol{c}$.

A formula F is said to be in *Clark normal form* (relative to the list $\boldsymbol{c}$ of intensional constants) if it is a conjunction of sentences of the form

$$\forall \boldsymbol{x}(G \rightarrow p(\boldsymbol{x})) \tag{6.3}$$

and

$$\forall \boldsymbol{x} y(G \rightarrow f(\boldsymbol{x}) = y) \tag{6.4}$$

one for each intensional predicate p and each intensional function f, where $\boldsymbol{x}$ is a list of distinct object variables, y is an object variable, and G is an arbitrary formula that has no free variables other than those in $\boldsymbol{x}$ and y.

The *completion* of a formula F in Clark normal form (relative to $\boldsymbol{c}$) is obtained from F by replacing each conjunctive term (6.3) with $\forall \boldsymbol{x}(p(\boldsymbol{x}) \leftrightarrow G)$ and each conjunctive term (6.4) with $\forall \boldsymbol{x} y(f(\boldsymbol{x}) = y \leftrightarrow G)$.

The *dependency graph* of F (relative to $\boldsymbol{c}$) is the directed graph that

- has all members of $\boldsymbol{c}$ as its vertices, and
- has an edge from c to d if, for some strictly positive occurrence of $G \rightarrow H$ in F,
  - c has a strictly positive occurrence in H, and
  - d has a strictly positive occurrence in G.

We say that F is *tight* (on $\boldsymbol{c}$) if the dependency graph of F (relative to $\boldsymbol{c}$) is acyclic.

The following theorem is a generalization of Theorem 12 from (Bartholomew & Lee, 2012).

Theorem 16 *For any formula F in Clark normal form that is tight on $\boldsymbol{c}$, an interpretation I that satisfies $\exists xy(x \neq y)$ is a model of $SM[F;\boldsymbol{c}]$ iff I is a model of the completion of F relative to $\boldsymbol{c}$.*

Theorem 16 can be applied to formulas in non-Clark normal form if they can be rewritten in Clark normal form. The following theorem is useful in extending completion to formulas that are not in Clark normal form.

Theorem 17 *(Bartholomew & Lee, 2012, Theorem 1) For any first-order formulas F and G, if G has no strict positive occurrence of a constant from $\boldsymbol{c}$, $SM[F \wedge G;\boldsymbol{c}]$ is equivalent to $SM[F;\boldsymbol{c}] \wedge G$.*

Theorem 16 is applicable to ASPMT formulas as well. Since F in Example 27 is tight on Speed_1, according to Theorem 16, $\mathrm{SM}[F;\mathrm{Speed}_1]$ is equivalent to the following SMT instance with the same background theory:

$$\begin{aligned}\mathrm{Speed}_1 = x \;\leftrightarrow\; &(\mathrm{Speed}_0 = x \;\wedge\; \neg\neg(\mathrm{Speed}_1 = x)) \\ &\vee (x = \mathrm{Speed}_0 + 3\times \mathrm{Duration}) \wedge \mathrm{Accelerate}\ .\end{aligned}$$

Comparison with Clingcon

Clingcon semantics (Gebser et al., 2009) combines answer set programs with arbitrary constraints.

Remind that a *constraint satisfaction problem* (CSP) is a tuple (V, D, C), where V is a set of *constraint variables* with the respective *domain* D, and C is a set of constraints of the form

$$\langle (v_1, \ldots, v_n), R \rangle, \tag{6.5}$$

such that $v_i \in V (1 \leq i \leq n)$ and $R \subseteq \mathrm{Dom}(v_1) \times \cdots \times \mathrm{Dom}(v_n)$.

In view of the definition of SM from the previous section, we identify V with object constants in $\sigma \setminus \sigma_{bg}$. D contains the $sort(v)$ for every $v \in V$. We represent a constraint (6.5) as a formula $F(v_1, \ldots, v_n)$ of the signature $\sigma^{bg} \cup V$ where $F(x_1, \ldots, x_n)$ is a formula of the signature σ^{bg} and $F(v_1, \ldots, v_n)$ is obtained from $F(x_1, \ldots, x_n)$ by substituting the constants $(v_1, \ldots, v_n)$ for $(x_1, \ldots, x_n)$. We call $F(v_1, \ldots, v_n)$ a *formula over* $(v_1, \ldots, v_n)$.

We now review the syntax and the semantics of constraint answer sets by (Gebser et al., 2009). We find it convenient to understand constraint answer sets as a first-order interpretations.

A *clingcon program* Π with with a constraint satisfaction problem (V, D, C) is a set of rules of the form

$$H \leftarrow B, N, Cn, \tag{6.6}$$

where H, B are sets of positive propositional literals, N is a set of negative propositional literals, and Cn is a set of constraints from C, possibly preceded by not.

We identify an interpretation of σ as the tuple $\langle A, X \rangle$ where A is the interpretation over function constants and X is a set of propositional atoms in $\sigma \setminus \sigma^{bg}$.

Given a clingcon program Π, an interpretation $I = \langle A, X \rangle$, we define the *reduct of* Π *relative to* X *and* A (denoted by Π_A^X) as the set of rules

$$H \leftarrow B,$$

where $H \leftarrow B, N, Cn$ is in Π such that

- $A \cup I^{bg} \models Cn$, and
- $X \models N$.

We say that a set X of propositional atoms is a *constraint answer set* of Π relative to A if X is a minimal model of Π_A^X.

Example 17 continued *Consider the Clingcon program* Π*:*

$$triangle(obj);$$

$$rightTriangle(obj) \leftarrow triangle(obj), sq(side_1(obj)) + sq(side_2(obj)) = sq(side_3(obj)).$$

Let A be a mapping such that $A(side_1(obj)) = 3$, $A(side_1(obj)) = 4$, $A(side_1(obj)) = 5$ and let $X = \{triangle(obj), rightTriangle(obj)\}$. We check that X is a constraint answer set relative to A because X is the minimal model of Π_A^X which is

$$triangle(obj);$$
$$rightTriangle(obj) \leftarrow triangle(obj).$$

We identify a clingcon program Π as a formula which is the conjunction of the rules in Π. The following proposition reformulate the clingcon semantics in terms of SM.

Proposition 49 *Let Π be a clingcon program with CSP (V, D, C), let $\boldsymbol{p}$ be a set of propositional constants that occur in Π and $I = \langle A, X\rangle$ an interpretation of the signature $V \cup \boldsymbol{p}$. $I \models_{bg} SM[\Pi; \boldsymbol{p}]$ iff X is a constraint answer set of Π relative to A.*

Example 17 Continued. *Let A and X be the same as above. We check that $\langle A, X\rangle \models_{bg} SM[\Pi; triangle(obj)]$.*

6.2 Enhancing $\mathcal{C}$+ for Continuous Changes

Syntax

We consider a many-sorted first-order signature σ that is partitioned into three signatures: the set σ^{fl} of object constants called *fluent constants*, the set σ^{act} of object constants called *action constants*, and the background signature σ^{bg}. The signature σ^{fl} is further partitioned into the set σ^{sim} of *simple* fluent constants and the set σ^{sd} of *statically determined* fluent constants. We assume the same syntax of formulas as in Section 6.1. A *fluent formula* is a formula of signature $\sigma^{fl} \cup \sigma^{bg}$. An *action formula* is a formula of $\sigma^{act} \cup \sigma^{bg}$ that contains at least one action constant and no fluent constants.

A *static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{6.7}$$

where each F and G are fluent formulas. An *action dynamic law* is an expression of the form (6.7) in which F is an action formula and G is a formula. A *fluent dynamic law* is an

expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{6.8}$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants. A *causal law* is a static law, or an action dynamic law, or a fluent dynamic law. A *$\mathcal{C}$+ action description* is a finite set of causal laws.

An action description is *definite* if the head F of every causal law (6.7) and (6.8) is an atomic formula that is $(\sigma^{fl} \cup \sigma^{act})$-plain. Throughout this chapter we consider definite action descriptions only, which covers the fragment of $\mathcal{C}$+ that is implemented in CCalc.

Semantics

In (Giunchiglia et al., 2004) the semantics of $\mathcal{C}$+ is defined in terms of nonmonotonic propositional causal theories, in which every constant has a finite domain. The semantics of the enhanced $\mathcal{C}$+ below is similar to the one in (Giunchiglia et al., 2004) except that it is defined in terms of ASPMT in place of causal theories. This reformulation is essential for the language to represent continuous changes as it is not limited to finite domains only.

For a signature σ and a nonnegative integer i, expression $i : \sigma$ is the signature consisting of the pairs $i : c$ such that $c \in \sigma$, and the sort of $i : c$ is the same as the sort of c. Similarly, if s is an interpretation of σ, $i : s$ is an interpretation of $i : \sigma$ such that $c^s = (i : c)^{i:s}$.

For any definite action description D of signature $\sigma^{fl} \cup \sigma^{act} \cup \sigma^{bg}$ and any nonnegative integer m, the ASPMT program D_m is defined as follows. The signature of D_m is $0 : \sigma^{fl} \cup \cdots \cup m : \sigma^{fl} \cup 0 : \sigma^{act} \cup \cdots \cup (m-1) : \sigma^{act} \cup \sigma^{bg}$. By $i : F$ we denote the result of inserting $i :$ in front of every occurrence of every fluent and action constant in a formula F. ASPMT program D_m is the conjunction of

$$\neg\neg\, i : G \rightarrow i : F$$

for every static law (6.7) in D and every $i \in \{0, \ldots, m\}$, and for every action dynamic law (6.7) in D and every $i \in \{0, \ldots, m-1\}$;

$$\neg\neg\, (i+1) : G \wedge i : H \rightarrow (i+1) : F$$

for every fluent dynamic law (6.8) in D and every $i \in \{0, \ldots, m-1\}$.

The transition system represented by an action description D consists of states (vertices) and transitions (edges). A *state* is an interpretation s of σ^{fl} such that $0 : s \models_{bg} \mathrm{SM}[D_0;\ 0{:}\sigma^{sd}]$. A *transition* is a triple $\langle s, e, s' \rangle$, where s and s' are interpretations of σ^{fl} and e is an interpretation of σ^{act}, such that

$$(0{:}s) \cup (0{:}e) \cup (1{:}s') \models_{bg} \mathrm{SM}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})].$$

The following theorems extend Propositions 7 and 8 from (Giunchiglia et al., 2004) by referring to functional stable model semantics. They justify the soundness of our reformulation of $\mathcal{C}$+.

Theorem 18 *For every transition $\langle s, e, s' \rangle$, s and s' are states.*

Theorem 19

$$(0{:}s_0) \cup (0{:}e_0) \cup (1{:}s_1) \cup (1{:}e_1) \cup \cdots \cup (m{:}s_m)$$
$$\models_{bg} SM[D_m;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl}) \cup (1{:}\sigma^{act}) \cup \cdots \cup (m{-}1{:}\sigma^{act}) \cup (m{:}\sigma^{fl})]$$

iff each triple $\langle s_i, e_i, s_{i+1} \rangle$ $(0 \leq i < m)$ is a transition.

According to the above theorems, each stable model of D_m corresponds to a path of length m in the transition system represented by D. At the same time, each path in the transition system of D can be mapped to a stable model of D_m. As a result, the stable models of D_m correctly represent the semantics of D.

It is not difficult to check that the ASPMT program D_m that is obtained from action description D is always tight. In view of the theorem on completion (Section 6.1), D_m can be represented in the language of SMT as the next section demonstrates.

Reasoning about Continuous Changes in $\mathcal{C}$+

In order to represent continuous changes in the enhanced $\mathcal{C}$+, we distinguish between steps and real clock times. We assume the theory of reals as the background theory, and introduce a simple fluent constant Time of sort $\mathcal{R}_{\geq 0}$, which denotes the clock time,

and an action constant Dur of sort $\mathcal{R}_{\geq 0}$, which denotes the time elapsed between the two consecutive states. We postulate the following causal laws:

$$\begin{aligned} &\textbf{exogenous } \mathrm{Time}, \mathrm{Dur}, \\ &\textbf{constraint } \mathrm{Time} = t + t' \textbf{ after } \mathrm{Time} = t \wedge \mathrm{Dur} = t' \ . \end{aligned} \tag{6.9}$$

These causal laws are shorthand for

$$\begin{aligned} &\textbf{caused } \mathrm{Time} = t \textbf{ if } \mathrm{Time} = t \ , \\ &\textbf{caused } \mathrm{Dur} = t \textbf{ if } \mathrm{Dur} = t \ , \\ &\textbf{caused } \bot \textbf{ if } \neg(\mathrm{Time} = t + t') \textbf{ after } \mathrm{Time} = t \wedge \mathrm{Dur} = t' \end{aligned}$$

where t, t' are variables of sort $\mathcal{R}_{\geq 0}$. (See Appendix B in (Giunchiglia et al., 2004) for the abbreviations of causal laws.)

Continuous changes can be described as a function of duration using fluent dynamic laws of the form

$$\textbf{caused } c = f(\boldsymbol{x}, \boldsymbol{x}', t) \textbf{ if } \boldsymbol{c}' = \boldsymbol{x}' \textbf{after } (\boldsymbol{c} = \boldsymbol{x}) \wedge (\mathrm{Dur} = t) \wedge G$$

where (i) c is a simple fluent constant, (ii) $\boldsymbol{c}$, $\boldsymbol{c}'$ are lists of fluent constants, (iii) $\boldsymbol{x}$, $\boldsymbol{x}'$ are lists of object variables, (iv) G is a formula, and (v) $f(\boldsymbol{x}, \boldsymbol{x}', t)$ is a term constructed from σ^{bg}, and variables in $\boldsymbol{x}$, $\boldsymbol{x}'$, and t.

For instance, the fluent dynamic law

$$\begin{aligned} &\textbf{caused } \mathrm{Distance} = d + 0.5 \times (v' + v) \times t \textbf{ if } \mathrm{Speed} = v' \\ &\qquad \textbf{after } \mathrm{Speed} = v \wedge \mathrm{Distance} = d \wedge \mathrm{Dur} = t \end{aligned}$$

describes how fluent $\mathrm{Distance}$ changes according to the function of real time.

Consider the following problem by Lifschitz.[1]

> If the accelerator of a car is activated, the car will speed up with constant acceleration A until the accelerator is released or the car reaches its maximum speed MS, whichever comes first. If the brake is activated, the car will slow down with acceleration $-A$ until the brake is released or the car stops, whichever comes first. Otherwise, the speed of the car remains constant. The problem asks to

[1] http://www.cs.utexas.edu/vl/tag/continuous_problem

find a plan satisfying the following condition: at time 0, the car is at rest at one end of the road; at time T, it should be at rest at the other end.

Notation: A, MS are real numbers

Simple fluent constants:	Domain:
Speed, $\mathrm{Distance}$, Time	$\mathcal{R}_{\geq 0}$
Action constants:	Domain:
$\mathrm{Accelerate}$, $\mathrm{Decelerate}$	Boolean
Dur	$\mathcal{R}_{\geq 0}$

caused $\mathrm{Speed}=v+A\times t$ **after** $\mathrm{Accelerate}\wedge\mathrm{Speed}=v\wedge\mathrm{Dur}=t$
caused $\mathrm{Speed}=v-A\times t$ **after** $\mathrm{Decelerate}\wedge\mathrm{Speed}=v\wedge\mathrm{Dur}=t$
caused $\mathrm{Distance}=d+0.5\times(v'+v)\times t$ **if** $\mathrm{Speed}=v'$
after $\mathrm{Speed}=v\wedge\mathrm{Distance}=d\wedge\mathrm{Dur}=t$
constraint $\mathrm{Time}=t+t'$ **after** $\mathrm{Time}=t\wedge\mathrm{Dur}=t'$
constraint $\mathrm{Speed}\leq MS$

inertial Speed
exogenous Time, c for every action constant c

Figure 6.2: Car Example in $\mathcal{C}$+

A $\mathcal{C}$+ description of this example is shown in Figure 6.2. The actions $\mathrm{Accelerate}$ and $\mathrm{Decelerate}$ has direct effects on Speed and indirect effects on $\mathrm{Distance}$. According to the semantics in Section 6.2, the description is turned into the ASPMT program with the theory of reals as the background theory, which can be further rewritten in Clark normal form. Some occurrences of $\neg\neg$ can be dropped without affecting stable models, which results in the program in Figure 6.3.

The program can be viewed as $F\wedge G$ where F is the conjunction of the rules that has $i+1:\mathrm{Speed}$ in the heads, and G is the conjunction of the rest rules. In view of Theorem 20, the stable models and $F\wedge G$ are the same as the stable models of F that satisfies G. By Theorem 16, the completion of the program relative to $i+1:\mathrm{Speed}$ is equivalent to the following formula

$$\begin{aligned} i+1:\mathrm{Speed}=x \leftrightarrow\ & \big(x=(i:\mathrm{Speed}+A\times i:\mathrm{Dur})\ \wedge\ i:\mathrm{Accelerate}\big) \\ & \vee\big(x=(i:\mathrm{Speed}-A\times i:\mathrm{Dur})\ \wedge\ i:\mathrm{Decelerate}\big) \\ & \vee\big(i+1:\mathrm{Speed}=x\ \wedge\ i:\mathrm{Speed}=x\big). \end{aligned}$$

Notation: x, v, t are variables of sort $\mathcal{R}_{\geq 0}$; y is a variable of sort Boolean.
Intensional object constants: $i{:}\mathrm{Speed}$ for $i > 0$

$$i{+}1{:}\mathrm{Speed}{=}x \leftarrow (x = v{+}A{\times}t) \wedge i{:}(\mathrm{Accelerate} \wedge \mathrm{Speed}{=}v \wedge \mathrm{Dur}{=}t)$$
$$i{+}1{:}\mathrm{Speed}{=}x \leftarrow (x = v{-}A{\times}t) \wedge i{:}(\mathrm{Decelerate} \wedge \mathrm{Speed}{=}v \wedge \mathrm{Dur}{=}t)$$
$$i{+}1{:}\mathrm{Distance}{=}x \leftarrow (x = d{+}0.5{\times}(v'{+}v){\times}t)$$
$$\wedge\, i{+}1{:}\mathrm{Speed}{=}v' \wedge i : (\mathrm{Speed}{=}v \wedge \mathrm{Distance}{=}d \wedge \mathrm{Dur}{=}t)$$
$$\neg\neg((i{+}1{:}\mathrm{Time}){=}(i{:}\mathrm{Time}) + (i{:}\mathrm{Dur}))$$
$$\neg\neg(i{:}\mathrm{Speed} \leq MS)$$
$$i{+}1{:}\mathrm{Speed}{=}x \leftarrow \neg\neg(i{+}1{:}\mathrm{Speed}{=}x) \wedge i{:}\mathrm{Speed}{=}x$$
$$i{:}\mathrm{Time}{=}t \leftarrow \neg\neg(i{:}\mathrm{Time}{=}t)$$
$$i{:}c{=}y \leftarrow \neg\neg(i{:}c{=}y) \qquad \text{for every action constant } c$$

Figure 6.3: Car Example in ASPMT

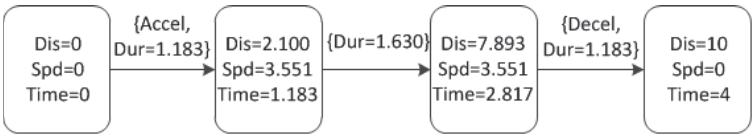

Figure 6.4: A Path in the Transition System of Car Example.

Variable x in the formula can be eliminated by equivalent transformations using equality:

$$i{:}\mathrm{Accelerate}{=}\boldsymbol{t} \rightarrow i{+}1{:}\mathrm{Speed}{=}(i{:}\mathrm{Speed}{+}A{\times}i{:}\mathrm{Dur})$$
$$i{:}\mathrm{Decelerate}{=}\boldsymbol{t} \rightarrow i{+}1{:}\mathrm{Speed}{=}(i{:}\mathrm{Speed}{-}A{\times}i{:}\mathrm{Dur})$$
$$(i{+}1 : \mathrm{Speed} = (i{:}\mathrm{Speed}{+}A{\times}i{:}\mathrm{Dur}) \wedge i{:}\mathrm{Accelerate}{=}\boldsymbol{t})$$
$$\vee(i{+}1 : \mathrm{Speed} = (i{:}\mathrm{Speed}{-}A{\times}i{:}\mathrm{Dur}) \wedge i{:}\mathrm{Decelerate}{=}\boldsymbol{t})$$
$$\vee(i{:}\mathrm{Speed} = i{+}1{:}\mathrm{Speed})\ .$$

This formula can be encoded in the input language of SMT solvers. The shortest plan found by iSAT [2] on this input formula when the road length is 10, $A = 3, MS = 4, K = 4$ is shown in Figure 6.4.

Reasoning about Additive Fluents

Additive fluents are fluents with numerical values such that the effect of several concurrently executed actions on it can be computed by adding the effects of the individual actions. Lee and Lifschitz [(2003a)] show how to describe additive fluents in $\mathcal{C}$+ by understanding "incre-

[2] http://isat.gforge.avacs.org/index.html

ment laws" as shorthand for some causal laws. However, some additive fluents are real-valued, and cannot be represented in the language described in (Lee & Lifschitz, 2003a) as it is limited to finite domains only. This made the discussion of additive fluents in (Lee & Lifschitz, 2003a) limited to integer domains only. For example, in (Lee & Lifschitz, 2003a) the effect of firing multiple jets on the velocity of a spacecraft is described by "increment laws"

$$\mathrm{Fire}(j) \textbf{ increments } \mathrm{Vel}(ax) \textbf{ by } n/Mass \textbf{ if } \mathrm{Force}(j, ax) = n \ ,$$

where Mass stands for an integer constant. The duration of firing action is assumed to be 1, and all components of the position and the velocity vectors at any time are assumed to be integers, and even the forces applied are limited to integers. Obviously these are too strong assumptions.

These limitations are not present in our SMT-based computation of the enhanced $\mathcal{C}$+. The representation in (Lee & Lifschitz, 2003a) can be straightforwardly extended to handle continuous motions by distinguishing between steps and real time as in the previous section. For example, we can describe the effect that firing multiple jets has on the velocity of a spacecraft by

$$\mathrm{Fire}(j) \textbf{ increments } \mathrm{Vel}(ax) \textbf{by } n/\mathrm{Mass} \times t \textbf{ if } \mathrm{Force}(j, ax) = n \wedge \mathrm{Dur} = t \ . \tag{6.10}$$

In general, an incremental law is of the form

$$a \textbf{ increments } c \textbf{ by } f(\boldsymbol{x}, t) \textbf{ if } \boldsymbol{d} = \boldsymbol{x} \wedge \mathrm{Dur} = t \wedge G \tag{6.11}$$

where

- a is a Boolean action or fluent constant;
- c is an additive fluent constant;
- $\boldsymbol{d}$ is a list of fluent constants, and $\boldsymbol{x}$ is a list of corresponding variables;
- $f(\boldsymbol{x}, t)$ is an arithmetic expression over $\boldsymbol{x}$ and the duration t;
- G is a formula that contains no Boolean action constants.

The increment law (6.11) is understood in terms of causal laws of the form (6.7) and (6.8) by using auxiliary action constants $Contr(a, c)$ where c is an additive fluent constants and a is an c-contributing constant. We

- replace each increment law (6.11) with the action dynamic law

$$\textbf{caused } Contr(a, c) = f(\boldsymbol{x}_{\boldsymbol{cur}}, d) \textbf{ if } \boldsymbol{c}_{\boldsymbol{cur}} = \boldsymbol{x}_{\boldsymbol{cur}} \wedge Duration = d \wedge G, \tag{6.12}$$

- for every auxiliary constant $Contr(a, c)$, add the action dynamic law

$$\textbf{caused } Contr(a, c) = 0 \textbf{ if } Contr(a, c) = 0, \tag{6.13}$$

- add the fluent dynamic laws

$$\textbf{caused } c = v + \Sigma_a v_a \textbf{ if } \top \textbf{ after } c = v \wedge \bigwedge_a Contr(a, c) = v_a, \tag{6.14}$$

  for every additive fluent constant c, every $v \in Dom(c)$ and every function $a \rightarrow v_a$ that maps each c-contributing constant a to an element of the domain of Contr(a,c) so that $v + \Sigma_a v_a$ is in the domain of c.

The sum and the multiple conjunction in (6.14) range over all c-contributing constants a.

Example 28 *The following example from (Lee & Lifschitz, 2003a) describe the use of additive fluents.*

> *A spacecraft is not affected by any external forces. It has two jets and the force that can be applied by each jet along each axis is at most 2. The current position of is* $(0, 0, 0)$*, and its current velocity is* $(0, 1, 1)$*. How can it get to* $(0, 3, 2)$ *within 2 seconds? Assume the mass is 2.*

The $C+$ encoding of the example is shown in Figure 6.5. Figure 6.6 shows the basic program that corresponds to the program in Figure 6.5. The corresponding ASPMT

program is shown in Figure 6.7. $Fire(j)$ is a contribution constant $Speed(ax)$. The increment law (6.10) is understood as the set of laws

$$\begin{aligned}
&\textbf{caused } \mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))=x/\mathrm{Mass}\times t \\
&\qquad\qquad \textbf{if } \mathrm{Fire}(j)\wedge \mathrm{Force}(j,ax)=x\wedge \mathrm{Dur}=t \\
&\textbf{caused } \mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))=0 \textbf{ if } \mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))=0 \\
&\textbf{caused } \mathrm{Speed}(ax)=v_1+v_2+v_3 \textbf{ if } \top \textbf{ after } v_1=\mathrm{Speed}(ax)\wedge \\
&\qquad v_2=\mathrm{Contr}(\mathrm{Fire}(J_1),\mathrm{Speed}(ax))\wedge v_3=\mathrm{Contr}(\mathrm{Fire}(J_2),\mathrm{Speed}(ax)),
\end{aligned} \tag{6.15}$$

which in turn can be understood as the following ASPMT program:

$$\begin{aligned}
&i{:}\,\mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))=x/\mathrm{Mass}\times t \\
&\qquad\qquad \leftarrow i{:}\,\mathrm{Fire}(j)\wedge i{:}\,\mathrm{Force}(j,ax)=x\wedge i{:}\,\mathrm{Dur}=t \\
&i{:}\,\mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))=0 \leftarrow \neg\, i{:}\,\mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))\neq 0 \\
&i{+}1{:}\,\mathrm{Speed}(ax)=s \leftarrow s=(i{:}\,\mathrm{Speed}(ax)+i{:}\,\mathrm{Contr}(\mathrm{Fire}(J_1),\mathrm{Speed}(ax)) \\
&\qquad\qquad +i{:}\,\mathrm{Contr}(\mathrm{Fire}(J_2),\mathrm{Speed}(ax))).
\end{aligned} \tag{6.16}$$

Completion of (6.16) is the conjunction of it with

$$\begin{aligned}
&i{:}\,\mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))=0\vee \\
&(i{:}\,\mathrm{Contr}(\mathrm{Fire}(j),\mathrm{Speed}(ax))=\ (i{:}\,\mathrm{Force}(j,ax)\times i{:}\ \mathrm{Dur}/\mathrm{Mass})\wedge i{:}\,\mathrm{Fire}(j)).
\end{aligned}$$

The definition of additive fluent in from (Lee & Lifschitz, 2003a) restricts contributing constant to be action constant. We slightly generalize to allow Boolean fluent constants as well. This is generalization allows us to encode process as we show in the next section. Our $\mathcal{C}$+ representation of the spacecraft example is essentially an enhancement of the one in Figure 6.2 in that it allows concurrent accelerations.

Table 6.1 and Table 6.2 compare the performance of SMT-based computation of $\mathcal{C}$+ vs. existing implementations of $\mathcal{C}$+: CCalc and cplus2asp in terms of running time and the number of variables and clauses. System cplus2asp translates $\mathcal{C}$+ into ASP programs and use gringo and clasp for computation. For the sake of comparison, we assume that the duration of each action is exactly 1 unit of time so that the plans found by the systems are of the same kind. We assume that initially the spacecraft is rest at coordinate $(0,0,0)$. The task is to find a plan such that at each integer time t, the spacecraft is at (t^2,t^2,t^2).

Notation: $j \in \{J_1, J_2\}$, $ax \in \{X, Y, Z\}$

Simple fluent constants:	Domain:
$Pos(ax)$	$\mathcal{R}$
$Time$	$\mathcal{R}_{\geq 0}$
Action constants:	**Domain:**
$\mathrm{Fire(j)}$, $\mathrm{Force(j,ax)}$	Boolean
Dur	$\mathcal{R}_{\geq 0}$
Additive fluent constants:	**Domain:**
Speed	$\mathcal{R}$

$\mathrm{Fire(j)}$ **increments** $\mathrm{Speed(ax)}$ **by** $n/Mass \times t$ **if** $\mathrm{Force}(j, ax) = n \ \wedge \mathrm{Dur} = t$.
caused $\mathrm{Pos(ax)} = p + (0.5 \times (v + \mathrm{Speed(ax)}) \times t)$
after $\mathrm{Speed(ax)} = v \wedge \mathrm{Pos(ax)} = p \wedge \mathrm{Dur} = t$.
always $\mathrm{Force(j,ax)} = 0 \leftrightarrow \neg \mathrm{Fire(j)}$.
constraint $\mathrm{Time} = t + t'$ **after** $\mathrm{Time} = t \wedge \mathrm{Dur} = t'$
exogenous Time
exogenous c for every action constant c

Figure 6.5: Spacecraft Example in Additive $\mathcal{C}$+

Steps	CCalc v2.0	cplus2asp v1.0	$\mathcal{C}$+ in iSAT v1.0
	Run Time (grounding+solving)	Run Time (grounding+solving)	Run Time last/total
1	0.16 (0.12+0.00)	0.01 (0.01+0)	0/0
2	0.57 (0.40+0.00)	0.03 (0.03+0)	0/0
3	10.2 (2.62+6)	0.43 (0.23+0.2)	0/0
4	505.86 (12.94+479)	12.55 (3.18+9.37)	0/0
5	failed (51.10+failed)	73.07 (15.85+57.22)	0/0.03
6	time out	3020.85 (62.38+2958.47)	0/0.03
10	time out	time out	0.03/0.09
50	time out	time out	0.09/1.39
100	time out	time out	0.17/5.21
200	time out	time out	0.33/21.96

Table 6.1: Experimental Results (Running Time) on Spacecraft Example

The plan involve having the maximal acceleration at each step. We further restrict duration of each action is exactly 1 unit of time so that the program can be run using previous $C+$ solvers. A 2 hours timeout is assumed for all systems. The first set is based on the $C+$ solver $CCALC$. The run time consists of grounding time (first number) and solving time (second number). Both time increase exponential according to the number of steps. When the step is 5, the grounded program is too large so that the solver does not return any solution. The second set is based on $cplus2asp$. It is a recent improvement of $CCALC$ by smart grounding. We can see that there is a significant speed up for test. However, the increase of all attributes are still exponential. The test takes too long to return solutions for

Notation: $j \in \{J_1, J_2\}$, $ax \in \{X, Y, Z\}$

Simple fluent constants:	Domain:
$Pos(ax)$	$\mathcal{R}$
$Time$	$\mathcal{R}_{\geq 0}$
Action constants:	**Domain:**
$\mathrm{Fire(j)}$, $\mathrm{Force(j,ax)}$	Boolean
Dur	$\mathcal{R}_{\geq 0}$
Additive fluent constants:	**Domain:**
Speed	$\mathcal{R}$

caused $\mathrm{Speed(ax)} = v_1 + v_2 + v_3$ **if** $v_1 = \mathrm{Speed(ax)} \wedge$
$v_2 = \mathrm{Contr}(\mathrm{Fire}(J_1), \mathrm{Speed(ax)}) \wedge v_3 = \mathrm{Contr}(\mathrm{Fire}(J_2), \mathrm{Speed(ax)})$
caused $\mathrm{Contr}(\mathrm{Fire(j)}, \mathrm{Speed(ax)}) = n/Mass \times t$
if $\mathrm{Fire(j)} \wedge \mathrm{Force(j, ax)} = n \wedge \mathrm{Dur} = t$
caused $\mathrm{Contr}(\mathrm{Fire(j)}, \mathrm{Speed(ax)}) = 0$ **if** $\mathrm{Contr}(\mathrm{Fire(j)}, \mathrm{Speed(ax)}) = 0$
caused $\mathrm{Pos(ax)} = p + (0.5 \times (v + \mathrm{Speed(ax)}) \times t)$
after $\mathrm{Speed(ax)} = v \wedge \mathrm{Pos(ax)} = p \wedge \mathrm{Dur} = t$.
always $\mathrm{Force(j,ax)} = 0 \leftrightarrow \neg \mathrm{Fire(j)}$.
constraint $\mathrm{Time} = t + t'$ **after** $\mathrm{Time} = t \wedge \mathrm{Dur} = t'$
exogenous Time
exogenous c for every action constant c

Figure 6.6: Spacecraft Example in Basic $\mathcal{C}$+

Notation: s, x are variables of sort $\mathcal{R}$, MS is a real number;
$j \in \{J_1, J_2\}$, $ax \in \{X, Y, Z\}$.
Intensional object constants: $:\mathrm{Pos(ax)}$ for $i > 0$

$i{+}1 : \mathrm{Speed(ax)} = s \leftarrow s = (i : \mathrm{Speed(ax)} + i : \mathrm{Contr}(\mathrm{Fire}(J_1), \mathrm{Speed(ax)})$
$+ i : \mathrm{Contr}(\mathrm{Fire}(J_2), \mathrm{Speed(ax)}))$
$i : \mathrm{Contr}(\mathrm{Fire(j)}, \mathrm{Speed(ax)}) = x \leftarrow i : \mathrm{Fire(j)} \wedge (i : \mathrm{Force(j,ax)} \times i : \mathrm{Dur}/\mathrm{Mass}) = x$
$i : \mathrm{Contr}(\mathrm{Fire(j)}, \mathrm{Speed(ax)}) = 0 \leftarrow \neg i : \mathrm{Contr}(\mathrm{Fire(j)}, \mathrm{Speed(ax)}) \neq 0$
$i{+}1 : \mathrm{Pos(ax)} = (i : \mathrm{Pos(ax)} + 0.5 \times i : \mathrm{Dur} \times (i{+}1 : \mathrm{Speed(ax)} + i : \mathrm{Speed(ax)}))$
$\leftarrow \neg (i : \mathrm{Force(j,ax)} = 0 \leftrightarrow \neg i : \mathrm{Fire(j)})$
$i{+}1 : \mathrm{Time} = i : \mathrm{Time} + i : \mathrm{Dur}$
$\leftarrow \neg (0 \leq i : \mathrm{Speed} \leq MS)$

Figure 6.7: Spacecraft Example in ASPTM

steps greater than 6. The third set is based on our framework using the SMT solver iSAT. Since iSAT uses iterative deepening search, the run time contains the actual time for the last step and the accumulated time. We can see that all attributes increase linearly according to the number of steps. The run time is still very short for a large steps number. From the experiments, it is clear that our approach significantly improves previous approaches

Steps	CCalc v2.0	cplus2asp v1.0	$\mathcal{C}$+ in iSAT v1.0
	# atoms / clauses	# atoms / rules	# variables / clauses
1	488 / 1872	1864 / 2626	(42+53) / 182
2	3262 / 14238	6673 / 12035	(82+98) / 352
3	32772 / 155058	42778 / 92124	(122+143) / 520
4	204230 / 992838	228575/ 503141	(162+188) / 688
5	897016 / 4410186	949240/ 2060834	(202+233) / 856
6	–	3179869/ 6790167	(242+278) / 1024
10	–	–	(402+458) / 1696
50	–	–	(2002+2258) / 8416
100	–	–	(4002+4508) / 16816
200	–	–	(8002+9008) / 33616

Table 6.2: Experimental Results (Instance Size) on Spacecraft Example

both in expressivity of the language and the efficiency of computation. The experiment was performed on Intel Core 2 Duo CPU 3.00 GHz with 4 GB RAM running on Ubuntu version 11.10. It shows clear advantage of the SMT-based computation of $\mathcal{C}$+ for this example.

Representing Processes in $\mathcal{C}+$

The language $\mathcal{C}$+ is flexible enough to represent the *start-process-end* model (Reiter, 1996; Fox & Long, 2006a), where instantaneous actions may initiate or terminate processes. Processes run over time and have a continuous effect on numeric fluents. They are initiated and terminated either by the direct action of the agent or by events triggered in the environment.

The model can be encoded in $\mathcal{C}+$ by representing a process as a Boolean fluent p. An instantaneous event e_{start} causes the fluent to be true which starts the process

$$e_{start} \textbf{ causes } p \ .$$

Similarly, an action or event e_{end} causes the fluent to be false which terminates the process

$$e_{end} \textbf{ causes } \neg p \ .$$

The process fluent p is inertial, meaning that the process continues until there is an action or event stops it

$$\textbf{inertial } p \ .$$

The process fluent p determines the changes of additive variables c in terms of incremental laws

$$p \textbf{ increments } c \textbf{ by } f(\boldsymbol{x}_{\boldsymbol{c}cur}, t) \textbf{ if } \boldsymbol{c}_{\boldsymbol{c}cur} = \boldsymbol{x}_{\boldsymbol{c}cur} \wedge Dur = t \wedge G.$$

For example, the process $On(Tap_1)$ increases the water level by the flow rate $W(Tap_1)$ times the duration

$$On(Tap_1) \textbf{ increments } Level \textbf{ by } W(Tap_1) \times t \textbf{ if } Dur = t. \tag{6.17}$$

The start action e_{start} and end actions e_{end} are instantaneous

$$e_{start} \textbf{ causes } Dur = 0$$
$$e_{end} \textbf{ causes } Dur = 0.$$

Example 29 *The following example describes the process of two taps filling a water tank.*

There are two taps above a water tank with a leak. When Tab_i is turned on, water fills the tank at the constant rate of W_i. The leak causes water to be drained at the constraint rate of V. The water level $Level$ must be maintained between the minimum constant level Low and the maximum constant level $High$. Assume $W_1 = W_2 = 0.2$, $V = 0.3$, $Low = 0.5$, $High = 1$. Initially, $Level = 1$, $Time = 0$. Find a plan such that when $Time = 5$, water level is 0.75.

Figure 6.8 describes the two-taps water tank example in $\mathcal{C}+$ with additive fluents. The process of Tap_i filling the tank can be modeled by the fluent $On(Tap_i)$. An instantaneous action $TurnOn(Tap_i)$ initiates the process while another instantaneous action $TurnOff(Tap_i)$ terminates it. When the fluent $On(Tap_i)$ is true, it contributes to increasing the water level by W_i times duration. $Leaking$ on the other hand is another process that decreases the water level.

The semantics of the $C+$ program in Figure 6.8 is defined by the ASPMT program in Figure 6.9. Completion of the program relative to the intensional constants are the

conjunction of the rules and the following formula

$$i{:}\,Contr(On(x),Level){=}0 \ \vee (i{:}\,Contr(On(x),Level){=}\,W(x) \times i{:}\,Dur \wedge i{:}\,On(x))$$

$$i{:}\,Contr(Leaking,Level){=}0 \ \vee (i{:}\,Contr(Leaking,Level){=}V \times i : Dur \wedge i{:}\,Leaking)$$

$$i{+}1{:}\,On(x) \ \rightarrow (i{:}\,On(x) \vee i{:}\,TurnOn(x))$$

$$\neg i{+}1{:}\,On(x) \ \rightarrow (\neg i{:}\,On(x) \vee i{:}\,TurnOff(x))$$

$$i{+}1{:}\,Leaking \ \rightarrow i{:}\,Leaking$$

$$\neg i{+}1{:}\,Leaking \ \rightarrow \neg i{:}\,Leaking.$$

$x \in \{\mathrm{Tap}_1, \mathrm{Tap}_2\}$; $W(x)$, V, Low, $High$ are real numbers;
t, t' are variables of sort $\mathcal{R}_{\geq 0}$.

$\mathrm{TurnOn}(x)$ **causes** $\mathrm{On}(x)$; $\mathrm{TurnOff}(x)$ **causes** $\neg\mathrm{On}(x)$
$\mathrm{TurnOn}(x)$ **causes** $\mathrm{Dur}{=}0$; $\mathrm{TurnOff}(x)$ **causes** $\mathrm{Dur}{=}0$

$\mathrm{On}(x)$ **increments** Level **by** $W(x){\times}t$ **if** $\mathrm{Dur}{=}t$
$\mathrm{Leaking}$ **decrements** Level **by** $V{\times}t$ **if** $\mathrm{Dur}{=}t$

constraint $Low{\leq}\,\mathrm{Level} \wedge \mathrm{Level}{\leq}\,High$
inertial $\mathrm{On}(x), \mathrm{Leaking}$
exogenous c for every action constant c

exogenous Time
constraint $\mathrm{Time}{=}t + t'$ **after** $\mathrm{Time}{=}t \wedge \mathrm{Dur}{=}t'$

Figure 6.8: Two Taps Water Tank Example $\mathcal{C}$+

An output from iSAT corresponds to a path in the transition system described by the $C+$ program in Figure 6.8 is shown in Figure 6.10. The solution contains 5 steps and the solving time is 0.07 second.

6.3 Hybrid Automata and $\mathcal{C}$+

Hybrid Automata

In the following, we review the definition of Hybrid Automata. The definition follows from the one in (Henzinger, 1996).

A *Hybrid Automaton* H consists of the following components:

Notation: y, v_1, v_2, v_3 are variables of sort $\mathcal{R}$.
Intensional object constants: $i\!:\!\mathrm{Leaking}$, $i\!:\!\mathrm{Contr(Leaking,Level)}$,
$i\!:\!\mathrm{Contr(On(x),Level)}$, $i\!:\!\mathrm{On(x)}$ for $i > 0$

$i{+}1\!:\!\mathrm{On(x)} \leftarrow i\!:\!\mathrm{TurnOn(x)}$
$\neg i{+}1\!:\!\mathrm{On(x)} \leftarrow i\!:\!\mathrm{TurnOff(x)}$
$i\!:\!\mathrm{Contr(On(x),Level)}{=}y \leftarrow y{=}\mathrm{W(x)}{\times}t \wedge i\!:\!\mathrm{Dur}{=}t \wedge i\!:\!\mathrm{On(x)}$
$i\!:\!\mathrm{Contr(On(x),Level)}{=}0 \leftarrow \neg\, i\!:\!\mathrm{Contr(On(x),Level)} \neq 0$
$i\!:\!\mathrm{Contr(Leaking,Level)}{=}y \leftarrow y{=}{-}\mathrm{V}{\times}t{\wedge}\!:\!\mathrm{Dur}{=}t \wedge i\!:\!\mathrm{Leaking}$
$i\!:\!\mathrm{Contr(Leaking,Level)}{=}0 \leftarrow \neg\, i\!:\!\mathrm{Contr(Leaking,Level)} \neq 0$
$i{+}1\!:\!\mathrm{Level}{=}x{+}v_1{+}v_2{+}v_3 \leftarrow i\!:\!\mathrm{Level}{=}x \wedge i\!:\!\mathrm{Contr}(\mathrm{On}(\mathrm{Tap}_1),\mathrm{Level})){=}v_1$
$\wedge i\!:\!\mathrm{Contr}(\mathrm{On}(\mathrm{Tap}_2),\mathrm{Level}){=}v_2 \wedge i\!:\!\mathrm{Contr(Leaking,Level)}{=}v_3$
$i{+}1\!:\!\mathrm{Time}{=}y \leftarrow i\!:\!\mathrm{TurnOn(x)} \wedge i\!:\!\mathrm{Time}{=}y$
$i{+}1\!:\!\mathrm{Time}{=}y \leftarrow i\!:\!\mathrm{TurnOff(x)} \wedge i\!:\!\mathrm{Time}{=}y$
$i{+}1\!:\!\mathrm{Time}{=}t{+}t' \leftarrow i\!:\!\mathrm{Dur}{=}t \wedge i\!:\!\mathrm{Time}{=}t'$
$i{+}1\!:\!\mathrm{Leaking} \leftarrow i\!:\!\mathrm{Leaking} \wedge \neg\neg i{+}1\!:\!\mathrm{Leaking}$
$\neg i{+}1\!:\!\mathrm{Leaking} \leftarrow \neg i\!:\!\mathrm{Leaking} \wedge \neg\, i{+}1\!:\!\mathrm{Leaking}$
$i{+}1\!:\!\mathrm{On(x)} \leftarrow i\!:\!\mathrm{On(x)} \wedge \neg\neg i{+}1\!:\!\mathrm{On(x)}$
$\neg i{+}1\!:\!\mathrm{On(x)} \leftarrow \neg i\!:\!\mathrm{On(x)} \wedge \neg i{+}1\!:\!\mathrm{On(x)}$

Figure 6.9: Two Taps Water Tank Example in ASPTM

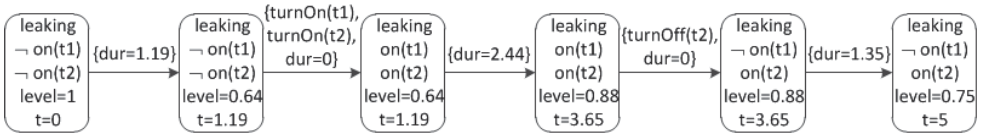

Figure 6.10: A Path in the Transition System of Two Taps Water Tank Example.

- **Variables.** A finite list $X = (x_1, \ldots, x_n)$ of variables whose values are in $\mathcal{R}$. X defines the continuous components. We write $\dot{X}$ for the list $(\dot{x}_1, \ldots, \dot{x}_n)$ of dotted variables, representing first derivatives during continuous change, and X' for the list $(x'_1, \ldots, x'_n)$ of primed variables, representing the set of "next" variables.

- **Directed Graph.** A finite directed graph $\langle V, E\rangle$, where V is the set of locations and E is a finite set of edges between locations.

- **Invariant, initial and flow conditions.** Three vertex labeling functions, init, inv, and flow, that assign to each location $v \in V$ three arithmetic formulas:

  - Each $inv(v)(X)$ is a formula over X which constraints the value of the continuous part of the state while the location is v.

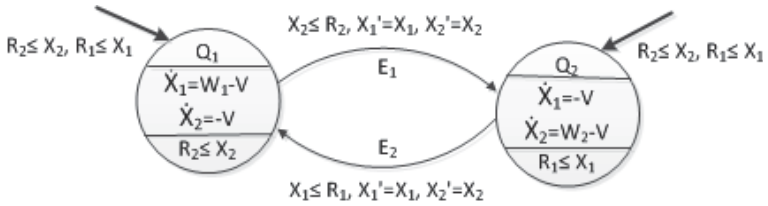

Figure 6.11: Hybrid Automata for Water Tank System.

  - Each $init(v)(X)$ is a formula over X that defines the initial condition.
  - Each $flow(v)(\dot{X}, X)$ is a formula over $X \cup \dot{X}$, which constraints the continuous variables and their first derivatives.

- **Jump conditions.** Each $jump(e)(X', X)$, where $e \in E$, is a formula over $X \cup X'$, which specifies the precondition and postcondition of the edge.

- **Events.** A finite set Σ of events and a function, $event : E \to \Sigma$, that assigns to each edge an unique event.

Example 30 *Consider the following Water Tank System example from (Lygeros, 2004).*

The automata consists of two variables $X = (X_1, X_2)$*, two events* $\Sigma = \{E_1, E_2\}$ *and two locations* $V = \{Q_1, Q_2\}$*. The flow, initial, invariants and jump conditions are as shown in the graph. For example,* $flow(Q_1)(\dot{X}_1, \dot{X}_2, X_1, X_2)$ *is* $\dot{X}_1 = W - V_1 \wedge \dot{X}_2 = -V_2$*,* $init(Q_1)(X_1, X_2)$ *is* $X_1 \geq R_1 \wedge X_2 \geq R_2$*.* $inv(Q_1)(X_1, X_2)$ *is* $X_2 \geq R_2$*.* $event(Q_1, Q_2) = E_1$*,* $jump(Q_1, Q_2)(X_1', X_2', X_1, X_2)$ *is* $X_2 \leq R_2 \wedge X_1' = X_1 \wedge X_2' = X_2$*.*

A *labelled transition system* consists of the following components:

- **State Space.** A (possibly infinite) set, Q, of states and a subset, $Q_0 \subseteq Q$ of initial states.

- **Transition Relation.** A (possibly infinite) set, A, of labels. For each label $a \in A$ a binary relation $\to^a$ on the state space Q. Each triple $q \xrightarrow{a} q'$ is called a transition.

The *Hybrid Transition System* T_H of a Hybrid Automaton H is the *labelled transition system* with components Q, Q_0, A and $\xrightarrow{a}$, for each $a \in A$, defined as follows:

- Define $Q, Q_0 \subseteq V \times \mathcal{R}^n$ such that $(v, \boldsymbol{n})$ is in Q iff $inv_v(\boldsymbol{n})$ is true, and $(v, \boldsymbol{n})$ is in Q_0 iff both $inv_v(\boldsymbol{n})$ and $init_v(\boldsymbol{n})$ are true.
- $A = \Sigma \cup \mathcal{R}_{\geq 0}$.
- For each event $\sigma \in \Sigma$, define $(v, \boldsymbol{n}) \xrightarrow{\sigma} (v', \boldsymbol{n}')$ iff there is an edge $(v, v') \in E$ such that: (1) the formula $jump_{(v,v')}(\boldsymbol{n}, \boldsymbol{n}')$ is true, and (2) $\mathrm{event}(v, v') = \sigma$.
- For each $\delta \in \mathcal{R}_{\geq 0}$, define $(v, \boldsymbol{n}) \xrightarrow{\delta} (v, \boldsymbol{n}')$ iff there is a differentiable function $f : [0, \delta] \to \mathcal{R}^n$, with the first derivative $\dot{f} : (0, \delta) \to \mathcal{R}^n$ such that: (i) $f(0) = \boldsymbol{n}$ and $f(\delta) = \boldsymbol{n}'$, and (ii) for all reals $\epsilon \in (0, \delta)$, both $inv_v(f(\epsilon))$ and $flow_v(f(\epsilon), \dot{f}(\epsilon))$ are true.

Example 30 Continued Let $W = 0.75$, $V_1 = V_2 = 0.5$, $R_1 = R_2 = 0$. Transition system of the automata contains the following path

$$(Q_1, (X_1 = 0, X_2 = 1)) \xrightarrow{2} (Q_1, (X_1 = 0.5, X_2 = 0)).$$

There is a differentiable function $f : [0, 2] \to \mathcal{R}^2$ such that for any $\epsilon \in (0, 2)$, $f(\epsilon) = (0 + 0.25 \times \epsilon, 1 - 0.5 \times \epsilon)$ which corresponds to the functions on X_1 and X_2 respectively. $\dot{f(\epsilon)} = (0.25, -0.5)$. Note that $flow(Q_1)(\dot{X_1}, \dot{X_2}, X_1, X_2)$ is $\dot{X}_1 = 0.25 \wedge \dot{X}_2 = -0.5$. We check $flow(Q_1)(\dot{f}(\epsilon), f(\epsilon))$ is $(0.25 = 0.25) \wedge (-0.5 = -0.5)$ and $inv(Q_1)(f(\epsilon))$ is $X_2 \geq 0$ which is true because $0 < X_2 < 1$ when $\epsilon \in (0, 2)$.

A *Linear Hybrid Automaton* is a Hybrid Automaton where (1) all the conditions are Boolean combinations of linear inequalities, and (2) the flow conditions contain variables in $\dot{X}$ only, and (3) $flow(v)(\dot{X}, X)$ is a conjunction of linear inequalities. For example, the automaton shown in Figure 6.11 is a linear Hybrid Automaton.

Linear Hybrid Automata in $\mathcal{C}$+ Modulo Theories

Given a linear hybrid automaton H, we assume that the background theory is the theory of reals extended with members of V as object constants. Consider a signature σ consisting of

- for each arithmetic variable X_i in H, a simple fluent constant X_i of sort $\mathcal{R}$;

- for each event $e \in \Sigma$, a Boolean action constant e;
- for each vertex v, a Boolean action constant $E(v)$;
- a simple fluent constant Time of sort $\mathcal{R}_{\geq 0}$;
- an action constant Dur of sort $\mathcal{R}_{\geq 0}$;
- a simple fluent constant Loc of sort V.

A $\mathcal{C}$+ action description describing hybrid automaton H consists of the following causal laws. Below $\boldsymbol{x}$ represents a list of variables of sort $\mathcal{R}$.

(i) For each vertex $v \in V$, the causal laws that represent invariant condition:

$$\textbf{constraint } inv_v(\boldsymbol{X}) \textbf{ if } \mathrm{Loc} = v$$

and flow condition:

$$\textbf{constraint } flow_v((\boldsymbol{X} - \boldsymbol{x})/t) \textbf{ if } \mathrm{Loc} = v \textbf{ after } \mathrm{Loc} = v \wedge \mathrm{Dur} = t \wedge t > 0 \wedge \boldsymbol{X} = \boldsymbol{x}$$

$$\textbf{constraint } \boldsymbol{X} = \boldsymbol{x} \textbf{ if } \mathrm{Loc} = v \textbf{ after } \mathrm{Loc} = v \wedge \neg E(v) \wedge \mathrm{Dur} = 0 \wedge \boldsymbol{X} = \boldsymbol{x}$$

where $(\boldsymbol{X} - \boldsymbol{x})/t$ represents the tuple of expressions $(X_i - x_i)/t$. Also include

$$\textbf{default } \neg E(v).$$

$E(v)$ is an auxiliary action constant that records whether any event happend from the location v. If $E(v)$ is false, the flow condition applies; otherwise the jump condition applies.

(ii) For each edge $(v, v') \in E$, causal laws for the jump condition.

$$\textbf{constraint } jump_{(v,v')}(\boldsymbol{x}, \boldsymbol{X}) \textbf{ after } \mathrm{event}(v, v') \wedge \boldsymbol{X} = \boldsymbol{x} \ .$$

(iii) For each edge (v, v') in E,

$$\begin{aligned}
&\textbf{exogenous } \mathrm{event}(v, v') \\
&\textbf{nonexecutable } \mathrm{event}(v, v') \textbf{ if } \mathrm{Loc} \neq v \\
&\mathrm{event}(v, v') \textbf{ causes } \mathrm{Loc} = v' \wedge \mathrm{Dur} = 0 \wedge E(v).
\end{aligned}$$

(iv) **exogenous** X_i for each simple fluent constant X_i; causal law **inertial** Loc; and causal law (6.9).

$q \in \{Q_1, Q_2\}$; t, t' are variables of sort $\mathcal{R}_{\geq 0}$.

Simple fluent constants:	Domain:
X_1, X_2, Time	$\mathcal{R}_{\geq 0}$
Loc	$\{Q_1, Q_2\}$
Action constants:	Domain:
E_1, E_2, $E(q)$	Boolean
Dur	$\mathcal{R}_{\geq 0}$

% inv
constraint $X_2 \geq R_2$ **if** $\mathrm{Loc} = Q_1$
constraint $X_1 \geq R_1$ **if** $\mathrm{Loc} = Q_2$

% flow
constraint $((X_1{-}x_1)/t, (X_2{-}x_2)/t) = (W{-}V_1, {-}V_2)$ **if** $\mathrm{Loc}{=}Q_1$
  after $\mathrm{Loc}{=}Q_1 \wedge (X_1, X_2) = (x_1, x_2) \wedge \mathrm{Dur}{=}t \wedge t > 0$
constraint $(X_1, X_2) = (x_1, x_2)$ **if** $\mathrm{Loc}{=}Q_1$
  after $\neg E(Q_1) \wedge \mathrm{Loc}{=}Q_1 \wedge \mathrm{Dur}{=}0 \wedge (X_1, X_2) = (x_1, x_2)$
constraint $((X_1{-}x_1)/t, (X_2{-}x_2)/t) = ({-}V_1, W{-}V_2)$ **if** $\mathrm{Loc}{=}Q_2$
  after $\mathrm{Loc}{=}Q_2 \wedge (X_1, X_2) = (x_1, x_2) \wedge \mathrm{Dur}{=}t \wedge t > 0$
constraint $(X_1, X_2) = (x_1, x_2)$ **if** $\mathrm{Loc}{=}Q_2$
  after $\neg E(Q_2) \wedge \mathrm{Loc}{=}Q_2 \wedge \mathrm{Dur}{=}0 \wedge (X_1, X_2) = (x_1, x_2)$

% jump
constraint $(X_1, X_2) = (x_1, x_2) \wedge (x_2 \leq R_2)$ **after** $E_1 \wedge (X_1, X_2) = (x_1, x_2)$
constraint $(X_1, X_2) = (x_1, x_2) \wedge (x_1 \leq R_1)$ **after** $E_2 \wedge (X_1, X_2) = (x_1, x_2)$

exogenous $X_1, X_2, E_1, E_2, \mathrm{Time}, \mathrm{Dur}$
nonexecutable E_1 **if** $\mathrm{Loc} \neq Q_1$
nonexecutable E_2 **if** $\mathrm{Loc} \neq Q_2$
E_1 **causes** $\mathrm{Loc}{=}Q_2 \wedge \mathrm{Dur}{=}0 \wedge E(Q_1)$
E_2 **causes** $\mathrm{Loc}{=}Q_1 \wedge \mathrm{Dur}{=}0 \wedge E(Q_2)$
inertial Loc; **default** $\neg E(Q_1)$; **default** $\neg E(Q_2)$
constraint $\mathrm{Time}{=}t + t'$ **after** $\mathrm{Time}{=}t \wedge \mathrm{Dur}{=}t'$

Figure 6.12: Hybrid Automaton of Water Tank (Figure 6.11) in $\mathcal{C}+$

Example 30 Continued The Linear Hybrid Automata in Example 30 can be encoded in the $\mathcal{C}$+ program shown in Figure 6.12. It is semantics is defined in terms of the $ASPTM$ program in Figure 6.3. Completion of the program is the conjunction of the program with

Figure 6.13: Water Tank Example in ASPMT

Intensional constants: $i : Location$ for $1 \leq i$
Note $q \in \{Q_1, Q_2\}$.

$(i : X_2) \geq (i : R_2) \leftarrow (i : Location) = Q_1$
$((i+1 : X_1) - (i : X_1))/d = (W - V_1) \wedge ((i+1 : X_2) - (i : X_2))/d = -V_2$
$\quad \leftarrow (i+1 : Location) = Q_1, (i : Location) = Q_1, (i : Duration) = d, d > 0$
$(i+1 : X_1) = (i : X_1) \wedge (i+1 : X_2) = (i : X_2)$
$\quad \leftarrow (i+1 : Location) = Q_1, (i : Location) = Q_1, (i : Duration) = 0$
$((i+1 : X_1) = (i : X_1) \wedge (i+1 : X_2) = (i : X_2)$
$\quad \wedge (i : X_2 \leq R_2) \wedge v = Q_1) \leftarrow E_1 \wedge (i : Location) = v$

$(i : X_1) \geq (i : R_1) \leftarrow (i : Location) = Q_2$
$((i+1 : X_1) - (i : X_1))/d = -V_1 \wedge ((i+1 : X_2) - (i : X_2))/d = (W - V_2)$
$\quad \leftarrow (i+1 : Location) = Q_2, (i : Location) = Q_2, (i : Duration) = d, d > 0$
$(i+1 : X_1) = (i : X_1) \wedge (i+1 : X_2) = (i : X_2)$
$\quad \leftarrow (i+1 : Location) = Q_2, (i : Location) = Q_2, (i : Duration) = 0$
$((i+1 : X_1) = (i : X_1) \wedge (i+1 : X_2) = (i : X_2)$
$\quad \wedge (i : X_1 \leq R_1) \wedge v = Q_2) \leftarrow E_2 \wedge (i : Location) = v$

$(i+1 : Location) = q \leftarrow (i : Location) = q, \text{not } (i+1 : Location) \neq q$
$(i+1 : Location) = Q_2 \leftarrow (i : E_1)$
$(i+1 : Location) = Q_1 \leftarrow (i : E_2)$
$(i+1 : Time) = (i : Time) \leftarrow i : E_1$
$(i+1 : Time) = (i : Time) \leftarrow i : E_2$
$(i+1 : Time) = y \leftarrow y = d + x, (i : Duration) = d, (i : Time) = x$

---

the following formula

$$(i+1 : Location) = (i : Location) \vee$$
$$((i+1 : Location) = Q_2 \wedge (i : E_1)) \vee$$
$$((i+1 : Location) = Q_1 \wedge (i : E_2))$$

The completion formula can be computed by SMT solvers. An output from iSAT corresponds to a path in the transition system is shown in Figure 6.15. The solution contains 5 steps and the solving time is 0.31 second.

Let H be a linear hybrid automata, T_H the labelled transition system of H, and D_H the $\mathcal{C}$+ description representing H. Let

$$p = (v_0, \boldsymbol{n}_0) \xrightarrow{\sigma_0} (v_1, \boldsymbol{n}_1) \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{m-1}} (v_m, \boldsymbol{n}_m)$$

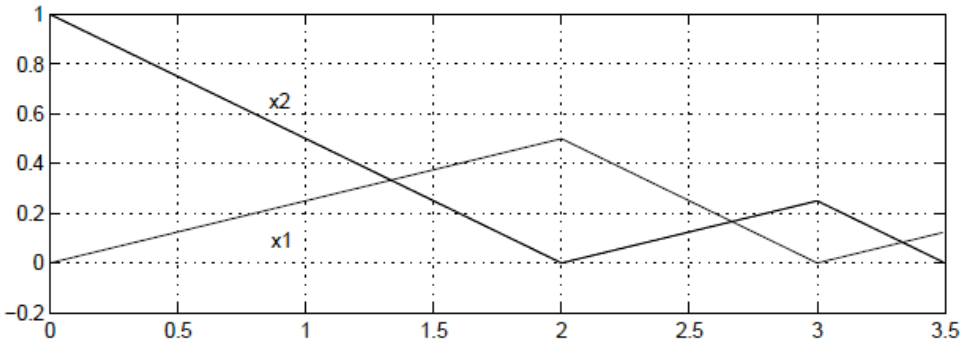

Figure 6.14: An execution of the Water Tank Example.

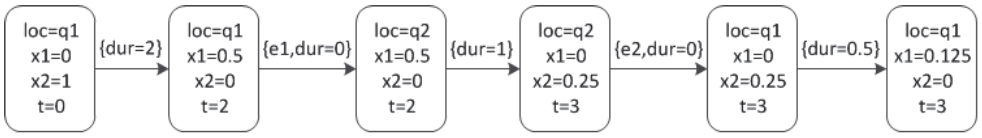

Figure 6.15: A Path in the Transition System of Water Tank Example.

be a path in T_H. We consider

$$p' = (s_0, e_0, s_1, e_1, \ldots, s_m)$$

where each s_i is an interpretation of σ^{fl} and each e_i is an interpretation of σ^{act} such that, for $i = 0, \ldots m-1$,

- $s_0 \models_{bg} (\mathrm{Loc}, \boldsymbol{X}, \mathrm{Time}) = (v_0, \boldsymbol{n}_0, 0)$;
- $s_{i+1} \models_{bg} (\mathrm{Loc}, \boldsymbol{X}, \mathrm{Time}) = (v_{i+1}, \boldsymbol{n}_{i+1}, t + d)$, where

  1) $s_i \models_{bg} \mathrm{Time} = t$, and

  2) d is σ_i if σ_i is a number, and 0 otherwise.

- 1) if $\sigma_i = \mathrm{event}(v_i, v_{i+1})$ then $e_i \models_{bg} \mathrm{Dur} = 0$, and, for all Boolean actions a, we have $e_i \models_{bg} a$ iff a is $\mathrm{event}(v_i, v_{i+1})$ or $E(v_i)$;

  2) if $\sigma_i \in \mathcal{R}_{\geq 0}$ then $e_i \models_{bg} \mathrm{Dur} = \sigma_i$, and, for all Boolean action a, $e_i \not\models_{bg} a$;

Proposition 50 *p' is a path in the transition system D_H.*

Initial conditions of a Linear Hybrid automata can be encoded in ASPMT formulas: for each $v \in V$, the formula

$$Loc = v \rightarrow init_v(X).$$

We denote the set of such formula by INIT.

Let

$$q = (s_0, e_0, s_1, e_1, \ldots, e_m)$$

be a path in the transition system of D_H such that $s_0 \models_{bg} \mathrm{INIT}$ and $v_0 = (Loc)^{s_0}$. Consider

$$q' = (v_0, \boldsymbol{n}_0) \xrightarrow{\sigma_0} (v_1, \boldsymbol{n}_1) \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{m-1}} (v_m, \boldsymbol{n}_m)$$

where each $v_i \in V$ and each $\boldsymbol{n}_i \in \mathcal{R}^n$ for $i = 0, \ldots m$,

- $s_i \models_{bg} (\mathrm{Loc}, \boldsymbol{X}) = (v_i, \boldsymbol{n}_i)$,

- 1) if $e_i \models_{bg} \mathrm{event}(v_i, v_{i+1})$ then $\sigma_i = \mathrm{event}(v_i, v_{i+1})$;

  2) otherwise $\sigma_i = \mathrm{Dur}^{e_i}$.

Proposition 51 *p' is a path in the transition system T_H.*

Note that only a fragment of the language $\mathcal{C}$+ is enough to describe Hybrid automata. Statically determined fluents, indirect effects, and concurrently executed actions are not utilized.

The completion formula can be computed by SMT solvers. An output from iSAT corresponds to a path in the transition system is shown in Figure 6.15. The solution contains 5 steps and the solving time is 0.09 second.

6.4 Related Work and Conclusion

The framework of ASPMT presented in this chapter is a novel combination of several recent developments: functional stable model semantics, constraint answer set solving, SMT solving, and hybrid reasoning about dynamic systems.

PDDL 2.1 (Fox & Long, 2003) introduced numeric fluents and durative actions to represent and reason about continuous time and resource. The framework is further extended to allow autonomous processes and event in PDDL+ (Fox & Long, 2006b). While the former is similar to our simple encoding approach, we showed that the start-process-stop model in PDDL+ can be represented in our framework by representing a process as an inertial fluent. (Shin & Davis, 2005) extended SAT-based planning framework to cover an extension of PDDL+ language using SAT-based arithmetic constraint solvers. In (Shin & Davis, 2005), durative actions are always understood as the start action, continuous action and end action.

In (Bu, Cimatti, Li, Mover, & Tonetta, 2010), the authors showed that bounded model checking of Linear Hybrid Automata can be implemented using SMT solvers. While (Bu et al., 2010) directly encode a Linear Hybrid Automata into SMT theory, we first encode it using the high level $\mathcal{C}$+ Modulo Theories and use completion to turn it into the input language of SMT solvers. Since $\mathcal{C}$+ has built-in transition semantics, we find our encoding more intuitive. On the other hand, (Bu et al., 2010) also considers shallow synchronization semantics of a Hybrid Automata network.

Our approach is similar to the action language H (Chintabathina, 2008) which is a recent extension of action language to reason about continuous process. One notable difference is there, each state represents an interval of time, rather than a particular timepoint. Instead of using SMT solvers, an implementation of H is by translation into the language $\mathcal{AC}$ (Mellarkod et al., 2008), which extends ASP with constraints. Action language $\mathcal{H}$ does not have action dynamic laws, and consequently does not allow additive fluents.

In action language H, fluents are separated into two kinds: discrete fluents and continuous fluents. While a discrete fluent describes the static properties in each state,

a continuous fluent describes a property that changes with time within a state. This is similar to the approach presented in (Pinto, 1994) in the framework of situation calculus. Natural actions and continuous changes in situation calculus were studied in (Pinto, 1994; Reiter, 1996), which do not have explicit transition system semantics. In situation calculus, a dynamic world is modeled as progressing through a series of situations as a result of various actions being performed within the world. Actions are assumed to be instantaneous and actions with durations are modeled in terms of two durationless actions which initiate and terminate a process fluent. As we show in Section 6.2, process can be modeled in our framework as well. Our language provides more flexibility by allowing durative actions as well.

Event Calculus (Kowalski & Sergot, 1986) provides a powerful framework for representing and reasoning about actions and their effects. It was extended by Shanahan (1990) to model continuous changes. As described in the water sink example in (Shanahan, 1990), within an event, some actions (like tap-on, tap-off) can initiate or terminate processes (e.g filling), which in turn can contribute to rate of change in some fluents (e.g the level of water). This is similar to our process model. The concept of an event make it flexible when dealing with concurrency. On the other hand, additive fluents in our formalism give us more natural representation when there are multiple contributing processes. In the case of (Shanahan, 1990), the filling process has to be terminated and restart with a different rate.

In (Lee & Palla, 2012b, 2012a), situation calculus and event calculus were reformulated in ASP. We expect that the techniques we developed in this chapter can be applied for more effective computation of these formalisms using SMT solvers.

6.5 Proofs
Useful Lemmas and Theorem

Lemma 83 *Formula* $\widehat{\boldsymbol{c}} = \boldsymbol{c} \to F^*(\widehat{\boldsymbol{c}}) \leftrightarrow F$ *is logically valid.*

Proof. By induction on F. ∎

Theorem 20 *[Theorem 1, (Bartholomew & Lee, 2012)] For any first-order formulas* F *and* G*, if* G *is negative on* $\boldsymbol{c}$*,* $SM[F \wedge G; \boldsymbol{c}]$ *is equivalent to* $SM[F; \boldsymbol{c}] \wedge G$*.*

Proof of Theorem 18

Let $\boldsymbol{c}$ be a list of object constants and let F be a formula that is the conjunction of formulas of the form

$$\forall \boldsymbol{z}(c = t \leftarrow B) \tag{6.18}$$

where c is an object constant, t is a term that does not contain any constant from $\boldsymbol{c}$, B is a formula and $\boldsymbol{z}$ is the set of all free variables in t and B. By F', we denote the formula obtained from F by replacing every formula (6.18) where $c \in \boldsymbol{c}$ by the formula

$$\forall x \boldsymbol{z}(c = x \leftarrow x = t \wedge B) \tag{6.19}$$

where x is a new variable. By $\mathrm{Comp}[F; \boldsymbol{c}]$, we denote the formula obtained from F' by replacing all implications (6.19) where $c \in \boldsymbol{c}$ by the formula

$$\forall x \big(c = x \leftrightarrow \bigvee_{(6.19) \in F'} \exists \boldsymbol{z}(x = t \wedge B)\big).$$

Lemma 84 *If* F *as defined above is tight, then for any interpretation* I *that satisfies* $\exists xy(x \neq y)$*,* $I \models_{bg} SM[F; \boldsymbol{c}]$ *iff* $I \models_{bg} Comp[F; \boldsymbol{c}]$*.*

Proof. It is clear that $I \models_{bg} \mathrm{SM}[F; \boldsymbol{c}]$ iff $I \models_{bg} \mathrm{SM}[F'; \boldsymbol{c}]$. F' is strongly equivalent to the conjunction of

$$\bigwedge_{c \in \boldsymbol{c}} \forall x \big(c = x \leftarrow \bigvee_{(6.19) \in F'} \exists \boldsymbol{z}(x = t \wedge B)\big) \tag{6.20}$$

and

$$\bigwedge_{c \notin \boldsymbol{c}} \forall \boldsymbol{z}(c = t \leftarrow B)\ . \tag{6.21}$$

Since (6.21) is negative on $\boldsymbol{c}$, by Theorem 20, $I \models_{bg} \mathrm{SM}[F; \boldsymbol{c}]$ iff I satisfies $\mathrm{SM}[(6.20); \boldsymbol{c}]$ and $I \models_{bg} (6.21)$. If F is tight, it is immediate that (6.20) is also tight. Since (6.20) is in Clark normal form, by Theorem 16, $I \models_{bg} \mathrm{SM}[(6.20); \boldsymbol{c}]$ iff I satisfies

$$\bigwedge_{c \in \boldsymbol{c}} \forall x \big(c = x \leftrightarrow \bigvee_{(6.19) \in F'} \exists \boldsymbol{z}(x = t \wedge B)\big) \tag{6.22}$$

$I \models_{bg} (6.21) \wedge (6.22)$ iff $I \models_{bg} \mathrm{Comp}[F; \boldsymbol{c}]$. ∎

Let D be an action description and D_m be the corresponding ASPMT formula.

$$\mathrm{Comp}[D_m;\ (0\!:\!\sigma^{sd}) \cup (0\!:\!\sigma^{act}) \cup (1\!:\!\sigma^{fl}) \cup (1\!:\!\sigma^{act}) \cup \cdots \cup (m{-}1\!:\!\sigma^{act}) \cup (m\!:\!\sigma^{fl})]$$

is the conjunction of the following formulas for $0 \leq i \leq m$ and $0 \leq j \leq m-1$:

- formula $SD(i)$, which is the conjunction of

$$\forall x \big(i\!:\!sd{=}x \ \leftrightarrow\ i\!:\! \bigvee_{\substack{\text{static law } (6.7) \in D \\ H \text{ is } sd=t}} \exists \boldsymbol{z}(x = t \wedge G)\big), \tag{6.23}$$

  for each statically determined constant sd where $\boldsymbol{z}$ is the set of all free variables in (6.7),

- formula $AD(j)$, which is the conjunction of

$$\forall x \big(j\!:\!a{=}x \ \leftrightarrow\ j\!:\! \bigvee_{\substack{\text{action dynamic law } (6.7) \in D \\ H \text{ is } a=t}} \exists \boldsymbol{z}(x = t \wedge G)\big), \tag{6.24}$$

  for each action constant a where $\boldsymbol{z}$ is the set of all free variables in (6.7),

- formula $FD(i)$, which is the conjunction of

$$\begin{aligned} \forall x \big((i{+}1)\!:\!sim{=}x \ \leftrightarrow\ (i{+}1)\!:\! &\bigvee\nolimits_{\substack{\text{static law } (6.7) \in D \\ H \text{ is } sim=t}} \exists \boldsymbol{z}(x = t \wedge G) \\ &\vee \bigvee\nolimits_{\substack{\text{fluent dynamic law } (6.8) \in D \\ H \text{ is } sim=t}} \exists \boldsymbol{z}'(x = (i{+}1)\!:\!t \wedge (i{+}1)\!:\!G \wedge i\!:\!H)\big) \end{aligned} \tag{6.25}$$

  for each simple fluent constant sim where $\boldsymbol{z}$ is the set of all free variables in (6.7) and $\boldsymbol{z}'$ is the set of all free variables in (6.8),

- formula $SIM(0)$, which is the conjunction of

$$\forall x\big(0{:}\,sim{=}x \;\leftarrow\; 0{:}\bigvee_{\substack{\text{static law } (6.7)\in D\\ H \text{ is } sim{=}t}} \exists \boldsymbol{z}(x = t \wedge G)\big) \tag{6.26}$$

  for each simple fluent constant sim where $\boldsymbol{z}$ is the set of all free variables in (6.7).

Theorem 18 *For every transition $\langle s, e, s'\rangle$, s and s' are states.*

Proof. Since $\langle s, e, s'\rangle$ is a transition, by definition,

$$0{:}s \cup 0{:}e \cup 1{:}s' \models_{bg} \mathrm{SM}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})]. \tag{6.27}$$

Since D_1 is tight, by Lemma 84, (6.27) is equivalent to

$$0{:}s \cup 0{:}e \cup 1{:}s' \models_{bg} \mathrm{Comp}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})]. \tag{6.28}$$

Note that $\mathrm{Comp}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})]$ is the conjunction of $SIM(0)$, $SD(0)$, $AD(0)$, $FD(0)$ and $SD(1)$. From (6.28), it follows that

$$0{:}s \models_{bg} SIM(0) \wedge SD(0), \tag{6.29}$$

and

$$0{:}s \cup 0{:}e \cup 1{:}s' \models_{bg} FD(0) \wedge SD(1). \tag{6.30}$$

(6.29) is the same as saying that

$$0{:}s \models_{bg} \mathrm{Comp}[D_0;\ 0{:}\sigma^{sd}]. \tag{6.31}$$

(a) From (6.30), it follows that $1 : s' \models_{bg} SD(1)$, which can be rewritten as $0 : s' \models_{bg} SD(0)$.

(b) From (6.30), it follows that $1 : s' \models_{bg} SIM(1)$, which can be rewritten as $0 : s' \models_{bg} SIM(0)$.

From (a) and (b), we conclude

$$0{:}s' \models_{bg} SIM(0) \wedge SD(0)$$

which is the same as saying that

$$0{:}s' \models_{bg} \mathrm{Comp}[D_0;\ 0{:}\sigma^{sd}]. \tag{6.32}$$

Note that D_0 is tight. From (6.31) and (6.32), by Lemma 84,

$$0{:}s \models_{bg} \mathrm{SM}[D_0;\ 0{:}\sigma^{sd}],$$
$$0{:}s' \models_{bg} \mathrm{SM}[D_0;\ 0{:}\sigma^{sd}]$$

follows. As a result, s and s' are states. ∎

Proof of Theorem 19

Theorem 19

$$\begin{aligned} &0{:}s_0 \cup 0{:}e_0 \cup 1{:}s_1 \cup 1{:}e_1 \cup \cdots \cup m{:}s_m \models_{bg} SM[D_m;\ (0{:}\sigma^{sd}) \\ &\cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl}) \cup (1{:}\sigma^{act}) \cup \cdots \cup (m{-}1{:}\sigma^{act}) \cup (m{:}\sigma^{fl})] \end{aligned} \tag{6.33}$$

iff each triple $\langle s_i, e_i, s_{i+1}\rangle$ $(0 \leq i < m)$ *is a transition.*

Proof. By induction on m.

Base Case: When $m = 1$, clear from definition.

Assumption: Assume that the statement of the theorem holds for $m = k$ $(1 \leq k)$.

Inductive Case: We will show that the statement of the theorem holds for $m = k + 1$.

From left to right: Assuming (6.33), by I.H.

$$\langle s_0, e_0, s_1\rangle, \ldots, \langle s_{k-1}, e_{k-1}, s_k\rangle$$

are transitions. We are to show that $\langle s_k, e_k, s_{k+1}\rangle$ is a transition. Since D_{k+1} is tight, by Lemma 84, (6.33) is equivalent to

$$\begin{aligned} &0{:}s_0 \cup 0{:}e_0 \cup \cdots \cup k{:}e_k \cup (k+1){:}s_{k+1} \models_{bg} \mathrm{Comp}[D_{k+1}; \\ &(0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl}) \cup (1{:}\sigma^{act}) \cup \cdots \cup (k{:}\sigma^{act}) \cup (k{+}1{:}\sigma^{fl})]. \end{aligned} \tag{6.34}$$

From (6.34), it follows that

$$k{:}s_k \models_{bg} SIM(k) \wedge SD(k), \tag{6.35}$$

$$k{:}s_k \cup k{:}e_k \models_{bg} AD(k), \tag{6.36}$$

$$k{:}s_k \cup k{:}e_k \cup (k{+}1){:}s_{k+1} \models_{bg} FD(k). \tag{6.37}$$

It is immediate from above that

$$0{:}s_k \models_{bg} SIM(0) \wedge SD(0), \tag{6.38}$$

$$0{:}s_k \cup 0{:}e_k \models_{bg} AD(0), \tag{6.39}$$

$$0{:}s_k \cup 0{:}e_k \cup 1{:}s_{k+1} \models_{bg} FD(0) \tag{6.40}$$

follows. This is equivalent to saying that

$$0{:}s_k \cup 0{:}e_k \cup 1{:}s_{k+1} \models_{bg} \mathrm{Comp}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})],$$

which by Lemma 84 is equivalent to saying that

$$0{:}s_k \cup 0{:}e_k \cup 1{:}s_{k+1} \models_{bg} \mathrm{SM}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})].$$

By definition, $\langle s_k, e_k, s_{k+1}\rangle$ is a transition.

From right to left: Assume that each triple $\langle s_i, e_i, s_{i+1}\rangle$ $(0 \leq i \leq k)$ is a transition. By I.H.

$$\begin{aligned} 0{:}s_0 \cup 0{:}e_0 \cup 1{:}s_1 \cup 1{:}e_1 \cup \cdots \cup k{:}s_k \models_{bg} \mathrm{SM}[D_k;\ (0{:}\sigma^{sd}) \\ \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl}) \cup (1{:}\sigma^{act}) \cup \cdots \cup (k{-}1{:}\sigma^{act}) \cup (k{:}\sigma^{fl})]. \end{aligned} \tag{6.41}$$

Since D_k is tight, by Lemma 84, (6.41) is equivalent to

$$\begin{aligned} 0{:}s_0 \cup 0{:}e_0 \cup 1{:}s_1 \cup 1{:}e_1 \cup \cdots \cup k{:}s_k \models_{bg} \mathrm{Comp}[D_k; \\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl}) \cup (1{:}\sigma^{act}) \cup \cdots \cup (k{-}1{:}\sigma^{act}) \cup (k{:}\sigma^{fl})]. \end{aligned} \tag{6.42}$$

Note that $\langle s_k, e_k, s_{k+1}\rangle$ is also a transition, by definition,

$$0{:}s_k \cup 0{:}e_k \cup 1{:}s_{k+1} \models_{bg} \mathrm{SM}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})]. \tag{6.43}$$

Since D_1 is tight, by Lemma 84, (6.43) is equivalent to saying that

$$0{:}s_k \cup 0{:}e_k \cup 1{:}s_{k+1} \models_{bg} \mathrm{Comp}[D_1;\ (0{:}\sigma^{sd}) \cup (0{:}\sigma^{act}) \cup (1{:}\sigma^{fl})]. \tag{6.44}$$

(6.44) is equivalent to saying (6.38), (6.39) and (6.40). It is immediate that (6.35), (6.36) and (6.37) follow. We observe that the conjunction of (6.35), (6.36), (6.37) and (6.42) is equivalent to saying that

$$\begin{aligned} &0:s_0 \cup 0:e_0 \cup 1:s_1 \cup 1:e_1 \cup \cdots \cup (k+1):s_{k+1} \models_{bg} \mathrm{Comp}[D_m; \\ &(0:\sigma^{sd}) \cup (0:\sigma^{act}) \cup (1:\sigma^{fl}) \cup (1:\sigma^{act}) \cup \cdots \cup (k:\sigma^{act}) \cup (k+1:\sigma^{fl})]. \end{aligned} \tag{6.45}$$

This follows from the fact that (6.42) implies $k:s_k \models_{bg} SD(k) \wedge FD(k)$, which in turn implies (6.35). From (6.45), by Lemma 84 again, we derive (6.33). ∎

Proof of Proposition 50

Lemma 85 *Let H be a linear hybrid automaton and*

$$(v, \boldsymbol{n}) \xrightarrow{\sigma} (v, \boldsymbol{n}')$$

be a transition in T_H such that $\sigma \in \mathcal{R}_{\geq 0}$. $f(t) = \boldsymbol{n} + t * (\boldsymbol{n}' - \boldsymbol{n})/\sigma$ is a linear differentiable function from $[0, \sigma]$ to $\mathcal{R}^n$, with the first derivative $\dot{f} : [0, \sigma] \rightarrow \mathcal{R}^n$ such that: (1) $f(0) = \boldsymbol{n}$ and $f(\sigma) = \boldsymbol{n}'$ and (2) for all reals $\epsilon \in (0, \sigma)$, both $inv(v)(f(\epsilon))$ and $flow(v)(\dot{f}(\epsilon))$ are true.

Proof. We check that f satisfies the above conditions:

- $f(t)$ is differentiable over $[0, \sigma]$.
- It is clear that $f(0) = \boldsymbol{n}$ and $f(\sigma) = \boldsymbol{n}'$.
- Since $(v, \boldsymbol{n})$ and $(v', \boldsymbol{n}')$ are states of T_H, it follows that $inv_v(f(0))$ and $inv_v(f(\sigma))$ are true. Since $inv_{v_i}(\boldsymbol{X})$ is a conjunction of linear inequalities, the values of $\boldsymbol{X}$ that satisfies $inv_{v_i}(\boldsymbol{X})$ must form a convex region in $\mathcal{R}^n$. Since $f(t)$ is a linear function, it follows that for any $\epsilon \in (0, \sigma)$, $inv_v(f(\epsilon))$ is true.
- Since $(v, \boldsymbol{n}) \xrightarrow{\sigma} (v, \boldsymbol{n}')$ is a transition in T_H, it follows that there is a function f' such that (1) f' is differentiable in $[0, \sigma]$, (2) for any $\epsilon \in (0, \sigma)$, $flow_v(\dot{f}'(\epsilon))$ is true, (3) $f'(0) = \boldsymbol{n}$ and $f'(\sigma) = \boldsymbol{n}'$. Since f' is continuous on $[0, \sigma]$ (differentiability implies continuity) and differentiable on $(0, \sigma)$, by mean value theorem[3], there is a

[3] http://en.wikipedia.org/wiki/Mean_value_theorem

point $c \in (0, \sigma)$ such that $\dot{f}'(c) = (\boldsymbol{n}' - \boldsymbol{n})/\sigma$. Consequently, $flow_v((\boldsymbol{n}' - \boldsymbol{n})/\sigma)$ is true. As a result, we get $flow(v)(\dot{f}(\epsilon))$ is true for all $\epsilon \in (0, \sigma)$.

∎

Lemma 86 *For each $i \geq 0$, s_i is a state in the transition system of D_H.*

Proof. By definition, we are to show that

$$0 : s_i \models_{bg} \mathrm{SM}[(D_H)_0; \emptyset], \tag{6.46}$$

while $\mathrm{SM}[(D_H)_0; \emptyset]$ is equivalent to the conjunctions of

$$0 : inv_v(\boldsymbol{X}) \leftarrow 0 : Loc = v \tag{6.47}$$

for each location $v \in V$. Since p is a path, for each $i \geq 0$, $(v_i, \boldsymbol{n}_i)$ is a state in T_H. By the definition of hybrid transition systems, $inv_{v_i}(\boldsymbol{n}_i)$ is true. Note that $s_i \models_{bg} (\mathrm{Loc}, \boldsymbol{X}) = (v_i, \boldsymbol{n}_i)$. It is clear that $0 : s_i \models_{bg} (6.47)$. ∎

Lemma 87 *For each $i \geq 0$, $\langle s_i, e_i, s_{i+1} \rangle$ is a transition.*

Proof. By definition, we are to show that

$$0 : s_i \cup 0 : e_i \cup 1 : s_{i+1} \models_{bg} \mathrm{SM}[(D_H)_1; 0 : \sigma^{act} \cup 1 : \sigma^{sim}]. \tag{6.48}$$

We check that $(D_H)_1$ is tight. By Lemma 84, (6.48) is equivalent to

$$0 : s_i \cup 0 : e_i \cup 1 : s_{i+1} \models_{bg} \mathrm{Comp}[(D_H)_1; 0 : \sigma^{act} \cup 1 : \sigma^{sim}] \tag{6.49}$$

where $\mathrm{Comp}[(D_H)_1; 0 : \sigma^{act} \cup 1 : \sigma^{sim}]$ is equivalent to the conjunction of the following formulas:

- Formula INV, which is the conjunction of

$$k : inv_v(\boldsymbol{X}) \leftarrow k : Loc = v \tag{6.50}$$

for each $k \in \{0, 1\}$ and each $v \in V$;

- Formula FLOW, which is the conjunction of

$$flow_v((1{:}\boldsymbol{X} - 0{:}\boldsymbol{X})/t) \leftarrow 1{:}Loc{=}v \wedge 0{:}Loc{=}v \wedge 0{:}Dur{=}t \wedge t > 0 \tag{6.51}$$

and

$$1{:}\boldsymbol{X} = 0{:}\boldsymbol{X} \leftarrow \bigwedge_{\substack{(v,v')\in E \\ \text{for some } v'}} \neg\, 0{:}event(v,v') \wedge 1{:}Loc{=}v \wedge 0{:}Loc{=}v \wedge 0{:}Dur{=}0 \tag{6.52}$$

for each $v \in V$;

- Formula JUMP, which is the conjunction of

$$\begin{aligned} jump_{(v,v')}(0{:}\boldsymbol{X}, 1{:}\boldsymbol{X}) \wedge 0{:}Loc{=}v \wedge 1{:}Loc{=}v' \wedge 0{:}Dur{=}0 \\ \leftarrow 0{:}event(v,v') \end{aligned} \tag{6.53}$$

for each $(v, v') \in E$;

- Formula LOC, which is the conjunction of

$$1{:}Loc{=}v \rightarrow 0{:}Loc{=}v \vee \bigvee_{\substack{(v',v)\in E \\ \text{for some } v'}} 0{:}event(v',v) \tag{6.54}$$

for each $v \in V$;

- Formula EV, which is the conjunction of

$$0{:}E(v) \leftrightarrow \bigvee_{\substack{(v,v')\in E \\ \text{for some } v'}} 0{:}event(v,v') \tag{6.55}$$

for each location $v \in V$;

- Formula TIME, which is the formula

$$1{:}Time = 0{:}Time + 0{:}Dur. \tag{6.56}$$

We will check in the following that $0{:}s_i \cup 0{:}e_i \cup 1{:}s_{i+1}$ satisfies each one of them.

INV: From the fact that $(v_i, \boldsymbol{n}_i)$ and $(v_{i+1}, \boldsymbol{n}_{i+1})$ are states in T_H, by the definition of hybrid transition systems, $inv_{v_i}(\boldsymbol{n}_i)$ and $inv_{v_{i+1}}(\boldsymbol{n}_{i+1})$ are true. Note that $s_i \models_{bg} (\mathrm{Loc}, \boldsymbol{X}) = (v_i, \boldsymbol{n}_i)$ and $s_{i+1} \models_{bg} (\mathrm{Loc}, \boldsymbol{X}) = (v_{i+1}, \boldsymbol{n}_{i+1})$. As a result,

$$\begin{aligned} 0 : s_i &\models_{bg} 0{:}inv_v(\boldsymbol{X}) \leftarrow 0{:}Loc{=}v \\ 1 : s_{i+1} &\models_{bg} 1{:}inv_v(\boldsymbol{X}) \leftarrow 1{:}Loc{=}v. \end{aligned}$$

TIME: From the construction of s_i and s_{i+1}, $\mathrm{Time}^{s_{i+1}} = \mathrm{Time}^{s_i} + \mathrm{Dur}^{e_i}$. It is clear that $0{:}s_i \cup 0{:}e_i \cup 1{:}s_{i+1} \models_{bg} \mathrm{TIME}$.

We will show that $0 : s_i \cup 0 : e_i \cup 1 : s_{i+1}$ satisfies FLOW, JUMP, LOC and EV. From the definition of T_H, there are two cases for the value of σ_i:

Case 1: $\sigma_i = event(v_i, v_{i+1})$. It follows from the construction of p' that $(\mathrm{Dur})^{e_i} = 0$, $(event(v_i, v_{i+1}))^{e_i} = \boldsymbol{t}$ and $(event(v, v'))^{e_i} = \boldsymbol{f}$ for every other $(v, v') \in E$. Since FLOW is trivially satisfied, it is sufficient to consider only JUMP, LOC and EV.

From the fact that

$$(v_i, \boldsymbol{n}_i) \xrightarrow{\sigma_i} (v_{i+1}, \boldsymbol{n}_{i+1})$$

is a transition in T_H and that $\sigma_i = event(v_i, v_{i+1})$, it follows from the definition of hybrid transition systems that $jump_{(v_i, v_{i+1})}(\boldsymbol{n}_i, \boldsymbol{n}_{i+1})$ is true.

- JUMP: Note that $s_i \models_{bg} (\mathrm{Loc}, \boldsymbol{X}) = (v_i, \boldsymbol{n}_i)$ and $s_{i+1} \models_{bg} (\mathrm{Loc}, \boldsymbol{X}) = (v_{i+1}, \boldsymbol{n}_{i+1})$. It is immediate that $0{:}s_i \cup 1{:}s_{i+1} \models_{bg} jump_{(v_i, v_{i+1})}(0{:}\boldsymbol{X}, 1{:}\boldsymbol{X})$, $0{:}s_i \models_{bg} 0{:}Loc = v_i$ and $1{:}s_{i+1} \models_{bg} 1{:}Loc = v_{i+1}$. Since $\mathrm{Dur}^{e_i} = 0$ and $(event(v_i, v_{i+1}))^{e_i} = \boldsymbol{t}$, it is follows that $0{:}s_i \cup 0{:}e_i \cup 1{:}s_{i+1} \models_{bg} \mathrm{JUMP}$.
- LOC: Since $(\mathrm{Loc})^{s_{i+1}} = v_{i+1}$ and $(event(v_i, v_{i+1}))^{e_i} = \boldsymbol{t}$, it follows that $0{:}s_i \cup 0{:}e_i \cup 1{:}s_{i+1} \models_{bg} \mathrm{LOC}$ (take v to be v_{i+1}).
- EV: Since $(event(v_i, v_{i+1}))^{e_i} = \boldsymbol{t}$, $E(v_i)^{e_i} = \boldsymbol{t}$ and $a^{e_i} = \boldsymbol{f}$ for any other action a. It is clear that $0{:}e_i \models_{bg} \mathrm{EV}$ (take v to be v_i).

Case 2: $\sigma_i \in \mathcal{R}_{\geq 0}$. By the construction of p', $Dur^{e_i} = \sigma_i$ and $(event(v, v'))^{e_i} = \boldsymbol{f}$ for every $(v, v') \in E$. Since JUMP is trivially satisfied, it is sufficient to consider only FLOW and LOC.

From the fact that

$$(v_i, \boldsymbol{n}_i) \xrightarrow{\sigma_i} (v_{i+1}, \boldsymbol{n}_{i+1})$$

is a transition of T_H and that $\sigma_i \in \mathcal{R}_{\geq 0}$, it follows from the definition of hybrid transition systems that

(a) $v_i = v_{i+1}$, and

(b) there is a differentiable function $f : [0, \sigma_i] \to \mathcal{R}^n$, with the first derivative $\dot{f} : [0, \sigma_i] \to \mathcal{R}^n$ such that: (1) $f(0) = \boldsymbol{n}_i$ and $f(\sigma_i) = \boldsymbol{n}_{i+1}$ and (2) for all reals $\epsilon \in (0, \sigma_i)$, both $inv_v(f(\epsilon))$ and $flow_v(\dot{f}(\epsilon))$ are true.

We check the following:

- FLOW: consider two cases.
  - If $\sigma_i = 0$ then $Dur^{e^i} = 0$. It is sufficient to consider only (6.52). From (b), $\boldsymbol{n}_i = \boldsymbol{n}_{i+1} = f(0)$. Since $(\boldsymbol{X})^{s_i} = (\boldsymbol{X})^{s_{i+1}}$, it follows that $0 : s_i \cup 0 : e_i \cup 1 : s_{i+1} \models_{bg} (6.52)$.
  - If $\sigma_i > 0$, then $Dur^{e^i} > 0$. By Lemma 85, $f(t) = \boldsymbol{n}_i + t * (\boldsymbol{n}_{i+1} - \boldsymbol{n}_i)/\sigma_i$ is a differentiable function that satisfies all the conditions in (b). As a result, $flow_{v_i}((\boldsymbol{n}_{i+1} - \boldsymbol{n}_i)/\sigma_i)$ is true and thus $0 : s_i \cup 0 : e_i \cup 1 : s_{i+1} \models_{bg} flow_{v_i}((1 : \boldsymbol{X} - 0 : \boldsymbol{X})/Dur)$. It follows that $0 : s_i \cup 0 : e_i \cup 1 : s_{i+1} \models_{bg} (6.51)$.
- LOC: From $v_i = v_{i+1}$ and that $Loc^{s_i} = v_i$, $Loc^{s_{i+1}} = v_{i+1}$, it follows that $0 : s_i \cup 0 : e_i \cup 1 : s_{i+1} \models_{bg} \mathrm{LOC}$.
- EV: Since $a^{e_i} = \boldsymbol{f}$ for any other action a. It is clear that $0 : e_i \models_{bg} \mathrm{EV}$.

∎

Proposition 50 *p' is a path in the transition system of D_H.*

Proof. By Lemma 86, each s_i is a state of D_H. By Lemma 87, each $\langle s_i, e_i, s_{i+1} \rangle$ is a transition of D_H. So p' is a path in the transition system of D_H. ∎

Proof of Proposition 51

Lemma 88

(a) For each $i \geq 0$, $(v_i, \boldsymbol{n}_i)$ is a state in T_H, and

(b) $(v_0, \boldsymbol{n}_0)$ *is an initial state in* T_H*.*

Proof.

(a) By definition, we are to show that $inv_{v_i}(\boldsymbol{n}_i)$ is true. Since each s_i is a state in the transition system of D_H, by definition,

$$0{:}s_i \models_{bg} \mathrm{SM}[(D_H)_0; \emptyset]. \tag{6.57}$$

Note that $\mathrm{SM}[(D_H)_0; \emptyset]$ is equivalent to the conjunction of the formula:

$$0{:}inv_v(\boldsymbol{X}) \leftarrow 0{:}Loc = v \tag{6.58}$$

for each location $v \in V$. Since $(Loc)^{s_i} = v_i$, it follows that $s_i \models_{bg} inv_{v_i}(\boldsymbol{X})$. Since $\boldsymbol{X}^{s_i} = \boldsymbol{n}_i$, it follows that $inv_{v_i}(\boldsymbol{n}_i)$ is true.

(b) We are to show that $init_{v_0}(\boldsymbol{n}_0)$ is true. This is clear from the fact that $s_0 \models_{bg} \mathrm{INIT}$ and $(\boldsymbol{X})^{s_0} = \boldsymbol{n}_0$. ∎

Lemma 89 *For each* $0 \leq i \leq m$*,*

$$(v_i, \boldsymbol{n}_i) \xrightarrow{\sigma_i} (v_{i+1}, \boldsymbol{n}_{i+1}) \tag{6.59}$$

is a transition in T_H*.*

Proof. From the fact that (s_i, e_i, s_{i+1}) is a transition of D_H, by definition, we know that

$$0{:}s_i \cup 0{:}e_i \cup 1{:}s_{i+1} \models_{bg} \mathrm{SM}[(D_H)_1; 0{:}\sigma^{act} \cup 1{:}\sigma^{sim}]. \tag{6.60}$$

Since $(D_H)_1$ is tight, by Lemma 84, (6.60) holds iff

$$0{:}s_i \cup 0{:}e_i \cup 1{:}s_{i+1} \models_{bg} \mathrm{Comp}[(D_H)_1; 0{:}\sigma^{act} \cup 1{:}\sigma^{sim}]. \tag{6.61}$$

where $\mathrm{Comp}[(D_H)_1; 0{:}\sigma^{act} \cup 1{:}\sigma^{sim}]$ is equivalent to the conjunction of the formulas INV, FLOW, JUMP, LOC, TIME and EV.

Since $0 : e_i \models_{bg} \mathrm{JUMP}$, it is follows that if $(event(v, v'))^{e_i} = \boldsymbol{t}$ for some v' then $Loc^{s_i} = v$, $Loc^{s_{i+1}} = v'$. As a result, there can be at most one $event(v, v')$ such that $(event(v, v'))^{e_i} = \boldsymbol{t}$. Consider two cases:

Case 1: There exists an edge (v, v') such that $(event(v, v'))^{e_i} = \boldsymbol{t}$. From the above discussion, we know that $event(v, v')$ is the only event that is true. Since $Loc^{s_i} = v_i$ and $Loc^{s_{i+1}} = v_{i+1}$, it follows that (v, v') must be (v_i, v_{i+1}). As a result, $(event(v_i, v_{i+1}))^{e_i} = \boldsymbol{t}$ and $(event(v, v'))^{e_i} = \boldsymbol{f}$ for every other $(v, v') \in E$. It follows from the definition that $\sigma_i = event(v_i, v_{i+1})$.

(a) Since $0{:}s_i \cup 0{:}e_i \cup 1{:}s_{i+1} \models_{bg} \mathrm{JUMP}$, $\boldsymbol{X}^{s_{i+1}} = \boldsymbol{n}_{i+1}$ and $\boldsymbol{X}^{s_i} = \boldsymbol{n}_i$, it is immediate that $jump_{(v_i, v_{i+1})}(\boldsymbol{n}_{i+1}, \boldsymbol{n}_i)$ is true.

(b) By Lemma 88, $(v_i, \boldsymbol{n}_i)$ and $(v_{i+1}, \boldsymbol{n}_{i+1})$ are states.

From (a), (b) and the fact that $\sigma_i = event(v_i, v_{i+1})$, we conclude (6.59) is a transition.

Case 2: $(event(v, v'))^{e_i} = \boldsymbol{f}$ for every $(v, v') \in E$. By construction, $(Dur)^{e_i} = \sigma_i$ for some $\sigma_i \in \mathcal{R}_{\geq 0}$. By Lemma 88, $(v_i, \boldsymbol{n}_i)$ and $(v_{i+1}, \boldsymbol{n}_{i+1})$ are states. From LOC, it follows that $Loc^{s_i} = Loc^{s_{i+1}}$. As a result, $v_i = v_{i+1}$. We are to show that there is a differentiable function $f : [0, \sigma_i] \to \mathcal{R}^n$, with the first derivative $\dot{f} : [0, \sigma_i] \to \mathcal{R}^n$ such that: (1) $f(0) = \boldsymbol{n}_i$ and $f(\sigma_i) = \boldsymbol{n}_{i+1}$ and (2) for all reals $\epsilon \in (0, \sigma_i)$, both $inv_{v_i}(f(\epsilon))$ and $flow_{v_i}(\dot{f}(\epsilon))$ are true. Define $f(t) = \boldsymbol{n}_i + t * (\boldsymbol{n}_{i+1} - \boldsymbol{n}_i)/\sigma_i$. We check that f satisfies the above conditions:

- $f(t)$ is differentiable over $[0, \sigma_i]$.
- It is clear that $f(0) = \boldsymbol{n}_i$ and $f(\sigma_i) = \boldsymbol{n}_{i+1}$.
- We check that for any $\epsilon \in (0, \sigma)$, $inv_v(f(\epsilon))$ is true. From $i : s_i \cup (i+1) : s_{i+1} \models_{bg} (6.50)$, it follows that $inv_{v_i}(f(0))$ and $inv_{v_i}(f(\sigma_i))$ are true. Since $inv_{v_i}(\boldsymbol{X})$ is a conjunction of linear inequalities, the values of $\boldsymbol{X}$ that satisfies $inv_{v_i}(\boldsymbol{X})$ must form a convex region in $\mathcal{R}^n$. Since $f(t)$ is a linear function, it follows that for any $\epsilon \in (0, \sigma)$, $inv_v(f(\epsilon))$ is true.
- We check that for any $\epsilon \in (0, \sigma)$, $flow_{v_i}(\dot{f}(\epsilon))$ is true. We only consider the case where $\sigma_i > 0$ because otherwise is trivial (there is no $\epsilon \in (0, 0)$). From (6.51), it follows that $flow_{v_i}((f(\sigma_i) - f(0))/\sigma_i)$ is true. Since $f(t)$ is a linear function, it

follows that for any $\epsilon \in (0, \sigma_i)$, $\dot{f}(\epsilon) = (f(\sigma_i) - f(0))/\sigma_i$. As a result, $flow_{v_i}(\dot{f}(\epsilon))$ is true.

From above, we conclude that (6.59) is a transition. ∎

Proposition 51 *q' is a path in T_H.*

Proof. By Lemma 88 (a), each $(v_i, \boldsymbol{n}_i)$ is a state in T_H. By Lemma 89, each $(v_i, \boldsymbol{n}_i) \xrightarrow{\sigma_i} (v_{i+1}, \boldsymbol{n}_{i+1})$ is a transition in T_H. So q' is a path in T_H. If $s_0 \models_{bg} init_{v_0}$ then by Lemma 88 (b), $(v_0, \boldsymbol{n}_0)$ is a initial state in T_H. ∎

Proof of Proposition 49

Lemma 90 *For any clingcon program Π with CSP (V, D, C), any interpretation $I = \langle A, X \rangle$ of the signature $V \cup \boldsymbol{p}$, let Y be sets of X such that. Consider any rule $H \leftarrow B, N, Cn$ in Π,*

$$Y \models_{bg} (H \leftarrow B, N, Cn)^X_A$$

iff

$$\langle A, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models_{bg} (B \wedge N \wedge Cn)^*(\boldsymbol{q}) \rightarrow H^*(\boldsymbol{q}). \tag{6.62}$$

Proof. By Theorem 20, $\langle A, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models_{bg} N^*(\boldsymbol{q})$ iff $\langle A, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models_{bg} N$. Since Cn contains no intentional constants, $Cn^*(\boldsymbol{q})$ is the same as Cn. So (6.62) can be equivalently rewritten as

$$\langle A, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models_{bg} B^*(\boldsymbol{q}) \wedge N \wedge Cn \rightarrow H^*(\boldsymbol{q}). \tag{6.63}$$

Consider two cases.

Case 1: $A \not\models_{bg} Cn$ or $X \not\models_{bg} N$, $(H \leftarrow B, N, Cn)^X_A$ is equivalent to $\top$. It follows that $\langle A, X \cup Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \cup I^{bg} \not\models_{bg} N \wedge Cn$. Clearly, (6.63) holds.

Case 2: Otherwise, $A \models_{bg} Cn$ and $X \models_{bg} N$. As a result, $\langle A, X \rangle \models_{bg} N \wedge Cn$. $(H \leftarrow B, N, Cn)^X_A$ is $H \leftarrow B$. $Y \models_{bg} H \leftarrow B$ is equivalent to $Y^{\boldsymbol{p}}_{\boldsymbol{q}} \models_{bg} H^*(\boldsymbol{q}) \leftarrow B^*(\boldsymbol{q})$. And thus $\langle A, Y^{\boldsymbol{p}}_{\boldsymbol{q}} \rangle \models_{bg} B^*(\boldsymbol{q}) \rightarrow H^*(\boldsymbol{q})$, which in turn is equivalent to (6.63) when $\langle A, X \rangle \models_{bg} N \wedge Cn$. ∎

Lemma 91 *For any clingcon program* Π *with CSP* (V, D, C)*, any interpretation* $I = \langle A, X\rangle$ *of the signature* $V \cup \boldsymbol{p}$*,* $I \models_{bg} \Pi$ *iff* X *satisfies* Π_A^X*.*

Proof. Immediate from Lemma 90 and Lemma 83 when $Y = X$ and $\boldsymbol{p} = \boldsymbol{q}$. ∎

Proposition 49 *Let* Π *be a clingcon program with CSP* (V, D, C)*, let* $\boldsymbol{p}$ *be a set of propositional constants that occur in* Π *and* $I = \langle A, X\rangle$ *an interpretation of the signature* $V \cup \boldsymbol{p}$*.* $I \models_{bg} SM[\Pi; \boldsymbol{p}]$ *iff* X *is a constraint answer set of* Π *relative to* A*.*

Proof. X is a constraint answer set of Π relative to A iff

(i) $X \models_{bg} \Pi_A^X$, and

(ii) no proper subset Y of X satisfies Π_A^X.

On the other hand, $I \models_{bg} \mathrm{SM}[\Pi; \boldsymbol{p}]$ iff

(i') $I \models_{bg} \Pi$, and

(ii') I does not satisfy $\exists \boldsymbol{u}(\boldsymbol{u} < \boldsymbol{p} \wedge \Pi^*(\boldsymbol{u}))$.

It follows from Lemma 91 that condition (i) is equivalent to condition (i'). Assume (i'). Condition (ii) can be reformulated as: no proper subset Y of X satisfies rule $(H \leftarrow B, N, Cn)_A^X$ for each $H \leftarrow B, N, Cn \in \Pi$. Under the assumption (i'), condition (ii') can be reformulated as: there is no proper subset Y of X such that, for every rule $H \leftarrow B, N, Cn$ in Π, $\langle A, X \cup Y_{\boldsymbol{q}}^{\boldsymbol{p}}\rangle \models_{bg} (B \wedge N \wedge Cn)^*(\boldsymbol{q}) \rightarrow H^*(\boldsymbol{q})$. By Lemma 90, (ii) is equivalent to (ii'). ∎

Chapter 7

Conclusion

In this dissertation, we proposed a framework for extending ASP language and for integrating it with other computing paradigms. We observed that the traditional ASP semantics is too restrictive for new constructs like aggregates, dl-atoms and constraints. Also, a systematic way to study different extensions is needed.

To overcome the limitations in the existing extensions, we extend the first-order stable model semantics to formulas with generalized quantifiers, which cover aggregates, DL-atoms, constraints and SMT theory atoms as special cases. The framework provides a new top-down perspective on extensions of ASP and a systematic approach to study and extend non-monotonic languages.

This dissertation contributes to the researches in the area of logic programming and reasoning about actions in the following ways:

- By relating the first-order stable model semantics to first-order logic via loop formulas, we provided useful insights into first-order reasoning with stable models. We showed that when a formula is bounded or has finite complete set of loop, the reasoning under the first-order stable model semantics can be reduced to reasoning in first-order logic. The results were extended to cover aggregates and generalized quantifiers. These results allow us to compute non-Herbrand stable models using solvers from other computing paradigms.

- By reformulating the existing semantics on programs with aggregates in terms of propositional formulas, we presented a systematic way to study the properties of each semantics using the underlying general language. Guided by the reduction, we lifted up the stable model semantics and FLP semantics of aggregates to the first-order level. The general semantics avoid the issues of grounding and provide natural semantics for arbitrary recursive and nested aggregates. The study paved ways to more general semantics that unify the extensions of ASP.

- To provide a systematic approach for studying the other extensions of ASP, we ex-

tended the first-order semantics to formulas with generalized quantifiers and showed that the unifying framework naturally covers aggregates, DL-atoms, constraints and SMT theory atoms as special cases. The framework provides a theoretic foundation of extending ASP with other computing paradigms via a first-order semantics. This overcomes the limitation of existing approaches that are based on propositional ones. We also illustrated the unifying view by generalizing several important properties, such as the splitting theorem, the theorem on completion, the theorem on strong equivalence, the theorem on safety and the theorem on loop formulas to formulas with generalized quantifiers, which in turn can be applied to existing individual extensions. This saves the efforts of reproving them in each context.

- Towards a tight integration of ASP with SMT, we presented the ASPMT framework which allows us to represent non-monotonic functions with background interpretations. We bridged the gap between traditional ASP and functional view in constraints by referring to functional stable model semantics. Using the framework, we enhanced action language $\mathcal{C}+$ to handle reasoning about continuous changes. We demonstrated the expressiveness of the new language by modeling domains that reason about continuous changes, additive fluents and process. We also showed the computation advantage of the framework using some benchmark examples. Preliminary experiment results shows that our approach outperform existing implementations of $\mathcal{C}+$ by several orders of magnitude.

Some results presented in the dissertation were in the following venue:

- Relating first-order stable model semantics to first-order logic: (Lee & Meng, 2008, 2011).
- On semantics of aggregates: (Lee & Meng, 2009; Bartholomew et al., 2011).
- First-order stable model semantics for generalized quantified formulas: (Lee & Meng, 2012a, 2012b, 2012c).

Armando, A., & Compagna, L. (2002). Automatic sat-compilation of protocol insecurity via reduction to planning. In *Proceedings of the Joint International Conference on Formal Techniques for Networked and Distributed Systems 2002*, pp. 210–225.

Artikis, A., Sergot, M., & Pitt, J. (2003). Specifying electronic societies with the Causal Calculator. In Giunchiglia, F., Odell, J., & Weiss, G. (Eds.), *Proceedings of Workshop on Agent-Oriented Software Engineering III (AOSE)*, LNCS 2585, pp. 1–15. Springer.

Asuncion, V., Lin, F., Zhang, Y., & Zhou, Y. (2010). Ordered completion for first-order logic programs on finite structures. In *AAAI*, pp. 249–254.

Balduccini, M. (2009). Representing constraint satisfaction problems in answer set programming. In *Working Notes of the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*.

Baral, C., & Gelfond, M. (2000). Reasoning agents in dynamic domains. In Minker, J. (Ed.), *Logic-Based Artificial Intelligence*, pp. 257–279. Kluwer.

Baral, C., Gelfond, M., & Provetti, A. (1997). Reasoning about actions: laws, observations and hypotheses. *Journal of Logic Programming*, *31*, 201–244.

Bartholomew, M., & Lee, J. (2012). Stable models of formulas with intensional functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 2–12.

Bartholomew, M., Lee, J., & Meng, Y. (2011). First-order extension of the FLP stable model semantics via modified circumscription. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 724–730.

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.

Biere, A., Biere, A., Heule, M., van Maaren, H., & Walsh, T. (2009). *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands.

Bonatti, P. A. (2004). Reasoning with infinite stable models. *Artificial Intelligence*, *156*(1), 75–111.

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.

Bu, L., Cimatti, A., Li, X., Mover, S., & Tonetta, S. (2010). Model checking of hybrid systems using shallow synchronization. In *Proceedings of the 12th IFIP WG 6.1 international conference and 30th IFIP WG 6.1 international conference on Formal Techniques for Distributed Systems*, FMOODS'10/FORTE'10, pp. 155–169, Berlin, Heidelberg. Springer-Verlag.

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., & Patoglu, V. (2009). Bridging the gap between high-level reasoning and low-level control. In *LPNMR*, pp. 342–354.

Chen, Y., Lin, F., Wang, Y., & Zhang, M. (2006). First-order loop formulas for normal logic programs. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 298–307.

Chen, Y., Lin, F., Zhang, Y., & Zhou, Y. (2011). Loop-separable programs and their first-order definability. *Artificial Intelligence*, *175*(3-4), 890–913.

Chen, Y., Zhang, Y., & Zhou, Y. (2010). First-order indefinability of answer set programs on finite structures. In *AAAI*, pp. 285–290.

Chintabathina, S. (2008). Towards answer set prolog based architectures for intelligent agents.. In *AAAI'08*, pp. 1843–1844.

Clark, K. (1978). Negation as failure. In Gallaire, H., & Minker, J. (Eds.), *Logic and Data Bases*, pp. 293–322. Plenum Press, New York.

Eiter, T., Fink, M., Ianni, G., Krennwallner, T., & Schüller, P. (2011). Pushing efficient evaluation of hex programs by modular decomposition. In *Proceedings of the 11th international conference on Logic programming and nonmonotonic reasoning*, LPNMR'11, pp. 93–106, Berlin, Heidelberg. Springer-Verlag.

Eiter, T., Gottlob, G., & Veith, H. (1997a). Generalized quantifiers in logic programs. In *In Proceedings of the ESSLLI Workshop on Generalized Quantifiers, Aix-en-Provence*, pp. 72–98. Springer.

Eiter, T., Gottlob, G., & Veith, H. (1997b). Modular logic programming and generalized quantifiers.. In *LPNMR'97*, pp. 290–309.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008a). Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, *172*(12-13), 1495–1539.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008b). 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., Ianni, G., Schindlauer, R., & Tompits, H. (2006a). Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *ESWC*, pp. 273–287.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2006b). Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *ESWC*, pp. 273–287.

Eiter, T., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2004). Combining answer set programming with description logics for the semantic web. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Faber, W. (2005). Unfounded sets for disjunctive logic programs with arbitrary aggregates. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 40–52.

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)*.

Faber, W., Pfeifer, G., & Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, *175*(1), 278–298.

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. (2011a). Stable models and circumscription. *Artificial Intelligence*, *175*, 236–263.

Ferraris, P., Lee, J., & Lifschitz, V. (2011b). Stable models and circumscription. *Artificial Intelligence*, *175*, 236–263.

Ferraris, P., Lee, J., Lifschitz, V., & Palla, R. (2009a). Symmetric splitting in the general theory of stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 797–803.

Ferraris, P., Lee, J., Lifschitz, V., & Palla, R. (2009b). Symmetric splitting in the general theory of stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 797–803.

Ferraris, P., & Lifschitz, V. (2010). On the stable model semantics of firsr-order formulas with aggregates. In *NMR*.

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.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, *20*, 61–124.

Fox, M., & Long, D. (2006a). Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)*, *27*, 235–297.

Fox, M., & Long, D. (2006b). Modelling mixed discrete-continuous domains for planning. *J. Artif. Int. Res.*, *27*(1), 235–297.

Gebser, M., Ostrowski, M., & Schaub, T. (2009). Constraint answer set solving. In *Proceedings of International Conference on Logic Programming (ICLP)*, pp. 235–249.

Gebser, M., Lee, J., & Lierler, Y. (2006). Elementary sets for logic programs. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*.

Gebser, M., Lee, J., & Lierler, Y. (2011). On elementary loops of logic programs. *Theory and Practice of Logic Programming*, *11(6)*, 953–988.

Gebser, M., & Schaub, T. (2005). Loops: Relevant or redundant?. In *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, pp. 53–65.

Gelfond, M. (1993). Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence*.

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[1]. *Electronic Transactions on Artificial Intelligence*, *3*, 195–210.

Gelfond, M., & Lobo, J. (2008). Authorization and obligation policies in dynamic systems. In *ICLP*, pp. 22–36.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, *153(1–2)*, 49–104.

Giunchiglia, E., Lee, J., Lifschitz, V., & Turner, H. (2001). Causal laws and multi-valued fluents[2]. Unpublished draft.

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.

Henzinger, T. A. (1996). The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pp. 278–292.

Heymans, S., de Bruijn, J., Predoiu, L., Feier, C., & Nieuwenborgh, D. V. (2008). Guarded hybrid knowledge bases. *TPLP*, *8*(3), 411–429.

Hoehndorf, R., Loebe, F., Kelso, J., & Herre, H. (2007). Representing default knowledge in biomedical ontologies: application to the integration of anatomy and phenotype ontologies. *BMC Bioinformatics*, *8*, 1–12.

[1] http://www.ep.liu.se/ea/cis/1998/016/

[2] http://www.cs.utexas.edu/users/vl/papers/clmvf-long.ps

Janhunen, T., Liu, G., & Niemela, I. (2011). Tight integration of non-ground answer set programming and satisfiability modulo theories. In *Working notes of the 1st Workshop on Grounding and Transformations for Theories with Variables*.

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.

Karp, C. R. (1964). *Languages with expressions of infinite length*. North-Holland Amsterdam.

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.

Kunen, K. (1987). Negation in logic programming. *The Journal of Logic Programming*, *4*(4), 289 – 308.

Lee, J. (2004). Nondefinite vs. definite causal theories. In *Proceedings 7th Int'l Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 141–153.

Lee, J. (2005). A model-theoretic counterpart of loop formulas. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 503–508. Professional Book Center.

Lee, J., Lierler, Y., Lifschitz, V., & Yang, F. (2010). Representing synonymity in causal logic and in logic programming[3]. In *Proceedings of International Workshop on Nonmonotonic Reasoning (NMR)*.

Lee, J., & Lifschitz, V. (2003a). Describing additive fluents in action language $\mathcal{C}$+. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1079–1084.

Lee, J., & Lifschitz, V. (2003b). 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). A reductive semantics for counting and choice in answer set programming. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 1*, pp. 472–479. AAAI Press.

Lee, J., Lifschitz, V., & Palla, R. (2009). Safe formulas in the general theory of stable models. Unpublished Draft. `http://peace.eas.asu.edu/joolee/papers/safety.pdf`.

[3] http://peace.eas.asu.edu/joolee/papers/syn.pdf

Lee, J., & Lin, F. (2006). Loop formulas for circumscription. *Artificial Intelligence*, *170*(2), 160–185.

Lee, J., & Meng, Y. (2008). On loop formulas with variables. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, pp. 444–453.

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)*, *42*, 125–180.

Lee, J., & Meng, Y. (2012a). Stable models of formulas with generalized quantifiers. In *Proceedings of International Workshop on Nonmonotonic Reasoning (NMR)*. http://peace.eas.asu.edu/joolee/papers/smgq-nmr.pdf.

Lee, J., & Meng, Y. (2012b). Stable models of formulas with generalized quantifiers (preliminary report). In *Technical Communications of the 28th International Conference on Logic Programming*, pp. 61–71.

Lee, J., & Meng, Y. (2012c). Two new definitions of stable models of logic programs with generalized quantifiers. In *Working Notes of the 5th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*.

Lee, J., & Palla, R. (2010). Situation calculus as answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 309–314.

Lee, J., & Palla, R. (2011). Integrating rules and ontologies in the first-order stable model semantics (preliminary report). In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 248–253.

Lee, J., & Palla, R. (2012a). Reformulating temporal action logics in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Lee, J., & Palla, R. (2012b). Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *Journal of Artificial Inteligence Research (JAIR)*, *43*, 571–620.

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. (2002). Answer set programming and plan generation. *Artificial Intelligence*, *138*, 39–54.

Lifschitz, V. (2008). What is answer set programming?. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1594–1597. MIT Press.

Lifschitz, V., Morgenstern, L., & Plaisted, D. (2008). Knowledge representation and classical logic. In van Harmelen, F., Lifschitz, V., & Porter, B. (Eds.), *Handbook of Knowledge Representation*, pp. 3–88. Elsevier.

Lifschitz, V., Pearce, D., & Valverde, A. (2001). Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, *2*, 526–541.

Lifschitz, V., & Razborov, A. (2006). Why are there so many loop formulas?. *ACM Transactions on Computational Logic*, *7*, 261–268.

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.

Lin, F., & Wang, Y. (2008). Answer set programming with functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 454–465.

Lin, F., & Zhao, Y. (2002). ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 112–117. MIT Press.

Lin, F., & Zhao, Y. (2004). ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, *157*, 115–137.

Lindström, P. (1966). First-order predicate logic with generalized quantifiers. *Theoria*, *32*, 186–195.

Liu, G. (2009). Level mapping induced loop formulas for weight constraint and aggregate programs. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pp. 444–449.

Liu, G., Goebel, R., Janhunen, T., Niemelä, I., & You, J.-H. (2011). Strong equivalence of logic programs with abstract constraint atoms. In *Proceedings of the 11th international conference on Logic programming and nonmonotonic reasoning*, LPNMR'11, pp. 161–173, Berlin, Heidelberg. Springer-Verlag.

Liu, G., Janhunen, T., & Niemelä, I. (2012). Answer set programming via mixed integer programming. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Liu, L., Pontelli, E., Son, T. C., & Truszczynski, M. (2010). Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence*, *174*(3ÍC4), 295 – 315.

Liu, L., & Truszczynski, M. (2006). Properties and applications of programs with monotone and convex constraints. *J. Artif. Intell. Res. (JAIR)*, *27*, 299–334.

Lloyd, J., & Topor, R. (1984). Making Prolog more expressive. *Journal of Logic Programming*, *3*, 225–240.

Lygeros, J. (2004). Lecture notes on hybrid systems. Tech. rep..

Marek, V., & Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pp. 375–398. Springer Verlag.

Marek, V. W., & Truszczynski, M. (2004). Logic programs with abstract constraint atoms. In *AAAI*, pp. 86–91.

McCain, N., & Turner, H. (1997). Causal theories of action and change. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 460–465.

Mellarkod, V. S., Gelfond, M., & Zhang, Y. (2008). Integrating answer set programming and constraint logic programming. *Ann. Math. Artif. Intell.*, *53*(1-4), 251–287.

Moore, G. (1997). The prehistory of infinitary logic: 1885Ű1955. In Chiara, M., Doets, K., Mundici, D., & Benthem, J. (Eds.), *Structures and Norms in Science*, Vol. 260 of *Synthese Library*, pp. 105–123. Springer Netherlands.

Mostowski, A. (1957). On a Generalization of Quantifiers. *Fundamenta Mathematicae*, *44*, 12–35.

Niemelä, I., & Simons, P. (2000). Extending the Smodels system with cardinality and weight constraints. In Minker, J. (Ed.), *Logic-Based Artificial Intelligence*, pp. 491–521. Kluwer.

Niemelä, I., Simons, P., & Soininen, T. (1999). Stable model semantics of weight constraint rules. In *PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING AND NONMONOTONIC REASONING (LPNMRą́99), VOLUME 1730 OF LECTURE*, pp. 317–331. Springer-Verlag. LNAI.

Nieuwenhuis, R., Oliveras, A., & Tinelli, C. (2006). Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(t). *J. ACM*, *53*(6), 937–977.

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.

Pearce, D. (1997). A new logical characterization of stable models and answer sets. In Dix, J., Pereira, L., & Przymusinski, T. (Eds.), *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, pp. 57–70. Springer.

Pednault, E. (1994). ADL and the state-transition model of action. *Journal of Logic and Computation*, *4*, 467–512.

Pelov, N., Denecker, M., & Bruynooghe, M. (2003). Translation of aggregate programs to normal logic programs.. In *Proceedings Answer Set Programming*.

Pelov, N., Denecker, M., & Bruynooghe, M. (2004). Partial stable models for logic programs with aggregates. In *LPNMR*, pp. 207–219.

Pelov, N., Denecker, M., & Bruynooghe, M. (2007). Well-founded and stable semantics of logic programs with aggregates. *TPLP*, *7*(3), 301–353.

Pinto, J. A. (1994). Temporal reasoning in the situation calculus..

Polleres, A., & Schindlauer, R. (2007). dlvhex-sparql: A sparql-compliant query engine based on dlvhex. In *2nd Int. Workshop on Applications of Logic Programming to the Web, Semantic Web and Web Services (ALPSWS2007*, pp. 332–347. Springer.

Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 2–13.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *J. ACM*, *12*, 23–41.

Rosati, R. (2005). On the decidability and complexity of integrating ontologies and rules. *J. Web Sem.*, *3*(1), 61–73.

Sabuncu, O., & Alpaslan, F. N. (2007). Computing answer sets using model generation theorem provers. Unpublished Draft.

Scott, D., T. A. (1958). The sentential calculus with infinitely long expressions. *Colloquium Mathematicae*, *6*(1), 165–170.

Shanahan, M. (1990). Representing continuous change in the event calculus. In *ECAI*, pp. 598–603.

Shen, Y.-D. (2011). Well-supported semantics for description logic programs. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 1081–1086.

Shen, Y.-D., You, J.-H., & Yuan, L.-Y. (2009). Characterizations of stable model semantics for logic programs with arbitrary constraint atoms. *TPLP*, *9*(4), 529–564.

Shin, J.-A., & Davis, E. (2005). Processes and continuous change in a sat-based planner. *Artif. Intell.*, *166*(1-2), 194–253.

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.

Son, T. C., Pontelli, E., & Tu, P. H. (2007). Answer sets for logic programs with arbitrary abstract constraint atoms. *J. Artif. Intell. Res. (JAIR)*, *29*, 353–389.

Tarski, A. (1958). Remarks on predicate logic with infinitely long expressions. *Colloquium Mathematicae*, *6*(1), 171–176.

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.

Truszczyński, M. (2010). Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. *Artificial Intelligence*, *174*(16-17), 1285–1306.

Truszczynski, M. (2012). Connecting first-order asp and the logic fo(id) through reducts. In *Correct Reasoning*, pp. 543–559.

Turner, H. (2003). Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming*, *3(4,5)*, 609–622.

Wang, Y., You, J.-H., Lin, F., Yuan, L. Y., & Zhang, M. (2010a). Weight constraint programs with evaluable functions. *Annals of Mathematics and Artificial Intelligence*, *60*, 341–380.

Wang, Y., You, J.-H., Yuan, L.-Y., & Shen, Y.-D. (2010b). Loop formulas for description logic programs. *TPLP*, *10*(4-6), 531–545.

Westerståhl, D. (2008). Generalized quantifiers. In *The Stanford Encyclopedia of Philosophy (Winter 2008 Edition)*. URL = <http://plato.stanford.edu/archives/win2008/entries/generalized-quantifiers/>.

You, J.-H., & Liu, G. (2008). Loop formulas for logic programs with arbitrary constraint atoms. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 584–589.

Zhang, Y., & Zhou, Y. (2010). On the progression semantics and boundedness of answer set programs. In *KR*.