FIRST-ORDER ANSWER SET PROGRAMMING AND CLASSICAL FIRST-ORDER LOGIC

A THESIS
SUBMITTED TO THE UNIVERSITY OF WESTERN SYDNEY
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Vernon Asuncion
February 2013
Except where otherwise indicated, this thesis is my own original work. I certify that this thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree.

________________________________________
Vernon Asuncion
May 2012
Acknowledgements

First and foremost, none of this would be possible without the help, guidance, encouragement, and support of my supervisor Yan Zhang. It is for this reason that I would like to thank Yan for all that he had provided to me from my days as a 3rd year undergraduate student, to my Honours year, and my most recent years as a Ph.D. candidate. I would especially like to thank Yan for keeping me on the correct path throughout my research work, which included him having to listen to my immature ideas.

I would also like to thank Yi Zhou for all the help and advice that he had given and in particular, for showing me the skills of paper writing. I also benefited from discussions with Yin Chen, Joohyung Lee, and Torsten Schaub. I would also like to mention the other useful discussion I had with the other ISL members.

Last (but not least), I would also like to thank my family for helping me throughout my studies, and for motivating me by setting me the challenge of achieving this.
Abstract

Answer Set Programming (ASP) is a form of declarative programming oriented towards difficult and primarily NP-hard search problems. Generally speaking, an ASP search problem is formulated as a theory of some language of formal logic. The formalization is designed in such a way that the problem description is usually separate from the problem instance. Thus, once a particular encoding of a problem instance is combined with the problem description, it results in some theory of formal logic whose models (under a particular semantics) corresponds to a solution of the problem instance.

This thesis is about the translation of ASP programs into classical first-order logic theories. Viewed in the context of expressing logic programs with variables into (classical) first-order logic, work of this direction goes all the way back to Clark who gave us what is now called the Clark’s completion semantics, on which this work is based. This work modifies the Clark’s completion in such a way that the models of the modified completion corresponds exactly to the answer sets. We then report on such an implementation of grounding completely on first-order theories rather than on programs (i.e., as is the case with traditional ASP solvers), and show that the new approach is quite competitive with current effective solvers on very large instances of the Hamiltonian circuit program. In addition, we further consider the translation of logic programs with aggregates (a very important building block of ASP) under the stable model semantics, into classical first-order theories.

Preferences play an important role in knowledge representation and reasoning. In the past decade, a number of approaches for handling preferences have been developed in various nonmonotonic reasoning formalisms, while adding preferences into ASP has been known to have promising advantages from both implementation and application viewpoints. This thesis also extends the notion of preferred ASP (i.e., answer set programming with preference relations among the rules), that is currently only formalized for
propositional programs, into the notion of first-order ASP. To this aim, we develop both a (first-order) fixpoint type characterization that is similar to Zhang and Zhou’s progression semantics, and a classical logic formulation so that the preferred answer sets are exactly those models of the logic formula. We further show that the fixpoint type characterization and classical logic formulation coincide.
Contents

Acknowledgements

Abstract

1 Introduction to Answer Set Programming
   1.1 Introduction ......................................................... 1
   1.2 Research Work in ASP ............................................. 3
       1.2.1 First-Order Answer Set Programming ......................... 6
       1.2.2 Stable Models of Arbitrary Formulas ....................... 10
       1.2.3 Loop Formulas Approach .................................... 15
       1.2.4 ASP with Aggregates ....................................... 17
       1.2.5 Preferred ASP .............................................. 19
       1.2.6 Important Preferred Program Frameworks ................ 21
   1.3 Structure of the Thesis ......................................... 26

2 First-Order ASP and Classical First-Order Logic .......................... 28
   2.1 Past Efforts in Linking Logic Programs and Classical Logics ........ 28
   2.2 The Clark’s Completion ......................................... 30
   2.3 The Ordered Completion ......................................... 34
       2.3.1 Positive Predicate Dependency Graphs and Tight Programs ........ 34
       2.3.2 Intuition of the Ordered Completion .................... 35
       2.3.3 Definition of the Ordered Completion .................... 36
       2.3.4 The Main Theorem ........................................ 41
       2.3.5 Normal Programs with Constraints ...................... 44
       2.3.6 Adding Choice Rules ..................................... 46
## 2.3.7 Arbitrary Structures and Disjunctive Programs

2.3.8 Related Work

## 2.4 Ordered Completion with Aggregates

2.4.1 The Stable Model Semantics for Aggregates

2.4.2 Extension of Ordered Completion for Aggregates

2.4.3 Externally Supported Sets with Aggregates

2.4.4 Theorem of Ordered Completion with Aggregates

2.4.5 Aggregate Constructs as Classical FO Logic

2.4.6 Related Work

## 3 Implementation and Experimental Results

3.1 Grounding ASP Programs

3.2 Domain Predicates and Guarded Variables

3.3 Ordered Completion vs Clark's Completion

3.4 Grounding Ordered Completion

3.4.1 Constructing Domains Using Partial Structures

3.4.2 Dynamic Domain Relations

3.4.3 Grounding on a Domain Predicate Sequence

3.4.4 Incorporating Classical Reasoning

3.4.5 Iteration and Conversion to SMT

3.5 Experimental Results

3.6 Further Remarks

## 4 Preferred FO Answer Set Programs

4.1 Lifting Preferred Answer Set Programming to First-Order

4.2 A Progression Based Semantics

4.3 Properties of Preferred Answer Sets

4.3.1 Grounding Preferred programs

4.3.2 Semantic Properties

4.4 Logical Characterizations

4.4.1 Formulas Representing Generating Rules and Program Completion

4.4.2 Well-Orderings on Generating Rules in Terms of Structures

4.4.3 Formalizing Progression
4.4.4 Incorporating Preference ........................................... 119
4.5 Related Work and Further Remarks ................................ 123

5 Summary of Contributions and Future Work ........................ 125
  5.1 Ordered Completion ................................................. 125
  5.1.1 Incorporating Aggregates ........................................ 127
  5.2 Solver Implementation .............................................. 129
  5.3 Preferred ASP ..................................................... 130
  5.4 Future Work ....................................................... 131

A Proofs .............................................................................. 134
  A.1 Proofs for Chapter 2 ................................................ 134
    A.1.1 Proof of Lemma 1 ............................................... 134
    A.1.2 Proof of Theorem 2 ............................................ 136
    A.1.3 Proof of Lemma 2 ............................................... 138
    A.1.4 Proof of Theorem 3 ............................................ 141
  A.2 Proofs for Chapter 4 ................................................ 145
    A.2.1 Proof of Proposition 14 ....................................... 145
    A.2.2 Proof of Theorem 6 ............................................ 150
    A.2.3 Proof of Lemma 3 ............................................... 151
    A.2.4 Proof of Lemma 4 ............................................... 152
    A.2.5 Proof of Theorem 10 .......................................... 153
    A.2.6 Proof of Theorem 11 .......................................... 161

References ................................................................. 170
Chapter 1

Introduction to Answer Set Programming

1.1 Introduction

Answer Set Programming (ASP) is a form of declarative programming oriented towards difficult and primarily NP-hard search problems. Generally speaking, an ASP search problem is formulated as a theory of some language of formal logic. The formalization is designed in such a way that the problem description is usually separate from the problem instance. Thus, once a particular encoding of a problem instance is combined with the problem description, it results in some theory of formal logic whose models (under a particular semantics) corresponds to a solution of the problem instance. ASP turned out to be particularly useful in knowledge-intensive applications such as automated product configuration [TSNS03], decision support for the space shuttle [NBG+01], and inferring phylogenetic trees [BEE+07].

ASP programs are logic programs that are usually interpreted under the stable model (also called answer set) semantics as introduced by Gelfond and Lifschitz in [GL88]. Unlike its Prolog logic programming counterpart, ASP programs are fully declarative in the sense that neither the ordering of the rules in the programs or the ordering of literals in the rules have any effect on the answer sets. In fact, it even goes further that it has, if any, only a negligible effect on the computation of answer sets. This is due to the clear separation of the problem description from that of the problem instance, since the fully declarative nature
of the problem description allows one to apply the necessary preprocessing even before the problem instance had arrived. The effectiveness of this approach is due to the reason that the problem instance is usually many magnitudes bigger in size than the problem description. Thus, as already mentioned above, any preprocessing on the problem description will only have a very negligible effect in the whole computation process.

In the traditional manner, the answer set semantics of logic programs with variables is defined in two steps. First, since the answer set semantics was originally defined by Gelfond and Lifschitz in [GL88] based on the syntax of propositional logic programs, it was necessary to first ground the program by replacing all the variables mentioned in it by domain constants of the corresponding problem instance. The answer set of the program is then defined as the minimal model of the program’s so-called “Gelfond-Lifschitz reduction” (named after its inventors). The Gelfond-Lifschitz reduction (or just reduction when clear from the context) is a “simplification” of the program where only positive atoms are mentioned in the “reduced” program. This two step process is also reflected in the way current ASP solver systems (softwares for finding solutions to ASP encodings) work by also having the two step process of first grounding the underlying program, and then on the actual solving of the resulting propositional program. Currently, the softwares used in each steps, called grounders and solvers, respectively, have already reached such a high level of performance that it is now possible to use these on ASP encodings corresponding to problems of practical importance.

The notion of the Gelfond-Lifschitz reduction and answer sets was extended to arbitrary formulas by Pearce in [Pea96] via the so-called equilibrium logic and the logic of here-and-there (HT) that is based on Kripke models. Generally speaking, the logic of HT generalizes the notion of the Gelfond-Lifschitz reduction from the restricted syntax of rules of logic programs into arbitrary propositional formulas. The so-called equilibrium models then correspond to the minimal models of the propositional formula’s reducts. This notion of reduct was then solely defined in terms of classical propositional theories by Ferraris in [Fer05]. Ferraris in [Fer05] defined the reduct $F^X$ of an (arbitrary) propositional formula $F$ in terms of a set $X$ of atoms by a recursive definition that is obtained via a similar method in [Pea96], but where the formula was already simplified by replacing its appropriate subformulas by $\bot$ (falsity), depending on whether or not $X$ satisfies the particular subformula.
The works of Pearce [Pea96] and Ferraris [Fer05] opened up the avenue for defining the answer set semantics of first-order (FO) formulas. In fact, Pearce et. al. in [PV05] extended the answer set semantics to first-order formulas via the so-called first-order equilibrium logic. Then, by viewing answer set programs with variables as first-order formulas, it is now possible to define a truly first-order answer set semantics for “first-order” answer set programs. Then a surprising result of Ferraris et. al. in [FLL07] is the direct encoding of the first-order equilibrium models as a classical second-order (SO) sentence via the SM operator. The SM operator takes any arbitrary first-order formula \( \varphi \), and then translates it into a (universal) second-order sentence \( \text{SM}(\varphi) \) such that the models of \( \text{SM}(\varphi) \) are exactly the answer sets of \( \varphi \). Then, by again viewing answer set programs with variables as first-order formulas, it is now possible to define the first-order answer set semantics of programs in terms of classical logic.

1.2 Research Work in ASP

In this section, we describe a brief history of the current research that had occured in the paradigm of ASP. To this aim, we first introduce the formal syntax of answer set programs, along with the formal definition of stable models (or answer sets).

A normal answer set program (or just program when clear from the context) \(^1\) \( \Pi \) of some propositional signature \( \mathcal{L} \) is a finite set of rules \( r \) of the form

\[
a \leftarrow b_1, \ldots, b_l, \neg c_1, \ldots, \neg c_m, \tag{1.1}
\]

where \( a, b_i \) (\( 1 \leq i \leq l \)), and \( c_i \) (\( 1 \leq i \leq m \)) are propositional atoms from \( \mathcal{L} \). The atom \( a \) is called the head of \( r \) and is denoted by \( \text{Head}(r) \), the set \( \{ b_1, \ldots, b_l, \neg c_1, \ldots, \neg c_m \} \) is called the body of \( r \) and is denoted by \( \text{Body}(r) \). The set \( \text{Body}(r) \) can be further subdivided into two sets \( \text{Pos}(r) \) and \( \text{Neg}(r) \), called the positive and negative bodies of \( r \) respectively, such that \( \text{Pos}(r) = \{ b_1, \ldots, b_l \} \) and \( \text{Neg}(r) = \{ c_1, \ldots, c_m \} \). When \( \text{Head}(r) \) is non-existent (i.e., \( r \) has no head), then \( r \) is called a constraint.

The semantics of answer set programming was originally defined in terms of its Gelfond-Lifschitz reduction, named after its inventors [GL88], and is referred to as the stable models semantics, or what we will mostly refer to as the answer sets semantics. Given a program

\(^1\)We assume all answer set programs to be “normal” unless stated otherwise.
Π and an interpretation \( I \subseteq \mathcal{L} \), by \( \Pi^I \), we denote the (Gelfond-Lifschitz) reduction of \( \Pi \) such that \( \Pi^I \) is obtained from \( \Pi \) by:

1. Deleting all rules \( r \in \Pi \) from \( \Pi \) where \( \text{Neg}(r) \cap I \neq \emptyset \);
2. Transforming all remaining rules \( r \) of the form (1.1) into \( a \leftarrow b_1, \cdots, b_l \), i.e., simply deleted the “negative” part of its body.

Then clearly, \( \Pi^I \) is now a positive (or negation free) program. Now by \( \widehat{\Pi}^I \), we denote the propositional formula

\[
\bigwedge_{r \in \Pi^I, r = \neg b_1, \cdots, b_l} (b_1 \land \cdots \land b_l \rightarrow a). \tag{1.2}
\]

Then \( \widehat{\Pi}^I \) is usually referred to as the logical closure \( \Pi^I \) (note that \( \widehat{\Pi}^I \) is a propositional formula while \( \Pi^I \) is not). Then \( I \) is said to be a stable model (or answer set) of \( \Pi \) iff:

1. \( I \models \widehat{\Pi}^I \);
2. For all other interpretations \( I' \subset I \) (i.e., \( I' \) is a strict-subset \( I \)), \( I' \not\models \widehat{\Pi}^I \).

In other words, an interpretation \( I \subseteq \mathcal{L} \) is a stable model of \( \Pi \) iff it is a minimal model of the closure of its reduct, i.e., \( \widehat{\Pi}^I \).

Example 1 Assume we have \( \mathcal{L} = \{a, b, c, d\} \) and \( \Pi \) as the program

\[
\begin{align*}
a & \leftarrow \neg c \\
b & \leftarrow a, \neg d \\
c & \leftarrow \neg a \\
d & \leftarrow \neg b.
\end{align*}
\]

Then with \( I = \{a, b\} \), we show that \( I \) is an answer set of \( \Pi \). Indeed, since for the rules “\( c \leftarrow \neg a \)” and “\( d \leftarrow \neg b \)” we have that \( a \in I \) and \( b \in I \), then these two rules will be deleted from \( \Pi \) so they will not be in the reduct \( \Pi^I \). Then through the deletion of the negative bodies of the remaining two rules “\( b \leftarrow a, \neg d \)” and “\( a \leftarrow \neg c \)” (i.e., to end
up with the two rules “\(b \leftarrow a\)” and “\(a \leftarrow\)” respectively), we have the reduct \(\Pi^I\) to be the (negation free) program:

\[
\begin{align*}
a \leftarrow \\
b &\leftarrow a.
\end{align*}
\]

Then \(\hat{\Pi}^I\) is simply the propositional formula \(a \land (a \rightarrow b)\). It is then not too difficult to see that \(\{a, b\}\) is also a minimal model of \(a \land (a \rightarrow b)\) so that \(I\) is an answer set of \(\Pi\). Note that when the body is empty, as in the rule “\(a \leftarrow\)” of \(\Pi^I\), then \(a\) is what is so-called a fact, and where facts are always assumed to be present in any answer set.

\(\square\)

Incidentally, the (Gelfond-Lifschitz) reduction of a program is a way of viewing the “not” operator as \textit{negation by failure}. The “not” operator differs in meaning from the classical ‘\(\neg\)’ connective (i.e., “not” is simply not denoted by ‘\(\neg\)” in programs) in that an atom \(a\) under “\(\neg a\)” is assumed to be non-provable from the \textit{reduct} of the program. To see the difference, let \(\Pi\) be the program

\[
\begin{align*}
b &\leftarrow a \\
&\leftarrow \neg a.
\end{align*}
\]

Then, if we view “not” as the classical connective ‘\(\neg\)” and the rules “\(b \leftarrow a\)” and “\(\leftarrow \neg a\)” as the classical implications “\(a \rightarrow b\)” and “\(\neg a \rightarrow \bot\)” respectively, we have that \(\hat{\Pi}\) is the propositional formula

\[
(a \rightarrow b) \land (\neg a \rightarrow \bot)
\]

\(\equiv (a \rightarrow b) \land a.\)

Then under this context, \(\{a, b\}\) is a minimal model of \(\hat{\Pi}\). Now let us consider the case where “not” is viewed as negation by failure. Then we have that \(\Pi^{\{a,b\}}\) is the reduced program

\[
b \leftarrow a,
\]
i.e., the constraint “← not a” was deleted since $a \in \{a, b\}$. Then $\Pi^{(a,b)}$ in this case is simply the propositional formula

$$a \rightarrow b,$$

and such that $\{}$ (i.e., the empty set) is its minimal model. So in this case, since $\{} \neq \{a, b\}$, then we have that $\{a, b\}$ is not an answer set of $\Pi$.

### 1.2.1 First-Order Answer Set Programming

A recent enhancement to answer set programming is the so-called first-order (FO) answer set programming. In traditional answer set programming approaches, a program with variables is simply viewed as a shorthand of its Herbrand instantiation. For example, assume $\Pi$ to be the following program with variables $x$, $y$, and $z$, and constants $a$, $b$, and $c$:

$$
E(a, b) \leftarrow \\
E(b, c) \leftarrow \\
T(x, y) \leftarrow E(x, y) \\
T(x, z) \leftarrow E(x, y), T(y, z).
$$

Then $\Pi$ is simply viewed as a shorthand for the propositional program

$$
E(a, b) \leftarrow \\
E(b, a) \leftarrow \\
T(a, a) \leftarrow E(a, a) \\
T(a, b) \leftarrow E(a, b) \\
T(b, a) \leftarrow E(b, a) \\
T(b, b) \leftarrow E(b, b) \\
T(a, a) \leftarrow E(a, a), T(a, a) \\
T(a, a) \leftarrow E(a, b), T(b, a)
$$

(1.3)
which was obtained from $\Pi$ by substituting the constants $a$, $b$, and $c$ for the variables $x$, $y$, and $z$ in all possible ways. Thus, in this context, each of the constants $a$, $b$, and $c$ are interpreted as their Herbrand interpretation, i.e., we think of $a$, $b$, and $c$ as literally constants $a$, $b$, and $c$ respectively. This is in contrast with that of FO logic since the constants $a$, $b$, and $c$ can stand for different domain objects. Hence, under this notion, the traditional approach of answer set programming with variables is not truly FO.

To get over this representational limit, a truly FO characterization of answer set programs is achieved by using a kind of minimality expression about the extents of some its predicates. This is achieved by a SO sentence that universally quantifies over these “intended” predicates. Hence, through a universal SO sentence, one will now be able to define the answer sets of a FO program without any reference to the grounding (or “proposition-alization”) of the program. Before proceeding, we first formally define the syntax of FO answer set programs.

A first-order normal answer set program (or simply call FO program) $\Pi$ is a finite set of rules $r$ of the form

$$
\alpha \leftarrow \beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots, \text{not } \gamma_m,
$$

such that $\alpha$, $\beta_i$ ($1 \leq i \leq l$), and $\gamma_i$ ($1 \leq i \leq m$) are atomic formulas of the form $P(x)$ where $x$ is a tuple of terms (i.e., can be variables or constants) whose length matches the arity of $P$. As in the case of propositional programs, the atom $\alpha$ is called the head of $r$ and is denoted by $\text{Head}(r)$, the set of literals $\{\beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots, \text{not } \gamma_m\}$ is called the body of $r$ and is denoted by $\text{Body}(r)$. The set $\text{Body}(r)$ can be further subdivided into $r$’s “positive” and “negative” bodies. Hence by $\text{Pos}(r)$, we denote the set $\{\beta_1, \ldots, \beta_l\}$, which

\[\text{not } \gamma_i \quad (1 \leq i \leq m)\]

is a literal.
is called the *positive body* of \( r \), and by \( \text{Neg}(r) \), we denote the set \( \{ \gamma_1, \ldots, \gamma_m \} \), which is called the *negative body* of \( r \). For convenience, for a given program \( \Pi \), we denote by \( \tau(\Pi) \) as the signature of the program \( \Pi \). As in the case of propositional programs, when the \( \text{Head}(r) \) is non-existent (i.e., the rule has no head), then \( r \) is what is so-called a *constraint*.

In our notion here of FO answer set programs, we distinguish between the so-called *extensional* and *intensional* predicates. The intensional predicates are simply those predicates that are mentioned in the heads of some rules of the program and the other predicates are referred to as extensional. Conceptually, the “extensional” predicates are viewed as the input and the “intensional” predicates as the output of the program. For instance, assume we have \( \Pi \) to be the well-known program that computes the transitive closure of a graph \( G = (V, E) \):

\[
\begin{align*}
T(x, y) & \leftarrow E(x, y) \\
T(x, z) & \leftarrow E(x, y), T(y, z),
\end{align*}
\]

such that our signature \( \tau(\Pi) \) in this case is \( \{ T, E \} \). Then under this context, given a graph structure \( G = (\text{Dom}(G), E^G) \) such that \( \text{Dom}(G) \) are the graph’s vertices and \( E^G \) its edge relations, the program \( \Pi \) will output the transitive closure of those edge relations of \( G \).

For convenience, in addition to denoting the signature of \( \Pi \) to be \( \tau(\Pi) \), we also denote by \( \mathcal{P}_{\text{ext}}(\Pi) \) and \( \mathcal{P}_{\text{int}}(\Pi) \) as the sets of *extensional* and *intensional* predicate symbols respectively.

From here on, we reserve a special type of extensional predicate called the *equality predicate*, denoted by the symbol ‘\( = \)’ and of arity 2, and such that we assume that ‘\( = \)’ is in \( \tau(\Pi) \) for any program \( \Pi \) (i.e., whether or not ‘\( = \)’ is actually mentioned in \( \Pi \)). Moreover, for any given \( \tau(\Pi) \)-structure \( \mathcal{M} \), we assume that \( =^\mathcal{M} \) is the interpretation

\[
\{(a, a) \mid a \in \text{Dom}(\mathcal{M})\}
\]

(i.e., the *equality relations*) and such that we take \( = (x, y) \), or \( x = y \) using the usual *infix notation*, to mean that \( x \) is equal to \( y \).

Now we introduce the *stable models* semantics of FO programs. So for this purpose,
CHAPTER 1. INTRODUCTION TO ANSWER SET PROGRAMMING

for a given FO program \( \Pi \), set \( \widehat{\Pi} \) as the FO sentence

\[
\bigwedge_{r \in \Pi, r := \alpha \leftarrow \beta_1, \ldots, \beta_l, \neg \gamma_1, \ldots, \neg \gamma_m} \forall x_r (\beta_1 \land \cdots \land \beta_l \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \rightarrow \alpha)
\]  

(1.5)

where for a rule \( r \in \Pi \), \( x_r \) denotes all the variables occurring in \( r \). Then \( \widehat{\Pi} \) is usually referred to as the universal closure of \( \Pi \). Now assume that the set of intensional predicates \( P_{\text{int}}(\Pi) \) is the set \( \{P_1, \ldots, P_n\} \) and let \( P = P_1 \cdots P_n \) such that \( P \) is a tuple of distinguishable intensional predicates. Then, set \( U = U_1 \cdots U_n \) to be another tuple of fresh (i.e., they are not in \( \tau \)) distinguishable predicates such that each \( U_i \) for \( 1 \leq i \leq n \) matches the arity of \( P_i \). Now define the SO formula \( \widehat{\Pi}^*(U) \) as

\[
\bigwedge_{r \in \Pi, r := \alpha \leftarrow \beta_1, \ldots, \beta_l, \neg \gamma_1, \ldots, \neg \gamma_m} \forall x_r (\beta_1^* \land \cdots \land \beta_l^* \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \rightarrow \alpha^*),
\]  

(1.6)

where for an atomic formula \( P(x) \): if \( P = P_i \) for some \( 1 \leq i \leq n \) (i.e., \( P \) is intensional) then \( P(x)^* = U_i(x) \), and \( P(x)^* = P(x) \) otherwise (i.e., if \( P \) is extensional). So finally, by \( SM(\Pi) \) (i.e., “SM” for “stable model”), denote the universal SO sentence

\[
\widehat{\Pi} \land \forall U (U < P \rightarrow \neg \widehat{\Pi}^*(U)),
\]  

(1.7)

where \( U < P \) denotes the FO sentence

\[
\bigwedge_{1 \leq i \leq n} \forall x_i (U_i(x_i) \rightarrow P(x_i)) \land \neg \bigwedge_{1 \leq i \leq n} \forall x_i (P_i(x_i) \rightarrow U_i(x_i)),
\]  

(1.8)

which basically states that \( U_i \subseteq P_i \) for all \( 1 \leq i \leq n \) and that \( U_j \subseteq P_j \) (strict subset) for some \( 1 \leq j \leq n \). Then a \( \tau(\Pi) \)-structure \( \mathcal{M} \) is said to be a stable model or answer set of \( \Pi \) iff \( \mathcal{M} \models SM(\Pi) \).

Informally speaking, the SO sentence \( SM(\Pi) \) is a “first-order” way of expressing a minimal model of \( \Pi \)’s reduct. Indeed, the formula \( \widehat{\Pi}^*(U) \) corresponds in a way to the reduct and such that we look for a “minimal” model of it. This is expressed by the fact that we look for a \( \tau(\Pi) \)-structure \( \mathcal{M} \) that satisfies the universal closure \( \widehat{\Pi} \) and where there cannot exist another \( \tau(\Pi) \)-structure \( \mathcal{M}' \) that is smaller in the extents of the intensional predicates that satisfies \( \widehat{\Pi}^*(U) \), i.e., as expressed by \( \forall U (U < P \rightarrow \neg \widehat{\Pi}^*(U)) \). It is not too
difficult to show that our notion of $SM(\Pi)$ here corresponds to that as proposed in [FLL11] when restricted to the syntax of (the universal closures of) normal answer set programs.

Novel as this truly FO characterization seems, it has one major drawback, which is the fact that we resorted to a universal SO sentence, i.e., we universally quantified over the extents of the intensional predicates. From even just the model checking viewpoint, it is immediately apparent that in verifying if a candidate structure $M$ is a model $SM(\Pi)$, one will have to consider all possible structures $M'$ for which we have $P_i^{M'} \subset P_i^M$ for some $1 \leq i \leq n$. Even under the context of finite structures, this will result in exponential time in the worst case, i.e., since there are an asymptotically exponential number of possible strict subsets of a given set. So a natural question to ask is: can the stable models semantics of first-order answer set programs be defined without resorting to a universal second-order sentence? Fortunately, in this work, we provide a positive answer to this question. In fact, until now and to the best of our knowledge, our work is the first such one. As will be seen in Chapter 2, we will show a polynomial translation of FO programs into FO sentence with fresh predicates [ALZZ12].

1.2.2 Stable Models of Arbitrary Formulas

Propositional equilibrium logic

Pearce extended the notion of the stable models semantics to arbitrary propositional formulas via his so-called equilibrium logic [Pea96]. Equilibrium logic is based on the logic of here-and-there (HT).\(^3\) To define the semantics of propositional HT, first let us assume a propositional signature $L$ and interpretations $H \subseteq L$ (i.e., ‘$H$’ for “here”) and $T \subseteq L$ (i.e., ‘$T$’ for “there”) and where $H \subseteq T$. Then the tuple $\langle H, T \rangle$ is called a here-and-there interpretation, or simply a HT interpretation. The satisfaction relation $\langle H, T \rangle \models_{HT} \varphi$, for some propositional formula $\varphi$ of $L$, is now recursively defined as follows:

- If $\varphi = \bot$, then $\langle H, T \rangle \not\models_{HT} \varphi$;
- If $\varphi = \top$, then $\langle H, T \rangle \models_{HT} \varphi$;
- If $\varphi = a$ for some $a \in L$, then $\langle H, T \rangle \models_{HT} \varphi$ iff $a \in H$;
- If $\varphi = \psi \land \xi$, then $\langle H, T \rangle \models_{HT} \varphi$ iff $\langle H, T \rangle \models_{HT} \psi$ and $\langle H, T \rangle \models_{HT} \xi$;

\(^3\)For simplicity here, we do not refer to the more rigorous notion of Kripke structures.
• If \( \varphi = \psi \lor \xi \), then \(<H,T> \models_{HT} \varphi \) iff \(<H,T> \models_{HT} \psi \) or \(<H,T> \models_{HT} \xi \);

• If \( \varphi = \psi \rightarrow \xi \), then \(<H,T> \models_{HT} \varphi \) iff both conditions are satisfied:
  
  - \(<H,T> \not\models_{HT} \psi \) or \(<H,T> \models_{HT} \xi \);
  
  - \( T \models \psi \rightarrow \xi \) (i.e., where \( \models \) in here is simply the classical satisfaction relation),

and where we view \( \neg \varphi \) as the shorthand for \( \varphi \rightarrow \bot \), i.e., the connective ‘\( \neg \)' is not in our language since we view \( \neg \varphi \) as \( \varphi \rightarrow \bot \). Then a HT interpertation \(<H,T>\) is said to be an equilibrium model of \( \varphi \) iff:

1. \(<H,T> \models_{HT} \varphi \) and where \( H = T \), i.e., we require the “here” world to be equal to the “there” world;

2. For all other interpretations \( H' \) where \( H' \subset H \), we have that \(<H',T> \not\models_{HT} \varphi \), i.e., the HT interpretation is “here” minimal.

From [Pea96], it also follows that the stable models of an arbitrary propositional formula \( \varphi \) are precisely its equilibrium models. Note that this generalizes the stable models semantics from the “syntactical” restriction of propositional programs to propositional formulas of arbitrary (syntactical) structures.

For intuition, let us refer back to the concept of the Gelfond-Lifschitz reduction, or simply the reduction, of a propositional program \( \Pi \). In the HT interpretation \(<H,T>\), the world “here” plays the role of the reduct via the interpretation \( T \) and such that the minimality requirement of this “here” world corresponds to the minimal models of this reduct. To see this, assume \( r \) to be the rule

\[
\begin{align*}
b & \leftarrow \text{not } a.
\end{align*}
\]

Then viewing \( r \) as the “classical” implication and “not” as the “classical” ‘\( \neg \)’ connective, gives us the propositional formula

\[
\neg a \rightarrow b.
\]

Then since we view \( \neg a \) as \( a \rightarrow \bot \) in the logic of HT, this further gives us

\[
(a \rightarrow \bot) \rightarrow b.
\]
Now let us assume a HT interpretation \( \langle H, T \rangle \) such that \( H = T = \{a\} \). Then, by the definition of the HT satisfaction relation \( \models_{HT} \), we have that \( \langle H, T \rangle \models_{HT} (a \rightarrow \bot) \rightarrow b \) iff:

- \( \langle H, T \rangle \not\models_{HT} a \rightarrow \bot \) or \( \langle H, T \rangle \models_{HT} b \);
- \( T \models a \land (a \rightarrow \bot) \rightarrow b \).

Then since \( \langle H, T \rangle \not\models_{HT} b \), we must have \( \langle H, T \rangle \not\models_{HT} a \rightarrow \bot \). Then, since we have \( \langle H, T \rangle \not\models_{HT} a \rightarrow \bot \) iff

- \( \langle H, T \rangle \models_{HT} a \) and \( \langle H, T \rangle \not\models_{HT} \bot \), or
- \( T \not\models a \rightarrow \bot \),

and where \( a \rightarrow \bot \equiv \neg a \) (i.e., classically), then \( T \not\models a \rightarrow \bot \equiv \neg a \) always holds.

Thus, because of this condition (i.e., where this condition corresponds to the activation of the reduct), we will have that \( \langle H', T \rangle \models_{HT} (a \rightarrow \bot) \rightarrow b \) for all interpretations \( H' \) where \( H' \subseteq H \). Therefore, it follows from the definition of an equilibrium model that \( a \) is not a stable model (or answer set) of \( \Pi \). Hence, even though \( \{a\} \) is a minimal model of the “classical view” of the rule “\( b \leftarrow \neg a \)” (i.e., which is the propositional formula “\( \neg a \rightarrow b \)”), it is not necessarily a “here” minimal HT interpretation.

**First-order equilibrium logic**

This notion of equilibrium models was extended to the first-order case of arbitrary FO formulas in [PV05] by extending to a FO notion of the logic of here-and-there (HT). Assuming a signature \( \tau \), a FO HT interpretation is a pair of \( \tau \)-structures \( \langle H, T \rangle \) such that:

1. \( \text{Dom}(H) = \text{Dom}(T) \);
2. \( c^H = c^T \) for each constant symbol \( c \in \tau \);
3. \( R^H \subseteq R^T \) for each relation symbol \( R \in \tau \).

As in the propositional case, the structures \( H \) and \( T \) corresponds to the worlds “here” and “there” respectively. Then the FO HT satisfaction relation \( \langle H, T \rangle \models_{HT} \varphi(x)[x/a] \) for some FO formula \( \varphi \) is defined recursively as follows:
• If \( \varphi(x) = \bot \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \not\models_{HT} \varphi(x)[x/a] \);

• If \( \varphi(x) = \top \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi(x)[x/a] \);

• If \( \varphi(x) = R(t_1, \ldots, t_k) \) for some relational symbol \( R \in \tau \) and tuple of terms \( (t_1, \ldots, t_k) \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi(x)[x/a] \) iff \( (b_1, \ldots, b_k) \in R^H \) (i.e., the “here” structure) such that for \( 1 \leq i \leq k \):

\[
 b_i = \begin{cases} 
 a_j & \text{if } t_i = x_j \text{ for some } 1 \leq j \leq s \\
 c^T & \text{if } t_i = c \text{ for some constant } c \in \tau
\end{cases}
\]

• If \( \varphi = (\psi \land \xi) \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi(x)[x/a] \) iff \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \psi(x)[x/a] \) and \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \xi(x)[x/a] \);

• If \( \varphi = (\psi \lor \xi) \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi(x)[x/a] \) iff \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \psi(x)[x/a] \) or \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \xi(x)[x/a] \);

• If \( \varphi = \exists x \psi \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi(x)[x/a] \) iff there exist some \( a \in \text{Dom}(\mathcal{T}) \) such that \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \psi(xx)[xx/aa] \) (i.e., where \( xx = x_1 \cdots x_s x \) and \( aa = a_1 \cdots a_s a \) are extensions of \( x \) and \( a \) respectively);

• If \( \varphi = \forall x \psi \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi(x)[x/a] \) iff for all \( a \in \text{Dom}(\mathcal{T}) \), we have \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \psi(xx)[xx/aa] \);

• If \( \varphi = (\psi \rightarrow \xi) \), then \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi(x)[x/a] \) iff:

\[
 - \langle \mathcal{H}, \mathcal{T} \rangle \not\models_{HT} \psi(x)[x/a] \text{ or } \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \xi(x)[x/a] \\
 - \mathcal{T} \models (\psi \rightarrow \xi)[x/a] \text{ (i.e., where } \models \text{ is simply the “classical” FO satisfaction relation)}
\]

and where we view \( \neg \varphi \) as a shorthand for \( \varphi \rightarrow \bot \) as usual. Note that when \( \varphi \) is a sentence (i.e., has no free variables), then the HT satisfaction relation is simply \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi \).

It follows from [PV05] that a HT interpretation \( \langle \mathcal{H}, \mathcal{T} \rangle \) is an equilibrium model of a FO sentence \( \varphi \) iff:

1. \( \langle \mathcal{H}, \mathcal{T} \rangle \models_{HT} \varphi \) and where for each relation symbol \( R \in \tau \), we have that \( R^H = R^T \), i.e., loosely speaking, “here” and “there” are the same;
2. For all other \( \tau \)-structures \( \mathcal{H}' \) such that:

\begin{enumerate}
\item \( \text{Dom}(\mathcal{H}') = \text{Dom}(\mathcal{H}) \);
\item \( c^{\mathcal{H}'} = c^{\mathcal{H}} \) for each constant symbol \( c \in \tau \);
\item \( R^{\mathcal{H}'} \subseteq R^{\mathcal{H}} \) for each relation symbol \( R \in \tau \);
\item \( R^{\mathcal{H}'} \subset R^{\mathcal{H}} \) for some relation symbol \( R \in \tau \),
\end{enumerate}

we have that \( \langle \mathcal{H}', T \rangle \not\models_{HT} \varphi \), i.e., this is the minimal “here” condition part.

As in the propositional case, it also follows from [PV05] that the equilibrium models of a FO sentence \( \varphi \) are precisely its stable models.

As a classical second-order formula

In [FLL11], this notion of FO “equilibrium models” was directly encoded into a classical formula via their so-called \( SM \) operator. This was achieved by encoding the minimal “here” world condition of the “reduct” via a universal second-order (SO) sentence. For a given FO formula \( \varphi \) of signature \( \tau \), \( SM(\varphi) \) is the following (classical) SO sentence:

\[
\varphi \land \forall \mathbf{U}(\mathbf{U} < \mathbf{P} \rightarrow \neg \varphi^*(\mathbf{U})),
\]

where:

\begin{itemize}
\item \( \mathbf{P} = P_1 \ldots P_n \) is a tuple of distinguishable predicates such that \( P_1, \ldots, P_n \in \tau \);
\item \( \mathbf{U} = U_1 \ldots U_n \) is a fresh (i.e., they are not in \( \tau \)) tuple of distinguishable predicates such that the arity of \( U_i \) matches the arity of \( P_i \) for \( 1 \leq i \leq n \);
\item \( \varphi^*(\mathbf{U}) \) is defined recursively based on \( \varphi \) as follows:
\end{itemize}

\begin{itemize}
\item If \( \varphi = P(\mathbf{x}) \) for some tuple of terms \( \mathbf{x} \) (i.e., can contain both variables and constants), then \( \varphi^*(\mathbf{U}) = U_i(\mathbf{x}) \) if \( P = P_i \) for some \( 1 \leq i \leq n \), and \( \varphi^*(\mathbf{U}) = P(\mathbf{x}) \) otherwise;
\item If \( \varphi = \odot \psi \) for \( \odot \in \{ \forall, \exists \} \), then \( \varphi^*(\mathbf{U}) = \odot \psi^*(\mathbf{U}) \);
\item If \( \varphi = \psi \odot \xi \) for \( \odot \in \{ \land, \lor \} \), then \( \varphi^*(\mathbf{U}) = \psi^*(\mathbf{U}) \odot \xi^*(\mathbf{U}) \);
\item If \( \varphi = \psi \rightarrow \xi \), then \( \varphi^*(\mathbf{U}) = (\psi^*(\mathbf{U}) \rightarrow \xi^*(\mathbf{U})) \land (\psi \rightarrow \xi) \),
\end{itemize}
and such that we view \( \neg \varphi \) as \( \varphi \rightarrow \bot \) as usual.

Then from [FLL11], it was shown that a \( \tau \)-structure \( M \) satisfies \( SM(\varphi) \) iff the HT interpretation \( \langle M, M \rangle \) is an equilibrium model of \( \varphi \). Thus, it follows from the notion of FO equilibrium models that \( M \models SM(\varphi) \) iff \( M \) is a stable model of \( \varphi \).

It follows that if \( \varphi \) is an implication free formula (i.e., free of the ‘→’ connective), then \( \varphi^*(U) \) is simply the formula obtained from \( \varphi \) by replacing every occurrences of atoms \( P_i(x) \) in it (where \( 1 \leq i \leq n \) and \( x \) is a tuple of terms whose length matches the arity of \( P_i \)) by \( U_i(x) \). Therefore, if \( \varphi \) is implication free, then we simply have that \( SM(\varphi) \) corresponds to the expression of the minimal models of \( \varphi \) with respect to the predicates in the tuple \( P \).

The scenario now becomes different when implications are introduced into \( \varphi \) since from above, we have that

\[
(\psi \rightarrow \xi)^*(U) = (\psi^*(U) \rightarrow \xi^*(U)) \land (\psi \rightarrow \xi),
\]

i.e., we also make a copy of \((\psi \rightarrow \xi)\) along with \((\psi^*(U) \rightarrow \xi^*(U))\). Note now how this corresponds to the logic of HT. Indeed, loosely speaking, we have that the “\((\psi^*(U) \rightarrow \xi^*(U))\)” part corresponds to the “here” world and the “\((\psi \rightarrow \xi)\)” part as the “there” world. In fact, in reference to a FO HT interpretation \( \langle H, T \rangle \), we can loosely view the relations of those predicates from the tuple \( U \) as those of the structure \( H \), while those relations of the predicates from \( P \) as those of \( T \). Thus, as in the case of equilibrium logic, this notion corresponds to the “activation” of the reduct of \( \varphi \) under the interpretations of those predicates from \( P \), i.e., intuitively speaking, we fix \( P \) and minimize \( U \).

Incidentally, it is also important to note the work in [LZ11] where they captured the stable models of arbitrary formulas in terms of circumscription [Lif85, McR86]. In terms of the ‘\( \ast \)’ operator, the formula in [LZ11] follows the form \( CIRC(\varphi^*(U)) \land U = P \) where \( CIRC(\varphi^*(U)) = \varphi^*(U) \land \forall U(U < P \rightarrow \neg \varphi^*(U)) \) and \( U = P \) stand for \((U < P) \land (P < U)\).

### 1.2.3 Loop Formulas Approach

In Lin and Zhao’s work [LZ04], the answer sets of propositional normal programs was defined in terms of the so-called loops and loop formulas. Generally speaking, the loop formulas \( LF(\Pi) \) of a (propositional) program \( \Pi \) is a way of strengthening the Clark’s
CHAPTER 1. INTRODUCTION TO ANSWER SET PROGRAMMING

completion of (propositional) programs so that it corresponds exactly to the answer.

More formally, let $\Pi$ be a propositional program and $\tau(\Pi)$ be the set of all the propositional atoms in it, i.e., its signature. Now let $L \subseteq \tau(\Pi)$. Then by $LF(L)$, define the following propositional formula

$$ \bigwedge_{a \in L} a \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) \in L, \text{Pos}(r) \cap L = \emptyset} Body(r), \quad (1.10) $$

where for a (propositional) rule $r$ of the form (1.1), $\overline{Body(r)}$ denotes the conjunctions $b_1 \land \cdots \land b_l \land \neg c_1 \land \cdots \land \neg c_m$ of literals in $Body(r)$. Intuitively speaking, $LF(L)$ is viewed as the external support of $L$ in the sense that if all the atoms in $L$ are true, then it must be supported by some rule $r \in \Pi$ for which $\text{Pos}(r) \cap L = \emptyset$, i.e., the “external” support. This notion was then extended to all the possible subsets $L \subseteq \tau(\Pi)$ such that $LF(\Pi)$ denotes the formula

$$ \bigwedge_{L \subseteq \tau(\Pi)} LF(L), $$

i.e., all the loop formulas of $\Pi$. It was then shown in [LZ04] that the formula $\overline{\Pi} \land LF(\Pi)$ (i.e., where $\overline{\Pi}$ is the logical closure of $\Pi$) exactly captures the answer sets of $\Pi$.

This notion of loop formulas for normal (propositional) programs was then extended to disjunctive programs [FLL06] (and to the more general programs with nested expressions). This was achieved by modifying the definition of a loop in such a way that a program is turned into the corresponding propositional formula by simply adding loop formulas directly to the conjunctions of the rules, which eliminated the intermediate step of forming the program’s Clark’s completion.

Up to this point, most of the work about loop formulas have been restricted to the propositional case. That is, variables in programs are first eliminated by grounding, through which the loop formulas are then formed from the resulting ground program. For this reason, loop formulas are only defined as formulas in propositional logic. A relatively closer to first-order form of loop formulas was given in Chen et al.’s definition of loop formulas [CLWZ06]. This notion of loop formulas in [CLWZ06] is different since the loop formulas are obtained from the original programs without first converting into a ground
program so that the variables remain. So in this sense, this notion of loop formulas are closer to “first-order.” However though, since the the underlying semantics of the logic programs still referred to grounding, then such loop formulas is still understood as schema for the set of propositional loop formulas. In the same way Ferraris et. al in [FLL06] extended the notion of loop formulas to the more general case of disjunctive programs and programs with nested expressions, this notion of “first-order” loop formulas in [CLWZ06] is also extended in Lee et. al. [LM11] to first-order disjunctive programs, and then to arbitrary first-order formulas.

1.2.4 ASP with Aggregates

One of the reasons that ASP has emerged as a predominant declarative programming paradigm in the area of knowledge representation and logic programming [Bar03] is due to its rich modeling language [GKKS09]. One important ASP building block is the so-called aggregate constructs. Some particular well known constructs of these aggregates are the cardinality and weight constraints [SNS02], which plays an important role in many applications. Aggregates are currently a hot topic in ASP not only because of their significance, but also since there is no uniform understanding of the concept of an aggregate [Fer11]. The reason why aggregates are crucial in answer set solving is twofold. First, it can simplify the representation task. For many applications, one can write a simpler and more elegant logic program by using aggregates, for instance, the job scheduling program [PSE04]. More importantly, it can improve the efficiency of ASP solving [GKKS09]. Normally, the program using aggregates can be solved much faster [FPL+08].

At the propositional level, a weight constraint atom is a construct of the form

\[ M \leq \{p_1 = n_1, \ldots, p_k = n_k\} \leq N, \]  

(1.11)

where \( p_1, \ldots, p_n \) are propositional atoms, and \( M, N, n_1, \ldots, n_k \) are numbers from \( \mathbb{Z} \) (the set of integers). The \( n_i \) of each corresponding \( p_i \) (for \( 1 \leq i \leq k \)) is known as the “weight” associated with \( p_i \). Intuitively, (1.11) means that the sum of all the \( n_i \) for which a corresponding \( p_i \) is true must be more than or equal to \( M \), but less than or equal to \( N \). More
formally, for a given set of propositional atoms $I$, we say that

$$I \models M \leq \{p_1 = n_1, \ldots, p_k = n_k\} \leq N$$

if and only if

$$M \leq \sum_{p_i \in I \cap \{p_1, \ldots, p_k\}} n_i \leq N.$$ 

The case for the cardinality constraint is the same as the weight constraint but where $n_i = 1$ for $1 \leq i \leq k$. An example of an ASP program with a weight constraint atom is the following:

\begin{align*}
  & p \leftarrow 2 \leq \{q = 1, r = 2\} \leq 3 \quad (1.12) \\
  & q \leftarrow p \quad (1.13) \\
  & q \leftarrow r \quad (1.14) \\
  & r \leftarrow, \quad (1.15)
\end{align*}

where rule (1.12) states that if the corresponding weights of the true atoms in $\{q, r\}$ is between 2 and 3 inclusively (i.e., by the weight constraint atom $2 \leq \{q = 1, r = 2\} \leq 3$), then we also have $p$.

As it is well known in the literatures, aggregate constructs can contain variables although, since the definition of their semantics still referred to grounding, it is still essentially propositional. An example of an ASP program with aggregates containing variables is the following well known company control program [FPL11]:

\begin{align*}
  ControlsStk(x, x, y, n) &\leftarrow OwnsStk(x, y, n) \\
  ControlsStk(x, y, z, n) &\leftarrow Company(x), Controls(x, y), OwnsStk(y, z, n) \\
  Controls(x, z) &\leftarrow Company(x), Company(z) \\
  \sum \{n : \exists y ControlsStk(x, y, z, n)\} &> 50,
\end{align*}

which is clearly a program with variables in the aggregate atom

$$\sum \{n : \exists y ControlsStk(x, y, z, n)\} > 50.$$
Traditionally, the answer sets of programs with aggregates was computed by first grounding the program, and then on computing the answer sets of the resulting propositional program (as in the case of the aggregate free programs). It is for this reason that the traditional approach is not truly first-order. In the recent years, the first-order stable semantics was extended to include aggregates by Lee et. al. in [LM09]. As this notion neither referred to grounding or fixpoint, it is viewed as truly first-order. Even more recently, the FLP aggregate semantics of Faber, Leone, and Pfeifer [FPL11], as initially defined via grounding, was also uplifted to the first-order case by Bartholomew et. al. in [BLM11].

It should be noted that although the recent characterizations of first-order logic programs with aggregates are viewed as truly first-order, they are defined via a SO sentence. Another of the main contributions of this thesis is that we provide a translation of ASP programs with a significant subclass of aggregates, called monotone and anti-monotone, into classical FO logic [AZZar]. We emphasize that although we only consider these subclass of aggregates, they are indeed powerful because, as far as we had check, almost all the aggregates mentioned in benchmark programs [CIR+11] belong to this class.

1.2.5 Preferred ASP

Getting back again to the case of propositional program, the expressive power of answer set programming was even enhanced by incorporating preferences among the rules.\(^4\) This brings us to the realm of the so-called preferred answer set programming. A preferred propositional program \(P\) is a structure \((\Pi, \prec^P)\) where \(\Pi\) is propositional program and \(\prec^P\) is an asymmetric and transitive partial relation among the rules of \(\Pi\). So in other words, a preferred propositional program \(P = (\Pi, \prec^P)\) is a propositional program \(\Pi\) together with a strict-partial ordering \(^5\) among its rules. For example, assume we have \(P\) to be the preferred program \((\Pi, \prec^P)\) such that \(\Pi\) is the propositional program

\[
(1) \quad b \leftarrow \neg c, a \\
(2) \quad c \leftarrow \neg b \\
(3) \quad a \leftarrow \neg d,
\]

\(^4\)Theoretically, some formalisms enhance expressive power since they step outside the complexity class of the underlying ASP programming framework, e.g., the approaches proposed in [Rin95] and [ZF97].

\(^5\)Note that the strict-partial order need not be total.
and where \((1) <^P (2) <^P (3)\), i.e., \(<^P = \{(1, (2)), (2, (3)), (1, (3))\}\) where (1), (2), and (3) stands for the rules “\(b \leftarrow \text{not } c, a\)”, “\(c \leftarrow \text{not } b\)”, and “\(a \leftarrow \text{not } d\)”, respectively.

Then the strict-partial order \(<^P\) on \(\Pi\) states that: rule (1) is more preferred than rule (2); (2) is more preferred than (3); and (1) is more preferred than (3) (i.e., by the transitive closure of \(\{(1), (2)\}\) and \(\{(2), (3)\}\)).

As in the traditional way of viewing a program with variables, the usual case of viewing a program with variables and preferences as simply a shorthand of its Herbrand instantiation poses a bottleneck. For instance, assume we have \(\mathcal{P} = (\Pi, <^P)\) such that \(\Pi\) is the program with variables

\[
\begin{align*}
(1) & \quad P(x, y) \leftarrow Q(x, y) \\
(2) & \quad P(x, z) \leftarrow Q(x, y), Q(y, z)
\end{align*}
\]

and where \((1) <^P (2)\). Then the grounding (or “propositionalizing”) of \(\Pi\) to a singleton constant \(\{a\}\) renders (1) and (2) to collapse to the same propositional rule “\(P(a, a) \leftarrow Q(a, a)\)” violating the intuition of the preference \((1) <^P (2)\). In fact, due to this condition and in some preferred answer frameworks, they require certain auxiliary conditions such as “well behaved” or “consistent,” which are simply assumptions that such a rule collapse will not happen in the grounding of the program, e.g., as found in [BE99].

In this work, we provide a truly first-order characterization of preferred FO programs. We achieve this in two ways: (1) by a FO fixpoint definition; (2) by a classical SO logic sentence. In addition, we also show that the fixpoint and classical logic characterizations are equivalent. Moreover, when only restricted to the confines of finite structures, the SO sentence reduces down to an existential SO sentence. In fact, this SO sentence is polynomial in the length of the underlying FO preferred program. This now allows us, for the first time (to the best of our knowledge), to reason about FO preferred programs at a truly first-order level. Furthermore, we achieved this FO preferred answer set characterizations by generalizing important preferred propositional program frameworks to the FO level, e.g., as those mentioned in [SW03].
1.2.6 Important Preferred Program Frameworks

In this section, we briefly review some important preferred program frameworks of the past years. We emphasize though the fact that all these frameworks are only defined for propositional preferred programs.

**Brewka and Eiter’s framework and Principles I and II**

Brewka and Eiter studied essential issues in *preferred default reasoning* and argued that all preferred answer set reasoning approaches should satisfy the following two basic principles [BE99]:

**Principle I.** Let \( A_1 \) and \( A_2 \) be two answer sets of a preferred propositional program \( \mathcal{P} = (\Pi, \prec^P) \) generated by the rules \( R \cup \{ r_1 \} \subseteq \Pi \) and \( R \cup \{ r_2 \} \subseteq \Pi \) respectively, where \( r_1, r_2 \not\in R \). If \( r_1 \) is preferred over \( r_2 \), then \( A_2 \) is not a preferred answer set of \( \Pi \).

**Principle II.** Let \( A \) be a preferred answer set of a preferred propositional program \( \mathcal{P} = (\Pi, \prec^P) \) and \( r \) a rule in \( \Pi \) such that there exist an \( a \in Pos(r) \) for which \( a \not\in A \). Then \( A \) is a preferred answer set of \( \mathcal{P}' = (\Pi, \prec^{P'}) \) whenever \( \prec^{P'} \) agrees with \( \prec^P \) on the preferences among the rules in \( \Pi \).

Brewka and Eiter observed that some preferred answer set reasoning frameworks do not satisfy these principles and proposed their own preferred semantics that satisfies these principles. Their preferred answer set semantics is based on a reduction of the underlying program to one where only negative bodies are left in the program’s rules. That is, while the Gelfond-Lifschitz reduction leaves us with a “reduced” program whose rules only contain a positive body (i.e., the negative part was deleted), the Brewka-Eiter reduction, or just BE-reduction, leaves us with a “reduced” program whose rules only contain a negative body (i.e., this time it is the positive part that is deleted). It should be noted that the preferred program framework as developed by Brewka and Eiter had been viewed as an important proposal in preferred program reasoning since it satisfies the two basic principles addressed above [BE99].

In order to define the notion of the Brewka and Eiter preferred (or just BE-preferred) program semantics, for now, it will be necessary to assume a preferred propositional program \( \mathcal{P} = (\Pi, \prec^P) \) where \( \prec^P \) is a strict-total order on \( \Pi \). In such a case, we refer to \( \mathcal{P} \)
as a fully preferred propositional program. For convenience, from here on, we assume all propositional programs to be of signature $\mathcal{L}$.

**Definition 1 (BE-reduction)** [BE99] Let $\mathcal{P} = (\Pi, <^P)$ be a fully preferred propositional program and $I \subseteq \mathcal{L}$ an interpretation $I$ of $\mathcal{L}$. Then $\mathcal{P}' = (I\Pi, <^{P'})$ denotes the fully preferred program such that $I\Pi$ is the set of rules obtained from $\Pi$ by:

1. deleting every rules having an atom $a$ in its positive body where $a \notin I$, and
2. removing from each remaining rules their positive body,

and where $<^{P'}$ is obtained from $<^P$ by the map $f : \Pi \rightarrow \Pi$ such that $r_1' <^{P'} r_2'$ iff $f(r_1') <^P f(r_2')$, and where $f(r') = r$ is the first rule in $\Pi$ with respect to $<^P$ such that $r'$ results from $r$ by step 2.

It should be noted that $<^{P'}$ is still a strict-total order on $I\Pi$ above. Thus, having defined the BE-reduction, we are now ready to define the BE-preferred semantics for propositional programs. Generally speaking, the BE-preferred semantics is defined through an immediate consequence operator$^6$

$$C_P : 2^\mathcal{L} \rightarrow 2^\mathcal{L}$$

such that an answer set $A \subseteq \mathcal{L}$ of $\Pi$ satisfies the preferences iff $C_P(A) = A$. Before proceeding to define the consequence operator $C_P$, it will first be helpful to introduce the following notion. For a given fully preferred program $\mathcal{P} = (\Pi, <^P)$, the rules in $\Pi$ corresponds to a unique ordinal number as given by $|\Pi|$, and thus, to an enumeration $r_1, \cdots, r_{|\Pi|}$ of the elements of $\Pi$. Therefore, we also use the notation $\{r_\alpha\}^{<^P}$ to represent $(\Pi, <^P)$.

**Definition 2 [BE99]** Let $\mathcal{P} = (\Pi, <^P)$ be a fully preferred propositional program and $I$ an interpretation $I \subseteq \mathcal{L}$. Moreover, let $\mathcal{P}' = (I\Pi, <^{P'}) = \{r_\alpha\}^{<^{P'}}$ where $I\Pi$ and $<^{P'}$ is as

$^6$Note that $2^\mathcal{L}$ denotes the set of all the subsets of $\mathcal{L}$, i.e., $2^\mathcal{L} = \{S \mid S \subseteq \mathcal{L}\}$. 

defined in Definition 1. Then the sequence $S_\alpha$ is defined inductively as follows:

$$S_0 = \emptyset;$$

$$S_{\alpha+1} = \begin{cases} S_\alpha, & \text{if } r_{\alpha+1} \text{ is defeated by } S_\alpha, \text{ or} \\ Head(r_{\alpha+1}) \in S \text{ and } r_{\alpha+1} \text{ is defeated by } S, \\ S_\alpha \cup \{\text{Head}(r_{\alpha+1})\}, & \text{otherwise,} \end{cases}$$

Then $C_P(S) = S|_{\Pi}$, where for rule $r$ and a set $S \subseteq \mathcal{L}$, we take “$r$ is defeated by $S$” to mean that $\text{Neg}(r) \cap S \neq \emptyset$.

Then it follows from [BE99] that for a fully preferred program $P = (\Pi, \prec^P)$, an answer set $A$ of $\Pi$ is a preferred answer set of $P$ iff $C_P(A) = A$.

In cases where $P$ is not a fully preferred program (i.e., $\prec^P$ is not a strict-total order but rather a strict-partial order), then the BE-preferred framework is defined by extending $P$ to a fully preferred one of $P' = (\Pi, \prec^{P'})$ such that $\prec^P \subseteq \prec^{P'}$. Thus, an answer set $A$ of $\Pi$ is then said to be a preferred answer set of a partially-ordered preferred program $P = (\Pi, \prec^P)$ iff there exists a fully preferred program $P' = (\Pi, \prec^{P'})$ where $\prec^P \subseteq \prec^{P'}$ such that $A$ is a preferred answer set of $P'$.

**Proposition 1** [BE99] The BE-preferred answer set framework satisfies Principles I and II above.

**Proposition 2** [AZ09] On finite propositional programs, the BE-preferred answer sets can be captured by propositional formulas without extending the language’s original vocabulary, i.e., no auxiliary atoms.

**Torsten and Wang’s unifying frameworks**

In [SW03], Torsten and Wang provided a unifying framework that linked three important preferred frameworks. We refer to these as follows:

**BE-preferred** : with “BE” standing for Brewka and Eiter and refers to the framework as already mentioned above;

**DST-preferred** : with “DST” standing for Delgrande, Schaub and Tompits, which refers to the preferred program framework as introduced in [DST00];
**WZL-preferred**: with “WZL” standing for Wang, Zhou and Lin, which refers to the preferred framework as proposed in [WZL00].

Torsten and Wang unified these three important frameworks by relating the *explicit* preference relations among the rules with the so-called notion of “groundedness”. In this notion of “groundedness,” a program $\Pi$ is said to be “grounded” if there exist a strict-total order $T = (\Gamma^l_\Pi, <^T)$ of the generating rules of $\Pi$ under $I$ (i.e., the set $\Gamma^l_\Pi$ where $\Gamma^l_\Pi = \{ r \mid \text{Pos}(r) \subseteq I \text{ and } \text{Neg}(r) \cap I = \emptyset\}$) such that $\text{Pos}(r) \subseteq \{ \text{Head}(r') \mid r' <^T r \}$ for each rule $r \in \Gamma^l_\Pi$. It was found in [SW01] that the DST-preferred framework captures the notion of “groundedness” by simply having the strict-total order corresponding to the program’s “groundedness” to respect the preference relations among the rules of the program. This is made precise in the following definition:

**Definition 3 (DST-preferred answer sets)** [SW03] An interpretation $I \subseteq \mathcal{L}$ is said to be a DST-preferred answer set of a preferred propositional program $\mathcal{P} = (\Pi, <^P)$ iff:

1. $I$ is an answer set of $\Pi$;
2. there exist a strict-total order $T = (\Gamma^l_\Pi, <^T)$ of $\mathcal{P}$ such that:

   (a) $\text{Pos}(r) \subseteq \{ \text{Head}(r') \mid r' <^T r \}$ for each rule $r \in \Gamma^l_\Pi$ (i.e., the groundedness condition);

   (b) $r_1 <^P r_2$ implies $r_1 <^T r_2$ for each rule $r_1, r_2 \in \Gamma^l_\Pi$ (i.e., the total-order $T$ respects the preference relations of $\mathcal{P}$);

   (c) for each $r_1 \in \Pi \setminus \Gamma^l_\Pi$ and $r_2 \in \Gamma^l_\Pi$ where $r_1 <^P r_2$, either:

      i. $\text{Pos}(r_1) \not\subseteq I$ or

      ii. $\text{Neg}(r_1) \cap \{ \text{Head}(r) \mid r <^T r_2 \} \neq \emptyset$.

As already briefly mentioned above, Conditions (2) (a) and (b) simply corresponds to the “groundedness” of $\Pi$ under the strict-total order $T$. On the other hand, Condition (2) (c) simply insures that for each non-generating rule $r_1$ (i.e., those rules not in $\Gamma^l_\Pi$) and generating rule $r_2$ (i.e., $r_2 \in \Gamma^l_\Pi$), and such that $r_1$ is more preferred than $r_2$ (i.e., $r_1 <^P r_2$), we have that either: (i) $r_1$ is not derivable since $\text{Pos}(r_1) \not\subseteq I$ or (ii) $r_2$ is already “defeated”.

---

7We refer to this notion of groundedness by “groundedness.”

8It should be noted that [SW03] is the TPLP-2003 journal counterpart of the IJCAI-2001 paper [SW01].
by a rule $r$ (i.e., “defeated” as in $\text{Head}(r) \in \text{Neg}(r)$) where $r <^T r_2$. Intuitively speaking, Condition (2) (c) above simply insures that the non-application of more preferred rules are already settled before considering the application of the lesser preferred ones.

It was found in [SW03] that the WZL-preferred framework is captured by this notion of “groundedness” through a slight modification of Definition 3. In fact, it was found that the WZL-preferred framework is nothing more than the DZL-preferred framework but with the property that it exploits a weaker form of “groundedness.” This notion is made precise in the following definition:

**Definition 4 (WZL-preferred answer sets)** [SW03] An interpretation $I \subseteq L$ is said to be a WZL-preferred answer set of a preferred propositional program $\mathcal{P} = (\Pi, <^P)$ iff:

1. $I$ is an answer set of $\Pi$;

2. there exist a strict-total order $\mathcal{T} = (\Gamma^I \Pi, <^T)$ of $\mathcal{P}$ such that:
   
   (a) $\text{Pos}(r) \subseteq \{\text{Head}(r') \mid r' <^T r\}$ or $\text{Head}(r) \in \{\text{Head}(r') \mid r' <^T r\}$ for each rule $r \in \Gamma^I \Pi$ (i.e., the weaker groundedness condition);

   (b) $r_1 <^P r_2$ implies $r_1 <^T r_2$ for each rule $r_1, r_2 \in \Gamma^I \Pi$ (i.e., the total-order $\mathcal{T}$ respects the preference relations of $\mathcal{P}$);

   (c) for each $r_1 \in \Pi \setminus \Gamma^I \Pi$ and $r_2 \in \Gamma^I \Pi$ where $r_1 <^P r_2$, either:

   i. $\text{Pos}(r_1) \not\subseteq I$ or

   ii. $\text{Neg}(r_1) \cap \{\text{Head}(r) \mid r <^T r_2\} \neq \emptyset$ or

   iii. $\text{Head}(r_1) \in \{\text{Head}(r) \mid r <^T r_2\} \neq \emptyset$.

It is not too difficult to see that Definition 4 of the WZL-preferred framework only differs from Definition 3 of the DST-preferred through Conditions (2) (a) and (c). Indeed, Condition (2) (a) of Definition 4 has the more liberal condition that it is enough for $\text{Head}(r) \in \{\text{Head}(r') \mid r' <^T r\}$ for each $r \in \Gamma^I \Pi$, which is in contrast with that of (2) (a) of Definition 3 where we explicitly required that $\text{Pos}(r) \subseteq \{\text{Head}(r') \mid r' <^T r\}$. Similarly, Condition (2) (c) of Definition 4 also has the more liberal condition that it is enough for $\text{Head}(r_1) \in \{\text{Head}(r) \mid r <^T r_2\} \neq \emptyset$ in order to settle the non-applicability of $r_1$.

Relating now to the BE-preferred framework, it was also found in [SW03] that the BE-preferred framework can also be characterized in terms of this “groundedness” property.
In fact, of the three BE, DST, and WZL-preferred frameworks, it was found that the BE-preferred possesses the weakest of all the "groundedness" requirement. To see this, let us consider the precise definition of the BE-preferred answer sets as found in [SW03]:

**Definition 5 (BE-preferred answer sets version of Schaub and Wang) [SW03]** An interpretation $I \subseteq \mathcal{L}$ is said to be a BE-preferred answer set of a preferred propositional program $\mathcal{P} = (\Pi, <^P)$ iff:

1. $I$ is an answer set of $\Pi$;
2. there exist a strict-total order $\mathcal{T} = (\Gamma^I, <^T)$ of $\mathcal{P}$ such that:
   - $r_1 <^P r_2$ implies $r_1 <^T r_2$ for each rules $r_1, r_2 \in \Gamma^I_\Pi$ (i.e., the total-order $\mathcal{T}$ respects the preference relations of $\mathcal{P}$);
   - for each $r_1 \in \Pi \setminus \Gamma^I_\Pi$ and $r_2 \in \Gamma^I_\Pi$ where $r_1 <^P r_2$, either:
     1. $\text{Pos}(r_1) \not\subseteq I$ or
     2. $\text{Neg}(r_1) \cap \{\text{Head}(r) \mid r <^T r_2\} \neq \emptyset$ or
     3. $\text{Head}(r_1) \subseteq I$.

Clearly, it can be seen that the "groundedness" property is outright omitted in Definition 5 of the BE-preferred framework. In fact, for any preferred program $\mathcal{P} = (\Pi, <^P)$, with $\mathcal{AS}_{DST}(\mathcal{P}), \mathcal{AS}_{WZL}(\mathcal{P})$ and $\mathcal{AS}_{BE}(\mathcal{P})$ standing for the set of all the DST, WZL and BE-preferred answer sets of $\mathcal{P}$, respectively, we have the following relationships from [SW03]:

$$\mathcal{AS}_{DST}(\mathcal{P}) \subseteq \mathcal{AS}_{WZL}(\mathcal{P}) \subseteq \mathcal{AS}_{BE}(\mathcal{P}).$$

Thus, the relative "weak" or "strong" characterizations of the DST, WZL, and BE-preferred answer sets rest entirely upon this notion of "groundedness."

It should be noted that [SW03] also provided fixpoint type characterizations of those three aforementioned preferred answer set frameworks. We delve deeper into these fixpoint type characterizations in Chapter 4.

1.3 Structure of the Thesis

The rest of this thesis is organized as follows. Chapter 2 addresses the main theme of the thesis that is about the translation of logic programs under the stable model semantics
(answer set programs) to classical first-order logic. Section 2.4 then further extends our first-order logic translation to consider logic programs with aggregate constructs. Chapter 3 describes our first implementation of the answer set solver Groc [ALZZ12], that works in a different manner to the traditional approach by first translating a logic program into a first-order sentence, and then grounding the sentence while incorporating “classical” reasoning. Experimental results are also reported in this chapter. Chapter 4 studies the problem of preferred first-order answer set programming, which is first-order logic programming where the rules in the programs can have a preference relation. As already mentioned above, adding preferences to rules with variables (i.e., a “first-order rule”) can introduce the additional problem where the preference relation among two rules can collapse to a single propositional rule, if the semantics of preferred first-order programs is defined via grounding. We address this problem in Chapter 4 by lifting some well known preferred propositional answer set programming frameworks to first-order. We achieve this by introducing a first-order progression (or fixpoint) type definition, and also through a classical second-order sentence such that on finite structures, the second-order sentence can be reduced down to first-order. Chapter 5 finally summarizes the contributions provided by this thesis. Long proofs of lemmas and theorems are left to Appendix A for a more fluent reading.
Chapter 2

First-Order ASP and Classical First-Order Logic

2.1 Past Efforts in Linking Logic Programs and Classical Logics

This chapter is about translating first-order (FO) programs under the stable models semantics [FLL11] to classical FO logic. Viewed in the context of expressing FO programs into classical FO logic, work on this direction goes all the way back to Clark [Cla77] who gave us what is now called the Clark’s completion semantics, on which our work is based.

As mentioned in Section 1.2.1, the “stable models” semantics of FO (normal) programs is defined by a (universal) SO sentence via the $SM(\Pi)$ formula. Recall that the $SM(\Pi)$ formula is a universal SO sentence of the form

$$\widehat{\Pi} \wedge \forall U (U < P \rightarrow \neg \widehat{\Pi}^*(U))$$

that captures the notion of the minimal models of the “reduct” (i.e., $\widehat{\Pi}^*(U)$) to the first-order level. As also mentioned in Section 1.2.1, in this work, we provide a translation of FO programs under the stable models semantics to classical FO logic sentences, which we do by introducing extra predicates that are polynomial in the size of the original number of intensional predicates of the underlying program. In fact, to the best of our knowledge, it provides for the first time a translation from FO (normal) programs (under the stable
models semantics) to classical FO logic sentences.

In terms of the stable models semantics, Clark’s completion semantics is, in general, too weak in the sense that not all classical models of the “Clark’s completion” are also stable models, unless the programs are “tight” [Fag94]. Various ways to remedy this have been proposed, particularly in the propositional case of programs without variables. This is due to the recent interest in answer set programming and the prospect of using SAT solvers to compute the answer sets, e.g., [LZ04]. Our work though considers FO programs and the prospect of capturing the answer sets of these programs in (classical) FO logic. Thus, under this new direction, we can now use the mature developments of the area of automated FO theorem proving to reason about FO programs for the first time. This will be beneficial in the simplification of FO programs for the purpose of optimization. For instance, there had already been substantial work done on the direct use of FO reasoning for the optimization of the grounding process of (classical) FO sentences, e.g., [WMD10].

In the past years, for the case of propositional programs, a way of expressing programs into classical propositional logic is by introducing extra propositional symbols, i.e., “expanding” the signature of the underlying language and translating the program into a classical propositional logic formula. An example of this is the Ben-Eliyahu and Dechter’s translation [BED94] of propositional programs to (classical) propositional logic that is polynomial in space but uses $O(n^2)$ extra propositional symbols. Another direction of mapping propositional programs to propositional logic is via the well-known notion of loop formulas of Lin and Zhao [LZ04]. The “loop formulas” approach does not introduce extra propositional symbols into the corresponding (classical) logic formula but is exponential in the number of the propositional symbols of the underlying program in the worst case. In the realm of FO programs, Chen et al. [CLWZ06] extended the notions of “loops” and “loop formulas” to the FO case and showed that on finite domains, the answer sets of a FO program can be captured by its Clark’s completion and all its FO loop formulas. However, in general, a program may have an infinite number of loops and loop formulas. Furthermore, the approach in [CLWZ06] still only treated the semantics of FO programs as the usual notion of “programs with variables” (i.e., still only as a shorthand of its Herbrand instantiation as mentioned in Section 1.2.1) since the variables of the underlying programs are viewed as just place holders and such that rules with variables are schemas. Hence, in

\[1\text{More discussions on “tight” programs in Section 2.3.1.}\]
this sense, the FO programs considered in [CLWZ06] are not truly FO. But this is the best that one can possibly hope for if no extra predicates are used. For instance, it is well-known that the program

\[
\begin{align*}
T(x, y) & \leftarrow E(x, y) \\
T(x, z) & \leftarrow E(x, y), T(y, z)
\end{align*}
\]

that computes the transitive closure of the relations under \(E\) (and thus, can be easily written as a FO program) cannot be captured by any finite first-order theory on finite structures [EF99].

The situation now becomes different if we introduce extra predicates for the corresponding classical FO logic formulas. One of the main technical results of this thesis is that, by using some additional predicates that keep track of the derivation order from the bodies to the heads of the underlying program, we can modify Clark’s completion into what we call the \textit{ordered completion}, which exactly captures the answer set semantics on finite structures.

### 2.2 The Clark’s Completion

Firstly, recall that as mentioned in Section 1.2.1, we view the predicate symbols of a signature \(\tau(\Pi)\) of a given program \(\Pi\) as being either of an \textit{extensional} or \textit{intensional} predicate, which we denote by \(P_{\text{ext}}(\Pi)\) and \(P_{\text{int}}(\Pi)\), respectively. To recall, the intensional predicates \(P_{\text{int}}(\Pi)\) of \(\Pi\) are those predicates that appear in the head of some rule in \(\Pi\) and where those other predicates in \(P_{\text{ext}}(\Pi)\) are the extensionals, i.e., the extensional predicates do not occur in the head of any rule of \(\Pi\). Note that under this notion, we have that \(P_{\text{ext}}(\Pi) \cap P_{\text{int}}(\Pi) = \emptyset\), i.e., the sets \(P_{\text{ext}}(\Pi)\) and \(P_{\text{int}}(\Pi)\) partitions the predicates symbols of \(\tau(\Pi)\).

For convenience and without loss of generality, we assume in this Chapter (i.e., Chapter 2) that every program mentioned is in a \textit{normalized form} in the sense that for each intensional predicate \(P\), there is a tuple \(x\) of distinct variables that matches the arity of \(P\) and such that for each rule, if its head mentions \(P\), then the head must be \(P(x)\).

We now introduce the notion of the \textit{Clark’s completion} of a program. Our definition of the Clark’s completion is standard except that we do not make the completions for the
extensional predicates.

Given a program $\Pi$, the Clark’s Completion of $\Pi$ (or simply completion when clear from the context), denoted $\text{Comp}(\Pi)$, is the following FO sentence:

$$\bigwedge_{P \in P_{\text{int}}(\Pi)} \forall x(P(x) \leftrightarrow \bigvee_{\substack{r \in \Pi, \\ \text{Head}(r) = P(x)}} \exists y_r \text{Body}(r)), \quad (2.1)$$

where for a rule $r \in \Pi$ of the form (1.4) (i.e., see Section 1.2.1):

- $y_r$ is the tuple of local variables in the rule $r$, i.e., those variables that is only mentioned in $\text{Body}(r)$ and not in $\text{Head}(r)$;

- $\text{Body}(r)$ is the formula

$$\beta_1 \land \cdots \land \beta_l \land \neg \gamma_1 \land \cdots \land \neg \gamma_m.$$

**Example 2** Let $\Pi_{\text{TC}}$ be the following program that computes the transitive closure of a given graph structure of predicate symbol $E$:

$$T(x, y) \leftarrow E(x, y)$$
$$T(x, y) \leftarrow E(x, z), T(z, y),$$

such that $E$ is the only extensional predicate of $\Pi_{\text{TC}}$ which represents the edge relations of the graph, and $T$ is the only intensional predicate of $\Pi_{\text{TC}}$. Ideally, the intensional predicate computes the transitive closure (i.e., all the paths) of the given graph. The Clark’s completion $\text{Comp}(\Pi_{\text{TC}})$ of $\Pi_{\text{TC}}$ is the following FO sentence:

$$\forall xy(T(x, y) \leftrightarrow (E(x, y) \lor \exists z(E(x, z) \land T(z, y)))).$$
Example 3 Now assume $\Pi_{PA}$ as the program that computes the parent-ancestor relationships

\[\Pi_{PA} :\]
\[\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, y)\]
\[\text{Ancestor}(x, y) \leftarrow \text{Ancestor}(x, z), \text{Ancestor}(z, y),\]

such that for the predicates $\text{Parent}$ and $\text{Ancestor}$, we take $\text{Parent}(x, y)$ to mean that $x$ is a parent of $y$, and $\text{Ancestor}(x, y)$ to mean $x$ is an ancestor of $y$. Then we have the Clark’s completion $\text{Comp}(\Pi_{PA})$ of $\Pi_{PA}$ to be the FO sentence

\[\forall xy (\text{Ancestor}(x, y) \leftrightarrow (\text{Parent}(x, y) \lor \exists z (\text{Ancestor}(x, z) \land \text{Ancestor}(z, y))))\].

Informally speaking,

\[\forall xy ((\text{Parent}(x, y) \lor \exists z (\text{Ancestor}(x, z) \land \text{Ancestor}(z, y))) \rightarrow \text{Ancestor}(x, y))\] (2.2)

means that if either: (1) $x$ is a parent of $y$; or (2) $\exists z$ (i.e., “there exist” some other person) such that $x$ is an ancestor of $z$ and where $z$ is an ancestor of $y$, then we have that $x$ is an ancestor of $y$. On the other hand,

\[\forall xy (\text{Ancestor}(x, y) \rightarrow (\text{Parent}(x, y) \lor \exists z (\text{Ancestor}(x, z) \land \text{Ancestor}(z, y))))\].

means that if $x$ is an ancestor $y$, then either: (1) $x$ is a parent of $y$; or (2) $\exists z$ (i.e., “there exist” some other person) such that $x$ is an ancestor of $z$ and where $z$ is an ancestor of $y$. Note that (2.2) corresponds to the universal closure of $\Pi_{PA}$ as mentioned in Section 1.2.1, i.e., it corresponds to the sentence $\widehat{\Pi_{PA}}$.

Now, let us consider the following “parent” structure

\[\mathcal{P} = (\text{Dom}(\mathcal{P}), \text{Parent}^\mathcal{P})\]

(i.e., our extensional or input database) where:

- $\text{Dom}(\mathcal{P}) = \{\text{Ann}, \text{Bob}, \text{Chris}\}$, i.e., our domain is the set comprising of the persons Ann, Bob and Chris;
• \( \text{Parent}^P = \{(\text{Ann}, \text{Bob}), (\text{Bob}, \text{Chris})\} \), i.e., which implies that: (1) Ann is the mother of Bob; and (2) Bob is the father of Chris.\(^2\)

Now, let us consider two \( \tau(\Pi_{PA}) \)-structures \(^3\)

\[
A_1 = (\text{Dom}(A_1), \text{Parent}^{A_1}, \text{Ancestor}^{A_1})
\]

and

\[
A_2 = (\text{Dom}(A_2), \text{Parent}^{A_2}, \text{Ancestor}^{A_2})
\]

such that:

• \( \text{Dom}(A_1) = \text{Dom}(A_2) = \text{Dom}(P) = \{\text{Ann}, \text{Bob}, \text{Chris}\} \);

• \( \text{Parent}^{A_1} = \text{Parent}^{A_2} = \text{Parent}^P = \{(\text{Ann}, \text{Bob}), (\text{Bob}, \text{Chris})\} \);

• \( \text{Ancestor}^{A_1} = \{(\text{Ann}, \text{Bob}), (\text{Bob}, \text{Chris}), (\text{Ann}, \text{Chris})\} \);

• \( \text{Ancestor}^{A_2} = \{(\text{Ann}, \text{Bob}), (\text{Bob}, \text{Chris}), (\text{Ann}, \text{Chris}), (\text{Chris}, \text{Bob}), (\text{Chris}, \text{Ann}), (\text{Bob}, \text{Ann})\} \).

Then it is not too difficult to show that \( A_1 \models \text{Comp}(\Pi_{PA}) \) and \( A_2 \models \text{Comp}(\Pi_{PA}) \). Moreover, it is also not too difficult to see that \( A_1 \) corresponds to the transitive closure of the “parent” structure \( P \) and that \( A_2 \) does not. For instance, the structure \( A_2 \) states that Chris is an ancestor of Ann, which we know is not the case since we get from the “parent” structure \( P \) that Ann must be the grandmother of Chris, i.e., since \( (\text{Ann}, \text{Bob}) \in \text{Parent}^P \) and \( (\text{Bob}, \text{Chris}) \in \text{Parent}^P \). In fact, we have that \( A_1 \) is an answer set of \( \Pi_{PA} \) and that \( A_2 \) is not. Thus, even though \( A_1 \) and \( A_2 \) both satisfy the Clark’s completion \( \text{Comp}(\Pi_{TC}) \) of \( \Pi_{PA} \), only \( A_1 \) is an answer set of \( \Pi_{PA} \) while \( A_2 \) is not.

As it turns out, the reason for this is because \( A_1 \) is a “minimal” model of \( \Pi_{PA} \) and \( A_2 \) is not.

\[^2\text{This is under the assumptions that Ann is a female, and both Bob and Chris are males.}\]

\[^3\text{Structures of the signature of } \Pi_{PA} \text{ (i.e., } \{\text{Parent}, \text{Ancestor}\} \text{) which could be thought of as our intensional “expansion.”}\]
Hence, through Example 3, it can be easily seen why the Clark’s completion is, in general, too weak in capturing the answer set semantics in the sense that there are some models of the Clark’s completion that are not answer sets of the underlying program.

### 2.3 The Ordered Completion

In this section, we show how we can strengthen the Clark’s completion so that its models correspond exactly to the answer sets. To do this, we first define the notion of the *positive predicate dependency* graph of a given FO program.

#### 2.3.1 Positive Predicate Dependency Graphs and Tight Programs

Thus, let $\Pi$ be a FO program. Then we define the graph structure $G^+_\Pi = (\text{Dom}(G^+_\Pi), E^{G^+_\Pi})$ such that:

- $\text{Dom}(G^+_\Pi) = P_{\text{int}}(\Pi)$, i.e., the intensional predicates of $\Pi$;
- $E^{G^+_\Pi} = \{ (P, Q) \mid \text{there is a rule } r \in \Pi \text{ mentioning } P \text{ in } \text{Head}(r) \text{ and } Q \text{ in } \text{Pos}(r) \}$.

This is usually referred in the literature as the *positive dependency* graph of $\Pi$, (and hence the superscript ‘+’ in $G^+_\Pi$).

**Example 4** For example, with $\Pi$ the following FO program:

\[
\begin{align*}
P(y, x) & \leftarrow T(x, z), \text{not } S(x, z) \\
T(x, y) & \leftarrow Q(x, y) \\
Q(x, y) & \leftarrow P(x, z), \text{not } S(x, z) \\
R(x, y) & \leftarrow S(x, y) \\
P(x, y) & \leftarrow R(x, y),
\end{align*}
\]

we have that $G^+_\Pi = (\text{Dom}(G^+_\Pi), E^{G^+_\Pi})$ such that:

- $\text{Dom}(G^+_\Pi) = P_{\text{int}}(\Pi) = \{P, Q, R, T\}$, i.e., and where $S$ is the only extensional predicate of $\Pi$;
- $E^{G^+_\Pi} = \{ (P, R), (Q, P), (T, Q), (P, T) \}$. 


We say that a program $\Pi$ is tight if the transitive closure of the edge relations of $G_\Pi^+$ is cycle or loop free. That is, there do not exist a relation $(x, x) \in \text{TC}(E_{G_\Pi^+})$ where $\text{TC}(E_{G_\Pi^+})$ denotes the transitive closure extension of the edge relations in $E_{G_\Pi^+}$, i.e., the minimal transitive relations extension of $\text{TC}(E_{G_\Pi^+})$. For instance, with $\Pi$ the program from Example 4, we have that $\Pi$ is non-tight since $(P, P)$ is in 

$$\text{TC}(E_{G_\Pi^+}) = \{(P, R), (Q, P), (T, Q), (P, T), (P, Q), (Q, T), (T, P), (P, P), (Q, Q), (T, T)\}.$$ 

It should be noted that this notion automatically holds (i.e., non-tight) when $x = y$ for some edge relation $(x, y) \in E_{G_\Pi^+}$, i.e., this is usually referred to as a self loop or cycle.

**Theorem 1** [Fag94] If a FO program $\Pi$ is tight, then a $\tau(\Pi)$-structure $M$ is an answer set of $\Pi$ iff $M \models \text{Comp}(\Pi)$.

Hence, from [Fag94], we have that the classical models of the Clark’s completion corresponds exactly to the answer sets on tight programs, although for non-tight programs (i.e., as in the program $\Pi$ of Example 4), this is not always the case.

### 2.3.2 Intuition of the Ordered Completion

In a nutshell, the ordered completion is a translation of FO programs into classical FO logic sentences that is achieved by modifying the Clark’s completion and introducing extra predicates. The main technical property of our new translation is that for each finite
FO program, our translation yields a FO sentence that exactly captures the finite answer sets of the program. The ideas behind our translation can be best illustrated by simple propositional programs. Consider the propositional program

\[ p \leftarrow q \]

\[ q \leftarrow p. \]

Then the program has the one answer set \( \emptyset \), although its Clark’s completion \( p \leftrightarrow q \) has two models: \( \{p, q\} \) and \( \emptyset \). To remedy this problem, we introduce four extra symbols \( T_{pq}, T_{pp}, T_{qq} \) and \( T_{qp} \) (read, for e.g., \( T_{pq} \) as from \( p \) to \( q \)), and translate this propositional program into the following propositional formula:

\[
\begin{align*}
(p \rightarrow q) \land (q \rightarrow p) & \quad \text{(2.3)} \\
(p \rightarrow (q \land T_{qp} \land \neg T_{pq})) & \quad \text{(2.4)} \\
(q \rightarrow (p \land T_{pq} \land \neg T_{qp})) & \quad \text{(2.5)} \\
(T_{pq} \land T_{qp} \rightarrow T_{pp}) & \quad \text{(2.6)} \\
(T_{qp} \land T_{pq} \rightarrow T_{qq}) & \quad \text{(2.7)}
\end{align*}
\]

Formula (2.3) is the “classical” logic encoding of the two rules “\( p \leftarrow q \)” and “\( q \leftarrow p \)”;

(2.4) is similar to the other side of the Clark’s completion for \( p \) except that we add \( T_{qp} \) and \( \neg T_{pq} \), which intuitively means that: for \( p \) to be true, then \( q \) must be true and it must be the case that \( q \) is used to derive \( p \) and not the other way around; (2.5) is similar in that it is the other side of the Clark’s completion for \( q \) but with the extra assertions of \( T_{pq} \) and \( T_{qp} \) (i.e., same meaning as above); and finally, (2.6) and (2.7) are simply about the condition that those “\( T \)” atoms satisfies the notion of transitivity.

### 2.3.3 Definition of the Ordered Completion

Now we introduce the notion of ordered completion. So let \( \Pi \) be a FO program with a set of intensional predicates \( \mathcal{P}_{\text{int}}(\Pi) \). Then for each pair of intensional predicates \( (P, Q) \) (i.e., note that \( P \) and \( Q \) can be the same), we introduce a new predicate \( \leq_{PR} \), called the comparison predicate, whose arity is the sum of the arities of \( P \) and \( Q \). The intuitive meaning of \( \leq_{PR}(x, y) \), read as from \( P(x) \) to \( Q(y) \), is that \( P(x) \) was used to derive \( Q(y) \).
Definition 6 (Ordered completion) Let $\Pi$ be a FO program. Then the ordered completion of $\Pi$, denoted by $OC(\Pi)$, is the following FO sentence:

$$
\forall P \in P_{int}(\Pi) \forall x \left( \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r \text{Body}(r) \rightarrow P(x) \right) \quad (2.8)
$$

$$
\forall P \in P_{int}(\Pi) \exists y \left( P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r (\text{Body}(r) \land Pos(r) < P(x)) \right) \quad (2.9)
$$

$$
\land Trans(\Pi), \quad (2.10)
$$

where we have borrowed notations as used in the definition of the Clark’s completion (see Section 2.2), and where:

- $Pos(r) < P(x)$ denotes the formula:

$$
\bigwedge_{Q(y) \in Pos(r), Q \in P_{int}(\Pi)} \leq_{QP}(y,x) \land \neg \leq_{PQ}(x,y), \quad (2.11)
$$

which intuitively means that the atoms in $Pos(r)$ were used to derive $P(x)$ and not the other way around;

- $Trans(\Pi)$ denotes the formula:

$$
\bigwedge_{P,Q,R \in P_{int}(\Pi)} \forall xyz (\leq_{PQ}(x,y) \land \leq_{QR}(y,z) \rightarrow \leq_{PR}(x,z)), \quad (2.12)
$$

which intuitively means that the comparison atoms satisfies a notion of “transitivity.”

Clearly, for a finite program $\Pi$, $OC(\Pi)$ is a finite FO sentence. Moreover, the predicates occurring in $OC(\Pi)$ are all the predicates occurring in $\Pi$ (i.e., $P_{ext}(\Pi) \cup P_{int}(\Pi)$) and together with the set of all the comparison predicates $\{ \leq_{PQ} \mid P,Q \in P_{int}(\Pi) \}$. 
Note that the Clark’s completion of a predicate can be rewritten as two parts:

\[
\bigwedge_{P \in \mathcal{P}} \forall x \left( \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \exists y \overline{\text{Body}(r)} \rightarrow P(x) \right) \tag{2.13}
\]

\[
\bigwedge_{P \in \mathcal{P}} \forall x \left( P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \exists y \overline{\text{Body}(r)} \right). \tag{2.14}
\]

Thus, apart from the “transitivity axioms” \(\text{Trans}(\Pi)\), we have that \(\text{OC}(\Pi)\) and \(\text{Comp}(\Pi)\) really only differ through Formulas (2.9) and (2.14), i.e., since (2.8) and (2.13) are basically the same formulas. It can be seen that (2.9) introduces some assertions on the comparison predicates (i.e., via “\(P_1 \prec P_2\)”), which intuitively means that there exist derivation paths from the intensional atoms of the positive body to the head and not the other way around. In addition (2.10) (i.e., \(\text{Trans}(\Pi)\)) simply means that the comparison predicates satisfy a notion of “transitivity.”

**Proposition 3** Let \(\Pi\) be a FO program. Then \(\text{OC}(\Pi)\) introduces \(m^2\) new predicates whose arities are no more than \(2s\), and the size of \(\text{OC}(\Pi)\) is \(O(s \times m^3 + s \times n)\), where: \(m\) is the number of intensional predicates of \(\Pi\); \(s\) is the maximal arity of the intensional predicates of \(\Pi\); and \(n\) is the length of \(\Pi\).

**Example 5** Recall the parent-ancestor program \(\Pi_{PA}\) presented in Example 3. In this case, since the only intensional predicate is \(\text{Ancestor}\), then we only need to introduce one additional predicate \(\leq_{\text{Ancestor}\text{Ancestor}}\), whose arity is 4, i.e., since the arity of \(\text{Ancestor}\) is 2.
The ordered completion \( OC(\Pi_{PA}) \) of \( \Pi_{PA} \) is then the following FO sentence:

\[
\forall xy((\text{Parent}(x, y) \lor \exists z(\text{Ancestor}(x, z) \land \text{Ancestor}(z, y)))
\rightarrow \text{Ancestor}(x, y))
\land \forall xy(\text{Ancestor}(x, y)
\rightarrow (\text{Parent}(x, y) \lor
\exists z(\text{Ancestor}(x, z) \land \text{Ancestor}(z, y) \land \leq_{\text{Ancestor}}(x, z, x, y)
\land \neg \leq_{\text{Ancestor}}(x, y, x, z)
\land \text{Ancestor}(z, y) \land \leq_{\text{Ancestor}}(z, y, x, y)
\land \neg \leq_{\text{Ancestor}}(x, y, z, y)))
\land \forall uvwyz(\leq_{\text{Ancestor}}(u, v, w, x) \land \leq_{\text{Ancestor}}(w, x, y, z)
\rightarrow \leq_{\text{Ancestor}}(u, v, y, z)).
\] (2.17)

Intuitively, one can understand \( \leq_{\text{Ancestor}}(u, v, w, x) \) to mean that \( \text{Ancestor}(u, v) \) is used to establish \( \text{Ancestor}(w, x) \). So Formula (2.16) means that for \( \text{Ancestor}(x, y) \) to be true, then either \( \text{Parent}(x, y) \) (the base case), or inductively, for some \( z \), both \( \text{Ancestor}(x, z) \) and \( \text{Ancestor}(z, y) \) are true and where they are both used to establish \( \text{Ancestor}(x, y) \) but not the other way around, i.e., as expressed by the comparison predicate assertions

\[
\leq_{\text{Ancestor}}(x, z, x, y) \land \neg \leq_{\text{Ancestor}}(x, y, x, z)
\] and

\[
\leq_{\text{Ancestor}}(z, y, x, y) \land \neg \leq_{\text{Ancestor}}(x, y, z, y),
\]
corresponding to \( \text{Ancestor}(x, z) \) and \( \text{Ancestor}(z, y) \) respectively.

To see how this axiom works, consider again the “parent” structure \( P \) (i.e., the extensional (or input) database) and the “intensional expansion” \( \tau(\Pi_{PA}) \)-structure \( A_2 \) of Example 3. As already discussed in Example 3, we have that \( A_2 \) is not an answer set of \( \Pi_{PA} \). Now we show how this corresponds to the ordered completion \( OC(\Pi_{PA}) \) of \( \Pi_{PA} \). Indeed, let us consider the ancestor relation \( (\text{Chris, Ann}) \in \text{Ancestor}^{A_2} \). Then from (2.16), since
(Chris, Ann) \notin Parent^{A_2} \text{ (i.e., and where } Parent^{A_2} \text{ is our extensional database), there must be some } z \text{ (i.e., another person) such that}

\[ Ancestor(Chris, z) \land Ancestor(z, Ann), \]

i.e., Chris is an ancestor of ‘z’ and ‘z’ is an ancestor of Ann. Then since we have \( Ancestor^{A_2} = \{(Ann, Bob), (Bob, Chris), (Ann, Chris), (Chris, Bob), (Chris, Ann) (Bob, Ann)\} \), we have that \( z \) can only be Bob since \( (Chris, Bob), (Bob, Ann) \in Ancestor^{A_2} \). But then, since this implies that we have

\[ Ancestor(Chris, Bob) \land Ancestor(Bob, Ann), \]

by the additional assertions as enforced by the comparison predicate atoms, we must also have that

\[ \leq Ancestor Ancestor (Chris, Bob, Chris, Ann) \]
\[ \land \neg \leq Ancestor Ancestor (Chris, Ann, Chris, Bob) \]
\[ \land \leq Ancestor Ancestor (Bob, Ann, Chris, Ann) \]
\[ \land \neg \leq Ancestor Ancestor (Chris, Ann, Bob, Ann) \] (2.18)

is true. Now, to see why this is a contradiction, let us consider the fact that (Bob, Ann) is also in \( Ancestor^{A_2} \). Then again by (2.16), since (Bob, Ann) \( \notin Parent^{A_2} \), there must be some \( z \) such that

\[ Ancestor(Bob, z) \land Ancestor(z, Ann). \]

Then it follows by the interpretation \( Ancestor^{A_2} \) that \( z \) can only be Chris since \( (Bob, Chris), (Chris, Ann) \in Ancestor^{A_2} \), so that we have

\[ Ancestor(Bob, Chris) \land Ancestor(Chris, Ann). \]

Then similarly, by the additional assertions as enforced by the comparison predicate atoms,
we must further have the following to also be true:

\[
\begin{align*}
\leq_{\text{Ancestor Ancestor}} (\text{Bob, Chris, Bob, Ann}) \\
\land \neg \leq_{\text{Ancestor Ancestor}} (\text{Bob, Ann, Bob, Chris}) \\
\land \leq_{\text{Ancestor Ancestor}} (\text{Chris, Ann, Bob, Ann}) \\
\land \neg \leq_{\text{Ancestor Ancestor}} (\text{Bob, Ann, Chris, Ann}).
\end{align*}
\]

Then this is a contradiction since we already had that

\[
\leq_{\text{Ancestor Ancestor}} (\text{Bob, Ann, Chris, Ann})
\]

must be true from (2.18). So therefore, we had just showed that \( A_2 \not\models OC(\Pi_{PA}) \), which corresponds to the fact that \( A_2 \) is not an answer set of \( \Pi_{PA} \). This in turn implies that the interpretation \( \text{Ancestor}^{A_2} \) of \( \text{Ancestor} \) under \( A_2 \) is not an ancestral relations of the “parent” extensional structure \( \mathcal{P} \).

\( \square \)

2.3.4 The Main Theorem

In this section, we now introduce some of the main technical results of this thesis. To achieve this, it will be helpful to first introduce the notion of an “expansion” of a structure. Thus, let \( \tau_1 \) and \( \tau_2 \) be arbitrary signatures such that \( \tau_1 \subseteq \tau_2 \). Then we say that a \( \tau_2 \)-structure \( M_2 \) is an expansion of a \( \tau_1 \)-structure \( M_1 \) iff:

- \( \text{Dom}(M_2) = \text{Dom}(M_1) \);
- \( c^{M_2} = c^{M_1} \) for each constant symbol \( c \in \tau_1 \);
- \( P^{M_2} = P^{M_1} \) for each predicate (or relational) symbol \( P \in \tau_1 \).

Loosely speaking, we can think of the expansion \( M_2 \) of \( M_1 \) as simply the addition of the constant and predicate symbols from \( \tau_2 \setminus \tau_1 \), and their associated interpretations, to the structure \( M_1 \). As an example, consider again the “parent” structure \( \mathcal{P} \) and the “intensional expansion” structures \( A_1 \) and \( A_2 \) of Example 3. Then it follows that both \( A_1 \) and \( A_2 \) are expansions of \( \mathcal{P} \) on the signature \( \{ \text{Ancestor} \} \), i.e., where \( \text{Ancestor} \) is the intensional
predicate. Symmetrically, by the restriction of $\mathcal{M}_2$ on the signature $\tau_1$, denoted $\mathcal{M}_2|_{\tau_1}$, we denote the $\tau_1$-structure that is obtained from $\mathcal{M}_2$ by only considering those symbols (both constants and predicates) that are in $\tau_1$. Again as an example, with the “parent” structure $\mathcal{P}$ and its associated “intensional expansions” $\mathcal{A}_1$ and $\mathcal{A}_2$ of Example 3, while $\mathcal{A}_1$ and $\mathcal{A}_2$ are expansions of $\mathcal{P}$, we have that $\mathcal{A}_1|_{\{\text{Parent}\}} = \mathcal{A}_2|_{\{\text{Parent}\}} = \mathcal{P}$, i.e., we only consider the predicate $\text{Parent}$ in the structures $\mathcal{A}_1$ and $\mathcal{A}_2$ to obtain $\mathcal{P}$.

Borrowing ideas from the notions of loops and loop formulas [CLWZ06], we now introduce the notion of an externally supported set of ground atoms.\(^4\) To achieve this, we first introduce the following notion about a set of ground atoms. Let $\Pi$ be a program and $\mathcal{A}$ a structure of some signature $\sigma$ such that $\tau(\Pi) \subseteq \sigma$. Then by $[\Pi_{\text{int}}]\mathcal{A}$, we denote the following set of ground atoms:

$$\{ P(a) \mid a \in \mathcal{A}, P \in [\Pi_{\text{int}}]\mathcal{A} \}.$$  

We are now ready to introduce the notion of an externally supported set.

**Definition 7 (Externally supported set)** Let $\Pi$ be a program and $\mathcal{A}$ a structure of some signature $\sigma$ such that $\tau(\Pi) \subseteq \sigma$. Then we say a set $S \subseteq [\Pi_{\text{int}}]\mathcal{A}$ is externally supported under $\Pi$ and $\mathcal{A}$ (or just externally supported) if there exist some ground atom $P(a) \in S$ such that for some rule $P(x) \leftarrow \text{Body}(r) \in \Pi$ with local variables $y_r$, and assignment of the form $x y_r \rightarrow a b_r$ (under $\mathcal{A}$), we have that:

1. $\mathcal{A} \models \text{Body}(r)[x y_r/ab_r]$;

2. $\text{Pos}(r)[x y_r/ab_r] \cap S = \emptyset$,

where with a slight abuse of notation, $\text{Pos}(r)[x y_r/ab_r]$ denotes the set of ground atoms $\{ Q(b) \mid Q(b) \text{ corresponds to } \beta[x y_r/ab_r], \beta \in \text{Pos}(r) \}$, i.e., the ground atoms corresponding to $\text{Pos}(r)$ under the assignment $x y_r \rightarrow a b_r$.

Informally speaking, an externally supported set $S$, under some program $\Pi$ and corresponding structure $\mathcal{A}$, is a set of ground atoms that can be justified without any dependence on itself, i.e., hence the “external” support. Note that by Definition 7, a set $S \subseteq [\Pi_{\text{int}}]^{A}$ is not externally supported if for all (ground atom) $P(a) \in S$, rule $P(x) \leftarrow \text{Body}(r) \in \Pi$ with local variables $y_r$, and assignment of the form $x y_r \rightarrow a b_r$ such that $\mathcal{A} \models \text{Body}(r)[x y_r/ab_r]$, that $\text{Pos}(r)[x y_r/ab_r] \cap S \neq \emptyset$.

\(^4\)By a ground atom, we refer to the construct of the form $P(a)$ where $P$ is a predicate symbol and $a$ is a tuple of domain elements (of some associated structure) that matches the arity of $P$. 
CHAPTER 2. FIRST-ORDER ASP AND CLASSICAL FIRST-ORDER LOGIC

The following lemma now relates the notion of an externally supported set to the answer sets.

**Lemma 1** Let $\Pi$ be a program. Then a $\tau(\Pi)$-structure $A$ is an answer set of $\Pi$ iff $A \models \hat{\Pi}$ and all sets $S \subseteq [P_{int}(\Pi)]^A$ is externally supported.

**Proof:** See Appendix A.1.1. □

We now introduce another of the key result of this thesis. Generally speaking, the following result links the notion of externally supported sets of ground atoms to the answer sets of a program.

**Theorem 2** Let $\Pi$ be a FO program and $\sigma_{\leq} = \{\leq_{PQ} | P, Q \in P_{int}(\Pi)\}$ (i.e., the set of all the comparison predicate symbols). Then a finite $\tau(\Pi)$-structure $A$ is an answer set of $\Pi$ iff there exist an expansion $A'$ of $A$ on the signature $\sigma_{\leq}$ such that $A' \models OC(\Pi)$.

**Proof:** See Appendix A.1.2. □

From the proof of the main theorem, we see that the basic idea of the ordered completion is that each atom in an answer set must be justified step-by-step. In this sense, a finite structure $A$ is an answer set of a FO program $\Pi$ iff it is a model of $\Pi$ and that also satisfies the following conditions:

**Downgrading:** every ground atom $P(a)$ in $A$ has some supports from earlier stages. The “support” part is ensured by the Clark’s completion, and where the “earlier stages” part is ensured by adding some assertions on the comparison predicates (see Formula (2.9));

**Loop-free:** the above downgrading procedure does not contain a loop. This is ensured by $Trans(\Pi)$, which states that the comparison predicates satisfy transitivity;

**Well-foundedness:** the downgrading procedure will end at some step. This is ensured by the finiteness, i.e., only finite structures are taken into account.

Thus, together with the above three conditions, each ground atom $P(a)$ in a finite answer set $A$ can be justified step-by-step, in which the track of justifying this atom is captured by the comparison predicates.
2.3.5 Normal Programs with Constraints

Note that from the definitions of both the Clark’s and ordered completion, we require the heads of all the rules of the underlying program to be existent. Now recall from Section 1.2.1 that when the head of a rule is empty, then the rule is what is so-called a constraint, i.e., rules of the form

\[ \leftarrow \beta_1, \cdots, \beta_l, \text{not} \gamma_1, \cdots, \text{not} \gamma_m, \tag{2.19} \]

such that \( \beta_i \) (for \( 1 \leq i \leq l \)) and \( \gamma_j \) (for \( 1 \leq j \leq m \)) are atoms. A model is said to satisfy the above constraint if it satisfies the corresponding sentence:

\[ \forall y \neg (\beta_1 \land \cdots \land \beta_l \land \neg \gamma_1 \land \cdots \land \neg \gamma_m), \]

where \( y \) is the distinctive tuple of all the variables occurring in (2.19).

A normal program with constraints is then a finite set of rules and constraints of the form (2.19). The answer set (or stable models) semantics can be extended to normal program with constraints: a model is an answer set of a program with constraints if it is an answer set of the program and that it also satisfies all the constraints.

Both the Clark’s completion and our ordered completion can be extended to programs with constraints by simply adding \(^5\) the corresponding constraints to the respective completions.

**Example 6** The following program \( \Pi_{\text{REACH}} \) checks whether all the nodes of a given graph can be reached from a given initial node:

\[
\begin{align*}
\text{Reach}(x) & \leftarrow x = a \\
\text{Reach}(x) & \leftarrow \text{Reach}(y), \text{Edge}(y, x) \\
& \leftarrow \text{not Reach}(x),
\end{align*}
\]

where:

- \( \text{Edge} \) is the only extensional predicate representing the edges of the graph;

\(^5\)By “adding,” we mean *conjuncting*. 
• \(a\) is a constant representing the initial node;

• \(\text{Reach}\) is the only intensional predicate representing whether a node can be reached from \(a\).

Then \(\Pi_{\text{REACH}}\) has an answer set iff all the nodes in the graph can be reached from \(a\). Then the ordered completion \(OC(\Pi_{\text{REACH}})\) of \(\Pi_{\text{REACH}}\) and the sentence corresponding to the constraint “\(\leftarrow \neg \text{Reach}(x)\)” is the following FO sentence:

\[
\forall xy \left( x = a \lor (\text{Edge}(y, x) \land \text{Reach}(y)) \rightarrow \text{Reach}(x) \right) \\
\land \forall x (\text{Reach}(x) \rightarrow x = a \lor \exists y (\text{Reach}(y) \land \text{Edge}(y, x) \land \leq \text{Reach}(y, x) \land \neg \leq \text{Reach}(x, y))) \\
\land \forall xyz (\leq \text{Reach}(x, y) \land \leq \text{Reach}(y, z) \rightarrow \leq \text{Reach}(x, z)) \\
\land \forall x \text{Reach}(x).
\]

\(\square\)

**Proposition 4** Let \(\Pi\) be a FO program with its set of constraints denoted by \(C\). Then a \(\tau(\Pi)\)-structure \(A\) is an answer set of \(\Pi\) iff there exist an expansion \(A'\) of \(A\) on the signature \(\sigma_\leq = \{\leq_{PQ} | P, Q \in \mathcal{P}_{\text{int}}(\Pi)\}\) such that \(A' \models OC(\Pi) \land \hat{C}\), where \(\hat{C}\) is the sentence

\[
\bigwedge_{r: \leftarrow \beta_1, \ldots, \beta_l, \neg \gamma_1, \ldots, \neg \gamma_m \in \Pi} \forall y_r \neg (\beta_1 \land \cdots \land \beta_l \land \neg \gamma_1 \land \cdots \land \neg \gamma_m)
\]

such that \(y_r\) is the distinctive tuple of all the variables occurring in \(r\).

\(\textbf{Proof:}\) Partition \(\Pi\) into the sets \(\Pi'\) and \(C\) such that \(\Pi' = \Pi \setminus C\), i.e., the non-constraint rules. Then \(A\) is an answer set of \(\Pi\) iff \(A\) is an answer set of \(\Pi' \cup C\) iff \(A\) is an answer set of \(\Pi'\) and is a model of \(\hat{C}\) iff there exist an expansion \(A'\) of \(A\) on the signature \(\sigma_\leq\) such that \(A' \models OC(\Pi')\) (i.e., by Theorem 2) and \(A' \models \hat{C}\) (i.e., since \(\hat{C}\) do not mention any comparison predicates) iff \(A' \models OC(\Pi) \land \hat{C}\) (i.e., \(OC(\Pi) \land \hat{C} = OC(\Pi') \land \hat{C}\) since \(OC(\Pi) = OC(\Pi')\) by the definition of \(OC(\Pi)\)). \(\square\)
2.3.6 Adding Choice Rules

Another widely used extension of normal program is to allow choice rules of the following form:

\[ \{P(x)\}, \]

where \( P \) is a predicate and \( x \) is the tuple of variables associated with \( P \). Intuitively, this choice rule of \( P \) means that the intensional predicate \( P \) can be interpreted arbitrarily in the stable models.

**Example 7** Consider the following program \( \Pi_{HC} \) with constraints and choice rules for computing all Hamiltonian circuit of a graph:

\[
\begin{align*}
\{hc(x, y)\} & \\
\leftarrow hc(x, y), \neg E(x, y) & \\
\leftarrow hc(x, y), hc(x, z), y \neq z & \\
\leftarrow hc(y, x), hc(z, x), y \neq z & \\
R(x) & \leftarrow hc(a, x) & \\
R(x) & \leftarrow R(y), hc(y, x) & \\
& \leftarrow \neg R(x),
\end{align*}
\]

where:

- \( E \) is the only extensional predicate representing the edges of the graph;
- \( a \) is a constant representing a particular node in the Hamiltonian circuit;
- \( hc(x, y) \) is an intensional predicate representing the Hamiltonian circuit;
- \( R(x) \) is an intensional predicate to check that all vertices are in the Hamiltonian circuit.

In particular, the first rule of the program, \( \{hc(x, y)\} \), is a choice rule to guess a possible Hamiltonian circuit. Then using the notion of ordered completion, we can encode \( \Pi_{HC} \) as a
FO sentence as follows:

\[ \forall x (hc(a, x) \rightarrow R(x)) \land \forall y (hc(y, x) \land R(y) \rightarrow R(x)) \]
\[ \land \forall x (R(x) \rightarrow hc(a, x) \lor \exists y (R(y) \land hc(y, x) \land \leq RR (y, x) \land \neg \leq RR (x, y))) \]
\[ \land \forall x y z (\leq RR (x, y) \land \leq RR (y, z) \iff \leq RR (x, z)) \]
\[ \land \forall x y (hc(x, y) \land E(x, y)) \]
\[ \land \forall x y z (hc(x, y) \land hc(x, z) \land y \neq z) \]
\[ \land \forall x y z (hc(y, x) \land hc(z, x) \land y \neq z) \]
\[ \land \forall x R(x), \]

which is simply the ordered completion of \( \Pi_{nc} \) obtained by only making the ordered completion of the intensional predicate \( R \) and not of the intensional predicate \( hc \). This is because we are free to choose arbitrarily whether or not to include an extent of \( hc \) in the model. \( \square \)

The following proposition shows that programs with choice rules can also be captured by their ordered completion.

**Proposition 5** Let \( \Pi \) be a FO program with its set of constraints denoted by \( C \) and \( \text{Choice}(\sigma) \) the set of choice rules of every predicate in \( \sigma \) where \( \sigma \subseteq P_{\text{int}}(\Pi) \). Then a \( \tau(\Pi) \)-structure \( A \) is an answer set of \( \Pi \cup \text{Choice}(\sigma) \) iff there exist an expansion \( A' \) of \( A \) such that \( A' \models OC(\Pi) \land \widehat{C} \) with \( \widehat{C} \) as defined in Proposition (4) and where this time, \( OC(\Pi) \) is the following FO sentence:

\[ \bigwedge_{P \in P_{\text{int}}(\Pi)} \forall x \left( \forall r \in \Pi, \text{Head}(r) = P(x) \right) \exists y_r (\text{Body}(r) \rightarrow P(x)) \) \tag{2.20} 
\[ \land \bigwedge_{P \in P_{\text{int}}(\Pi), P \not\in \sigma} \forall x \left( P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \exists y_r (\text{Body}(r) \land \text{Pos}(r) < P(x)) \right) \) \tag{2.21} 
\[ \land \text{Trans}(\Pi), \] \tag{2.22}
and where $\text{Trans}(\Pi)$ is defined as

$$\bigwedge_{P,Q,R \in \mathcal{P}_m(\Pi), P,Q,R \notin \sigma} \forall x y z \left( \leq_{P} (x,y) \land \leq_{Q} (y,z) \rightarrow \leq_{R} (x,z) \right).$$  \hspace{1cm} (2.23)

Loosely speaking, we can think of incorporating choice rules into the ordered completion by not considering those “choice” intensional predicates of $\sigma$ in Formulas (2.21) and (2.22), i.e., and where (2.22) is fulfilled by (2.23).

**Proof:** This assertion follows directly from Proposition 4 and the following fact: the answer sets of $\Pi \cup \text{Choice}(\sigma)$ are exactly the same as the answer sets of $\Pi^*$ where $\Pi^*$ is the program obtained from $\Pi$ by rewriting each rule of the form:

$$\alpha \leftarrow \beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots, \text{not } \gamma_m,$$

whose head mentions predicates from $\sigma$, into the following constraint:

$$\leftarrow \beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots, \text{not } \gamma_m, \text{not } \alpha.$$

□

2.3.7 Arbitrary Structures and Disjunctive Programs

It is worth mentioning that the correspondence between classical FO models of ordered completions and answer sets of programs only holds on finite structures. In general, the result does not hold if infinite structures are allowed. For instance, the ordered completion $\text{OC}(\Pi_{TC})$ of the program $\Pi_{TC}$ of Example 2, that computes the Transitive Closure (TC), does not capture TC on some infinite structures [ALZZ12].

**Proposition 6 [ALZZ12]** There does not exist a FO formula whose signature contains the signature of TC, and where the reducts of all its models exactly corresponds to the answer sets of TC on arbitrary structures.

Disjunctive answer set programs are a very important extension of normal answer set programs for dealing with incomplete information [EGM97, GL91]. A disjunctive answer
**set program** is a finite set of *disjunctive rules* of the following form

\[ \alpha_1; \ldots; \alpha_n \leftarrow \beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots, \text{not } \gamma_m. \]  

(2.25)

Similarly to normal programs, we can distinguish intensional and extensional predicates here. The answer set semantics for a disjunctive program \( \Pi \) is defined by applying the \( SM \) operator to the universal closure \( \hat{\Pi} \) of \( \Pi \), where \( \hat{\Pi} \) is the following formula:

\[
\bigwedge_{r \in \Pi} \forall x_r (\beta_1 \land \cdots \land \beta_l \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \rightarrow \alpha_1 \lor \cdots \lor \alpha_n),
\]  

(2.26)

i.e., the answer sets of \( \Pi \) are exactly those models of \( SM(\hat{\Pi}) \), where \( SM(\hat{\Pi}) \) is defined as in Section 1.2.2.

A natural question then arises on whether or not the ordered completion can be extended to FO disjunctive programs under the stable models (or answer set) semantics. As it turns out, the answer is negative provided some well-recognized assumptions in the computational complexity theory are true. Actually, the following proposition shows a stronger result that there exist disjunctive programs that cannot be captured (under the stable model semantics) by any FO sentence, even with a larger signature.

**Proposition 7** [ALZZ12] *There exists a disjunctive program \( \Pi \) such that it cannot be captured by any FO sentence with the same or larger signature unless \( NP=coNP \). That is, there is no FO sentence \( \varphi \) whose signature contains that of \( \Pi \), and where the reduct of all its finite models are exactly the finite answer sets of \( \Pi \).*

### 2.3.8 Related Work

In this section, we discuss some related work, mostly other existing or possible translations from answer set programs under the stable model (answer set) semantics to classical logic. In fact, the intuitions behind most of the current translations are similar. The main differences are in the ways how these intuitions are formalized.
Other FO translations

As the focus of this work is to consider the FO case, we first review the existing work about translating FO answer set programs under the stable model (i.e., answer set) semantics to standard FO logic. To the best of our knowledge, the only such translation is the loop formula approach [CLWZ06, LM08]. From a syntactical viewpoint, the main difference between this approach and ours is that the ordered completion results in a finite FO theory (which can be represented as a single FO sentence) but uses auxiliary predicates, while the loop formula approach does not use any auxiliary predicates but in general results in an infinite FO theory.

From a semantical viewpoint, both approaches share some similar ideas. First of all, both of them are extended from Clark’s completion, and where the extended parts play a similar role to the elimination of those structures which are models of the Clark’s completion, but are not stable models of the answer set program. The main difference is that the loop formula approach uses loop formulas for this purpose, while the ordered completion uses additional comparison predicates to keep track of the derivation order. Secondly, they both require that every ground atom in a stable model must be justified by a certain derivation path. However, for this purpose, the loop formula approach further claims that every loop (so is every ground atom) must have some external supports, while the ordered completion approach explicitly enumerates such a derivation order (thus a derivation path) by the new comparison predicates.

Similar FO translations in Datalog

Another related work [Kol90] is in the area of finite model theory and fixed-point logic. Although fixed-point logic and normal answer set programming are not comparable, they have a common fragment, namely Datalog. Kolaitis [Kol90] showed that every fixed-point query is conjunctive definable on finite structures. That is, given any fixed-point query $Q$, there exists another fixed-point query $Q'$ such that the conjunctive query $(Q, Q')$ is implicitly definable on finite structures. As a consequence, every datalog query is also conjunctively definable on finite structures. From this result, although tedious, one can actually derive a translation from Datalog to FO sentences using some new predicates not in the signatures of the original datalog programs.

We will not go into details comparing our translation and the one derived from Kolaitis’
result since our focus here is on normal answer set programs. Suffice to say here that the two are different in many ways, not the least is that ours is based on Clark’s completion in the sense that some additional conditions are added to the necessary parts of intensional predicates, while the one derived from Kolaitis’ result is not. We mention this work because Kolaitis’ result indeed inspired our initial study on this topic. We speculated that if it is possible to translate datalog programs to FO sentences using some new predicates, then it must also be possible for normal answer set programs, and that if this is true, then it must be doable by modifying Clark’s completion. As it happened, this turned out to be the case.

**Translations in the propositional case**

The ordered completion can be viewed as a propositional translation from normal logic programs to propositional theories by treating each propositional atom as a 0-ary predicate. Several proposals in this direction have been proposed in the literature [BED94, Jan04, LM08, LZ04, Nie08]. An early attempt is due to Ben-Eliyahu and Dechter [BED94], who assigned an index (or level numbering) $\#p$ to each propositional atom $p$, and added the assertions $\#p < \#q$ to the Clark’s completion for each pair $(p, q)$ similar to the ordered completion, where $q$ is the head of a rule and $p$ ranges over all atoms in the positive body of a rule. A closely related work is recently proposed by Niemelä [Nie08], in which the level mappings and their comparisons are captured in difference logic, an extension of classical propositional logic. More precisely, each atom $p$ is assigned to a number $x_p$, meaning its level or stage. Then, the assertions $x_q - 1 < x_p$ are added to the Clark’s completion similar to Ben-Eliyahu and Dechter and the ordered completion. In addition, in both approaches, the optimization technique of exploiting strongly connected components is discussed.

Another translation, also sharing the basic idea of comparing stages (or indices), is due to Janhunen [Jan04], who proposed a simplified translation by level numbering as well. Different from the above approaches, Lin and Zhao [LZ03] translated an arbitrary normal answer set program equivalently to a tight program by first adding some new atoms, and then to use the Clark’s completion of the new program to capture the answer sets of the original one. Finally, the loop formula approach in the propositional case [LZ04] yields another translation from propositional normal answer programming to propositional logic. Again, the loop formula approach requires no new atoms. However, it is not polynomial in the sense that a program may have exponential loop formulas in worst case.
Comparisons with Ben-Eliyahu and Dechter’s and Niemelä’s work

Here, we discuss more about the relationships among the ordered completion, Ben-Eliyahu and Dechter’s translation, and Niemelä’s work since these three translations are very closely related, while the others are slightly different. In fact, the above three translations basically share the same intuitions in the propositional case. This is because all of them are modified from Clark’s completion by adding to it the comparisons of indices/levels/stages. Specifically, the comparisons are represented by $\leq_{pq} \land \neg \leq_{qp}$ in the ordered completion, $\#p < \#q$ in Ben-Eliyahu and Dechter’s translation and $x_q - 1 \geq x_p$ in Niemelä’s work, where in all the above approaches, $q$ is the head of a rule and $p$ ranges over all atoms in the positive body of a rule. Indeed, these assertions play the same role to state that the stage (or level) of $p$ should be less than the one of $q$. In this sense, the modified completion part of all these three approaches can be transformed from each other.

In Ben-Eliyahu and Dechter’s translation, one has to explicitly enumerate the “indices” $\#p$ and “comparisons” $\#p < \#q$ in propositional logic [BED94], which turns out to be rather complicated. This is not an issue for Niemelä’s work [Nie08] because the level numbering $x_p$ associated with the atoms and the comparisons $x_q - 1 \geq x_p$ can be directly represented by the built-in predicates within the language of difference logic. Finally, in the ordered completion, we do not introduce the indices directly but use additional atoms $\leq_{pq}$ in classical propositional logic to explicitly represent the comparisons $\leq_{pq} \land \neg \leq_{pq}$, which are further specified by the transitivity formulas. The similarities and differences among the three approaches can be better illustrated by the following example.

**Example 8** Let $\Pi$ be the following propositional program:

$$
\begin{align*}
p_1 & \leftarrow p_2 \\
p_2 & \leftarrow p_1 \\
p_1 & \leftarrow \text{not } p_3.
\end{align*}
$$

(2.27)

According to the definitions, the modified completion part of $\Pi$ for the ordered completion
is

\[(p_2 \lor \neg p_3 \rightarrow p_1) \land (p_1 \rightarrow p_2)\]
\[\land (p_1 \rightarrow p_2 \land [\leq p_{2p_1} \land \neg \leq p_{1p_2}] \lor \neg p_3)\]
\[\land (p_2 \rightarrow p_1 \land [\leq p_{1p_2} \land \neg \leq p_{2p_1}] ),\]

while for Ben-Eliyahu and Dechter’s translation, this is

\[(p_2 \lor \neg p_3 \rightarrow p_1) \land (p_1 \rightarrow p_2)\]
\[\land (p_1 \rightarrow p_2 \land [\# p_2 \land \# p_1] \lor \neg p_3)\]
\[\land (p_2 \rightarrow p_1 \land [\# p_1 \land \# p_2]),\]

and finally, for Niemelä’s work, this is

\[(p_2 \lor \neg p_3 \rightarrow p_1) \land (p_1 \rightarrow p_2)\]
\[\land (p_1 \rightarrow p_2 \land [x_{p_1} - 1 \geq x_{p_2}] \lor \neg p_3)\]
\[\land (p_2 \rightarrow p_1 \land [x_{p_2} - 1 \geq x_{p_1}]).\]

It can be observed that, the modified completion part of the three approaches can be easily obtained from each other. For instance, from Niemelä’s work to the ordered completion, one only needs to replace each subformula of the form $x_{p_1} - 1 \geq x_{p_2}$ (e.g., $x_{p_1} - 1 \geq x_{p_2}$) with its corresponding counterpart $\leq_{aq} \land \neg \leq_{pq}$ in the ordered completion ($\leq_{p_2p_1} \land \neg \leq_{p_1p_2}$ resp.).

The main difference among the three approaches is another part of the translation, namely how to encode those new indices and comparisons. The host formalism for both Ben-Eliyahu and Dechter’s translation and our ordered completion is propositional logic, but for Niemelä’s work, it is difference logic, which is an extension of classical propositional logic with linear constraints but not propositional logic itself. As a result, the encoding problem of comparisons for Niemelä’s work is not an issue because the comparisons, e.g., $x_{p_1} - 1 \geq x_{p_2}$, can be naturally represented in the language of difference logic with

\footnote{It can be observed that the new atoms $bd_{i_0}$ in Niemelä’s work are not necessary.}
the built-in predicate $\geq$.

In the ordered completion, we use additionally transitivity formulas among new atoms $\leq_{pq}$ for this purpose. For instance, for the program $\Pi$ of Example 8, the transitivity formula is:

$$\leq_{p_1p_2} \land \leq_{p_2p_1} \rightarrow \leq_{p_1p_1} \land \leq_{p_2p_1} \land \leq_{p_1p_2} \rightarrow \leq_{p_2p_2}.$$  

In Ben-Eliyahu and Dechter’s translation, one needs to explicitly encode the indices $\#p$ and the comparisons $\#p < \#q$ in classical propositional logic. This is rather complicated because one has to enumerate all the possibilities. For instance, for the program $\Pi$ of Example 8, the encoding of each index, e.g., $\#p_1$, is:

$$(p_1 = 1 \vee p_1 = 2) \land (p_1 = 1 \rightarrow \neg(p_1 = 2)),$$

and the encoding of each comparison, for instance $\#p_1 < \#p_2$, is:

$$p_1 = 1 \land p_2 = 2.$$

**FO definability and weak definability**

Since the ordered completion is about translating answer set programs to FO logic, it is closely related to the concepts of (first-order) definability for answer set programming [LM08, ZZ10].

A program is (first-order) definable (on finite structures) if there exists a first-order sentence of the signature of the program such that its (finite) models are exactly the (finite) stable models of the program. It is well-known that many programs are not first-order definable, e.g., the program TC in Example 2, both on arbitrary structures and on finite structures [EF99].

A weaker notion of FO definability is to allow new predicates. A program is (first-order) weakly definable (on finite structures) if there exists a FO sentence of a signature containing the signature of the program such that the reducts of its (finite) models on the signature of the program are exactly the (finite) stable models of the program. It is easy to

---

7 All the other transitive formulas are trivially true.
see that a program is weakly definable (on finite structures) iff it is defined by an existential second-order sentence (on finite structures).

The following result immediately follows from Theorem 2.

**Corollary 1** Every normal answer set program is weakly definable on finite structures. More precisely, every program is weakly defined by its ordered completion on finite structures.

However, as shown in Proposition 6, this result does not hold on arbitrary structures. For instance, the TC program is not weakly definable on arbitrary structures. In fact, following a similar proof, Proposition 6 can be extended to the following result.

**Proposition 8** On arbitrary structures, if a normal answer set program is not definable, then it is not weakly definable.

### 2.4 Ordered Completion with Aggregates

Aggregates are a very important building block of answer set programming that is widely used in many (practical) applications. The reason why aggregates are crucial in answer set solving is twofold. First, it can simplify the representation task. For many applications, one can write a simpler and more elegant logic program by using aggregates, for instance, the job scheduling program [PSE04]. More importantly, it can improve the efficiency of program solving [GKKS09]. Normally, the program using aggregates can be solved much faster [FPL+08].

Before we formally define the notion of a program with aggregates, it will first be necessary to enhance our definition of a structure. We do this by having a more general view of a structure to include some fixed interpretations on relations associated with the integer set \( Z \). For this purpose, we resort to many-sorted structures and typed first-order logic. In particular, we now assume our language to have the super-sort \( S = Z \cup D \) such that: \( Z \cap D = \emptyset \) (i.e., \( D \) contains no elements from the \( Z \)); \( Z \) is the set \( \{ \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots \} \) of integers; and \( D \) is the set of all possible non-integer domain elements.
CHAPTER 2. FIRST-ORDER ASP AND CLASSICAL FIRST-ORDER LOGIC

We now view a structure $A$ as a many-sorted structure of the form

\[ \{S, \text{Dom}(A)\}, \quad \ldots, -3^A, -2^A, -1^A, 0^A, 1^A, 2^A, 3^A, \ldots, +^A, \times^A, <^A, \leq^A, =^A, \geq^A, >^A, \]

\[ c_1^A, \ldots, c_m^A, R_1^A, \ldots, R_n^A, \quad (2.28) \]

with sorts $\{S, \text{Dom}(A)\}$ such that

\[ (\text{Dom}(A), c_1^A, \ldots, c_m^A, R_1^A, \ldots, R_n^A) \quad (2.29) \]

is a standard structure in first-order logic where $\text{Dom}(A)$ is a finite sub-sort of the super-sort $S = \mathbb{Z} \cup D$.\(^8\)

Note that in (2.28),
\[ \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots \]

are constant symbols such that
\[ \ldots, -3^A, -2^A, -1^A, 0^A, 1^A, 2^A, 3^A, \ldots \]
corresponds to their natural interpretations on $\mathbb{Z}$, i.e., $0^A = 0, 1^A = 1, 2^A = 2$, etc. In addition to these constants, we also have the function symbols $+$ and $\times$, and relational symbols $<, \leq, =, \geq$, and $>$ such that $+^A, \times^A, <^A, \leq^A, =^A, \geq^A, >^A$ stands for the "standard" interpretations on $\mathbb{Z}$. Note that since we view (2.29) as a sub-structure of $A$, then all the constants $c_1, \ldots, c_m$ and relations $R_1, \ldots, R_n$ only involve sorts of $\text{Dom}(A)$. Therefore, we view the other sub-structure

\[ (S, \ldots, -3^A, -2^A, -1^A, 0^A, 1^A, 2^A, 3^A, \ldots, +, \times, <^A, \leq^A, =^A, \geq^A, >^A) \quad (2.30) \]

of the super-sort $S$ as the fixed "standard" interpretation associated with number theory. For this reason and simplicity, when we refer to a structure, we only assume (2.29) and regard (2.30) as the fixed background structure on integers. In addition, as already mentioned

---

8$\text{Dom}(A)$ is a finite set that can contain both integer (i.e., those from $\mathbb{Z}$) and non-integer (i.e., those from $D$) elements.

9For instance, if one of $x$ or $y$ is a non-integer element, then we assume $x \leq y$ to evaluate to false, and $x + y$ (or $x \times y$) to evaluate to some non-integer element.
above, we only restrict ourselves to finite structures by assuming that \( \text{Dom}(A) \) is finite. Also for convenience, we denote by \( \sigma_{\mathbb{Z}} \) as the signature corresponding to the set of symbols 
\[
\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, +, \times, <, \leq, =, \geq, >\}
\]
associated with the standard interpretation on \( \mathbb{Z} \).

We now introduce the notion of aggregate atoms. For this purpose, it will again be necessary to further enhance our notion of a structure by also incorporating a new sort \( M_S \), which represents the set of all the finite multiset\(^{10} \) subsets of \( S \), i.e., where \( S \) is the super-sort of \( \mathbb{Z} \) and \( \text{Dom}(A) \). We also introduce the following additional functions into the signature \( \sigma_{\mathbb{Z}} \): \( \text{CARD} \), \( \text{SUM} \), \( \text{PROD} \), \( \text{MIN} \), and \( \text{MAX} \),\(^{11} \) which operates from \( M_S \) to \( S \). In particular, \( \text{CARD} \) is a function from \( M_S \) to \( \mathbb{Z} \), and \( \text{SUM} \), \( \text{PROD} \), \( \text{MIN} \), and \( \text{MAX} \) are \textit{partial} functions from \( M_S \) to \( S \) such that:

1. If \( M \in M_S \) is a multiset of \( \mathbb{Z} \) (i.e., only mentions element of integers), then with \( \text{OP} \in \{\text{SUM}, \text{PROD}, \text{MIN}, \text{MAX}\} \), we have \( \text{OP}(M) \in \mathbb{Z} \);

2. Otherwise, if \( M \in M_S \) is not a multiset of \( \mathbb{Z} \) (i.e., it mentions some non-integer elements), then with \( \text{OP} \in \{\text{SUM}, \text{PROD}, \text{MIN}, \text{MAX}\} \), we have that \( \text{OP}(M) \) is undefined.

In this work, we consider aggregate atoms \( \delta \) of the form
\[
\text{OP}\langle v : \exists w \text{Body}(\delta) \rangle \preceq N, \tag{2.31}
\]
where:

- \( \text{OP} \in \{\text{CARD}, \text{SUM}, \text{PROD}, \text{MIN}, \text{MAX}\} \);

- \( \text{Body}(\delta) \) (the body of \( \delta \)) is of the form
\[
Q_1(y_1), \ldots, Q_s(y_s), \text{not } R_1(z_1), \ldots, \text{not } R_t(z_t), \tag{2.32}
\]

where each \( Q_i(y_i) \) (1 \( \leq i \leq s \)) and \( R_j(z_j) \) (1 \( \leq j \leq t \)) are atoms that can be

\(^{10} \text{A multiset generalizes the notion of a set by allowing repeated elements. Hence conceptually, the usual notion of a set is the special case of a multiset where every element occurs at most once.} \)

\(^{11} \text{CARD, SUM, PROD, MIN, and MAX stands for cardinality, sum, product, minimum, and maximum, respectively.} \)
of predicates from $\sigma_Z$, but where all the variables are mentioned in some atom of a predicate from $\{R_1, \ldots, R_n\}$;\(^{12}\)

- $v$ and $w$ are distinctive tuples of variables mentioned in (2.32),
- $\preceq \in \{<, \leq, =, \geq, >\}$ is a comparison operator on $\mathbb{Z}$ (i.e., $\preceq \in \sigma_Z$);
- and finally, $N$ is a constant standing for a number in $\mathbb{Z}$ (i.e., $N \in \sigma_Z$ as well).

For convenience, we use $\text{Pos}(\delta)$ and $\text{Neg}(\delta)$ to denote the sets $\{Q_1(y_1), \ldots, Q_s(y_s)\}$ and $\{R_1(z_1), \ldots, R_t(z_t)\}$ respectively. By the free variables of $\delta$, we denote those variables mentioned in (2.32) but not in $v \cup w$. For instance, with $\delta$ the aggregate atom

$$\text{SUM}(x : \exists y P(x,y), \text{not } Q(y,z)) < 10, \quad (2.33)$$

then we have $z$ as its only free variable, i.e., we can also denote $\delta$ by $\delta(z)$. Informally speaking, (2.33) states that the sum of all $x$ such that there exist a $y$ where “$P(x,y), \text{not } Q(y,z)$” holds for some given $z$, is less than 10.

A FO program $\Pi$ with aggregates is then still a collection of rules $r$ of the form:

$$\alpha \leftarrow \beta_1, \cdots, \beta_l, \text{not } \gamma_1, \cdots, \text{not } \gamma_m, \quad (2.34)$$

(1.4) but where some of the $\beta_i$ ($1 \leq i \leq l$) or $\gamma_j$ ($1 \leq j \leq m$) can be aggregate atoms of the form (2.31). In addition, we can also assume that these atoms can involve symbols from $\sigma_Z$ but where each variable mentioned must occur in some atom from $\{R_1, \ldots, R_n\}$,\(^{13}\) and all function terms from $\sigma_Z$ (i.e., ‘$+$’ and ‘$\times$’) only occurs in the comparison operators $\prec$, $\leq$, $=\geq$, and $\succ$ of $\sigma_Z$.\(^{14}\) By $\tau(\Pi)$, we denote the signature of the symbols mentioned in $\Pi$ that are not in $\sigma_Z$ (this includes the aggregate functions).

\(^{12}\)This insures that all the variables only ranges over $\text{Dom}(A)$ since all atoms of $\{R_1, \ldots, R_n\}$ are of sort $\text{Dom}(A)$.

\(^{13}\)This insures that all variables are of sort $\text{Dom}(A)$ hence, only ranging over $\text{Dom}(A)$.

\(^{14}\)This insures the values of $x + y$ and $x \times y$ are only considered for the comparison operators of $\sigma_Z$. 

Example 9 Consider the following FO program \( \Pi \) with a simple aggregate atom

\[
\begin{align*}
Q(y) & \leftarrow \text{CARD}(x : P(x)) = 2 \\
P(x) & \leftarrow Q(x) \\
P(x) & \leftarrow R(x).
\end{align*}
\]

Here, \( \tau(\Pi) = \{P, Q\} \) and where \( P \) and \( Q \) are intensional while \( R \) is extensional. Rule (2.35) states that if \( P(x) \) holds exactly for two elements, then we have \( Q(y) \).

Now assume \( \delta \) to be an aggregate atom with free variables \( x \). Then we say that a structure \( \mathcal{A} \) (of appropriate signature) satisfies \( \delta(x)[x/a] \) for some \( a \in \text{Dom}(\mathcal{A})^{\lvert x \rvert} \) iff:

1. The multiset\(^{15}\)

\[
M = \{ (c[1] | \mathcal{A} \models \text{Body}(\delta)[xwv/abc], b \in \text{Dom}(\mathcal{A})^{\lvert w \rvert}, c \in \text{Dom}(\mathcal{A})^{\lvert v \rvert} ) \}
\]

is in the domain of the aggregate function \( \text{OP} \) (where \( \text{Body}(\delta) \) denotes the conjunctions of all the literals in \( \text{Body}(\delta) \) with ‘not’ turned into ‘\( \neg \)’);

2. \( \text{OP}(M) \preceq N \) holds.

Then the satisfaction of formulas with aggregates can be defined recursively as usual by viewing the aggregate atoms as a base case.

An aggregate function \( \text{OP} \) with respect to a comparison operator \( \preceq \) and number \( N \) is said to be: **monotone** if \( M_1, M_2 \in \text{Dom}(\text{OP}) \) (i.e., in the domain of \( \text{OP} \)), \( M_1 \subseteq M_2 \), and \( \text{OP}(M_1) \preceq N \) imply that \( \text{OP}(M_2) \preceq N \); and **anti-monotone** if \( M_1, M_2 \in \text{Dom}(\text{OP}) \), \( M_2 \subseteq M_1 \), and \( \text{OP}(M_1) \preceq N \) imply that \( \text{OP}(M_2) \preceq N \).

### 2.4.1 The Stable Model Semantics for Aggregates

By \( \hat{\Pi} \), we now define the *universal closure* of \( \Pi \) as the FO sentence

\[
\bigwedge_{r \in \Pi} \forall x_r(\beta_1 \land \cdots \land \beta_l \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \rightarrow \alpha),
\]

\(^{15}\)In the following, \( e[1] \) denotes the first component (or position) in the tuple \( e \).
where \( x_r \) denotes the tuple of distinctive free variables occurring in \( r \), and such that if \( \beta_i \) (1 \( \leq i \leq l \)) or \( \gamma_j \) (1 \( \leq j \leq m \)) is an aggregate atom \( \delta \) of the form (2.31), then \( \beta_i \) (or \( \gamma_j \)) is the aggregate atom of the form
\[
\text{OP}\left( v : \exists y \text{Body}(\delta) \right) \preceq N,
\] (2.39)
such that \( \text{Body}(\delta) \) denotes the conjunctions
\[
P_1(y_1) \land \cdots \land P_s(y_s) \land \neg Q_1(z_1) \land \cdots \land \neg Q_t(z_t)
\]
of literals in \( \text{Body}(\delta) \).

**Example 10** Let us consider again the FO program \( \Pi \) of Example 9. Then the universal closure \( \hat{\Pi} \) of \( \Pi \) is the following FO sentence:
\[
\forall y (\text{CARD}\langle x : P(x) \rangle = 2 \implies Q(y)) \quad (2.40)
\]
\[
\land \forall x (Q(x) \implies P(x)) \quad (2.41)
\]
\[
\land \forall x (R(x) \implies P(x)). \quad (2.42)
\]

\( \square \)

In the spirit of Section 1.2.1, also assuming \( P = P_1 \cdots P_n \) to be the tuple of distinctive intensional predicates and \( U = U_1 \cdots U_n \) as the tuple of fresh distinctive predicates of the same length as \( P \) (i.e., and such that the arity of \( U_i \) matches the arity of \( P_i \) for \( 1 \leq i \leq n \)), we now define \( \hat{\Pi}^*(U) \) as the following formula:
\[
\bigwedge_{r \in \Pi, r := \alpha \leftarrow \beta_1, \cdots , \beta_l, \text{not} \gamma_1, \cdots , \text{not} \gamma_m} \forall x_r (\beta_1^* \land \cdots \land \beta_l^* \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \implies \alpha^*), \quad (2.43)
\]
such that for an atom \( \rho \in \{ \alpha, \beta_1, \cdots , \beta_l \} \), if:

1. \( \rho = Q(x) \) where \( Q \in P_{\text{ext}}(\Pi) \), then \( \rho^* = \rho \); 
2. \( \rho = P_i(x) \) where \( P_i \in P_{\text{int}}(\Pi) \) (i.e., \( 1 \leq i \leq n \)), then \( \rho^* = U_i(x) \);
3. \( \rho \) is an aggregate atom \( \delta \) of the form (2.31), then \( \rho^* \) is the formula

\[
(\text{OP}(v : \exists w\text{Body}(\delta)^*) \leq N) \land (\text{OP}(v : \exists w\text{Body}(\delta)) \leq N),
\]

where \( \text{Body}(\delta)^* \) is the formula

\[
P_1(y_1)^* \land \cdots \land P_s(y_s)^* \land \neg Q_1(z_1) \land \cdots \land \neg Q_t(z_t),
\]

such that \( P_j(y_j)^* \) (\( 1 \leq j \leq s \)) is defined as in the aforementioned Items 1 and 2 above.

Then similarly to Section (1.2.1), by \( SM(\Pi) \) we denote the universal SO sentence

\[
\hat{\Pi} \land \forall U(U < P \rightarrow \neg \hat{\Pi}^*(U)).
\]

Then we say a \( \tau(\Pi) \)-structure \( \mathcal{A} \) is an answer set (or stable model) of the FO program \( \Pi \) with aggregates iff \( \mathcal{A} \models SM(\Pi) \). It should be noted here that our notion of \( SM(\Pi) \) is essentially equivalent to the one in [BLM11] when restricted to the syntax of normal programs with aggregate atoms of the form (2.31).

### 2.4.2 Extension of Ordered Completion for Aggregates

From a theoretical point of view, ordered completion [ALZZ10] makes important progress on understanding first-order answer set programming. Firstly, it shows that the stable model semantics is simply Clark’s completion plus derivation order. Secondly, it clarifies the relationship between first-order normal program and classical first-order logic. More precisely, every normal answer set program can be captured by a first-order sentence with some new predicates on finite structures. Surprisingly, this fails to be true on infinite structures or if no new predicates are allowed [ALZZ12] (see Section 2.3.7).

Ordered completion is not only theoretically interesting but also practically important. It initiates a new direction of program solvers by first translating a normal logic program to its ordered completion and then to work on finding a model of this first-order sentence [ALZZ12]. A first implementation shows that this new direction is promising as it performs surprisingly good on the Hamiltonian Circuit program [Nie99], especially on big instances.
However, ordered completion can hardly be used beyond the Hamiltonian Circuit program because it cannot handle aggregates - a very important building block in program that is widely used in many applications. As far as we are concerned, most benchmark programs contain aggregate constructs [CIR+11]. Aggregates are crucial in answer set solving because on one hand, it can simplify the representation task, and on the other hand, it can improve the efficiency of program solving [GKKS09].

In this section, we consider to incorporate aggregates in ordered completion. However, this is a challenging task. As shown in [ALZZ12], ordered completion cannot be extended for disjunctive programs because they are not in the same complexity level. Similarly, normal answer set programs with arbitrary aggregates has the same complexity level as disjunctive programs, which is beyond the complexity of naive normal programs. This suggests that, most likely, normal logic programs with arbitrary aggregates cannot be captured in first-order logic, thus not by ordered completion.

Hence, we are mainly focused on some special classes of aggregates, for instance, (anti)monotone aggregates. Even for this case, the task is still very complicated. One observation is that aggregate atoms can express some features of existential quantifiers, which is even more complicated than disjunctions in first-order program. For instance, recall the program specified in Example 9. The first rule, i.e., rule (2.35), is actually equivalent to the following rule $\exists^=2P(x) \rightarrow Q(y)$, where $\exists^=2P(x)$ is a shorthand of

$$
\exists x z(x \neq z \land P(x) \land P(z) \land \forall u(P(u) \rightarrow u = x \lor u = z)),
$$

meaning that $P(x)$ holds exactly for 2 elements.

Another observation is that monotone aggregate atoms contribute to the first-order loops [CLWZ06, LM09] of the program. Again, consider the program in Example 9. If we ignore the aggregate atoms, then this program has no loops. But the stable models of the program cannot be captured by its Clark’s completion. This means that the aggregate atom in rule (2.35) indeed plays a role to form a loop of the program.

Hence, the difficult part is to identify the gap between the aggregates that can be incorporated into ordered completion and those cannot. In this section, we show that a large variety of (anti)monotone aggregates can indeed be incorporated in ordered completion, which covers the types of aggregates in most benchmark programs, as far as we have checked.

We extend the notion of ordered completion for normal logic programs with such kind of
aggregates so that every stable model of a program is corresponding to a classical model of its enhanced ordered completion, and vice versa.

Formally, the aggregate functions we consider in this work are restricted as follows.

**Definition 8** For an aggregate atom of the form (2.31) where OP ∈ \{CARD, SUM, PROD, MIN, MAX\},

- **CARD** is a function from multisets of domain tuples to \(\mathbb{Z}^+ = \{0, 1, 2, 3, \ldots\}\) (non-negative integers), and it is defined as 0 on the empty multiset \(\emptyset\);
- **SUM** is a function from multisets of \(\mathbb{Z}^+\) to \(\mathbb{Z}^+\), and it is also defined as 0 on \(\emptyset\);
- **MIN** and **MAX** are functions from multisets of \(\mathbb{Z}\) to \(\mathbb{Z}\), and are undefined for \(\emptyset\);
- **PROD** is a function from multisets of \(\mathbb{N}\) to \(\mathbb{N}\) (i.e., \(\mathbb{N} = \{1, 2, 3, \ldots\}\), the natural number), and is defined as 1 on \(\emptyset\).

Now, we define the ordered completion for normal FO programs with aggregate atoms of the form (2.31) under the restrictions in Definition 8. For convenience, we use \(PosAgg(r)\) to denote the aggregate atoms from \(Pos(r)\); \(PosCardSumProd(r)\) to denote the cardinality, sum, and product aggregates from \(Pos(r)\); and \(PosMinMax(r)\) to denote the minimum and maximum aggregate atoms from \(Pos(r)\) respectively.

**Definition 9 (Ordered completion with aggregates)** Let \(\Pi\) be a FO program with aggregate atoms of form (2.31) under the restrictions in Definition 8. The modified completion of \(\Pi\), denoted by \(MComp(\Pi)\), is the following formula:

\[
\bigwedge_{P \in P_{int}(\Pi)} \forall x\big( \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \exists y_r \text{Body}(r) \rightarrow P(x) \big) \land \bigwedge_{P \in P_{int}(\Pi)} \forall x\big( P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \exists y_r \big[ \text{Body}(r) \land Pos(r) < P(x) \land PosAgg(r) < P(x) \big] \big)
\]

(2.44)

(2.45)

where:
\( Pos(r) < P(x) \) is
\[
\bigwedge_{Q(y) \in Pos(r) \setminus PosAgg(r), Q \in P_{\text{cut}}(\Pi)} (\leq QP(y, x) \land \neg \leq PQ(x, y));
\]

\( and \ PosAgg(r) < P(x) \) is
\[
\bigwedge_{\delta \in PosCardSumProd(r), \leq \in \{=, \geq\}} (\text{OP}(v : \exists w Body(\delta) \land Pos(\delta) < P(x)) \leq N) \land
\bigwedge_{\delta \in PosMinMax(r), \leq \in \{<, \leq\}} (\text{OP}(v : \exists w Body(\delta) \land Pos(\delta) < P(x)) \leq N),
\]

where for an aggregate atom \( \delta \) of the form (2.31) with body of the form (2.32), \( Pos(\delta) < P(x) \) stands for
\[
\bigwedge_{1 \leq i \leq s, Q_i \in P_{\text{int}}(\Pi)} (\leq_{Q_i P} (y_1, x) \land \neg \leq_{PQ_i} (x, y_1)). \tag{2.46}
\]

Finally, the ordered completion of \( \Pi \), denoted by \( OC(\Pi) \), is again \( MComp(\Pi) \land Trans(\Pi) \).

Let us take a closer look at Definition 9. First of all, for non-aggregate atoms, we treat them the same way as in the original definition of ordered completion. For aggregate atoms, we distinguish two cases. For those atoms \( \text{OP}(v : \exists w Body(\delta)) \leq N \) where \( \text{OP} \in \{\text{CARD, SUM, PROD}\} \) and \( \leq \in \{<, \leq\} \), and those atoms negatively occurring in the rule, we do not need to pay extra attention. However, for the rest of the positive aggregate atoms \( \delta \), we need to enforce the comparison assertions, i.e., formula (2.46). This is because, for the latter kind of aggregate atoms, we need to keep track of the derivation order. However, for the former kind, this is not necessary because they do not enforce a nonempty multiset.
Example 11 Consider again the program II in Example 9. Then $OC(\Pi)$ is

$$\forall y (\text{CARD}(x : P(x)) = 2 \rightarrow Q(y))$$  \hspace{1cm} (2.47)  
$$\forall x (Q(x) \rightarrow P(x)) \land \forall x (R(x) \rightarrow P(x))$$ \hspace{1cm} (2.48)  
$$\forall x (P(x) \rightarrow R(x) \lor (Q(x) \land \neg (Q(x) \land Q(x) < P(x))))$$ \hspace{1cm} (2.49)  
$$\forall y (Q(y) \rightarrow [\text{CARD}(x : P(x)) = 2] \land [\text{CARD}(x : P(x) \land P(x) < Q(y)) = 2])$$ \hspace{1cm} (2.50)  
$$\land \text{Trans}(\Pi),$$ \hspace{1cm} (2.51)

where $Q(x) < P(x)$ and $P(x) < Q(y)$ are the shorthands for $\leq_{QP} (x, x) \land \neg \leq_{PQ} (x, x)$ and $\leq_{PQ} (x, y) \land \neg \leq_{QP} (y, x)$ respectively. Notice that $OC(\Pi)$ can be represented in first-order logic since formula (2.47) is equivalent to $\forall y (\exists^2 x P(x) \rightarrow Q(y))$ and Formula (2.50) is equivalent to $\forall y (Q(y) \rightarrow \exists^2 x (P(x) \land P(x) < Q(y)))$.

Consider a structure $\mathcal{A}$ where $R^A = \emptyset$. Then, $\mathcal{A}$ is a stable model of $\Pi$ iff $P^A = Q^A = \emptyset$. This in fact corresponds to a model of $OC(\Pi)$. Otherwise, suppose $P^A \neq \emptyset$, say $P^A = \{a, b\}$. Then since we have $P(a)$, by (2.49), $Q(a) \land Q(a) < P(a)$ holds. Then by (2.50), $\exists^2 x (P(x) < Q(a))$ must also hold. Then since $P^A = \{a, b\}$ exactly contains 2 elements, then $P(a) < Q(a)$ must hold as well. This contradicts to $\text{Trans}(\Pi)$ and $Q(a) < P(a)$.

Consider a structure $\mathcal{A}$ with $R^A = \{a, b\}$. Then $\mathcal{A}$ is a stable model of $\Pi$ iff $\text{Dom}(\mathcal{A})$ only contains two elements and $P^A = Q^A = \{a, b\}$. Now we show that this in fact corresponds to the model of $OC(\Pi)$ as well. On one hand, if $\text{Dom}(\mathcal{A})$ only contains two elements and $P^A = Q^A = \{a, b\}$, it can be easily extended to a model of $OC(\Pi)$ by forcing $\forall xy P(x) < Q(y)$ to be true. On the other hand, suppose $\text{Dom}(\mathcal{A})$ contains more than 2 elements. Then $OC(\Pi)$ has no model since there are only two possible cases:

**Case 1:** $P^A = \{a, b\}$.

Then by (2.47), $Q$ holds for all domain elements. By (2.48), $P$ holds for all domain elements as well, a contradiction.

**Case 2:** $P^A \neq \{a, b\}$.

Then by (2.48), $\{a, b\} \subset P^A$. By (2.49), $Q^A$ is not empty. Then by (2.50), $\exists^2 x P(x)$ holds, a contradiction.

In fact, it can be verified that a finite structure $\mathcal{A}$ is a stable model of $\Pi$ iff it can be
expanded to a model of $OC(\Pi)$.

□

2.4.3 Externally Supported Sets with Aggregates

In Definition 7 of Section 2.3.4, we introduced the notion of the externally supported set of ground atoms, i.e., the sets $S \subseteq [P_{int}(\Pi)]^A$ where $[P_{int}(\Pi)]^A = \{ P(a) \mid a \in P^A, P \in P_{int}(\Pi) \}$ for some program $\Pi$ and associated structure $A$. We will now extend this notion to account for the occurrence of aggregate atoms. The following definition is towards this purpose.

**Definition 10** Let $\Pi$ be a FO program with aggregate atoms of form (2.31) and $A$ a structure of some signature $\sigma$ such that $\tau(\Pi) \subseteq \sigma$. Then we say that a set $S \subseteq [P_{int}(\Pi)]^A$ is externally supported (under $A$ and $\Pi$) if there exist some $P(a) \in S$ and rule $P(x) \leftarrow Body(r) \in \Pi$ with local variables $y_r$, such that for some assignment of the form $xy_r \rightarrow ab_r$, we have:

1. $A \models Body(r)[xy_r/ab_r]$;
2. $(Pos(r) \setminus PosAgg(r))[xy_r/ab_r] \cap S = \emptyset$;  
3. For all aggregate atom $\delta \in PosAgg(r)$ of the form (2.31),
   
   $OP(c_v[1] : A \models \overline{Body(\delta)}[xy_r,wv/ab_r,c_w,c_v], Pos(\delta)[xy_r,wv/ab_r,c_w,c_v] \cap S = \emptyset, c_w \in Dom(A)^{|w|}$ and $c_v \in Dom(A)^{|v|} \leq N$,

   holds.

Note that by Definition 10, a set $S \subseteq [P_{int}(\Pi)]^A$ is not externally supported if for all $P(a) \in S$, rule $P(x) \leftarrow Body(r) \in \Pi$ with local variables $y_r$, and assignments $xy_r \rightarrow ab_r$ such that $A \models Body(r)[xy_r/ab_r]$ (i.e., a “support” for $P(a)$), there exists some atom $\beta \in Pos(r)$ such that either

---

$\text{OP}(c_v[1]) : A \models \overline{Body(\delta)}[xy_r,wv/ab_r,c_w,c_v]$, $Pos(\delta)[xy_r,wv/ab_r,c_w,c_v] \cap S = \emptyset$, $c_w \in Dom(A)^{|w|}$ and $c_v \in Dom(A)^{|v|} \leq N$,

---

$\text{OP}(c_v[1])$ denotes the first component (or position) in the tuple $c_v$.  

$\text{OP}(c_v[1])$ denotes the atoms in $Pos(r)$ minus the ones from $PosAgg(r)$ (i.e., the aggregate ones), such that with a slight abuse of notation, $(Pos(r) \setminus PosAgg(r))[xy_r/ab_r]$ denotes the set of predicate instances $\{ Q(b) \mid \beta \in Pos(r) \setminus PosAgg(r), Q(b) \text{ corresponds to } \beta[xy_r/ab_r] \}$.

16In the following, $c_v[1]$ denotes the first component (or position) in the tuple $c_v$. Part 2
1. \( \beta = Q(y) \), where \( Q \in \mathcal{P}_{\text{int}}(\Pi) \) and \( Q(b) \in S \) corresponds to \( \beta[xy_r/ab_r] \), or

2. \( \beta \) is an aggregate atom \( \delta \) of the form (2.31) and

\[
\text{OP}(c_v[1] : A \models \text{Body}(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset) \leq N
\]  

(2.52)

does not hold, where \( \alpha \) is the assignment of the form \( xy_rwv \rightarrow ab_r c_w c_v \) such that \( c_w \in \text{Dom}(A)|w| \) and \( c_v \in \text{Dom}(A)|v| \).

Intuitively, if \( \text{OP} \langle c_v : A|\models \text{Body}(\delta)[\alpha] \rangle \preceq N \) holds (i.e., given \( A|\models \text{Body}(r)[xy_r/ab_r] \)), then (2.52) does not hold because there are not enough \( \text{Body}(\delta)[\alpha] \) with \( \text{Pos}(\delta)[\alpha] \) independent of those grounded atoms from \( S \). Then removing those intensional grounded atoms corresponding to the grounded atoms in \( S \) from \( A \) will make \( \text{OP} \langle c_v : A|\models \text{Body}(\delta)[\alpha] \rangle \preceq N \) false. Now consider the particular cases where \( \text{OP} \in \{\text{CARD}, \text{SUM}, \text{PROD}\} \). If \( \text{OP} \langle c_v : A|\models \text{Body}(\delta)[\alpha] \rangle \preceq N \) holds and (2.52) does not, then \( \preceq \) must be in \( \{=, \geq, >\} \) since \( \text{CARD}, \text{SUM} \) (on \( \mathbb{Z}^+ \)), and \( \text{PROD} \) (on \( \mathbb{N} \)) are (anti)monotonic. Now assume that \( \text{OP} \) is \( \text{MIN} \) and \( \preceq \) is \( \geq \) (or \( \text{OP} \) is \( \text{MAX} \) and \( \preceq \) is \( \leq \)). If \( \text{OP} \langle c_v : A|\models \text{Body}(\delta)[\alpha] \rangle \preceq N \) holds and (2.52) does not, then it must be the case that \( \{\{c_v : A|\models \text{Body}(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset\}\} = \emptyset \) since \( \text{MIN} \) and \( \text{MAX} \) are undefined for \( \emptyset \), i.e., since it is the only way for \( \delta \) to be false in this case.

The following lemma now relates the notion of externally supported to the answer sets of programs with aggregates.

**Lemma 2** Let \( \Pi \) be FO program with aggregate atoms of the form (2.31) under the restrictions in Definition 8, and \( A \) a finite \( \tau(\Pi) \)-structure. Then \( A \) is an answer set of \( \Pi \) iff \( A|\models \hat{\Pi} \) and every \( S \subseteq [\mathcal{P}_{\text{int}}(\Pi)]^A \) is externally supported.

**Proof:** See Appendix A.1.3. \( \square \)

### 2.4.4 Theorem of Ordered Completion with Aggregates

With the notion of externally supported set extended to aggregates, then through Lemma 2, we can now prove the following theorem about the enhanced ordered completion of programs with aggregates.
**Theorem 3** Let \( \Pi \) be FO program with aggregate atoms of the form (2.31) under the restrictions in Definition 8, and \( A \) a finite \( \tau(\Pi) \)-structure. Then \( A \) is an answer set of \( \Pi \) iff there exists an expansion \( A' \) of \( A \) such that \( A' \models OC(\Pi) \).

**Proof:** See Appendix A.1.4. \( \square \)

**Example 12** Let us consider the familiar company control program [FPL11]. We are given an input database corresponding to the predicate \( Company(x) \), denoting the companies involved. Along with all the companies involved, we are also given an input database corresponding to the predicate \( OwnsStk(x, y, n) \), denoting the percentage of shares from company \( y \) that is owned by company \( x \). We say a company \( x \) controls a company \( y \) if the sum of the shares from \( y \) that is either directly owned by \( x \), or by companies also owned by \( x \), is more than 50%. This problem yields the following program \( \Pi_{\text{ctrl}} \):

\[
\begin{align*}
\text{ControlsStk}(x, x, y, n) & \leftarrow OwnsStk(x, y, n) \quad (2.53) \\
\text{ControlsStk}(x, y, z, n) & \leftarrow Company(x), Controls(x, y), OwnsStk(y, z, n) \quad (2.54) \\
\text{Controls}(x, z) & \leftarrow Company(x), Company(z) \\
\text{SUM}(n : \exists y \text{ControlsStk}(x, y, z, n)) & > 50, \quad (2.55)
\end{align*}
\]

where:

- \( \text{ControlStk}(x, y, z, n) \) denotes that company \( x \) controls \( n\% \) of \( z \) shares via company \( y \) (since \( x \) controls \( y \) and where \( y \) owns \( n\% \) of \( z \) shares);

- \( \text{Controls}(x, y) \) encodes that company \( x \) controls company \( y \). Note that by rule (2.55), \( \text{Controls}(x, z) \) can be derived if the sum of the elements in the multiset \( \{ n \mid \exists y \text{ ControlsStk}(x, y, z, n) \} \) is greater than 50%.
Then the ordered completion \(OC(\Pi_{ctrl})\) of \(\Pi_{ctrl}\) is given by the following FO sentence:

\[
\forall xyzn (\langle OwnsStk(x, z, n) \land x = y \rangle \lor \\
(Company(x) \land Controls(x, y) \land OwnsStk(y, z, n)) \rightarrow ControlsStk(x, y, z, n))
\]

\[
\land \forall xz (Company(x) \land Company(z) \land (\text{SUM}(n : \exists y ControlsStk(x, y, z, n)) > 50) \rightarrow Controls(x, y))
\]

\[
\land \forall xyzn (ControlsStk(x, y, z, n) \rightarrow [OwnsStk(x, z, n) \land x = y] \lor \\
[Company(x) \land Controls(x, y) \land OwnsStk(y, z, n) \land Controls(x, y) < ControlsStk(x, y, z, n)])
\]

\[
\land \forall xz (Controls(x, z) \rightarrow Company(x) \land Company(z) \land \\
(\text{SUM}(n : \exists y ControlsStk(x, y, z, n)) > 50) \land \\
(\text{SUM}(n : \exists y ControlsStk(x, y, z, n) \land ControlsStk(x, y, z, n) < Controls(x, z)) > 50)),
\]

where \(Controls(x, y) < ControlsStk(x, y, z, n)\) and \(ControlStk(x, y, z, n) < Controls(x, z)\) denotes the formulas \(\leq ControlsStk(x, y, z, n) < ControlsStk(x, y, z, n)\) and \(\leq ControlsStk(x, y, z, n) < ControlsStk(x, y, z, n)\) respectively. Then by Theorem 3, we have that the answer sets of \(\Pi_{ctrl}\) are exactly those \(\tau(\Pi_{ctrl})\)-structures \(A\) that can be expanded to a model of \(OC(\Pi_{ctrl})\).

\(\square\)

### 2.4.5 Aggregate Constructs as Classical FO Logic

Theorem 3 shows that normal answer set programs with a large variety of aggregates can be captured by their ordered completions. However, the ordered completion defined in Definition 9 is not exactly in first-order logic as it contains aggregate atoms. A natural question arises whether this can be further translated into classical first-order logic. We answer it positively in this section. That is, we show that programs with aggregates functions, restricted in Definition 8, can be further expressed in classical FO logic.

We first express the cardinality aggregate atoms of the form \(\text{CARD}(\langle x : \exists y Lt(\delta) \rangle \leq N)\).
into classical FO logic. In here, \( \widehat{Lt}(\delta) \) is a conjunction of literals that can be comparison literals. For convenience throughout this section, let us set \( \delta_{vw,N} \), for some number \( N \), to stand for the following formula:

\[
\forall vw(\widehat{Lt}(\delta) \rightarrow \bigvee_{1 \leq i \leq N} (vw = v_i w_i)),
\]

(2.56)

where each pair of tuples \( v_i \) and \( w_i \) are fresh tuples of distinctive variables matching the lengths of \( v \) and \( w \), respectively. Intuitively speaking, (2.56) simply encodes that all the \( vw \) satisfying \( \widehat{Lt}(\delta) \) must have a match with at least one of the tuples \( v_1 w_1 \ldots v_N w_N \).

**Definition 11 (CARD into classical FO logic)** Let \( \delta \) be the aggregate atom of the form \( \text{CARD}\langle v : \exists w \widehat{Lt}(\delta) \rangle \preceq N \). Then by \( \delta^{FO} \), we denote the following formula:

\[
\Phi_{\leq,N} \land \ 
( \exists vw \widehat{Lt}(\delta) \rightarrow \exists v_1 w_1 n_1 \ldots v_{N+1} w_{N+1} n_{N+1} \{ \ 
\land_{1 \leq i \leq N+1} \widehat{Lt}(\delta)[vw/v_i w_i] \land \land_{1 \leq i \leq N+1} (n_i = 1 \lor n_i = 0) \land \land_{1 \leq i < j \leq N+1} (v_i w_i = v_j w_j \rightarrow (n_i = 0 \lor n_j = 0)) \land (n_1 + \cdots + n_{N+1} \preceq N) \land \Psi_{\leq,N+1} \} ).
\]

(2.61)

In Definition 11, the formula \( \Phi_{\leq,N+1} \) in (2.57) is defined based on whether or not the aggregate atom

\[
\delta(x) = \text{CARD}\langle v : \exists w \widehat{Lt}(\delta) \rangle \preceq N
\]

can be satisfied by any structure \( A \), and corresponding assignment \( x \rightarrow a \) on \( x \), even if the multiset

\[
\{ \{ b \mid A \models \widehat{Lt}(\delta)[xvw/abc], b \in \text{Dom}(A)^{|v|} \text{ and } c \in \text{Dom}(B)^{|w|} \} \}
\]

(2.62)

is empty. For instance, if \( \preceq \) is \( > \) and \( N \geq 1 \), then we define \( \Phi_{\leq,N} \) to be \( \exists vw \widehat{Lt}(\delta) \). This encodes the fact that \( \delta(x)[x/a] \) cannot be satisfied by \( A \), under the assignment \( x \rightarrow a \), if (2.62) is empty since \( \preceq \) being \( \geq \) and \( N \geq 1 \) implies that there must at least be one \( v \) for which \( \exists vw \widehat{Lt}(\delta)[x/a] \) holds. On the other hand, if \( \preceq \in \{ <, \leq \} \), then we simply omit...
CHAPTER 2. FIRST-ORDER ASP AND CLASSICAL FIRST-ORDER LOGIC

\exists vw \widehat{Lt}(\delta) \text{ in (2.57)} (or equivalently, set } \exists vw \widehat{Lt}(\delta) \text{ as } \top, \text{ since in this case, the multiset (2.62) can be empty.}

About the formula \( \Psi_{\preceq,N+1} \) in (2.61), it is also defined based on the cases about \( \preceq \). If \( \preceq \in \{<,\leq,=\} \), then \( \Psi_{\preceq,N+1} = \delta_{v,w,N+1} \) (see Formula (2.56)). For the other cases (i.e., \( \preceq \in \{\geq,>\} \)), we simply omit \( \Psi_{\preceq,N+1} \) from the formula (or equivalently, set \( \Psi_{\preceq,N+1} = \top \)). Informally speaking, setting \( \Psi_{\preceq,N+1} = \delta_{v,w,N+1} \) for \( \preceq \in \{<,\leq,=\} \) simply insures that we had considered all possible instances of \( vw \) satisfying \( Lt(\delta) \) in \( v_1w_1 \ldots v_{N+1}w_{N+1} \).

In general, although Definition 11 seems a little complicated, the underlying idea is quite simple. We use \( n_i \) to count \( v_iw_i \) but only counting non-duplicated occurrences of \( v_iw_i \). In relation to the formula \( \Phi_{\preceq,N} \) mentioned above, \( \exists vw \widehat{Lt}(\delta) \) in (2.58) is to cover the case where \( \text{CARD} \) is defined on \( \emptyset \). In particular, note that if \( \Phi_{\preceq,N} = \exists vw \widehat{Lt}(\delta) \) in (2.57), then \( \exists vw \widehat{Lt}(\delta) \) must be true in (2.58), which enforces the consequent in (2.58). As already hinted above, the \( \exists v_1w_1n_1 \ldots v_{N+1}w_{N+1}n_{N+1} \) part in (2.58) is for the counting of the satisfiable instances of \( vw \). The \( \Psi_{\preceq,N+1} \) in (2.61) insures that every satisfiable instances of \( vw \) is considered in our counting (for the cases where \( \preceq \in \{<,\leq,=\} \)). The formula

\[ \bigwedge_{1 \leq i \leq N+1} (n_i = 1 \vee n_i = 0) \]

in (2.59) forces \( n_i \) to be only either 1 (count the corresponding instance \( v_iw_i \)) or 0 (not count it). The

\[ \bigwedge_{1 \leq i < j \leq N+1} (v_iw_i = v_jw_j \rightarrow (n_i = 0 \vee n_j = 0)) \]

formula in (2.60) means that for duplicated satisfiable instance \( vw \), we only count at most once, to prevent multiple countings of duplicated instances. Finally, the

\[ (n_1 + \cdots + n_{N+1} \preceq N) \]

formula in (2.61) insures that the counting result is consistent with \( \preceq \) and \( N \).

Through a minor modification of \( \delta^{FO} \) as described Definition 11, we can adopt it to also express the aggregate constructs involving \( \text{SUM} \) and \( \text{PROD} \) into classical FO logic.

**Definition 12 (SUM and PROD into classical FO logic)** Let \( \delta \) be the aggregate atom of the form \( \text{OP}(v : \exists w \widehat{Lt}(\delta)) \leq N \) where \( \text{OP} \in \{\text{SUM}, \text{PROD}\} \). Then by \( \delta^{FO} \), we denote the
following formula:

\[
\Phi_{\leq,N} \land \\
( \exists \mathbf{v} \mathbf{w} \hat{Lt}(\delta) \rightarrow \exists \mathbf{v}_1 \mathbf{w}_1 \ldots \mathbf{v}_{N+1} \mathbf{w}_{N+1} \{ \!
\!
\land \!
\sum_{1 \leq i \leq N+1} \hat{Lt}(\delta)[\mathbf{v} \mathbf{w}/\mathbf{v}_i \mathbf{w}_i] \land \!
\land \!
\sum_{1 \leq i \leq N+1} (n_i = v_i[1] \lor n_i = C) \!
\land \!
\land \!
\sum_{1 \leq i < j \leq N+1} (v_i w_i = v_j w_j \rightarrow (n_i = C \lor n_j = C)) \!
\land \!
\land \!
(n_1 \circ \cdots \circ n_{N+1} \leq N) \land \Psi_{\leq,N+1} \land \Theta_{\text{Dom}} \} ),
\]

where:

- the \( \Phi_{\leq,N} \) in (2.63) is defined as in Definition 11 of the \text{CARD} aggregate constructs;
- the \( C \) in (2.65) and (2.66) is such that \( C = 0 \) for \( \text{OP} = \text{SUM} \), and \( C = 1 \) for \( \text{OP} = \text{PROD} \);
- the \( \circ \) in (2.67) is such that \( \circ = + \) for \text{SUM} and \( \circ = \times \) for \text{PROD}.
- the \( \Psi_{\leq,N+1} \) in (2.67) is also as defined Definition 11 for the \text{CARD} aggregate constructs;
- and finally, the \( \Theta_{\text{Dom}} \) in (2.67) denotes the formula

\[
\forall \mathbf{v} \mathbf{w}(\hat{Lt}(\delta) \rightarrow v[1] \geq C),
\]

where \( v[1] \) (as also found in (2.65)) stands for the first position (or component) of the tuple \( v \). This formula simply insures that all the \( v[1] \) are numbers that are greater than or equal to \( C \) (with \( C = 0 \) for \text{SUM} and \( C = 1 \) for \text{PROD}).

To explain the intuition of Definition 12, let us consider the particular case where \( \text{OP} = \text{SUM} \) (\( \text{OP} = \text{PROD} \) follows similarly). Then \( C = 0 \) in (2.65) and (2.66). In the same way as in the definition of \( \delta^{\text{FO}} \) of the \text{CARD} aggregate constructs, we use the \( n_i \) to simulate the sum of the (first position) of the \( v_i w_i \) but only considering the non-duplicated occurrences. Hence similarly, the

\[
\sum_{1 \leq i \leq N+1} (n_i = v_i[1] \lor n_i = 0)
\]
formula in (2.65) (i.e., with \( C = 0 \) since \( \text{OP} = \text{SUM} \)) insures that \( n_i \) can only be either \( v_i[1] \) (include \( v_i[1] \) in the sum) or 0 (not include it). In addition, the

\[
\bigwedge_{1 \leq i < j \leq N+1} (v_1w_i = v_jw_j \rightarrow (n_i = 0 \lor n_j = 0))
\]

formula in (2.66) (i.e., with \( C = 0 \) since \( \text{OP} = \text{SUM} \)) insures that duplicated instance of \( v_1w_i \) is not considered in the sum. On the other hand, the \( \Theta_{\text{Dom}} \) in (2.67) (see Formula (2.68) for its definition) insures that all the (first position) of the satisfiable instances of \( vw \) are numbers that are greater than or equal to 0. Finally, as in the case of the \( \text{CARD} \) aggregate constructs (as described in Definition 11), the formula \( (n_1 + \cdots + n_{N+1} \preceq N) \) in (2.67) insures the summation result is consistent with \( \preceq \) and \( N \).

Note that with respect to the \( \text{CARD} \) aggregate constructs, we can also explicitly express these into classical FO logic without resorting to any symbols from \( \sigma_{\mathbb{Z}} \) (i.e., the signature associate with number theory). We achieve this by incorporating some ideas from [LLP08]. More precisely, we can alternatively define \( \delta^{\text{FO}} \), where \( \delta \) is a \( \text{CARD} \) aggregate atom, through the following formula

\[
\Phi_{\preceq,N} \land \{
\exists vw \overline{Lt}(\delta) \rightarrow \bigvee_{1 \leq k \leq N+1} \exists v_1w_1 \ldots v_kw_k \left( \bigwedge_{1 \leq i \leq k} \overline{Lt}(\delta)[vw/v_iw_i] \land \bigwedge_{1 \leq i < j \leq k} (v_iw_i \neq v_jw_j) \land \Psi_{\preceq,N+1} \right) \},
\]

where \( \Phi_{\preceq,N} \) and \( \Psi_{\preceq,N+1} \) is as defined in Definitions 11 and 12. Informally speaking, (2.71) simply considers all the possible non-duplicated instances \( v_1w_1 \ldots v_kw_k \) for lengths \( k \) of 1 to \( N + 1 \), for which \( \overline{Lt}(\delta)[vw/v_1w_1] \) holds for \( 1 \leq i \leq k \).

In contrast with the aggregate constructs for \( \text{CARD} \), \( \text{SUM} \), and \( \text{PROD} \), the aggregates involving \( \text{MIN} \) and \( \text{MAX} \) yields a much more simpler translation into classical FO logic. The following definition now provides the translation of the \( \text{MIN} \) and \( \text{MAX} \) aggregates into classical FO logic.

**Definition 13 (MIN and MAX into classical FO logic)** Let \( \delta \) be an aggregate atom of the
form (2.31) where $\text{OP} \in \{\text{MIN}, \text{MAX}\}$. Then by $\delta^{FO}$, we denote the following formula

$$\exists \mathbf{v} \mathbf{w} \hat{\text{Lt}}(\delta) \land \Theta_{\text{Dom}} \land \forall v_1 \{ \forall v_2 (\exists \mathbf{w} \hat{\text{Lt}}(\delta)[v^1/v_1] \land \exists \mathbf{w} \hat{\text{Lt}}(\delta)[v^1/v_2] \rightarrow v_1 \odot v_2 ) \rightarrow v_1 \leq N \},$$

(2.72)

where:

- $\Theta_{\text{Dom}}$ is now\(^{18}\) defined for this MIN and MAX cases as

$$\forall \mathbf{v} \mathbf{w} (\hat{\text{Lt}}(\delta) \rightarrow (\mathbf{v}[1] \leq 0 \lor \mathbf{v}[1] \geq 0)),$$

(2.74)

which simply ensures that $\mathbf{v}[1]$ (i.e., the first position of $\mathbf{v}$) is a number;

- with a slight abuse of notation, $v^1$ denotes the first position (or component) of the tuple $\mathbf{v}$ (i.e., $v^1 = \mathbf{v}[1]$), so that $\hat{\text{Lt}}(\delta)[v^1/v_1]$ (or $\hat{\text{Lt}}(\delta)[v^1/v_2]$) denotes the formula obtained from $\hat{\text{Lt}}(\delta)$ by substituting $v_1$ (or $v_2$) for the first component $\mathbf{v}[1]$ of $\mathbf{v}$;

- $\odot$ is $\leq$ if $\text{OP} = \text{MIN}$ and $\geq$ if $\text{OP} = \text{MAX}$.

Informally speaking, (2.72) encodes that the multiset associated with the aggregate atom $\delta$ is non-empty (i.e., via $\exists \mathbf{v} \mathbf{w} \hat{\text{Lt}}(\delta)$), and that it is a multiset only containing numbers (i.e., via $\Theta_{\text{Dom}}$, as defined in (2.74)). On the other hand, (2.73) simply encodes that the minimum (maximum) element $v_1$ of the multiset satisfies $v_1 \leq N$ (with minimum corresponding to MIN and maximum with MAX).

**Theorem 4** For a program $\Pi$ with aggregate atoms of the form (2.31) under the restrictions in Definition 8. Let $\text{OC}(\Pi)^{FO}$ be the formula obtained from $\text{OC}(\Pi)$ by replacing every aggregate atom $\delta$ in it by $\delta^{FO}$. Then a finite structure $\mathcal{A}$ is an answer set of $\Pi$ iff $\mathcal{A}$ can be expanded into a model of $\text{OC}(\Pi)^{FO}$.

**Proof:** This follows from the descriptions of $\delta^{FO}$ for the various cases of CARD, SUM, PROD, MIN, and MAX, as given in Definitions 11, 12, and 13. □

\(^{18}\)As opposed to (2.68) in Definition 12.
2.4.6 Related Work

Aggregates are extensively studied in the literature [AZZar, DTEFK09, BLM11, FPL11, EIST05, Fer11, PSE04]. Although the syntactic form of aggregates is usually presented in a first-order language, its semantics is normally defined propositionally via grounding. There are several major approaches: Ferraris’ semantics [Fer11], the FLP semantics [FPL11] (later extended for arbitrary formulas by Truszczynski [Tru10]), and the SP semantics [PSE04].

The Ferraris’ semantics and the FLP semantics (and its extension to arbitrary formulas by Truszczynski) are extended into the first-order case [FLL11, BLM11]. Our work captures the first one, also called the stable model semantics, in first-order logic. Certainly, it is interesting to consider whether the FLP semantics can be captured in first-order logic as well. We leave this to our future work. Nevertheless, it is worth mentioning that if the aggregate atoms only occurs in the positive bodies of rules and the bodies of these aggregates contain no negative atoms (this is actually the case in most benchmark programs), these semantics coincide.

Bartholomew et al. [BLM11] studied the aggregate semantics in first-order case via translating into second-order logic. This work is slightly different from ours. Syntactically, the former considers aggregate atoms as arbitrary formula while we only consider a special form, i.e., form (2.31). Semantically, the former combines the theory of first-order atoms and aggregate atoms into a unified one, while the latter defines them separately in the sense that the theory of aggregates is regarded as a background theory. The main reason is for simplicity. It can be shown that they essentially coincide when restricted into the specific aggregate forms.

In this work, we only focused on aggregate frameworks that are based on “translation like” semantics, i.e., either translated the corresponding logic program with aggregates into classical formulas (as in [BLM11] and [AZZar]) or to an aggregate free program (as in [DTEFK09] via the unfolding semantics). Another line of aggregate research worth mentioning are those ones based on fixpoint type characterizations. These line of work stems from the extension of the well-founded semantics of [GRS91] to logic programs with aggregates. The core of these type of approaches [DPB01, PDB07, SP07] is to extend the well-founded semantics to logic programs with aggregates by various extensions of the van Emden-Kowalski provability operator. In fact, we would mention that these approaches are
also applicable to arbitrary (i.e., includes non-monotone) aggregates. It will be very interesting to also explore for future work about how such approaches also relates to classical FO logic.
Chapter 3

Implementation and Experimental Results

3.1 Grounding ASP Programs

To the best of our knowledge, ordered completion provides for the first time a translation from FO normal answer set programs under the stable model semantics to classical FO logic. Significantly, this translation enables us to develop a new direction of ASP solvers by grounding on a program’s ordered completion instead of the program itself.

In this section, we report on a first implementation of such a solver. In order to be consistent with existing ASP solvers, we consider Herbrand structures and not arbitrary structures at this stage. This is because this first implementation does not yet consider constant symbols (or 0-ary functions) in the language, i.e. we currently only assume predicate symbols in the vocabulary. We also point out that we do not yet consider any kind of aggregate constructs in our language.

A grounding transforms an answer set program into an equivalent ground program where the equivalence is defined as having the same set of answer sets. In Lparse [SN01], the answer set programs are assumed to be domain restricted. That is, all the variables in its rules must occur in some positive domain predicate atom in the rule’s body. This syntactical restriction guarantees that the interpretations of all predicates are confined by the interpretations of the domain predicates (i.e., more specifically, confined by the natural joins of the interpretations of domain predicates). Hence in this way, the domain predicates
are viewed as the guards of the variables.

The input answer set programs of DLV [LPF+06] is a larger class of programs called the safe programs, which generalizes the notion of domain restricted programs by only restricting the variables of its rules to occur in a positive body atom of the rule (not necessarily to be a domain predicate atoms). In addition, DLV has the ability to handle disjunctive programs. As a front end, DLV employs a routine called the intelligent grounder (or IG), which uses techniques of rule rewriting and joins-projections reordering. In recent years, GrinGo [GST07] has emerged as a very effective ASP grounding system. GrinGo’s grounding procedure is based on DLV’s back-jumping algorithm [LPS04], and to avoid redundant rules, employs enhanced binder-splitting [GST07].

In this chapter, we provide a different manner of approach. Since we are grounding under the framework of classical FO sentences, we can incorporate classical reasoning directly in our grounding technique. Hence, compared to current ASP grounders such as DLV (as the front end), GrinGo, and Lparse that “reasons” about grounding only under the semantics of the ASP framework, we can do more. For instance, in our technique, the “domain” as constructed using the notion of domain predicates in Lparse can behave in a dynamic way. Hence, the “domain” as constructed during our grounding procedure can actually “shrink” as we progress through the grounding, which is in contrast with current existing ASP grounders where the domain remains static.

### 3.2 Domain Predicates and Guarded Variables

Recall the notion of the positive predicate dependency graph $G^+_\Pi = (\text{Dom}(G^+_\Pi), E^G^+_\Pi)$ of a program $\Pi$ as introduced in Section 2.3.1 where:

- $\text{Dom}(G^+_\Pi) = \mathcal{P}_{int}(\Pi)$, i.e., the intensional predicates of $\Pi$;
- $E^G^+_\Pi = \{(P, Q) \mid$ there is a rule $r \in \Pi$ mentioning $P$ in $\text{Head}(r)$ and $Q$ in $\text{Pos}(r)\}$.

Then we say that a predicate $P$ is a domain predicate under $\Pi$ if either $P$ is an extensional predicate, or $P$ is not involved in a predicate loop in the graph $G^+_\Pi$. By “predicate loop,” we mean there exists a predicate $Q \in \mathcal{P}_{int}(\Pi)$ (can also be $P$) such that $(P, Q), (Q, P) \in \text{TC}(E^G^+_\Pi)$, where $\text{TC}(E^G^+_\Pi)$ denotes the transitive closure of the edge relations $E^G^+_\Pi$ of the
CHAPTER 3. IMPLEMENTATION AND EXPERIMENTAL RESULTS

graph $G^+_Π$. Hence by default, all extensional predicates are domain predicates. By $P_{dom}(Π)$, we denote all the domain predicates of $Π$.

Given a rule $r$ with $x$ the unique tuple of free variables, we call a variable $x$ of $x$ guarded if $x$ appears in a positive body atom that is also of a domain predicate. A rule is guarded if all variables in it are guarded. So naturally, a program is guarded if all the rules in it are guarded.

**Example 13** Consider the well known program $Π_1$ that computes the transitive closure of a graph with extensional predicate $E$ and intensional $T$:

$$
\begin{align*}
    r_1 : T(x, y) & \leftarrow E(x, y) \\
    r_2 : T(x, y) & \leftarrow E(x, z), T(z, y).
\end{align*}
$$

(3.1)

Then all the variables in $r_1$ are guarded while $r_2$ contains the single non-guarded variable $y$. Therefore, $Π_1$ is not guarded.

□

All rules can be converted into a guarded rule without changing its meaning by adding the equality atom $x = x$ for every non-guarded variable $x$ to the rule’s positive body.

**Example 14** Program $Π_1$ can be converted into an equivalent guarded program $Π_2$ by adding $y = y$ into $r_2$’s positive body:

$$
\begin{align*}
    r_1 : T(x, y) & \leftarrow E(x, y) \\
    r_2 : T(x, y) & \leftarrow E(x, z), T(z, y), y = y.
\end{align*}
$$

(3.2)

□

3.3 Ordered Completion vs Clark’s Completion

Let us consider again the Clark’s completion of a predicate $P$ under some program $Π$ as given by

$$
\forall x \forall \Phi \left[ P(x) \leftrightarrow \bigvee_{r \in Π, Head(r)=P(x)} \exists y_r \Phi_{r, \text{Body}(x, y_r)}, \right]
$$

3.3 Ordered Completion vs Clark’s Completion

Let us consider again the Clark’s completion of a predicate $P$ under some program $Π$ as given by

$$
\forall x \forall \Phi \left[ P(x) \leftrightarrow \bigvee_{r \in Π, Head(r)=P(x)} \exists y_r \Phi_{r, \text{Body}(x, y_r)}, \right]
$$
where for our purpose in this chapter, $\Phi_{BODY}^{\text{r}}(x, y_r)$ will now denote the formula originally represented by $\overline{\text{Body}(r)}$ (see Section 2.2).

Note that as mentioned in Section 2.3.3, $P$’s completion can be equivalently written as

$$
\forall x (P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r \Phi_{BODY}^r(x, y_r)).
$$

(3.4)

In the ordered completion (see Definition 6), we simply replaced (3.4) by

$$
\forall x (P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r (\Phi_{BODY}^r(x, y_r) \land \Phi_{OC}^r(x, y_r))),
$$

(3.5)

where for our purpose now in this chapter, $\Phi_{OC}^r(x, y_r)$ will denote the formula originally represented by $P_{\text{Pos}(r)}(x, y_r)$ (see Definition 6).

Clearly, it can be seen that the “necessary” condition for a ground instance of $P$ in (3.5) is stronger than that in (3.4) due to the extra assertions enforced by the comparison atoms (i.e., via $\Phi_{OC}^r(x, y_r)$). Now for a predicate $P$, denote the sentence

$$
\forall x (P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r (\Phi_{BODY}^r(x, y_r) \land \Phi_{OC}^r(x, y_r))).
$$

(3.7)

by $\Phi_{P}^{OC}$. Then we call $\Phi_{P}^{OC}$ as the predicate ordered completion of $P$. Additionally for convenience, for a rule $r$ and predicate $P$, we also denote the formulas

$$
(\Phi_{BODY}^r(x, y_r) \rightarrow P(x))
$$

(3.8)
and

\[(P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \exists y_r (\Phi^\text{BODY}_r(x, y_r) \land \Phi^\text{OC}_r(x, y_r)))\]  (3.9)

by $\Phi^\text{CLO}_r(x, y_r)$ (where “CLO” for closure) and $\Phi^\text{SUP}_P(x)$ (where “SUP” for support) respectively, so that the predicate ordered completion of $P$ can now be represented by the following formula:

\[\Phi^\text{OC}_P = \bigwedge_{r \in \Pi, \text{Head}(r) = P(x)} \forall x y_r \Phi^\text{CLO}_r(x, y_r) \land \forall x \Phi^\text{SUP}_P(x).\]  (3.10)

Also for convenience, we sometimes denote the formula $\Phi^\text{BODY}_r(x, y_r) \land \Phi^\text{OC}_r(x, y_r)$ simply by $\Phi^\text{BODYOC}_r(x, y_r)$.

### 3.4 Grounding Ordered Completion

Structure in the traditional sense defines an interpretation as those extents of a predicate which we regard as true. Hence, those ones regarded as false are defined implicitly as those not in the interpretation. In this section, we introduce the notion of a partial structure, which also contains the notion of a “not-known.” Thus, with this notion in mind, we now define both true and false explicitly.

**Definition 14** A partial-structure $P$ of vocabulary $\tau$ is a tuple $P = (\text{Dom}(P), c^P_1, \ldots, c^P_m, P^P_1, \ldots, P^P_n)$ with constants $c^P_i \in \text{Dom}(P)$ (1 ≤ $i$ ≤ $m$) as usual, but where for each $P_j$ (1 ≤ $j$ ≤ $n$), we have $P^P_j = \{(P_j \cdot T)^P, (P_j \cdot F)^P, (P_j \cdot D)^P\}$ such that:

1. $(P_j \cdot T)^P, (P_j \cdot F)^P, \text{ and } (P_j \cdot D)^P$ are subsets of $\text{Dom}(P)^k$ with $k$ the arity of $P_j$;
2. $(P_j \cdot T)^P \cap (P_j \cdot F)^P = (P_j \cdot D)^P \cap (P_j \cdot F)^P = \emptyset$.

Intuitively speaking, for a predicate $P$, $(P \cdot T)^P$ captures the extents of those regarded as true while $(P \cdot F)^P$ the ones as false. Additionally, $(P \cdot D)^P$ (here, ‘$D$’ stands for domain) captures the notion of those extents that are either true or not known but not false (so hence, it can be that that $(P \cdot T)^P \cap (P \cdot D)^P \neq \emptyset$). Thus, $(P \cdot D)^P$ can be thought of as
a “weaker” form of \((P \cdot T)^P\). This will play a prominent role in our grounding procedure.

For convenience, we refer to \((P \cdot T)^P\), \((P \cdot F)^P\), and \((P \cdot D)^P\) simply as \(P^P \cdot T\), \(P^P \cdot F\), and \(P^P \cdot D\) respectively.

In the spirit of Lparse on constructing “domains” using domain predicates, for an intensional domain predicate \(P\) and input structure \(I\), our use of the “weaker” form of \(P^P \cdot T\) is to hold the current domain relations as constructed using the domain predicates. On the other hand, the relations as contained in \(P^P \cdot F\) and \(P^P \cdot D\) represents the instances of \(P\) that are already known to be true and false, respectively, in order to keep \(OC(\Pi)\) from being inconsistent (or unsatisfiable).

### 3.4.1 Constructing Domains Using Partial Structures

We now propose a procedure for grounding the formula \(\Phi^CLO_x, y_r\). First, unless otherwise stated, we set \(I\) and \(E\) to be structures defined as the follows:

- \(I = (\text{Dom}(I), c_I^1, \ldots, c_I^r, I^I_1, \ldots, I^I_s)\) is the input or known structure such that \(c_I^i \in \text{Dom}(I)\) \((1 \leq i \leq r)\) and \(I^I_j \subseteq \text{Dom}(I)|^{I_j}|\) \((1 \leq j \leq s)\) (i.e., this corresponds to the extensional structure);

- \(E = (\text{Dom}(E), c_E^1, \ldots, c_E^r, E^E_1, \ldots, E^E_t)\) is the expansional or output partial-structure that covers both the intensional and comparison predicates, and such that:

  1. \(c_I^i = c_E^i\) \((1 \leq i \leq r)\) (i.e., interpretations of the constants are the same);

  2. \(E^E_j \cdot F\), \(E^E_j \cdot D\), and \(E^E_j \cdot T\) \((1 \leq j \leq t)\) are subsets of \(\text{Dom}(E)|^{E_j}|\) (i.e., \(E^E_j \cdot F\), \(E^E_j \cdot D\), and \(E^E_j \cdot T\) are tuples of \(\text{Dom}(E)\) and of length \(|E_j|\)).

Thus, generally speaking, we have that \(I\) corresponds to the extensional structure of signature \(\tau_{ext}(\Pi)\) (hence we refer to this as the “input” structure), while \(E\) corresponds to the intensional structure plus the comparison predicates (hence we refer to these as the “expansional” structure).

In the well known classical problem of model expansion [PLTG07], a given structure (in our case, the extensional or input structure) is expanded with new relations (this corresponds in our case to the expansional or output structure) such that the new relations satisfies a given formula (the ordered completion in our case). It is important to note that

---

1This notation is borrowed from the way structure members are referred to in C++.
although our approach will be similar in principle to that of grounding for model expansion, our approach can also incorporate several techniques that had proved effective in traditional ASP systems.

For convenience, we say that a predicate $P \in \tau(\Pi)$ is of an input (or extensional) predicate if $P \in \tau_{ext}(\Pi)$, otherwise we say that it is of an expansional predicate (note that this covers the cases where $P$ is either an intensional or comparison predicate).

**Definition 15** Given a first-order formula $\Phi(x)$ of signature $\tau$ with free variables $x$, and input structure $I$ of signature $\tau' \subseteq \tau$, a domain for $\Phi(x)$ is a pair $(x, T)$ where $T \subseteq \text{DOM}(I)^{|x|}$ (i.e., a set of tuples of arity $|x|$). In most context, we refer to the pair $(x, T)$ as a table $T$ with rows corresponding to the tuples in $T$, and whose fields (or attributes) correspond to the vector $x$. Sometimes, when it is clear from the context, we simply use $T$ to refer to the $T$ in $(x, T)$.

For convenience, for a table $T = (x, T)$, we denote by $|T|$ as the size of $T$, i.e., $|T| = |T|$.

**Example 15** Let $x$ be the tuple of variables $xyz$ and $T$ be the corresponding (i.e., 3-ary) set of relations $\{(a, b, c), (a, b, b), (b, b, a), (a, c, c), (a, a, b)\}$. Then $T = (x, T)$ represents the following table:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

and where $|T| = |T| = 5$. □

For the rule $r$ of the formula $\Phi^{\text{clo}}_r(x, y_r)$, for each unguarded variable $x$ of $xy_r$, add the corresponding atom $x = x$ in its positive body to transform $r$ into a guarded rule. Now, assuming $\text{Head}(r) = P(x)$, Algorithm 1 describes our get domain procedure for obtaining a domain for $\Phi^{\text{body}}_r(x, y_r)$. For this purpose, we assume that $\Phi^{\text{body}}_r(x, y_r)$ is of the form

$$P_1(x_1) \land \cdots \land P_m(x_m) \land \neg Q_1(y_1) \land \cdots \land \neg Q_n(y_n).$$
Moreover, we can further assume without loss of generality that $P_1(x_1)$ is a domain predicate (i.e., note that in case $r$ is not guarded to start with via some variable, say $x$, adding the equality atom $x = x$ in its positive body gives it (at least) a domain predicate in its positive body). Note that get domain also considers the formula $\Phi_{\text{BODY}}^\text{OC}(x, y_r)$ (i.e., similar to $\Phi_{\text{BODY}}^\text{r}(x, y_r)$ in that it can contain the comparison atoms).

Algorithm 1: GetDomain

Data: The formula $\Phi_{\text{BODY}}^\text{r}(x, y)$ or $\Phi_{\text{BODY}}^\text{OC}(x, y)$, the input structure $I$, and the expansional partial-structure $E$. 

Result: A table $T$ with fields $xy_r$

1 begin
2 if $P_1$ is of an input (extensional) predicate then
3 \[ T \leftarrow [P_1^E]_{x_1} \]
4 else
5 \[ T \leftarrow [P_1^E \cdot D]_{x_1} \]
6 for $2 \leq i \leq m$ do
7 if $P_i$ is a domain predicate then
8 \[ T \leftarrow T \bowtie [P_i^E]_{x_i} \]
9 else
10 \[ T \leftarrow T \bowtie [P_i^E \cdot D]_{x_i} \]
11 else
12 \[ T \leftarrow T \bowtie [P_i^E \cdot F]_{x_i} \]
13 for $1 \leq j \leq n$ do
14 if $Q_j$ is an input predicate then
15 \[ T \leftarrow T \bowtie [Q_j^E]_{y_j} \]
16 else
17 \[ T \leftarrow T \bowtie [Q_j^E \cdot T]_{y_j} \]
18 return $T$

First, in Lines 2 to 5, we set the table $T$ such that if $P_1$ is an input (or expansional) predicate, then $T$ is the table $[P_1^E]_{x_1}$ (or $[P_1^E \cdot D]_{x_1}$), where $[P_1^E]_{x_1}$ (or $[P_1^E \cdot D]_{x_1}$) is the table corresponding to the interpretation $P_1^E$ (or $P_1^E \cdot D$) of $P_1$ with fields (i.e., attributes) $x_1$. Then for each atoms $P_i(x_i)$ for $2 \leq i \leq m$ (i.e., corresponds to $\text{Pos}(r)$), we take the natural join of $T$ with:

- $[P_i^E]_{x_i}$ if $P_i$ is of the input (extensional) predicate, or
• \([P^E_i \cdot D]_x\), if \(P_i\) is a both a domain and expansional predicate.

Additionally, if \(P_i\) is a non-domain and expansional predicate, then we take instead the complementary join of \(T\) with \([P^E_i \cdot F]_x\). The table \(T\) is then set to the new resulting table. On the other hand, for each atoms \(Q_j\) for \(1 \leq j \leq n\) (i.e., corresponds to \(\text{Neg}(r)\)) we take instead the complementary join of \(T\) with:

1. \([Q^E_j]_x\) if \(Q_j\) is input;
2. \([Q^F_j \cdot T]_x\) if \(Q_j\) is expansional. Finally, Line 19 returns the resulting table \(T\) with fields \(xy_r\).

**Example 16** Assume we have \(r\) as the rule

\[
P(w, z) \leftarrow I(w, x), E_1(y, z), E_2(y, z), \text{not } E_3(w, z),
\]

such that \(xy_r = wzxy\), \(\text{Head}(r) = P(w, z)\), and \(\Phi_r^{\text{BODY}}(x, y_r)\) the formula

\[
I(w, x) \land E_1(y, z) \land E_2(y, z) \land \neg E_3(w, z).
\]

In Addition, we have that:
1. \(I\) is of the input structure;
2. \(E_1\) is a domain predicate of the expansional partial-structure;
3. both \(E_2\) and \(E_3\) are non-domain predicates of the expansional partial-structure. Then we perform the following join operations:

\[
[I^E]_{wx} \times [E^E_1 \cdot D]_{yz} \bowtie^c [E^E_2 \cdot F]_{yz} \bowtie^c [E^E_3 \cdot T]_{wz},
\]

such that the resulting table with fields \((w, z, x, y)\) is the domain for \(\Phi_r^{\text{BODY}}(x, y_r)\). Additionally, to obtain a domain for \(P\) (i.e., \(P^E \cdot D\)), we project under the fields \((w, z)\)

\[
\pi_{w,z}(I^E_{wx} \times [E^E_1 \cdot D]_{yz} \bowtie^c [E^E_2 \cdot F]_{yz} \bowtie^c [E^E_3 \cdot T]_{wz}),
\]

and add the tuples corresponding to the resulting table to \(P^E \cdot D\).

\(\square\)

Informally speaking, compared to current approaches to grounding ASP programs, we also take into account the already “known” expansional relations (i.e., those of \(P^E \cdot F\) and \(P^E \cdot T\), while most current grounders only considers \(P^E \cdot D\).
3.4.2 Dynamic Domain Relations

Consider again the rule $r$ in Example 16 and such that this time, assume that $E_1^E \cdot D = E_2^E \cdot F$ (i.e., the “domain” for $E_1$ is the same as the false of $E_2$). Then $[E_1^E \cdot D]_{yz} \propto c [E_2^E \cdot F]_{yz} = \emptyset$, and thus, $[F]_{wz} \propto [E_1^E \cdot D]_{yz} \propto c [E_2^E \cdot F]_{yz} \propto c [E_3^E \cdot T]_{wz} = \emptyset$. Hence, this time, we have $P^E \cdot D = \emptyset$. This is our motivation for this section.

**Proposition 9** Let $T_1$, $T_2$, and $T_3$ be tables with fields $x$, $y$, and $z$ respectively, where $y \subseteq x$. Then $|T_1 \propto^c T_2 \propto T_3| \leq |T_1 \propto T_3|$.

**Proposition 10** Let $T_1$, $T_2$, and $T_3$ be the tables again from Proposition 9. Then computing $T_1 \propto^c T_2 \propto T_3$ is done in time $O(|T_1| \cdot |T_2| + |T_1 \propto^c T_2| \cdot |T_3|)$, while computing $T_1 \propto T_3$ is done in $O(|T_1| \cdot |T_3|)$.

**Proposition 11** Let $T_1$, $T_2$, and $T_3$ be the tables again from Proposition 9. Then $|T_1 \propto^c T_2| \leq |T_1| \cdot (1 - |T_2|/|T_1|)$ implies $|T_1| \cdot |T_2| + |T_1 \propto^c T_2| \cdot |T_3| \leq |T_1| \cdot |T_3|.$

That is, if it is the case that the inequality

$$|T_1 \propto^c T_2| \leq |T_1| \cdot (1 - |T_2|/|T_3|)$$

holds, then computing $T_1 \propto^c T_2 \propto T_3$ is more efficient than computing $T_1 \propto T_3$. Hence, if the ratio $|T_2|/|T_1|$ is small and $|T_1 \propto^c T_2| < |T_1|$ (i.e., $T_1$ is reduced from the complement of join with $T_2$), then it is highly probable that computing $T_1 \propto^c T_2 \propto T_3$ is faster than computing $T_1 \propto T_3$. Note that in usual practice, the known expansion relations (i.e., that takes the role of the table $T_2$) is generally a lot smaller in size than the “domain” relations (as played by $T_3$).

**Example 17** Assume we have $\Phi^R_{\text{DOMAIN}}(x, y_r)$ to be

$$I(w, x, y) \land E_1(y) \land \leq_{E_1 E_2} (y, w, x) \land \neg \leq_{E_2 E_1} (w, x, y) \land E_3(w, x, z),$$

such that: (1) $I$ is the input predicate; (2) $E_1$ is a non-domain and expansional predicate; (3) $E_3$ is a domain and expansional predicate; (4) $\leq_{E_1 E_2}$ and $\leq_{E_2 E_1}$ are comparison predicates. Then by the GETDOMAIN procedure, we perform the join operations

$$[F]_{wxy} \propto [E_1^E \cdot F]_{iy} \propto c [\leq_{E_1 E_2} E_1 \cdot F]_{wux} \propto c [\leq_{E_1 E_2} E_1 \cdot T]_{wxy} \propto [E_1^E \cdot D]_{wxz}.$$
Recall that for a predicate $P$, we denoted by $\Phi^\text{SUP}_P(x)$ as the following formula:

$$P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r(\Phi^\text{BODYOC}_r(x, y_r))$$

(i.e., the support for the instances of $P$). Then an instance $P(a)$ of $P$ will be known to be false (i.e., $a \in P^E \cdot F$), if

$$\bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r(\Phi^\text{BODYOC}_r(x, y_r))[x/a]$$

is false. As already mentioned above (i.e., Section 3.3), $\Phi^\text{SUP}_P(x)$ is a much stronger necessary condition than its Clark’s completion counter part, due to the extra assertions enforced by the comparison atoms (i.e., via the formula $\Phi^\text{OC}_r(x, y_r)$). Thus, to fully exploit this “stronger” property, we assume that each comparison predicates $\leq_{PQ}$ and $\leq_{QR}$ of the expansional partial-structure has the following properties:

1. $ab \in \leq_{PQ} \cdot T$ and $bc \in \leq_{QR} \cdot T$ implies $ac \in \leq_{PR} \cdot T$ (i.e., the transitive closure);

2. if either $P \neq Q$ or $a \neq b$ then $ab \in \leq_{PQ} \cdot T$ implies $ba \in \leq_{QP} \cdot F$, otherwise, $P = Q$ and $a = b$ implies $ab \notin \leq_{PQ} \cdot T$ and $ab \in \leq_{PQ} \cdot F$ (the asymmetric property).

For intuition, we can think of the comparison predicate relations as a dynamic set of relations that updates itself, as required, when new known relations are added, in order to satisfy the transitive and asymmetric properties (i.e., analogous to an oracle).

**Proposition 12** Checking whether the comparison predicates of the expansional partial-structure are transitively closed can be done in time

$$O\left(\sum_{P,Q,R \in \mathcal{P}_{\text{int}}(\Pi)} |T^E_{PQ} \cdot T| \cdot |T^E_{QR} \cdot T|\right).$$

**Proof:** For each triple $P, Q, R \in \mathcal{P}_{\text{int}}(\Pi)$, we perform the join operation $[T^E_{PQ} \cdot T]_{xy} \bowtie [T^E_{QR} \cdot T]_{yz}$ where $x, y,$ and $z$ are pairwise disjoint tuples of distinguished variables of length $|P|, |Q|,$ and $|R|$ respectively. We then define a new table $T_{PR}$ with fields $x$ and $z$, obtained
from the projection $\pi_{x,y}(\left[ T^\varepsilon_{PQ} \cdot T \right]_{xy} \bowtie \left[ T^\varepsilon_{QR} \cdot T \right]_{yz})$. Then if $T^\varepsilon_{PQ} \cdot T = T^\varepsilon_{PQ} \cdot T \cup T^\varepsilon_{PQ}$ for each pairs $P,Q \in \mathcal{P}_{\text{int}}(\Pi)$, the comparison relations are transitively closed. □

3.4.3 Grounding on a Domain Predicate Sequence

Definition 16 (Domain grounding sequence) Let $\Pi$ be a (first-order) answer set program with intensional predicates $\mathcal{P}_{\text{int}}(\Pi)$. Then a strict total-order $S = (\mathcal{P}_{\text{int}}(\Pi), <)$ on $\mathcal{P}_{\text{int}}(\Pi)$ is called a domain grounding sequence (or just grounding sequence) if for every pair of predicates $P_1, P_2 \in \mathcal{P}_{\text{int}}(\Pi)$ not in a predicate loop in $G^+_\Pi$, $P_1 < P_2$ implies that $P_2$ is not reachable from $P_1$ in the graph $G^+_\Pi$.

Intuitively speaking, a domain grounding sequence $S = (\mathcal{P}_{\text{int}}(\Pi), <^S)$ is a strict-total order on $\mathcal{P}_{\text{int}}(\Pi)$ such that for each pair of predicates $P, Q \in \mathcal{P}_{\text{int}}(\Pi)$ not involved in a predicate loop in $G^+_\Pi$, if $P <^S Q$ then $Q$ is not reachable from $P$ in $G^+_\Pi$. Informally, this makes sure that $Q$ is on a higher stratum than $P$. Hence, if $P$ and $Q$ are domain predicates such that $Q$ depends on $P$, then from $P$, we construct the domain for $Q$. That is, we use the relations from the interpretation $P^\varepsilon \cdot D$ of $P$ to obtain the “domain” for $Q$, which is $Q^\varepsilon \cdot D$ (i.e., via the table join and projection operations).

Definition 17 Given a FO program $\Pi$, assume that $\mathcal{P}_{\text{int}}(\Pi) = \{P_1, \ldots, P_n\}$. Without loss of generality, also assume that $S = (\mathcal{P}_{\text{int}}(\Pi), <^S)$ is a predicate grounding sequence of $\mathcal{P}_{\text{int}}(\Pi)$ such that $P_1 <^S \cdots <^S P_n$. We recursively define the grounding $G^r(\text{OC}(\Pi))$ of $\text{OC}(\Pi)$ as follows:

1. $G^r(\text{OC}(\Pi)) = \bigwedge_{1 \leq i \leq n} G^r(\Phi^\text{OC}_{P_i})$;

2. $G^r(\Phi^\text{OC}_{P_i}) = (\bigwedge_{r \in \Pi, \ Head(r) = P_i(x_i)} G^r(\Phi^\text{CLO}_r(x_i, y_r))) \land G^r(\Phi^\text{SUP}_{P_i}(x_i))$;

3. $G^r(\Phi^\text{CLO}_r(x, y_r)) = \bigwedge_{aa_r \in T} \Phi^\text{CLO}_r(x y_r/aa_r)$
   (where $T$ is the table as obtained by the GETDOMAIN procedure on $\Phi^\text{BODY}_r(x, y_r)$);
4. \( \mathcal{G}_r(\Phi_r^{\text{SUP}}(x)) = \bigwedge_{a \in P^r \cdot D} (P(a) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \mathcal{G}_r(\Phi_r^{\text{BODYOC}}(xy_r/ay_r))) \); \\
5. \( \mathcal{G}_r(\Phi_r^{\text{BODYOC}}(xy_r/ay_r)) = \bigvee_{a_i \in T} \Phi_r^{\text{BODYOC}}(xy_r/aa_r) \)

(where \( T \) is the table obtained via a similar method to the \textsc{GetDomain} procedure on \( \Phi_r^{\text{BODYOC}}(x_i)[x/a] \), and such that for each \( \Phi_r^{\text{CL}O}(x_iy_r/a_ia_r) = \Phi_r^{\text{BODY}} \rightarrow P_i(a_i) \), we add \( a_i \) to \( P_i^E \cdot D \), provided it is not the case that \( a_i \notin P_i^E \cdot F \).

Let us now take a closer look at Definition 17. Basically, given a predicate grounding sequence \( S = (P_{\text{int}}(\Pi), <^S) \) on \( P_{\text{int}}(\Pi) = \{P_1, \ldots, P_n\} \) such that \( P_1 <^S \ldots <^S P_n \), \( \mathcal{G}_r(\text{OC}(\Pi)) \) (i.e., the grounding of the whole ordered completion) is defined as the grounding of \( \Phi_{P_1}^{\text{OC}}, \ldots, \Phi_{P_n}^{\text{OC}} \) in the sequence \( 1, \ldots, n \). As with the usual technique of the use of domain predicates in traditional ASP grounders, this grounding sequence produces the “domain tuples” associated with the intensional predicates \( P_1, \ldots, P_n \). For every \( P_i \) (\( 1 \leq i \leq n \)), this is fulfilled by the grounding of each \( \Phi_r^{\text{CL}O}(x_i, y_r) \) for rules \( r \in \Pi \) such that \( \text{Head}(r) = P_i(x_i) \). Indeed, as indicated in the last sentence of Definition 17, each relation \( a_i \) corresponding to a grounded rule \( \Phi_r^{\text{CL}O}(x_iy_r/a_ia_r) = \Phi_r^{\text{BODY}} \rightarrow P_i(a_i) \), is added to \( P_i^E \cdot D \) (provided it is not known to be false), and where we use \( P_i^E \cdot D \) in the future join operations, i.e., to get the “domain” relations for other predicates.

As mentioned in Section 3.4.2, the set \( P_i^E \cdot D \) of domain relations can also reduce in size after the grounding of the corresponding \( \Phi_{P_i}^{\text{SUP}}(x_i) \), i.e., see Item 2 of Definition 17. This is because each domain relation \( a_i \in P_i^E \cdot D \) is verified in

\[
P_i(a_i) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) = P_i(x_i)} \mathcal{G}_r(\Phi_r^{\text{BODYOC}}(x_iy_r/a_ia_r))
\]

if it can be justified by

\[
\bigvee_{r \in \Pi, \text{Head}(r) = P_i(x_i)} \mathcal{G}_r(\Phi_r^{\text{BODYOC}}(x_iy_r/a_ia_r)),
\]

where \( \Phi_r^{\text{BODYOC}}(x_iy_r/a_ia_r) \) is a stronger condition than \( \Phi_r^{\text{BODY}}(x_iy_r/a_ia_r) \). Thus, in this sense, the “domain relations” can behave in a dynamic way, which can optimize future join operations due to some tables possibly shrinking in size.
3.4.4 Incorporating Classical Reasoning

For each application of $G_r$ on a formula $\Phi(x)$, we also do an application of $S$ on $G_r(\Phi(x))$ (i.e., $S(G_r(\Phi(x)))$) such that for a propositional formula $\varphi$, $S(\varphi)$ is recursively defined as follows:

1. if $\varphi = P(a)$, then:
   (a) if $P = I_i$ (for $1 \leq i \leq s$) is of an input (or extensional) predicate, then if $a \in I_i^T$ (or $a \notin I_i^T$), $S(\varphi) = \top$ (or $S(\varphi) = \bot$);\footnote{Recall that $I = (\text{Dom}(I), c_I^1, \cdots, c_I^r, I_1^I, \cdots, I_s^I)$ denotes the input (also called the extensional) structure (see beginning of Section 3.4.1).}
   (b) else, if $P = E_i$ (for $1 \leq i \leq t$) is of an expansional predicate, then:
      i. if $a \in E_i^T \cdot T$ (or $a \in E_i^F \cdot F$), then $S(\varphi) = \top$ (or $S(\varphi) = \bot$),
      ii. otherwise, $S(\varphi) = P(a)$;

2. else, if $\varphi = \neg(\psi)$, then $S(\varphi) = \neg S(\psi)$;

3. else, if $\varphi = \psi \land \xi$ (or $\varphi = \psi \lor \xi$), then $S(\varphi) = S(\psi) \land S(\xi)$ (or $S(\varphi) = S(\psi) \lor S(\xi)$),

and where $\top \land \varphi \equiv \bot \lor \varphi \equiv \varphi$, $\bot \land \varphi \equiv \bot$, and $\top \lor \varphi \equiv \top$ as usual.

In addition, if $S(\varphi)$ results in the conjunctive clause $l_1 \land \cdots \land l_k$ for $\varphi = G_r(\Phi^\text{Clo}_r(x, y_r))$ or $\varphi = G_r(\Phi^\text{Sup}_p(x))$, then $S(\varphi) = \top$ and such that for each $l_i$ (1 $\leq i \leq k$): if $l_i = P(a)$, then $a$ is added to $P^E \cdot T$; and if $l_i = \neg P(a)$, then $a$ is added to $P^E \cdot F$ and deleted from $P^E \cdot D$. This will then dynamically effect future join operations involving $P^E \cdot T$, $P^E \cdot F$, and $P^E \cdot D$. For convenience, we denote this immediate application of $S$ with each application of $G_r$ as $S \circ G_r$, where ‘$\circ$’ denotes function composition.

It should be noted from the definition of $S$ that $S(\varphi)$ is a formula with literals only corresponding to the instances of the expansional predicates (i.e., the input ones had been turned into $\top$ or $\bot$). In a nutshell, the operator $S$ simply does the simplification of the resulting propositional formula on the fly. Based on these simplifications, apart from reducing the size of the resulting proposition theory, we are also able to know more true and false “domain relations,” which can be useful in optimizing the future grounding of the other formulas.

\footnote{Recall that $E = (\text{Dom}(E), c_E^1, \cdots, c_E^r, E_1^E, \cdots, E_t^E)$ represents the expansional partial structure (see beginning of Section 3.4.1).}
3.4.5 Iteration and Conversion to SMT

With $\Gamma = (S \circ G_r)(\text{OC}(\Pi))$,\(^4\) $S$ can be applied again to $\Gamma$ to simplify it further, and possibly collect more known expansional relations using classical reasoning. It can even be that $S$ is iteratively applied to $\Gamma$ until $\Gamma$ cannot be simplified any further.

After $(S \circ G_r)(\text{OC}(\Pi))$, we are now at liberty to treat instances $P(a)$ for which $a \notin P^E \cdot D$ as false. That is, instance $P(a)$ for which either $a \in P^E \cdot F$ or $a \notin P^E \cdot D$ is now treated as $\bot$. This is because $P^E \cdot D$ corresponds to the relations that appear in the heads of instances of rules mentioning $P$. In fact, it will be necessary to make at least one application of $S$ to consider if all the instances (i.e., propositional atoms) occurring in the resulting propositional formula are mentioned in $P^E \cdot D$. This corresponds to the well-known notion that an atom $P(a)$ is in some answer set only if it is mentioned in some head of a grounded rule of the program. It is important to note that this is only necessary for the occurrences of the non-domain predicate instances since those of the domain predicates are already assured to be in the “domain relations” as it was already enforced by the predicate grounding sequence (see Section 3.4.3).

An obvious drawback of the grounded form of the ordered completion is the transitive axioms of the comparison predicates (see Formula 2.12), since this introduces $O(n^3)$ propositional clauses into the formula (with $n$ the number of intensional ground atoms). To get over this problem, we borrow some ideas from [Nie08]. Thus, instead of using a SAT solver to find a model of the theory, we instead use an SMT (Satisfiability Modulo Theories) [NOT99] solver. Through this way, instead of incorporating the $O(n^3)$ propositional clauses (of the transitive axioms) into the resulting propositional theory, we instead use the “built in” predicate ‘$\geq$’ of the SMT framework. We achieve this by converting each comparison predicate instance $\leq_{PQ} (a, b)$ into the inequality atom

$$x_{Q(b)} - 1 \geq x_{P(a)},$$

where $x_{Q(b)}$ and $x_{P(a)}$ are new variables ranging over the integers. Intuitively, the atom $x_{Q(b)} - 1 \geq x_{P(a)}$ encodes the notion that there exist mappings from the variables $x_{P(a)}$ and $x_{Q(b)}$ to the integers for which $x_{P(a)}$ is strictly less than $x_{Q(b)}$.

Thus, assuming $\Gamma = S((S \circ G_r)(\text{OC}(\Pi)))$ (i.e., we apply at least one iteration of $S$

\(^4\)Recall that ‘$\circ$’ denotes function composition.
to \((S \circ Gr)(OC(\Pi))\), then as already stated above, we can map \(\Gamma\) into a difference logic formula \(DIF(\Gamma)\) of the SMT framework.

**Theorem 5** Given a FO program \(\Pi\), input structure \(\mathcal{I}\) of signature \(\tau_{ext}(\Pi)\), and \((P_{int}(\Pi), <^S)\) a domain grounding sequence on its intensional predicates, let \(\Gamma = S((S \circ Gr)(OC(\Pi)))\) such that \(Gr(\Pi)\) is the grounding on the domain predicate sequence \((P_{int}(\Pi), <^S)\). Now, let \(DIF(\Gamma)\) (i.e., its mapping to difference logic). Then there exists an expansion \(\mathcal{I}'\) of \(\mathcal{I}\) on the signature \(\tau_{ext}(\Pi) \cup P_{int}(\Pi) = \tau(\Pi)\) such that \(\mathcal{I}'\) is an answer set of \(\Pi\) iff \(DIF(\Gamma)\) is satisfiable.

### 3.5 Experimental Results

![The Groc ASP solver main architecture.](image)

In the following, we report our first experiment with a prototype implementation. The goal of our experiments is to compare our new type of ASP solver based on ordered completion, as depicted in Figure 3.1, with a number of existing ASP solvers based on the traditional approach.

The input is divided into two parts: a FO program as well as a set of ground facts (i.e., an extensional database). The output is to return an answer set of the program based on the extensional database if there exists such one, and to return “no” otherwise.

In order to solve this problem, existing ASP solvers normally use a 2-step approach. First, a grounder is used to transform the FO program together with the extensional database to a propositional program. Here, we considered Gringo (version 3.03) [GST07] as the grounder. Second, a propositional ASP solver is called to compute an answer set. In this section, we consider three different propositional ASP solvers, including Clasp (version
2.0.1 [GKNS07], Cmodels (version 3.81) [Lie05], and Lp2diff (version 1.27) [JNS09] with Z3 (version 3.2.18).

Differently, our solver needs 3 steps. First, we translate a FO program to its ordered completion. As this step is normally very efficient and can be considered as off line, we do not count the time used in this step. Second, we have implemented a grounder, called Groc (version 1.0) [ALZZ12], to transform the ordered completion together with the extensional database to a propositional SMT theory. Finally, we call an SMT solver to compute a model of the SMT theory, which should be an answer set of the program based on the extensional database by Theorem 5. We use Z3 (version 3.2.18) as the SMT solver in this step.

We consider the Hamiltonian Circuit benchmark program, originally inspired by Niemelä [Nie99], given by the following:

\[
hc(x, y) \leftarrow \text{arc}(x, y), \text{not otherroute}(x, y)
\]
\[
\text{otherroute}(x, y) \leftarrow \text{arc}(x, y), \text{arc}(x, z), hc(x, z), y \neq z
\]
\[
\text{otherroute}(x, y) \leftarrow \text{arc}(x, y), \text{arc}(z, y), hc(z, y), x \neq z
\]
\[
\text{reached}(y) \leftarrow \text{arc}(x, y), hc(x, y), \text{reached}(x), \text{not init}(x)
\]
\[
\text{reached}(y) \leftarrow \text{arc}(x, y), hc(x, y), \text{not init}(x)
\]
\[
\text{vertex}(x), \text{not reached}(x),
\]

such that \text{arc}, \text{vertex}, \text{and init} are the extensional, and \text{hc}, \text{otherroute}, and \text{reached} are the intensional predicates. The current benchmark graph instances for the HC program normally contain no more than 150 nodes. Here, we instead consider much bigger graph instances. That is, we consider random graphs with nodes ranging from 200 to 1000, in which the numbers of edges are ten times the numbers of nodes. The graph instances are named as \text{rand} \_\text{nodes} \_\text{edges} \_\text{number}, where \text{rand} means this is a random graph, \text{nodes} represents the number of nodes in this graph, \text{edges} represents the number of edges in this graph, and \text{number} is the code of this graph in this category. For instance, \text{rand} \_200 \_2000 \_6 is a random graph with 200 nodes and 2000 edges, and is the 6-th graph instance in this

---

5http://www.cs.uni-potsdam.de/clasp/
6http://www.cs.utexas.edu/ tag/cmodels/
7http://www.tcs.hut.fi/Software/lp2diff/
9The system is found at http://staff.scms.uws.edu.au/~yzhou/?page_id=174
Table 3.1 reports some runtime data of our experiments on the HC program with those relatively big graph instances. The experiments were performed on a CENTOS version 2.16.0 LINUX platform with 2GB of memory and AMD Phenom 9950 Quad-Core processor running at 2.6GHz. For space reasons, we only report the overall time used by the following different approaches:

- Gringo as the Grounder and Clasp as the propositional ASP solver (gringo+clasp);
- Gringo as the grounder and Cmodels as the propositional ASP solver (gringo+cmodels);
- Gringo as the grounder, Lp2diff as the translator from propositional programs to SMT and Z3 as the SMT solver (Gringo+Lp2diff+Z3);
- finally, our solver by using Groc to ground the ordered completion together with the extensional databases, and then calling the SMT solver Z3 (groc+Z3).

We set the timeout threshold as 900 seconds, which is denoted by “—” in our experimental results.

In Table 3.1, the first column specifies the graph instances. In the second column, “y” (“n”) means that the corresponding graph has a (no) Hamiltonian Circuit, while “?” means that this problem instance is not solved by any approaches within limited time. The rest of the four columns records the overall time in seconds used by the four different approaches. It is worth mentioning that, normally, the grounding time (i.e., for Gringo and Groc) is much less than the solving time.

Table 3.1 shows that our solver compares favorably to the others on the Hamiltonian Circuit benchmark program, especially for those big graph instances. For 200 node random graphs, our solver seems not as good as Gringo+Clasp in general, but still looks slightly better than the other two. However, when the graph becomes bigger, our advantages emerge. For those 400 node and 600 node graphs, our solver clearly outperforms the other approaches. Moreover, for 1000 node random graphs, our solver is the only one capable of solving the problems within the time threshold. Also, it is interesting to take a closer look at the only instance with no answer sets, i.e., \textit{rand\_200\_2000\_3}. With our grounder Groc, the inconsistency can be identified immediately.
### Table 3.1: Experimental Results

<table>
<thead>
<tr>
<th>instances</th>
<th>Gringo + Clasp</th>
<th>Gringo + Cmodels</th>
<th>Gringo + Lp2diff + Z3</th>
<th>Groc + Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td>rand_200_2000_1</td>
<td>y 0.325</td>
<td>3.130</td>
<td>6.954</td>
<td>1.79</td>
</tr>
<tr>
<td>rand_200_2000_2</td>
<td>y 0.604</td>
<td>3.310</td>
<td>10.185</td>
<td>1.95</td>
</tr>
<tr>
<td>rand_200_2000_3</td>
<td>n 0.175</td>
<td>0.150</td>
<td>2.507</td>
<td>0.00</td>
</tr>
<tr>
<td>rand_200_2000_4</td>
<td>y 1.453</td>
<td>7.960</td>
<td>18.412</td>
<td>1.66</td>
</tr>
<tr>
<td>rand_200_2000_5</td>
<td>y 0.329</td>
<td>7.600</td>
<td>8.899</td>
<td>15.24</td>
</tr>
<tr>
<td>rand_400_4000_1</td>
<td>y ——</td>
<td>——</td>
<td>49.506</td>
<td>5.08</td>
</tr>
<tr>
<td>rand_400_4000_2</td>
<td>y 24.110</td>
<td>——</td>
<td>——</td>
<td>59.31</td>
</tr>
<tr>
<td>rand_400_4000_3</td>
<td>? ——</td>
<td>——</td>
<td>——</td>
<td>——</td>
</tr>
<tr>
<td>rand_400_4000_4</td>
<td>y ——</td>
<td>——</td>
<td>46.938</td>
<td>8.10</td>
</tr>
<tr>
<td>rand_400_4000_5</td>
<td>y ——</td>
<td>——</td>
<td>162.277</td>
<td>8.00</td>
</tr>
<tr>
<td>rand_600_6000_1</td>
<td>y 140.830</td>
<td>——</td>
<td>114.973</td>
<td>12.16</td>
</tr>
<tr>
<td>rand_600_6000_2</td>
<td>y ——</td>
<td>——</td>
<td>203.500</td>
<td>38.41</td>
</tr>
<tr>
<td>rand_600_6000_3</td>
<td>y ——</td>
<td>——</td>
<td>340.219</td>
<td>45.84</td>
</tr>
<tr>
<td>rand_600_6000_4</td>
<td>y ——</td>
<td>——</td>
<td>83.650</td>
<td>52.13</td>
</tr>
<tr>
<td>rand_600_6000_5</td>
<td>y ——</td>
<td>——</td>
<td>403.075</td>
<td>9.20</td>
</tr>
<tr>
<td>rand_1000_10000_1</td>
<td>y ——</td>
<td>——</td>
<td>——</td>
<td>324.22</td>
</tr>
<tr>
<td>rand_1000_10000_2</td>
<td>y ——</td>
<td>——</td>
<td>——</td>
<td>133.66</td>
</tr>
<tr>
<td>rand_1000_10000_3</td>
<td>y ——</td>
<td>——</td>
<td>——</td>
<td>99.32</td>
</tr>
<tr>
<td>rand_1000_10000_4</td>
<td>y ——</td>
<td>——</td>
<td>——</td>
<td>256.91</td>
</tr>
<tr>
<td>rand_1000_10000_5</td>
<td>y ——</td>
<td>——</td>
<td>——</td>
<td>295.89</td>
</tr>
</tbody>
</table>
More importantly, it is reasonable to believe that the performance of our solver can be further improved by employing more optimization techniques because this is our first implementation based on an entirely new method. Among them, one specific technique, both theoretically challenging and practically promising, is the simplification of the FO formula of ordered completion. As mentioned earlier, this step can be regarded as offline since it only needs to be done once for each program. Evident from some simple observations, the simplifications can sometimes significantly reduce the size of ordered completion, thus the grounded propositional SAT/SMT theory. To us, this is one of the most important work left in the future.

3.6 Further Remarks

In this chapter, we have showed that through ordered completion, we have been able to integrate ASP grounding techniques with those of first-order theories. Moreover, since ordered completion is a first-order theory, we are able to reason about grounding in the classical manner. Thus, we are now able to blur the distinction between the actual grounding and solving processes of ASP programming implementations. Among our key contributions is the exploitation of the comparison atoms to the full effect. This was achieved by explicitly considering the comparison atoms in the table join operations. Additionally, for maximum affect, we assumed that the “known” relations of the comparison predicates always satisfy the notions of transitivity and asymmetry. So in fact, our grounding procedure incorporates a sort of “background theory” that allows us to simulate a kind of built in “<” predicate as found in SMT solver systems. Furthermore, as already mentioned above, although theoretically challenging but practically promising, this work opens up the possibility for the direct simplification of the FO formula of ordered completion, which may be a viewed as a kind of offline reasoning.
Chapter 4

Preferred FO Answer Set Programs

4.1 Lifting Preferred Answer Set Programming to First-Order

Preferences play an important role in knowledge representation and reasoning. In the past decade, a number of approaches for handling preferences have been developed in various nonmonotonic reasoning formalisms, e.g., see a survey in [DSTW04], while adding preferences into answer set programming has promising advantages from both implementation and application viewpoints [DST03].

In recent years, as an important enhancement of traditional answer set programming approaches, FO programs have been intensively studied by researchers, e.g., [ALZZ12, FLL11, ZZ10]. FO answer set programming generalizes the traditional propositional answer set programming paradigm in the following aspects: syntactically it allows arbitrary FO formulas to be used to formalize the underlying problem scenario, while its semantics is characterized by a SO sentence which has certain similarity to circumscription [FLL11, LZ11].

An important research agenda in this direction is to redevelop important functionalities and properties, that have been successful in propositional answer set programming, to the new framework of FO answer set programming, e.g., [LM09]. In this work, we propose a semantic framework for preferred FO programs in the case of normal logic programs, and show how preference reasoning is properly captured under this new framework. In particular, we make the following contributions towards the development of the preferred
FO answer set programming:

- We propose a progression based preference semantics for FO programs. This semantics is a generalization of Zhang and Zhou's progression semantics for FO normal answer set programs [ZZ10], and which also extends Schaub and Wang's preference semantics for propositional answer set programming [SW03];

- We investigate essential semantic properties of preference reasoning under the proposed preferred FO answer set programming. In order to prove these important properties, we specifically consider the grounding of preferred answer set programs and establish its connections to the first-order case.

- Finally, we address the expressiveness of preferred FO answer set programming in relation to classical SO logic. In particular, we show that the proposed preferred semantics can be precisely represented by a SO sentence on arbitrary structures. Furthermore, by restricting on finite structures, the preferred semantics can be characterized by an existential SO sentence. As a consequence of this result, we know that on finite structures, a preferred FO normal program can always be represented by a FO sentence under an extended vocabulary.

4.2 A Progression Based Semantics

As in the case of preferred propositional programs, a preferred FO program $\mathcal{P}$ is a structure $(\Pi, <^\mathcal{P})$ where $<^\mathcal{P}$ is an asymmetric and transitive relation and such that this time, $\Pi$ is a FO program rather than a propositional one. Intuitively speaking, if $(r_1, r_2) \in <^\mathcal{P}$ (i.e., note that we sometimes write this as $r_1 <^\mathcal{P} r_2$), we mean that $r_1$ is more preferred than $r_2$.

That is, when we evaluate $\Pi$, we consider that $r_1$ has a higher priority than $r_2$ during the evaluation. However, this intuition is quite vague, and we need to make this precise.

To develop a semantics for preferred FO programs, we first introduce some useful notions. Let $\Pi$ be a FO program and $r$ a rule in $\Pi$. We denote by $\text{Var}(r)$ and $\text{Const}(r)$ to be the set of all variables and constants occurring in $r$ respectively. We also denote by $\text{Term}(r)$ as the set $\text{Var}(r) \cup \text{Const}(r)$. Consider a $\tau(\Pi)$-structure $\mathcal{M}$ and $r \in \Pi$. We define
an assignment $\eta$ in $\mathcal{M}$ with respect to $r$ as a function:

$$\eta : \text{Var}(r) \longrightarrow \text{Dom}(\mathcal{M}).$$

We may extend $\eta$ from arbitrary terms occurring in $r$ to $\text{Dom}(\mathcal{M})$ by mapping each constant occurring in $r$ to an element $c^\mathcal{M}$ in $\text{Dom}(\mathcal{M})$. We also define the set $\Sigma(\Pi)_\mathcal{M}$ as follows:

$$\Sigma(\Pi)_\mathcal{M} = \{(r, \eta) \mid r \in \Pi \text{ and } \eta : \text{Term}(r) \longrightarrow \text{Dom}(\mathcal{M})\}.$$ 

Basically, $\Sigma(\Pi)_\mathcal{M}$ contains all the rules of $\Pi$ but together with the references to all the possible associated assignments under the structure $\mathcal{M}$. Such assignment references will be essential to the development of the semantics for preferred programs.

Given a program $\Pi$, let $\mathcal{M}$ be a $\tau(\Pi)$-structure. We specify $\mathcal{M}^0(\Pi)$ to be a new $\tau(\Pi)$-structure obtained from $\mathcal{M}$ as follows:

$$\mathcal{M}^0(\Pi) = (\text{Dom}(\mathcal{M}), c^\mathcal{M}_1, \ldots, c^\mathcal{M}_r, P^\mathcal{M}_1^0, \ldots, P^\mathcal{M}_s^0, Q^\mathcal{M}_1^0, \ldots, Q^\mathcal{M}_n^0),$$

where $c^\mathcal{M}_i = c^\mathcal{M}_i$ for each constant $c_i$ of $\tau(\Pi)$ ($1 \leq i \leq r$), $P^\mathcal{M}_j^0 = P^\mathcal{M}_j$ for each extensional predicate $P_j$ in $\tau_{\text{ext}}(\Pi)$ ($1 \leq j \leq s$), and $Q^\mathcal{M}_k^0 = \emptyset$ for each intensional predicate $Q_k$ in $\tau_{\text{int}}(\Pi)$ ($1 \leq k \leq n$). Furthermore, for some $X \subseteq \Sigma(\Pi)_\mathcal{M}$, we also define $\lambda^{\mathcal{M}^0}(X)$ to be a $\tau(\Pi)$-structure generated from $\mathcal{M}^0$ and $X$ in the following way:

$$\lambda^{\mathcal{M}^0}(X) = \mathcal{M}^0(\Pi) \cup \{\text{Head}(r)\eta \mid (r, \eta) \in X\}.$$ 

We are now ready to present a progression based semantics for preferred FO programs.

**Definition 18 (Preferred evaluation stage)** Let $\mathcal{P} = (\Pi, \prec^\mathcal{P})$ be a preferred FO program and $\mathcal{M}$ a $\tau(\Pi)$-structure. The $t$-th preferred evaluation stage of $\mathcal{P}$ based on $\mathcal{M}$, denoted as
Γ^t(\mathcal{P})_\mathcal{M}, is a set of pairs \((r, \eta)\) (i.e., \(\Gamma^t(\mathcal{P})_\mathcal{M} \subseteq \Sigma(\Pi)_\mathcal{M}\), defined inductively as follows:

\[\Gamma^0(\mathcal{P})_\mathcal{M} = \{(r, \eta) \mid (1) \text{Pos}(r) \eta \subseteq \mathcal{M}^0(\Pi) \text{ and } \text{Neg}(r) \eta \cap \mathcal{M} = \emptyset; (2) \text{there do not exist a rule } r' \in \Pi \text{ and an assignment } \eta' \text{ such that } r' <^P r, \text{Pos}(r') \eta' \subseteq \mathcal{M} \text{ and } \text{Neg}(r') \eta' \cap \mathcal{M}^0(\Pi) = \emptyset}\};\]

\[\Gamma^{t+1}(\mathcal{P})_\mathcal{M} = \Gamma^t(\mathcal{P})_\mathcal{M} \cup \{(r, \eta) \mid (1) \text{Pos}(r) \eta \subseteq \lambda \mathcal{M}^0(\Gamma^t(\mathcal{P})_\mathcal{M}) \text{ and } \text{Neg}(r) \eta \cap \mathcal{M} = \emptyset; (2) \text{there do not exist a rule } r' \in \Pi \text{ and an assignment } \eta' \text{ such that } r' <^P r, (r', \eta') \notin \Gamma^t(\mathcal{P})_\mathcal{M}, \text{ and } \text{Pos}(r') \eta' \subseteq \mathcal{M} \text{ and } \text{Neg}(r') \eta' \cap \lambda \mathcal{M}^0(\Gamma^t(\mathcal{P})_\mathcal{M}) = \emptyset}\}.\]

Let us now take a closer look at Definition 18. First, \(\mathcal{M}^0(\Pi)\) contains all extensional relations as input, and any rule \(r\) with the highest priority, where its positive body is satisfied in \(\mathcal{M}^0(\Pi)\) (i.e., \(\text{Pos}(r) \eta \subseteq \mathcal{M}^0(\Pi)\)) and negative body is not satisfied in \(\mathcal{M}\) (i.e., \(\text{Neg}(r) \eta \cap \mathcal{M}^0(\Pi) = \emptyset\)), will be generated along with the underlying assignment \(\eta\) (i.e., as a tuple \((r, \eta)\)). This forms the initial stage \(\Gamma^0(\mathcal{P})_\mathcal{M}\).

Next, we now consider the structure \(\lambda \mathcal{M}^0(\Gamma^t(\mathcal{P})_\mathcal{M})\) that is obtained from the structure \(\mathcal{M}^0(\Pi)\) by expanding all the relations derived from the rules in \(\Gamma^t(\mathcal{P})_\mathcal{M}\) (with the associated assignments). Then the structure \(\lambda \mathcal{M}^0(\Gamma^t(\mathcal{P})_\mathcal{M})\) will be used as a basis to evaluate the preferred program \(\mathcal{P}\) at the \((t+1)\)-th stage. Condition (1) in \(\Gamma^{t+1}(\mathcal{P})_\mathcal{M}\) is quite straightforward: the positive body of the rule to be generated should be satisfied by the structure obtained from the previous evaluation stages, and the negative should not be satisfied in \(\mathcal{M}\). Condition (2) in \(\Gamma^{t+1}(\mathcal{P})_\mathcal{M}\), on the other hand, takes the preference into account. In particular, based on Condition (1), we require that a rule to be generated should meet two criteria: (a) no other rules with higher priorities have not been generated earlier; and (b) these rules’ positive bodies are already satisfied in \(\mathcal{M}\) and their negative bodies are not satisfied (not defeated) by the structure \(\lambda \mathcal{M}^0(\Gamma^t(\mathcal{P})_\mathcal{M})\) generated from the previous stages. Generally speaking, this strategy insures that when we consider a rule for application, other more preferred rules had already been settled in the sense that the more preferred rules had already been derived or that they cannot possibly ever be derived since either their positive bodies are not satisfied by \(\mathcal{M}\) or that they are already defeated by the structure of the previous stage.
Basically, Definition 18 may be viewed as the generalization of Zhang and Zhou’s progression semantics [ZZ10] for FO normal logic program by taking preference into account. We should also emphasize its inside connection to Schaub and Wang’s order preservation semantics for propositional preferred logic programs [SW03]. In particular, the Condition (2) in the specification of $\Gamma^{i+1}(P)_M$ is a first-order generalization of Schaub and Wang’s immediate consequence operator (i.e., operator $T_{\mathfrak{I},<,YX}$ in Definition 1 of [SW03]). Nevertheless, Schaub and Wang’s order preservation semantics may not simply be extended into our first-order case since during each evaluation stage, we must keep track of the assignments that were applied to all the rules in the progression evaluation. Without such assignment references, the rules generated from the evaluation stage will lose their preference features because rules with different priorities may be instantiated to the same grounded rule 1.

**Definition 19 (progression based preferred semantics)** Let $P = (\Pi, <^P)$ be a preferred FO program and $M$ a $\tau(\Pi)$-structure. Then $M$ is called a preferred answer set of $P$ iff $\lambda_M(\Gamma^\infty(P)_M) = M$.

**Example 18** Let us consider the following simple preferred program $P_1 = (\Pi_1, <^{P_1})$ where $\Pi_1$ is the following program:

$$r_1 : \text{Flies}(x) \leftarrow \text{Bird}(x), \text{not Cannot}_{.}\text{fly}(x)$$
$$r_2 : \text{Cannot}_{.}\text{fly}(x) \leftarrow \text{Penguin}(x), \text{not Flies}(x),$$

and where $<^{P_1} = \{(r_2, r_1)\}$ (i.e., $r_2 <^{P_1} r_1$). Let us now consider a finite structure $M$ were

$$\text{Dom}(M) = \{\text{cody, tweety}\};$$
$$\text{Bird}^M = \{\text{cody, tweety}\};$$
$$\text{Penguin}^M = \{\text{tweety}\};$$
$$\text{Flies}^M = \{\text{cody}\};$$
$$\text{Cannot}_{.}\text{fly}^M = \{\text{tweety}\}.$$}

Then $\text{Bird}$ and $\text{Penguin}$ are the extensional and $\text{Flies}$ and $\text{Cannot}_{.}\text{fly}$ are the intensional

---

1 This bottleneck notion was mentioned in Section 1.2.5.
predicates of \( \Pi_1 \). According to Definition 18, we have:

\[
\Gamma_0(P_1)_M = \{(r_2, \eta) \mid \eta: x \rightarrow \text{tweety}\};
\]

\[
\Gamma_2(P_1)_M = \Gamma_1(P_1)_M = \Gamma_0(P_1)_M \cup \{(r_1, \eta') \mid \eta': x \rightarrow \text{cody}\}.
\]

Then from Definition 19, we have:

\[
\lambda_{M^0}(\Gamma^2(P_1)_M) = M^0 \cup \{\text{Cannot\_fly(tweety)}, \text{Flies(cody)}\} = M.
\]

Thus, we have that \( M \) is a preferred answer set of \( P_1 \).

□

**Proposition 13** Let \( P = (\Pi, <^P) \) be a preferred program. If \( <^P = \emptyset \), then a structure \( M \) of \( \tau(\Pi) \) is a preferred answer set of \( P \) iff \( M \) is an answer set of \( \Pi \).

**Proof:** By restricting the preference relation \( <^P \) to be empty, the definition of \( \Gamma^t(P)_M \) is simplified to the following form:

\[
\Gamma^0(P)_M = \{(r, \eta) \mid \text{Pos}(r) \eta \subseteq M^0(\Pi) \text{ and } \text{Neg}(r) \eta \cap M = \emptyset\};
\]

\[
\Gamma^{t+1}(P)_M = \Gamma^t(P)_M \cup \{(r, \eta) \mid \text{Pos}(r) \subseteq \lambda_{M^0}(\Gamma^t(\Pi)_M) \text{ and } \text{Neg}(r) \eta \cap M = \emptyset\}.
\]

It is easy to observe the correspondence between \( \Gamma^t(P)_M \) and \( M^t(\Pi) \) in Definition 1 from [ZZ10]. Indeed, we can show that for each \( t \geq 0 \), \( \lambda_{M^0}(\Gamma^t(P)_M) = M^{t+1}(\Pi) \). Therefore, \( \lambda_{M^0}(\Gamma^\infty(P)_M) = M^\infty(\Pi) \). Then from Theorem 1 of [ZZ10], we conclude that \( M \) is a preferred answer set of \( P = (\Pi, \emptyset) \) iff \( M \) is an answer set of \( \Pi \). □

**Proposition 14** Let \( P = (\Pi, <^P) \) be a preferred program. If a structure \( M \) of \( \tau(\Pi) \) is a preferred answer set of \( P \), then \( M \) is also an answer set of \( \Pi \).

**Proof:** See Appendix A.2.1. □
4.3 Properties of Preferred Answer Sets

In this section, we study some essential properties of the preferred answer set semantics proposed in last section. Firstly, from Proposition 14, we know that for any preferred program $\mathcal{P} = (\Pi, <^\mathcal{P})$, its preferred answer sets must also be answer sets of $\Pi$. Now we take a closer look at the relationship between the existence of an answer set of $\Pi$ and of a preferred one of $\mathcal{P}$.

**Example 19** Let $\mathcal{P}_2 = (\Pi_2, <^{\mathcal{P}_2})$ be a preferred program such that $\Pi_2$ is the program:

\[
\begin{align*}
 r_1 & : P(x) \leftarrow Q(x) \\
 r_2 & : Q(x) \leftarrow,
\end{align*}
\]

and where $<^{\mathcal{P}_2} = \{(r_1, r_2)\}$, i.e., $r_1 <^{\mathcal{P}_2} r_2$. Note that $\Pi_2$ has no extensional predicate, and any structure $\mathcal{M}$ on $\tau(\Pi_2)$ where $P^\mathcal{M} = Q^\mathcal{M} = \text{Dom}(\mathcal{M})$ is an answer set of $\Pi_2$. But $\mathcal{P}_2$ has no preferred answer set. To see this, we consider Definition 18. For any $\mathcal{M}$ which is an answer set of $\Pi_2$, we have $\Gamma^0(\mathcal{P}_2)_\mathcal{M} = \Gamma^1(\mathcal{P}_2)_\mathcal{M} = \emptyset$. This follows that $\lambda^\mathcal{M}_0(\Gamma^\infty(\mathcal{P}_2)_\mathcal{M}) = \mathcal{M}'$ were $P^{\mathcal{M}'} = Q^{\mathcal{M}'} = \emptyset$. By Definition 19, $\mathcal{M}$ cannot be a preferred answer set of $\mathcal{P}_2$.

Now we consider another preferred program $\mathcal{P}_3 = (\Pi_3, <^{\mathcal{P}_3})$ such that $\Pi_3$ is the program:

\[
\begin{align*}
 r_1 & : P(y) \leftarrow P(x), Q(x) \\
 r_2 & : P(x) \leftarrow Q(x),
\end{align*}
\]

and where $r_1 <^{\mathcal{P}_3} r_2$. Here, $Q$ is the only extensional predicate of $\Pi_3$. It is not difficult to show that for any answer set $\mathcal{M}$ of $\Pi_3$ were $P^\mathcal{M} = Q^\mathcal{M} \neq \emptyset$, $\mathcal{M}$ cannot be a preferred answer set of $\mathcal{P}_3$. However, $\mathcal{P}_3$ has one preferred answer set $\mathcal{M}'$ in which $P^{\mathcal{M}'} = Q^{\mathcal{M}'} = \emptyset$. □

The preferred program $\mathcal{P}_2$ from Example 19 suggests that the existence of an answer set for a program $\Pi$ does not necessarily imply the existence of a preferred answer set for a corresponding preferred program $\mathcal{P} = (\Pi, <^\mathcal{P})$. On the other hand, the preferred program $\mathcal{P}_3$ seems to suggest that if the positive body for each rule of the program is not empty, then a preferred answer set always exists. The following proposition shows that this is the case.
Proposition 15 Let $\Pi$ be a program where for each rule $r \in \Pi$, $Pos(r) \neq \emptyset$. Then for any preferred program $\mathcal{P} = (\Pi, \prec^\mathcal{P})$ built upon $\Pi$, $\mathcal{P}$ has a preferred answer set.

4.3.1 Grounding Preferred programs

It has been observed that the answer sets of FO programs can always be obtained via its grounded correspondence [ALZZ12]. We would like to explore whether this approach is also suitable for preferred FO programs since a grounding based approach will not only have advantages in computing preferred answer sets in practice, but also provide an effective means of studying the semantic properties of the corresponding preferred FO programs.

Nevertheless, unlike the case of FO answer set programs, a naive grounding method does not work for preferred programs. Let us consider the following preferred program $\mathcal{P}_4 = (\Pi_4, \prec_{P_4})$ such that $\Pi_4$ is the program:

\[
\begin{align*}
   r_1 : & \quad P(x) \leftarrow Q(x), Q(y) \\
   r_2 : & \quad P(z) \leftarrow Q(z),
\end{align*}
\]

and where $r_1 <_{P_4} r_2$. If we simply ground $\Pi_4$ under a domain consisting of a singleton $\{a\}$, the grounded program would only contain one instance $\{P(a) \leftarrow Q(a)\}$, while the original preference relation $r_1 <_{P_4} r_2$ collapses with this instance \(^2\). This problem may be avoided by relating each rule of $\Pi$ with a corresponding tag predicate, as shown in the following.

Let $\mathcal{P} = (\Pi, \prec^\mathcal{P})$ be a preferred FO program of vocabulary $\tau(\Pi)$. A preferred program $\mathcal{P}' = (\Pi', \prec^{P'})$ is called the tagged preferred program of $\mathcal{P}$, if:

1. for each rule $r_i \in \Pi$ of the form:

\[
   r_i : \quad \alpha \leftarrow \beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots \text{,not } \gamma_m,
\]

$\Pi'$ contains its tagged rule of the form:

\[
   r_i : \quad \alpha \leftarrow \beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots \text{,not } \gamma_m, \text{Tag}_i(x)
\]

\(^2\)Note that this was also addressed in Section 1.2.5.
where \( x \) is the tuple of all variables occurring in \( r_i \) and \( {\text{Tag}}_i \) is a new extensional predicate not in \( \tau(\Pi) \);

2. \( \Pi' \) does not contain any other rules;

3. for any two rules \( r_i' \) and \( r_j' \) in \( \Pi' \), \( r_i' <^P r_j' \) iff \( r_i <^P r_j \), and where \( <^P \) is as specified in \( \mathcal{P} \).

Now let \( \mathcal{P} = (\Pi, <^P) \) be a preferred program of the vocabulary \( \tau(\Pi) \) and \( \mathcal{P}' = (\Pi', <^{P'}) \) be the tagged preferred program of \( \mathcal{P} \) of the vocabulary \( \tau(\Pi') = \tau(\Pi) \cup \{ {\text{Tag}}_1, \ldots, {\text{Tag}}_k \} \).

Given a structure \( \mathcal{M} \) of \( \tau \), we construct an expansion \( \mathcal{M}' \) of \( \mathcal{M} \) to \( \tau(\Pi') \) as follows:

1. \( \text{Dom}(\mathcal{M}') = \text{Dom}(\mathcal{M}) \);

2. For each predicate \( P \) and constant \( c \) in \( \tau \), \( P^M' = P^M \) and \( c^M' = c^M \);

3. For each \( n \)-ary \( {\text{Tag}}_i \) in \( \tau(\Pi') \) \((1 \leq i \leq k)\), \( {\text{Tag}}_i^M' = \text{Dom}(\mathcal{M}')^n \).

Let \( r' \in \Pi' \) and \( \eta \) an assignment on structure \( \mathcal{M}' \) of \( \tau(\Pi') \). We denote by \( r'\eta \) the ground instance of \( r' \) based on \( \eta \). We are now ready to define the grounding of a preferred answer set program.

**Definition 20** Let \( \mathcal{P} = (\Pi, <^P) \) be a preferred program, \( \mathcal{M} \) a structure of \( \tau(\Pi) \), \( \mathcal{P}' = (\Pi', <^{P'}) \) the tagged preferred program of \( \mathcal{P} \), and \( \mathcal{M}' \) the expansion of \( \mathcal{M} \) to \( \tau(\Pi') \) as described above. We say that a structure \( \mathcal{P}^* = (\text{Ground}(\Pi), \mathcal{M}', <^{P^*}) \) is the grounded preferred program of \( \mathcal{P} \) based on \( \mathcal{M} \), if:

1. \( \text{Ground}(\Pi)_\mathcal{M} = \{ r^* \mid r^* : \text{Head}(r) \eta \leftarrow \text{Body}(r) \eta, {\text{Tag}}(x) \eta, \text{where } r \in \Pi \text{ and } \eta \text{ is an assignment on } \mathcal{M}' \} \cup \{ P(a) \leftarrow \mid P \in \mathcal{P}_{\text{ext}}(\Pi) \text{ and } P(a) \in P^M \} \cup \{ {\text{Tag}}_i(a) \leftarrow \mid {\text{Tag}}_i(a) \in {\text{Tag}}_i^M(1 \leq i \leq k) \} \cup \{ a = a \leftarrow \mid a \text{ is an element of } \text{Dom}(\mathcal{M}') \}^4 \);

2. \( <^{P^*} = \{ (r^*_i, r^*_j) \mid r^*_i, r^*_j \in \text{Ground}(\Pi)_\mathcal{M} \text{ for some } r'_k, r'_l \in \Pi' \text{ such that there exist an assignment } \eta \text{ on } \mathcal{M}' \text{ satisfying } r^*_i = r'_k \eta, r^*_j = r'_l \eta \text{ and } r'_k <^{P'} r'_l \} \).

\(^3\)We assume that \( \Pi \) contains rules \( r_1, \ldots, r_k \).

\(^4\)Here we view \( a = a \) as a propositional atom.
Example 20 We consider the preferred program $P = (\Pi, <^P)$ from Example 18 again. Under the structure $M$ given in Example 18, the grounded preferred program $P^*_1 (\text{Ground}(\Pi), M, <^P)$ of $P_1$ is as follows:

$r^*_1 : \text{Flies}(\text{cody}) \leftarrow \text{Bird}(\text{cody}), \text{not Cannot}_ \_fly(\text{cody}), \text{Tag}_1(\text{cody})$
$r^*_2 : \text{Flies}(\text{tweety}) \leftarrow \text{Bird}(\text{tweety}), \text{not Cannot}_ \_fly(\text{tweety}), \text{Tag}_1(\text{tweety})$
$r^*_3 : \text{Cannot}_ \_fly(\text{cody}) \leftarrow \text{Penguin}(\text{cody}), \text{not Flies}(\text{cody}), \text{Tag}_2(\text{cody})$
$r^*_4 : \text{Cannot}_ \_fly(\text{tweety}) \leftarrow \text{Penguin}(\text{tweety}), \text{not Flies}(\text{tweety}), \text{Tag}_2(\text{tweety})$
$r^*_5 : \text{Bird}(\text{cody}) \leftarrow$
$r^*_6 : \text{Bird}(\text{tweety}) \leftarrow$
$r^*_7 : \text{Penguin}(\text{tweety}) \leftarrow$
$r^*_8 : \text{Tag}_1(\text{cody}) \leftarrow$
$r^*_9 : \text{Tag}_1(\text{tweety}) \leftarrow$
$r^*_{10} : \text{Tag}_2(\text{cody}) \leftarrow$
$r^*_{11} : \text{Tag}_2(\text{tweety}) \leftarrow$
$r^*_{12} : \text{cody} = \text{cody} \leftarrow$
$r^*_{13} : \text{tweety} = \text{tweety} \leftarrow$
$r^*_3 <^P r^*_1$
$r^*_3 <^P r^*_2$
$r^*_4 <^P r^*_1$
$r^*_4 <^P r^*_2$.

Now let us consider the preferred program $P_3 = (\Pi, <^P)$ in Example 19 under the finite structure $M = (\{a\}, \{P(a)\}, \{Q(a)\})$. It is clear that $M$ is an answer set of $\Pi_3$. By Definition 20, it is easy to see that the grounded preferred program $P^*_3 = (\text{Ground}(\Pi_3), M, <^P)$ of $P_3$ consists of the following rules and preferences:

$r^*_1 : P(a) \leftarrow Q(a), \text{Tag}_1(a)$
$r^*_2 : P(a) \leftarrow Q(a), \text{Tag}_2(a)$
$r^*_3 : Q(a) \leftarrow$
$r^*_4 : \text{Tag}_1(a) \leftarrow$
$r^*_5 : \text{Tag}_2(a) \leftarrow$
$r^*_6 : a = a \leftarrow$
$r^*_4 <^P r^*_1$.
It is noted that with the tag predicate introduced, the preferences from the original preferred program \( P_3 \) is preserved in its grounded preferred program \( P_3^* \).

Basically, a grounded preferred program is a program in propositional form and may contain infinite number of rules and preference relations if an infinite domain is considered. We will define the semantics for such grounded preferred programs and show that it coincides with the progression based semantics of preferred FO programs.

**Definition 21 (preferred answer sets of grounded programs)** Let \( P = (\Pi, <^P) \) be a grounded preferred program as obtained from Definition 20 and \( S \) a set of propositional atoms. The \( t \)-th preferred evaluation stage of \( P \) based on \( S \), denoted as \( \Delta^t(P)_S \), is a set of rules from \( \Pi \), i.e., \( \Delta^t(P)_S \subseteq \Pi \), defined inductively as follows:

\[
\Delta^0(P)_S = \{ r \mid (1) \text{ Pos}(r) = \emptyset \text{ and } \text{Neg}(r) \cap S = \emptyset; \\
(2) \text{ there does not exist a rule } r' \in \Pi \text{ such that } r' <^P r, \text{ Pos}(r') \subseteq S \text{ and } \text{Neg}(r') \cap \text{Head}(r) = \emptyset \}; \\
\Delta^{t+1}(P)_S = \Delta^t(P)_S \cup \{ r \mid (1) \text{ Pos}(r) \subseteq \text{Head}(\Delta^t(P)_S) \text{ and } \text{Neg}(r) \cap S = \emptyset; \\
(2) \text{ there does not exist a rule } r' \in \Pi \text{ such that } r' <^P r, r' \not\in \Delta^t(P)_S, \text{ and } \text{Pos}(r') \subseteq S \text{ and } \text{Neg}(r') \cap \text{Head}(\Delta^t(P)_S) = \emptyset \}.
\]

Then \( S \) is called a preferred answer set of \( P \) iff Head(\( \Delta^\infty(P)_S \)) = S.

**Example 21** Example 20 continued. It is easy to see that under Definition 21, the grounded preferred program \( P_1^* \) of \( P_1 = (\Pi_1, <^{P_1}) \) has the unique preferred answer set:

\[
\{ \text{Bird}(\text{cody}), \text{Bird}(\text{tweety}), \text{Penguin}(\text{tweety}), \text{Flies}(\text{cody}), \text{Cannot\_fly}(\text{tweety}), \\
\text{Tag}_1(\text{cody}), \text{Tag}_2(\text{cody}), \text{Tag}_1(\text{tweety}), \text{Tag}_2(\text{tweety}), \\
\text{cody} = \text{cody}, \text{tweety} = \text{tweety} \}.
\]

On the other hand, the grounded preferred program \( P_3^* \) of \( P_3 = (\Pi_1, <^{P_3}) \) has the unique preferred answer set \( \{ P(a), Q(a), \text{Tag}_1(a), \text{Tag}_2(a), a = a \} \).
The following result shows that the preferred answer sets of each FO preferred answer set program can be precisely computed through its grounded counterpart.

**Theorem 6** Let $\mathcal{P} = (\Pi, <^P)$ be a preferred FO program, $\mathcal{M}$ a structure of $\tau(\Pi)$, and $\mathcal{P}^* = (\text{Ground}(\Pi), M, <^{P^*})$ the grounded preferred answer set program of $(\Pi, <)$ based on $\mathcal{M}$ as defined in Definition 20. Then $\mathcal{M}$ is a preferred answer set of $\mathcal{P}$ iff there is a preferred answer set $S$ of $\mathcal{P}^*$ such that $S \cap \mathcal{M} = \mathcal{M}$.

**Proof:** See Appendix A.2.2 □

### 4.3.2 Semantic Properties

In this subsection, we study several specific properties of the preferred answer set semantics. As we will see, the grounded preferred answer set semantics provides a basis for our investigation. We first define the notion of generating rules. Let $\mathcal{P} = (\Pi, <^P)$ be a grounded preferred program and $S$ a set of propositional atoms. We say that a rule $r \in \Pi$ is a *generating rule* of $S$ if $\text{Pos}(r) \subseteq S$ and $\text{Neg}(r) \cap S = \emptyset$. Now consider $\mathcal{P} = (\Pi, <^P)$ to be a preferred FO program (i.e., $\Pi$ is a FO program) and $\mathcal{M}$ a structure of $\tau(\Pi)$. Then we say that a rule $r \in \Pi$ is a *generating rule* of $\mathcal{M}$ under the assignment $\eta$ if $\mathcal{M} \models \text{Pos}(r)\eta \land \neg\text{Neg}(r)\eta$, where $\text{Pos}(r)$ and $\neg\text{Neg}(r)$ denote the formulas $\beta_1 \land \cdots \land \beta_l$ and $\neg\gamma_1 \land \cdots \land \neg\gamma_m$ respectively.

**Lemma 3** Let $\mathcal{P} = (\Pi, <^P)$ be a grounded preferred program and $S$ an answer set of $\Pi$. Then the following two statements are equivalent:

1. $S$ is a preferred answer set of $\mathcal{P}$;
2. For each rule $r \in \Pi$, $r \in \Delta^\infty(\mathcal{P})_S$ iff $r$ is a generating rule of $S$.

**Proof:** See Appendix A.2.3 □

Now we have the following semantic characterization theorem for preferred FO programs.
CHAPTER 4. PREFERRED FO ANSWER SET PROGRAMS

Theorem 7 Let $\mathcal{P} = (\Pi, <^P)$ be a preferred FO program and a $\tau(\Pi)$-structure $M$ be an answer set of $\Pi$. Then the following two statements are equivalent:

1. $M$ is a preferred answer set of $\mathcal{P}$;
2. For each rule $r \in \Pi$, $(r, \eta) \in \Gamma^\infty(\mathcal{P})_M$ iff $r$ is a generating rule of $M$ under $\eta$.

Proof: For the given $\mathcal{P} = (\Pi, <^P)$ and its preferred answer set $M$, $\mathcal{P}$’s grounded preferred program is denoted as $\mathcal{P}^* = (\text{Ground}(\Pi)_M, <^{P^*})$ as defined in Definition 20. From the proof of Theorem 6, we can then construct a set of propositional atoms $S$ from $M$ such that $S$ is an answer set of $\mathcal{P}^*$ and $S = S \cap M$. Then based on Lemma 3, in order to prove this theorem, it is sufficient to prove the following two results:

Result 1: $(r, \eta) \in \Gamma^\infty(\mathcal{P})_M$ iff $r^* \in \Delta^\infty(\mathcal{P}^*)_S$, were

\[
r^*: \text{Head}(r)\eta \leftarrow \text{Body}(r)\eta, \text{Tag}(x)\eta,
\]

Result 2: $r$ is a generating rule of $M$ under $\eta$ iff $r^*$ is a generating rule of $S$.

From the definitions of $\Gamma^\infty(\mathcal{P})_M, \Delta^\infty(\mathcal{P}^*)_S$ and such that $\mathcal{P}^* = (\text{Ground}(\Pi)_M, <^{P^*})$, and together with Theorem 6, Results 1 and 2 are easily proved. □

Proposition 16 For preferred FO programs $\mathcal{P}_1 = (\Pi, <^{P_1})$ and $\mathcal{P}_2 = (\Pi, <^{P_2})$ where $<^{P_1} \subseteq <^{P_2}$, and a structure $M$ of $\tau(\Pi)$, if $M$ is a preferred answer set of $\mathcal{P}_2$, then $M$ is also a preferred answer set of $\mathcal{P}_1$.

Now we consider the existence of preferred answer sets of a preferred program. From Theorem 7, in order to see whether a structure $M$ is a preferred answer set of a given preferred program $\mathcal{P}$, we need to compute $\Gamma^\infty(\mathcal{P})_M$ and then to check all generating rules against $M$. It is always desirable to discovery some stronger sufficient conditions for the existence of preferred answer sets for which there is no need to undertake the computation of $\Gamma^\infty(\mathcal{P})_M$. The following lemma and theorem are towards this purpose.

Lemma 4 A grounded preferred program $\mathcal{P} = (\Pi, <^P)$ has a preferred answer set if there exist an answer set $S$ of $\Pi$ such that for each $(r_1, r_2) \in <^P$ were $\text{Head}(r_2) \cap (\text{Pos}(r_1) \cup \text{Neg}(r_1)) \neq \emptyset$, $r_2$ is not a generating rule of $S$. 
Proof: See Appendix A.2.4. □

Theorem 8  A preferred FO program $\mathcal{P} = (\Pi, <^\mathcal{P})$ has a preferred answer set if there exist an answer set $\mathcal{M}$ of $\Pi$ such that for each $(r_1, r_2) \in <^\mathcal{P}$ and for all assignments $\eta, \eta'$ of $\mathcal{M}$ satisfying $\text{Head}(r_2)\eta' \cap (\text{Pos}(r_1) \cup \text{Neg}(r_1))\eta \neq \emptyset$, $r_2$ is not a generating rule of $\mathcal{M}$ under $\eta'$.

Proof: We prove the result by showing that the given $\mathcal{M}$ is a preferred answer set of $\mathcal{P}$. For the given preferred FO program $\mathcal{P} = (\Pi, <^\mathcal{P})$ and an answer set $\mathcal{M}$ of $\Pi$, we consider the grounded preferred program $\mathcal{P}^* = (\text{Ground}(\Pi)_\mathcal{M}, <^\mathcal{P})$. From Theorem 6, we know that there is a preferred answer set $S$ of $\mathcal{P}^*$ where $S \cap \mathcal{M} = \mathcal{M}$. We can also show that for each pair of rules $r_1, r_2 \in \Pi$ where $r_1 <^\mathcal{P} r_2$ and assignments $\eta, \eta'$ of $\mathcal{M}$ such that $\text{Head}(r_2)\eta' \cap (\text{Pos}(r_1) \cup \text{Neg}(r_1))\eta \neq \emptyset$, there exist a corresponding pair of rules $r_1^*, r_2^* \in \text{Ground}(\Pi)$ such that $r_1^* <^{\mathcal{P}^*} r_2^*$ and $\text{Head}(r_2^*) \cap (\text{Pos}(r_1^*) \cup \text{Neg}(r_1^*)) \neq \emptyset$, and vice versa. We can further prove that a rule $r \in \Pi$ is not a generating rule of $\mathcal{M}$ under $\eta$ iff there is a corresponding rule $r^*$ in $\text{Ground}(\Pi)$ which is not a generating rule of $S$. Then from Lemma 4, it follows that $S$ is a preferred answer set of $\mathcal{P}^* = (\text{Ground}(\Pi), <^{\mathcal{P}^*})$. Finally, from Theorem 6, $\mathcal{M}$ is also a preferred answer set of $\mathcal{P} = (\Pi, <^\mathcal{P})$. □

It is clear that the sufficient condition represented in Theorem 8 is semantics based in the sense that an answer set (answer sets) of $\Pi$ has (have) to be computed and verified to decide the existence of a preferred answer set. Nevertheless, from Theorem 8, we can actually derive a purely syntactic sufficient condition which can be quite effective in deciding the existence of preferred answer sets. To make our claim precise, we first introduce a useful notion.

A substitution $\theta$ is a tuple of term replacements $(t_1/t_1', \ldots, t_k/t_k')$. By applying a substitution $\theta$ on a set of first-order formulas $X$, we obtain another set of formulas, denoted as $X\theta$, by replacing each term $t$ occurring in formulas of $X$ with $t'$ whenever $t/t' \in \theta$.

Definition 22 Let $P(t)$ be an atom, $X$ a finite set of atoms, $T_1$ and $T_2$ two finite sets of formulas of the forms $t_1 = t_2$ and $t_1 \neq t_2$ where $t$ is a tuple of terms, $t_1$ and $t_2$ are terms. We call $T_1$ and $T_2$ the sets of term binding constraints imposing on $P(t)$ and $X$ respectively. The pair $(P(t), X)$ is called unifiable under $T_1$ and $T_2$ if there exist a substitution $\theta$ such that $P(t) \in X\theta$ and $(X \cup T_2)\theta \cup T_1$ is consistent.
Example 22 Let us consider an atom $P(x,y)$. Set $X = \{P(x',y'), Q(y',z')\}$, and their term constraint sets $T_1 = \{x = y\}$ and $T_2 = \{x' \neq z', y' = c\}$ respectively. It is easy to see that $P(x,y)$ and $X$ are unifiable under $T_1$ and $T_2$ because we can find a substitution $\theta = (x'/x, y'/y)$, such that $(X \cup T_2)\theta = \{P(x,y), Q(y,z'), x \neq z', y = c\}$, and $\{P(x,y), Q(y,z'), x \neq z', y = c\} \cup \{x = y\}$ is consistent.

For a given FO program $\Pi$ and a rule $r \in \Pi$, we specify $r$’s set of term binding constraints as follows:

$$T(r) = \{t_1 = t_2 \mid t_1, t_2 \in Pos(r)\} \cup \{t_1 \neq t_2 \mid t_1, t_2 \in Neg(r)\}.$$ 

We also use $Pos(r)^{-e}$ and $Neg(r)^{-e}$ to denote the sets of all atoms of $Pos(r)$ and $Neg(r)$ except the equality atoms respectively.

**Theorem 9** Let $\mathcal{P} = (\Pi, <^P)$ be a preferred FO program. Then $\mathcal{P}$ has a preferred answer set if $\Pi$ has an answer set and where for each $r_1 <^P r_2$, neither $(\text{Head}(r_2), Pos(r_1)^{-e})$ nor $(\text{Head}(r_2), Neg(r_1)^{-e})$ are unifiable under $T(r_2)$ and $T(r_1)$.

**Proof:** From Definition 22, we see that for any two rules $r, r' \in \Pi$, if neither $(\text{Head}(r_2), Pos(r_1)^{-e})$ nor $(\text{Head}(r_2), Neg(r_1)^{-e})$ are unifiable under $T(r_2)$ and $T(r_1)$, then for any answer set $M$ of $\Pi$ and any two assignments $\eta$ and $\eta'$, $\text{Head}(r_2)\eta \cap (Pos(r_1) \cup Neg(r_1))\eta = \emptyset$. This implies that the condition of Theorem 8 holds. So under such situation, each answer set $M$ of $\Pi$ is actually a preferred answer set of $\mathcal{P} = (\Pi, <^P)$. □

4.4 Logical Characterizations

Since the answer set semantics for FO programs is defined via a SO sentence, it is a natural question to ask whether it is also possible to characterize the progression based semantics for preferred FO programs through a SO sentence. In this section, we will study this issue in detail. Our basic idea is that: we first propose an alternative SO sentence that precisely captures the answer set semantics of FO programs. Our formalization will be different from that of the original one as presented in [FLL11] for the case when restricted to normal logic programs. Instead, our SO formalism will be developed based on the intuition of the
progression based semantics [ZZ10]. Then we extend such SO formalism by taking the preference into account.

To begin with, we first introduce some useful notions. We suppose that from here on, each program \( \Pi \) is presented in a constant normalized form. That is, we assume that each proper atom occurring in a rule only contains a tuple of distinguishable variables. Note that every program can be rewritten into such a normalized form. For instance, if

\[
P(x_1, \cdots, x_{i-1}, c, x_{i+1}, \cdots, x_n)
\]

occurs in some rule \( r \) were \( c \) is a constant, then we simply replace (4.1) by

\[
P(x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_n)
\]

and then add \( x_i = c \) into \( r \)'s positive body. In this way, we can simply write \( x_P \) and \( x_r \) as the tuples of all distinguishable variables occurring in predicate \( P \) and rule \( r \) respectively.

Now consider a program consisting of the single rule

\[
r: S(x, y) \leftarrow E(a, y), x = a.
\]

Then this program can be equivalently rewritten to a normalized form

\[
r: S(x, y) \leftarrow E(z, y), x = a, z = a,
\]

where \( x_r = xyz \). Now let \( x = x_1 \cdots x_n \) and \( y = y_1 \cdots y_n \), then we also write \( x = y \) to denote the formula \( \bigwedge_{1 \leq i \leq n} x_i = y_i \).

For easier presentation and readability of formulas, we further assume that for any two rules \( r_1 \) and \( r_2 \) where \( r_1 \neq r_2 \) (that is, distinct rules), \( x_{r_1} \) and \( x_{r_2} \) are disjoint from each other. That is, the names of the variables in \( r_1 \) do not occur in \( r_2 \) (and vice versa). Note that in cases where we have to refer to the tuples \( x_{r_1} \) and \( x_{r_2} \) in a formula for which they have to be disjoint, we can always relabel the variables in \( x_{r_2} \) without explicitly stating it, if clear from the context.

### 4.4.1 Formulas Representing Generating Rules and Program Completion

Similar to what we introduced in section 4.2, here we present a FO formula to represent the notion of generating rules in terms of a structure. In particular, given a program \( \Pi \) and
CHAPTER 4. PREFERRED FO ANSWER SET PROGRAMS

a rule \( r \in \Pi \) of the form (1.4): \( \alpha \leftarrow \beta_1, \ldots, \beta_l, \text{not} \ \gamma_1, \ldots, \text{not} \ \gamma_m \), we specify a FO formula

\[
\varphi_r^{\text{GEN}}(x_r) = \widehat{\text{Pos}}(r) \land \neg \widehat{\text{Neg}}(r).
\]  

Clearly, for a given \( \tau(\Pi) \)-structure \( M \), \( M \models \varphi_r^{\text{GEN}}(x_\eta) \) iff \( \text{Pos}(r) \eta \subseteq M \) and \( \text{Neg}(r) \eta \cap M = \emptyset \).

**Example 23** Consider the program \( \Pi_5 \) consisting of the following rules:

\[
\begin{align*}
  r_1 & : S(x_1, y_1) \leftarrow E(z_1, y_1), x_1 = a, z_1 = a \\
  r_2 & : S(x_2, z_2) \leftarrow S(x_2, y_2), E(y_2, z_2'), z_2 = a, z_2' = b.
\end{align*}
\]

Then we have:

\[
\begin{align*}
  \varphi_{r_1}^{\text{GEN}}(x_1, y_1, z_1) & = E(z_1, y_1) \land x_1 = a \land z_1 = a \\
  \varphi_{r_2}^{\text{GEN}}(x_2, y_2, z_2, z_2') & = S(x_2, y_2) \land E(y_2, z_2') \land z_2 = a \land z_2' = b.
\end{align*}
\]

Now let \( \Pi \) be a FO program. Then we define the FO sentence \( \varphi_{\Pi}^{\text{COMP}} \) to be the completion of \( \Pi \),

\[
\varphi_{\Pi}^{\text{COMP}} = \bigwedge_{P \in P_{\text{int}}(\Pi)} \forall x_P (P(x_P) \Leftrightarrow \bigvee_{\begin{subarray}{c} r \in \Pi, \\
  \text{Head}(r) = P(y_P) \end{subarray}} \exists x_r (\varphi_r^{\text{GEN}}(x_r) \land x_P = y_P)),
\]  

where we assume that \( x_P \) is a tuple of distinguishable variables disjoint from \( y_P \).

**4.4.2 Well-Orderings on Generating Rules in Terms of Structures**

Given a FO program \( \Pi \) and a \( \tau(\Pi) \)-structure \( M \), let \( \Gamma(\Pi)_M \)\(^5\) denote the set:

\[
\{(r, \eta) \mid r \in \Pi, \text{Pos}(r) \eta \subseteq M \text{ and } \text{Neg}(r) \eta \cap M = \emptyset\},
\]

\(^5\)Note that \( \Gamma(\Pi)_M \) differs from \( \Gamma^t(\mathcal{P})_M \) in that \( \Gamma(\Pi)_M \) does not represent a progression stage of a preferred program \( \mathcal{P} \).
i.e., those generating rules under the structure $\mathcal{M}$. Then a well-order on $\Gamma(\Pi)_\mathcal{M}$ is a structure $\mathcal{W} = (\Gamma(\Pi)_\mathcal{M}, <^W)$ with domain $\Gamma(\Pi)_\mathcal{M}$ and binary relations $<^W$ on $\Gamma(\Pi)_\mathcal{M}$ that satisfies the following properties:

1. $x, y \in \Gamma(\Pi)_\mathcal{M}$ and $x \neq y$ implies $x <^W y$ or $y <^W x$ (totality);
2. $x <^W y$ and $y <^W z$ implies $x <^W z$ (transitivity);
3. $x <^W y$ implies $y \not<^W x$ (asymmetry);
4. and lastly, the SO axiom:

$$(\forall S \subseteq \Gamma(\Pi)_\mathcal{M})(S \neq \emptyset \rightarrow (\exists x \in S)((\forall y \in S)(x \neq y \rightarrow x <^W y))),$$

which expresses that every non-empty subset $S$ of $\Gamma(\Pi)_\mathcal{M}$ has a least element.

Note that when $\Gamma(\Pi)_\mathcal{M}$ is finite, any strict-total order of $\Gamma(\Pi)_\mathcal{M}$ is trivially a well-order, while this is not the case when $\Gamma(\Pi)_\mathcal{M}$ is infinite. In fact, since totality, transitivity and asymmetry makes up a strict-total order, then when only considering finite structures, we can drop the SO axiom corresponding to the least element property of each non-empty subset.

### 4.4.3 Formalizing Progression

We now propose a SO sentence that will simulate the progression semantics we defined earlier but without first taking the preferences on rules into account.

**Definition 23** Given a FO program $\Pi$, we define a SO formula $\varphi^{\text{PRO}}_\Pi(\rightarrow^S, S)$ as follows (here
"PRO" stands for progression\(^6\):

\[
\bigwedge_{r_1, r_2, r_3 \in \Pi} \forall x_{r_1} x_{r_2} x_{r_3} (\lnot \lnot <_{r_1 r_2} (x_{r_1}, x_{r_2}) \land \lnot \lnot <_{r_2 r_3} (x_{r_2}, x_{r_3}) \rightarrow \lnot <_{r_1 r_3} (x_{r_1}, x_{r_3})) \quad (4.4)
\]

\[
\bigwedge_{r_1, r_2 \in \Pi} \forall x_{r_1} x_{r_2} (\lnot <_{r_1 r_2} (x_{r_1}, x_{r_2}) \rightarrow \lnot <_{r_2 r_1} (x_{r_2}, x_{r_1})) \quad (4.5)
\]

\[
\bigwedge_{r_1, r_2 \in \Pi} \forall x_{r_1} x_{r_2} (\lnot <_{r_1 r_2} (x_{r_1}, x_{r_2}) \rightarrow (\varphi_{r_1}^{\text{GEN}} (x_{r_1}) \land \varphi_{r_2}^{\text{GEN}} (x_{r_2}))) \quad (4.6)
\]

\[
\bigwedge_{r \in \Pi} \forall x_r (\varphi_r^{\text{GEN}} (x_r) \rightarrow \varphi_r^{\text{SUP}} (\lnot <, x_r)) \quad (4.7)
\]

\[
\varphi_{\Pi}^{\text{WELLOR}} (\lnot <, S),
\]

where:

- for each \(r_1, r_2 \in \Pi\), the symbol \(\lnot <_{r_1 r_2}\) is a predicate variable of arity \(|x_{r_1}| + |x_{r_2}|\); \(^7\)

- \(\lnot <\) denotes the tuple of distinguishable predicate variables of the set \(\{\lnot <_{r_1 r_2} | r_1, r_2 \in \Pi\}\);

- \(\varphi_r^{\text{SUP}} (\lnot <, x_r)\) denotes the following formula (here “SUP” stands for support):

\[
\bigwedge_{P(x_P) \in \text{Pos}(r), \quad P \in \text{Pos}(\Pi)} \bigvee_{r' \in \Pi, \quad \text{Head}(r') = P(y_P)} \exists x_{r'} (\lnot <_{r' r'} (x_{r'}, x_r) \land x_P = y_P) \quad (4.9)
\]

(where in the case that \(r' = r\), we simply assume a relabeling of \(x_{r'}\) such that \(x_{r'}\) will be disjoint from \(x_r\));

\(^6\)In the following Formulas (4.4), (4.5) and (4.6), in the case that \(r_1 = r_2\), \(r_1 = r_3\), or \(r_2 = r_3\), we assume a relabling of \(x_{r_1}, x_{r_2},\) or \(x_{r_3}\) (however appropriate) such that they will now be disjoint from each other.

\(^7\)For clarity, we denote a predicate variable \(P\) with an accent \(\bar{P}\) to remind of the fact that we will be quantifying over these predicates.
• $\varphi_\Pi^{\text{WellOr}}(\vec{\tau}, \mathbf{S})$ denotes the following formula (here "WellOr" stands for well-ordered):

$$
\bigwedge_{r \in \Pi} \forall x_r (\vec{S}_r(x_r) \rightarrow \varphi_r^{\text{Gen}}(x_r)) \land \left( \bigvee_{r \in \Pi} \exists x_r \vec{S}_r(x_r) \rightarrow \varphi_r^{\text{Sup}}(\vec{\tau}, x_r) \right)
$$

$$
\bigwedge_{r \in \Pi} \forall x_r (\vec{S}_r(x_r) \rightarrow \varphi_r^{\text{Sup}}(\vec{\tau}, x_r))
$$

\[ (4.10) \]

where $\mathbf{S}$ denotes the tuple of distinguishable predicate variables of the set $\{ \vec{S}_r \mid r \in \Pi \}$ such that for each $r \in \Pi$, the arity of $\vec{S}_r$ is $|x_r|$, and $y_r$ (in the consequent above) is a relabeling of the distinct variables of $x_r$ such that $y_r$ is now disjoint from $x_r$.

Let us take a closer look at Definition 23. Basically, formula $\varphi_\Pi^{\text{Pro}}(\vec{\tau}, \mathbf{S})$ imposes a progression-like order on the set of generating rules with respect to a structure $\mathcal{M}$:

$$
\Gamma(\Pi, \mathcal{M}) = \{(r, \eta) \mid r \in \Pi, \text{Pos}(r) \eta \subseteq \mathcal{M} \text{ and } \text{Neg}(r) \eta \cap \mathcal{M} = \emptyset \} \text{ for an assignment } \eta \text{ of } \mathcal{M},
$$

which eventually establishes a correspondence to the sequence of progression sets $\Gamma^0(\Pi, \mathcal{M}), \Gamma^1(\Pi, \mathcal{M}) \Gamma^2(\Pi, \mathcal{M}), \ldots$ as we defined in Definition 18. We should emphasize that at this stage, no preferences among rules are considered.

In particular, Formulas (4.4) and (4.5) expresses the transitive and asymmetric properties respectively, while Formula (4.6) expresses the condition that the well-order only involves those of generating rules. Moreover, Formula (4.7) expresses that if some rule is a generating rule (with respect to certain structure and associated assignment), then it must be supported by rules generated in some earlier stages (i.e., $\varphi_r^{\text{Sup}}(\vec{\tau}, x_r)$). Finally, Formula (4.8) enforces a well-order (i.e., $\varphi_\Pi^{\text{WellOr}}(\vec{\tau}, S)$).

Specifically, (4.8) is fulfilled by Formula (4.10), which encodes that each non-empty subset of a generating rule has a least element. Indeed, the formula

$$
\bigwedge_{r \in \Pi} \forall x_r (\vec{S}_r(x_r) \rightarrow \varphi_r^{\text{Gen}}(x_r))
$$
encodes that each extent of each $\tilde{S}_r$ for $r \in \Pi$ are only those of generating rules, and

$$\bigvee_{r \in \Pi} \exists x_r \tilde{S}_r(x_r)$$

encodes that at least one of these subsets are non-empty, and while the consequence

$$\bigvee_{r' \in \Pi} \exists x_{r'}(\tilde{S}_{r'}(x_{r'}) \wedge \forall y_{r'}(\tilde{S}_{r'}(y_{r'}) \wedge y_{r'} \neq x_{r'} \rightarrow \tilde{<}_{r'}(x_{r'}, y_{r'})))$$

$$\wedge \bigwedge_{r'' \in \Pi, r'' \neq r'} \forall x_{r''} (\tilde{S}_{r''}(x_{r''}) \rightarrow \tilde{<}_{r''}(x_{r''}, x_{r''})))$$

encodes the existence of the least element.

**Proposition 17** The SO formula $\varphi_{\Pi}^{\text{PRO}}(\tilde{\prec}, S)$ is of length $O(n^3 + mn^2)$ where $m = |\text{ATOMS}(\Pi)|$ (i.e., all the atoms occurring in $\Pi$) and $n = |\Pi|$.

**Proof:** Formula (4.4) is of length $O(n^3)$, (4.6) is of length $O(n^2)$, (4.7) is of length $O(mn^2)$ and (4.8) is of length $O(n^2)$. $\square$

**Example 24** Consider the well-known program $\Pi_6$ that computes the transitive closure of a binary relation $E$:

$$r_1 : T(x_1, y_1) \leftarrow E(x_1, y_1)$$
$$r_2 : T(x_2, z_2) \leftarrow T(x_2, y_2), E(y_2, z_2).$$

Then:

$$\varphi_{\Pi_6}^{\text{COMP}} = \forall xy(T(x, y) \leftrightarrow \exists x_1 y_1 (\varphi_{r_1}^{\text{GEN}}(x_1, y_1) \wedge x = x_1 \wedge y = y_1)$$
$$\vee \exists x_2 y_2 z_2 (\varphi_{r_2}^{\text{GEN}}(x_2, y_2, z_2) \wedge x = x_2 \wedge y = z_2)),$$

where:

$$\varphi_{r_1}^{\text{GEN}}(x_1, y_1) = E(x_1, y_1),$$
$$\varphi_{r_2}^{\text{GEN}}(x_2, y_2, z_2) = T(x_2, y_2) \wedge E(y_2, z_2).$$
In particular, we also have that:
\[
\varphi_{\pi_1}^{\text{SUP}}(\vec{r}, x_1, y_1) = E(x_1, y_1),
\]
\[
\varphi_{\pi_2}^{\text{SUP}}(\vec{r}, x_2, y_2, z_2) = \exists x_3 y_3 z_3 \left( \forall r_{\pi_2}(x_3, y_3, z_3, x_2, y_2, z_2) \land x_2 = x_3 \land y_2 = z_3 \right) \\
\lor \exists x_1 y_1 \left( \forall r_{\pi_1}(x_1, y_1, x_2, y_2, z_2) \land x_2 = x_1 \land y_2 = y_1 \right),
\]
where the tuple \( x_3 y_3 z_3 \) is a disjoint relabeling of \( x_2 y_2 z_2 \) and \( \vec{r} \) is the tuple \( \vec{r}_{\pi_1} \vec{r}_{\pi_2} \vec{r}_{\pi_1} \vec{r}_{\pi_2} \vec{r}_{\pi_1} \vec{r}_{\pi_2} \) of predicate variables.

About the formula \( \varphi_{\Pi_6}^{\text{WELL}}(\vec{r}, S) \), we further have:
\[
\forall x_1 y_1 (\vec{S}_{\pi_1}(x_1, y_1) \rightarrow \varphi_{\pi_1}^{\text{GEN}}(x_1, y_1)) \land \forall x_2 y_2 z_2 (\vec{S}_{\pi_2}(x_2, y_2, z_2) \rightarrow \varphi_{\pi_1}^{\text{GEN}}(x_2, y_2, z_2)) \\
\land (\exists x_1 y_1 \vec{S}_{\pi_1}(x_1, y_1) \lor \exists x_2 y_2 z_2 \vec{S}_{\pi_2}(x_2, y_2, z_2)) \rightarrow \\
(\exists x_1 y_1 (\vec{S}_{\pi_1}(x_1, y_1) \land \forall x y (\vec{S}_{\pi_1}(x, y) \land (x_1 \neq x \lor y_1 \neq y)) \rightarrow \vec{r}_{\pi_1}(x_1, y_1, x, y)) \\
\land \forall x_2 y_2 z_2 (\vec{S}_{\pi_2}(x_2, y_2, z_2) \rightarrow \vec{r}_{\pi_1}(x_1, y_1, x_2, y_2, z_2)) \\
\lor \exists x_2 y_2 z_2 (\vec{S}_{\pi_2}(x_2, y_2, z_2) \land \forall x y z (\vec{S}_{\pi_2}(x, y, z) \land (x_2 \neq x \lor y_2 \neq y \lor z_2 \neq z) \rightarrow \\
\vec{r}_{\pi_1}(x_2, y_2, z_2, x, y, z)) \\
\land \forall x_1 y_1 (\vec{S}_{\pi_1}(x_1, y_1) \rightarrow \vec{r}_{\pi_1}(x_1, y_1)))
\]
where \( S \) is the tuple \( \vec{S}_{\pi_1} \vec{S}_{\pi_2} \) of predicate variables. Then \( \forall S \varphi_{\Pi_6}^{\text{WELL}}(\vec{r}, S) \) expresses that each non-empty subset of \( \Gamma(\Pi)_{\mathcal{M}} \) possesses a least element as induced by the relations of the predicates \( \vec{r}_{\pi_1} \vec{r}_{\pi_2} \vec{r}_{\pi_1} \vec{r}_{\pi_2} \vec{r}_{\pi_1} \vec{r}_{\pi_2} \). Note that the non-empty subsets of \( \Gamma(\Pi)_{\mathcal{M}} \) are implicitly encoded via the universally quantification of the predicate variables \( \vec{S}_{\pi_1} \) and \( \vec{S}_{\pi_2} \).

\[\square\]

**Theorem 10** Given a FO program \( \Pi \) and a \( \tau(\Pi) \)-structure \( \mathcal{M} \), \( \mathcal{M} \) is an answer set of \( \Pi \) iff \( \mathcal{M} \models \exists \vec{r} \forall S \varphi_{\Pi}(\vec{r}, S) \), where
\[
\varphi_{\Pi}(\vec{r}, S) = \varphi_{\Pi}^{\text{PRO}}(\vec{r}, S) \land \varphi_{\Pi}^{\text{COMP}}.
\]

**Proof:** See Appendix A.2.5. \[\square\]
4.4.4 Incorporating Preference

We have shown that the SO sentence $\exists \neg \rightarrow S^\Pi(\neg, S)$ proposed in Theorem 10 has precisely captured the answer set semantics for FO programs. In this subsection, we further extend this formula by embedding preferences among the rules of a preferred FO program.

We first define the following formula.

**Definition 24** Given a preferred FO program $P = (\Pi, <^P)$, we define the formula $\varphi^{\text{PREF}}(\neg)$ as follows (here “PREF” stands for preference):

$$\bigwedge_{r \in \Pi} \forall x_r (\varphi^\text{GEN}(x_r) \rightarrow \bigwedge_{r' <^P r} \forall x_{r'}((\varphi^\text{GEN}(x_{r'}) \rightarrow \overline{\sim}(<^r, x_{r'}, x_r)))$$

$$\land (\neg \varphi^\text{GEN}(x_{r'}) \rightarrow (\varphi^{\neg \text{POS}}(x_{r'}) \lor \varphi^{\text{DEF}}(\neg, x_{r'}, x_r))))),$$

where:

- $\varphi^{\neg \text{POS}}(x_{r'})$ denotes the formula (“$\neg \text{POS}$” stands for positive body not satisfied):

$$\bigvee_{P(x_p) \in Pos(r')} \neg P(x_p);$$

(4.12)

- $\varphi^{\text{DEF}}(\neg, x_{r'}, x_r) \ (\text{here “DEF” stands for defeated})$ denotes the formula

$$(\bigvee_{P(x_p) \in Neg(r'}, P \notin P_{\text{int}}(\Pi)} P(x_p)) \lor (\bigvee_{P(x_p) \in Neg(r'}, P \in P_{\text{int}}(\Pi)} \bigvee_{r'' \in \Pi, \text{Head}(r'') = P(y_p)} \exists x_{r''}(<^r, x_{r''}, x_r) \land x_p = y_p).$$

(4.13)

Intuitively speaking, Formula (4.11) encodes that if $r$ is a generating rule (under some structure and associated assignment), then for all rules $r'$ that is more preferred than $r$, and such that $r'$ is also a generating rule (under the structure and associated assignment), then we require that $r'$ should had already been derived earlier than $r$ in the progression stages. On the other hand, in the case that $r'$ is not a generating rule, then either the positive body of $r'$ is not satisfied (i.e., by (4.12)), or that it is defeated by some other rule $r''$ such that $r''$ is already derived earlier than $r$ in the progression stages (i.e., by (4.13)), which indicates that the head of $r''$ occurs in the negative body of $r'$. 
The following theorem now reveals the main result of this chapter.

**Theorem 11** Let $\mathcal{P} = (\Pi, <^\mathcal{P})$ be preferred FO program and $\mathcal{M}$ a $\tau(\Pi)$-structure. Then $\mathcal{M}$ is a preferred answer set of $\mathcal{P}$ iff $\mathcal{M} |\models \exists \overrightarrow{\mathbf{S}} \phi_P(\overrightarrow{\mathbf{S}}, \mathbf{S})$, where

$$\phi_P(\overrightarrow{\mathbf{S}}, \mathbf{S}) = \phi_P^{\text{REF}}(\overrightarrow{\mathbf{S}}) \land \phi_P^{\text{PRO}}(\overrightarrow{\mathbf{S}}, \mathbf{S}) \land \phi_P^{\text{COMP}}.$$

**Proof:** See Appendix A.2.6. □

Thus, by Theorem 11, we have that the fixpoints of the progression operator $\lambda_{M^0}(\Gamma^\infty(\mathcal{P}), \mathcal{M})$ (i.e., $\lambda_{M^0}(\Gamma^\infty(\mathcal{P}), \mathcal{M}) = \mathcal{M}$, corresponding to $\mathcal{M}$ a preferred answer set) are exactly the models of the SO sentence $\exists \overrightarrow{\mathbf{S}} \phi_P(\overrightarrow{\mathbf{S}}, \mathbf{S})$. In fact, when only considering finite structure, we even have that the universal SO sentence $\exists \overrightarrow{\mathbf{S}} \phi_P(\overrightarrow{\mathbf{S}}, \mathbf{S})$ reduces down to an existential SO sentence, as revealed in the following proposition.

**Proposition 18** On finite structures, every preferred FO program is precisely captured by an existential SO (ESO) formula.

**Proof:** We obtain an ESO formula $\exists \overrightarrow{\mathbf{S}} \psi_P(\overrightarrow{\mathbf{S}})$ from $\exists \overrightarrow{\mathbf{S}} \phi_P(\overrightarrow{\mathbf{S}}, \mathbf{S})$ by substituting the SO formula $\phi_P^{\text{TOTALOR}}(\overrightarrow{\mathbf{S}}, \mathbf{S})$ for $\phi_P^{\text{WELLOR}}(\overrightarrow{\mathbf{S}}, \mathbf{S})$ in $\phi_P(\overrightarrow{\mathbf{S}}, \mathbf{S})$ such that $\phi_P^{\text{TOTALOR}}(\overrightarrow{\mathbf{S}}, \mathbf{S})$ is given by:

$$\bigwedge_{r \in \Pi} \forall x_r, y_r (\phi_r^{\text{GEN}}(x_r) \land \phi_r^{\text{GEN}}(y_r) \land x_r \neq y_r \rightarrow (\overleftarrow{r}(x_r, y_r) \lor \overleftarrow{r}(y_r, x_r)))$$

$$\land \bigwedge_{r_1, r_2 \in \Pi, \ r_1 \neq r_2} \forall x_{r_1}, y_{r_1} (\phi_{r_1}^{\text{GEN}}(x_{r_1}) \land \phi_{r_2}^{\text{GEN}}(x_{r_2}) \rightarrow (\overleftarrow{r_1}(x_{r_1}, x_{r_2}) \lor \overleftarrow{r_2}(x_{r_2}, x_{r_1}))),$$

which, in conjunction with Formulas (4.4), (4.5) and (4.6), expresses a strict-total order on $\Gamma(\Pi)_M$. Then since well-orders are strict total-orders on finite structures, the result follows. □

**Example 25** On finite structures, let us consider once again the preferred program $\mathcal{P}_2 =$

---

\(^8\text{Where "TOTALOR" stands for total-order.}\)
\( (\Pi_2, \prec^{P_2}) \) of Example 19: \(^9\)

\[
\begin{align*}
    r_1 : & \quad P(x_1) \leftarrow Q(x_1) \\
    r_2 : & \quad Q(x_2) \leftarrow \\
    r_1 & \prec^{P_2} r_2.
\end{align*}
\]

We will now use Theorem 11 and Proposition 18 to show that \( \Pi_2 \) does not have a preferred answer set. Thus, given \( \Pi_2 \), we have \( \varphi_{\Pi_2}^{\text{PREF}}(\prec) \) to be:

\[
\begin{align*}
    \forall x_1 (\varphi_{r_1}^{\text{GEN}}(x_1) \rightarrow \top) \\
    \land \forall x_2 (\varphi_{r_2}^{\text{GEN}}(x_2) \rightarrow \forall x_1 ((\varphi_{r_1}^{\text{GEN}}(x_1) \rightarrow \prec^{r_1 r_2}(x_1, x_2))) \\
    \land (\neg \varphi_{r_1}^{\text{GEN}}(x_1) \rightarrow (\varphi_{r_1}^{\text{POS}}(x_1) \lor \varphi_{r_2 r_1}^{\text{DEF}}(\prec, x_2, x_1))))). \quad (4.14)
\end{align*}
\]

Then from \( \varphi_{\Pi_2}^{\text{PRO}}(\prec) \), we further have:

\[
\begin{align*}
    \forall x_1 (\varphi_{r_1}^{\text{GEN}}(x_1) \rightarrow \exists x_2 (\prec^{r_2} r_1(x_2, x_1) \land x_1 = x_2)), \quad (4.15)
\end{align*}
\]

i.e., the support for \( r_1 \). Then since we also have

\[
\forall x_1 x_2 (\prec^{r_1 r_2}(x_1, x_2) \rightarrow \varphi_{r_1}^{\text{GEN}}(x_1) \land \varphi_{r_2}^{\text{GEN}}(x_2)) \quad (4.16)
\]

by \( \varphi_{\Pi_2}^{\text{PRO}}(\prec) \) (i.e., via (4.6)), then (4.15) is equivalent to:

\[
\forall x_1 (\varphi_{r_1}^{\text{GEN}}(x_1) \rightarrow \exists x_2 (\prec^{r_2 r_1}(x_2, x_1) \land \varphi_{r_2}^{\text{GEN}}(x_2) \land x_1 = x_2)). \quad (4.17)
\]

Then from (4.14), we further get that (4.17) is equivalent to:

\[
\begin{align*}
    \forall x_1 (\varphi_{r_1}^{\text{GEN}}(x_1) & \rightarrow \exists x_2 (\prec^{r_2 r_1}(x_2, x_1) \land \varphi_{r_2}^{\text{GEN}}(x_2) \land \prec^{r_1 r_2}(x_1, x_2) \land x_1 = x_2)\).
\end{align*}
\]

\(^9\)Note that for our purpose here, we relabelled the variables in rules \( r_1 \) and \( r_2 \) to make them disjoint.
CHAPTER 4. PREFERRED FO ANSWER SET PROGRAMS

i.e., simply added the atom, \( \neg r_1 r_2(x_1, x_2) \), to the consequent. Then by (4.4), i.e., the transitive property, we further have that (4.18) is equivalent to

\[
\forall x_1 (\varphi_{r_1}^{\text{GEN}}(x_1) \rightarrow \exists x_2 (\neg r_2 r_1(x_2, x_1) \land \varphi_{r_2}^{\text{GEN}}(x_2) \land \neg r_1 r_2(x_1, x_2) \land \neg r_2 r_2(x_2, x_2) \land x_1 = x_2)),
\]

(4.19)
i.e., added the atom, \( \neg r_2 r_2(x_2, x_2) \), to the consequent. Then by the asymmetry as expressed by Formula (4.5) (i.e., which implies we cannot have \( \neg r_2 r_2(x_2, x_2) \)), we further have that

\[
\forall x_1 (\varphi_{r_1}^{\text{GEN}}(x_1) \rightarrow \bot) \equiv \forall x_1 \neg \varphi_{r_1}^{\text{GEN}}(x_1).
\]

(4.20)

Hence, since \( \varphi_{r_1}^{\text{GEN}}(x_1) = Q(x_1) \), then this implies that:

\[
\forall x_1 \neg Q(x_1).
\]

Therefore, since we also have \( \forall x_1 Q(x_1) \) by \( \varphi_{\Pi_2}^{\text{COMP}} \), then the sentence \( \exists \neg \psi_p(\bar{z}) \) is inconsistent (or unsatisfiable), which corresponds to the preferred program \( p_2 \) not having a preferred answer set as mentioned in Example 19.

□

The following proposition now reveals that when the preference relations among the rules in the preferred program is empty, then the models of the sentence \( \exists \neg \forall S \varphi_p(\bar{z}, S) \) corresponds exactly to the answer sets of the program.

**Proposition 19** For a preferred FO program \( P = (\Pi, <^P) \) where \( <^P = \emptyset \) and a \( \tau(\Pi) \)-structure \( M \models \exists \neg \forall S \varphi_p(\bar{z}, S) \) iff \( M \) is an answer set of \( \Pi \).

**Proof:** When \( <^P = \emptyset \), the sentence \( \exists \neg \forall S \varphi_p(\bar{z}, S) \) reduces to \( \exists \neg \forall S \varphi_{\Pi}(\bar{z}, S) \), since \( \varphi_{p}^{\text{DEF}}(\bar{z}) \) as given by:

\[
\bigwedge_{r \in \Pi} \forall x_r (\varphi_r^{\text{GEN}}(x_r) \rightarrow \bigwedge_{r' <^P r} \forall x_{r'} ((\varphi_{r'}^{\text{GEN}}(x_{r'}) \rightarrow \neg r_{r'}(x_{r'}, x_r)) \\
\land (\neg \varphi_{r'}^{\text{DEF}}(x_{r'}) \rightarrow (\varphi_{r'}^{\text{POS}}(x_{r'}) \lor \varphi_{r'}^{\text{DEF}}(\bar{z}, x_r, x_{r'}))))),
\]

becomes

\[
\bigwedge_{r \in \Pi} \forall x_r (\varphi_r^{\text{GEN}}(x_r) \rightarrow \top) \equiv \top.
\]
Therefore, along with Theorem 11, we have by Proposition 19 that all FO preferred programs can be precisely captured by SO sentences on arbitrary structures. In addition, by Proposition 18, when only finite structures are considered, we have that all FO preferred programs can be precisely captured by ESO sentences.

4.5 Related Work and Further Remarks

Handling preferences through propositional answer set programming has been studied by many researchers. Among the different proposals, the approach developed by Schaub and Wang [SW03] provided a unified preference framework that captures several important approaches including Brewka and Eiter’s and Delgrande’s et al’s [BE99, DST03].

Our framework generalizes Zhang and Zhou’s progression semantics for FO programs [ZZ10] to incorporate preference, while it also extends Schaub and Wang’s order preservation semantics for preferred propositional programs to the FO case. As mentioned in Sections 1.2.5 and 4.3.1, a naive grounding method extending previous preferred propositional program approaches seems not to work for defining a preferred FO program framework since simply grounding a preferred program \( \mathcal{P} = (\Pi, <^P) \) with \( r_1 <^P r_2 \) for some rules \( r_1, r_2 \in \Pi \) could render \( r_1 \) and \( r_2 \) to collapse to the same rule and hence lose the original preference intuition presented in the program.

Another important difference between preferred FO program and propositional program is the issue of dealing with infinite domains. In our proposed framework, we allow both finite and infinite domains, while only finite domains are possibly represented in preferred propositional program formulations. Although infinite domains are not often considered in practice, it is critical to gain a clear picture of the theoretic foundations of preferred FO program.

Other related work regarding preferred nonmonotonic reasoning on the FO level is that of prioritized circumscription. Prioritized circumscription [Lif85, McC86] is an alternative way of introducing circumscription by means of an ordering on tuples of predicates satisfying an axiom (can be any arbitrary first-order sentence). Hence, prioritized circumscription differs from ours in that we do not relate to any ordering of the tuples of predicates, but
rather, we relate directly to “ordering on formulas,” i.e., as represented by the universal closures of rules. Thus, our approach is more of a prescriptive analog on a FO level.

Another work with respect to FO nonmonotonic reasoning is the ordered completion of FO programs as discussed in Chapter 2. Though the logical formulation in here of preferred FO programs corresponds to a SO characterization that reduces down to an ESO formula on finite structures, the way that the underlying ESO formula is encoded here is quite different from that of the ordered completion. In fact, while ordered completion expresses the order of the derivation paths of predicates, the ESO formula encodes the derivation order on the rules itself, thus giving us the ability to express the notion of preference among the rules.
Chapter 5

Summary of Contributions and Future Work

5.1 Ordered Completion

One of the main contributions of this thesis is the introduction of the notion of ordered completion that exactly captures the answer set semantics of first-order normal answer set programs (see Chapter 2). As is well known in the literatures, the Clark’s completion of a FO (normal) ASP $\Pi$ is a classical FO sentence of the following form:

$$\bigwedge_{P \in \mathcal{P}_{int}(\Pi)} \forall x (P(x) \leftrightarrow \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y \overline{\text{Body}(r)}),$$

and where it is also well known that the Clark’s completion is too weak in the sense that not all models of it are answer sets of the underlying program. The ordered completion strengthens the Clark’s completion by introducing the so-called comparison predicates, that keep track of the derivation order of the intensional relations.

Semantically, the Clark’s completion of a predicate $P$ can be equivalently written into
two parts:

\[ \forall x ( \forall r \in \Pi, \text{Head}(r) = P(x) \rightarrow \exists y \text{Body}(r) \rightarrow P(x) ) \]  

(5.1)

\[ \land \forall x ( P(x) \rightarrow \exists r \in \Pi, \text{Body}(r), \exists y \text{Body}(r), \land \text{Pos}(r) < P(x)) \]  

(5.2)

where (5.1) corresponds to the universal closure of the predicate \( P \), and (5.2) to the necessary condition of an extent of \( P \). The so-called modified completion for the predicate \( P \) is obtained from the above form of the traditional predicate completion by replacing (5.2) with the following formula:

\[ \forall x ( P(x) \rightarrow \exists r \in \Pi, \text{Body}(r), \land \text{Pos}(r) < P(x)) \]

which simply enforces \( \text{Body}(r) \) with \( \text{Pos}(r) < P(x) \), where \( \text{Pos}(r) < P(x) \) denotes the comparison assertions

\[ \land \left( \leq_{QP}(y,x) \land \neg \leq_{PQ}(x,y) \right) \]

which intuitively mean that \( Q(y) \) is used to establish \( P(x) \) (i.e., the head of the rule) and not the other way around. From this, with \( MComp(\Pi) \) denoting all the conjunctions of all the modified completions of the intensional predicates of \( \Pi \), the ordered completion \( OC(\Pi) \) is then simply the conjunction of the two sentences \( MComp(\Pi) \land Trans(\Pi) \), where \( Trans(\Pi) \) stands for the formula

\[ \land \left( \forall x \forall y \forall z ( \leq_{PR}(x,z) \rightarrow \leq_{QR}(y,z) \rightarrow \leq_{PQ}(x,y)) \right) \]

which further encodes that the comparison relations satisfy a notion of “transitivity.” Then, by Theorem 2, we have that the models of \( OC(\Pi) \) corresponds exactly to the answer sets of \( \Pi \) on finite structures. We then extended this result to also include other important ASP constructs such as constraints and choice rules (see Propositions 4 and 5).

This result can now be summarized as follows:
Table 5.1: From normal ASP to FO logic

<table>
<thead>
<tr>
<th>Structures</th>
<th>New Predicates</th>
<th>Resulting Theory</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary</td>
<td>Allowed</td>
<td>No restriction</td>
<td>Does not exist</td>
</tr>
<tr>
<td>Finite</td>
<td>Not Allowed</td>
<td>Finite</td>
<td>Does not exist</td>
</tr>
<tr>
<td>Finite</td>
<td>Not Allowed</td>
<td>No restriction</td>
<td>Loop Formula</td>
</tr>
<tr>
<td>Finite</td>
<td>Allowed</td>
<td>Finite</td>
<td>Ordered Completion</td>
</tr>
</tbody>
</table>

For first-order normal answer set programs on finite structures,

$$\text{Answer Set} = \text{Clark’s Completion + Derivation Order}$$

$$= \text{Ordered Completion}.$$  

This seems to be a very tight result. First of all, as we have seen, this result cannot be extended to disjunctive answer set programs unless NP = coNP (see Proposition 7). For normal answer set programs, with this result, we now have a rather complete picture of mappings from logic programs to FO logic which is summarized by Table 5.1.

The significance of our ordered completion can be seen from both a theoretical and a practical point of view. To the best of our knowledge, it provides for the first time a translation from first-order normal answer programs under the stable model semantics to first-order sentences.

### 5.1.1 Incorporating Aggregates

Aggregates form one of the most important building blocks of ASP. Aggregates are currently a hot topic in ASP due to their wide range of practical applications. Another of the main contributions of this thesis is the extension of the ordered completion to include FO ASP programs with aggregate constructs.

In this work, we considered first-order aggregate atoms $\delta$ of the following form:

$$\text{OP}\langle v : \exists w \text{Body}(\delta) \rangle \leq N,$$

(5.3)
such that $\text{Body}(\delta)$ is of the form

$$Q_1(y_1), \ldots, Q_s(y_s), \not R_1(z_1), \ldots, \not R_t(z_t),$$

and where $\text{OP}$ can either be one of the aggregate functions $\text{CARD}$, $\text{SUM}$, $\text{PROD}$, $\text{MIN}$, and $\text{MAX}$ defined for the integers. However, some of these aggregates are non-monotone in general, and where we know that on the propositional level, programs with non-monotone aggregates are at least as complex as disjunctive programs [Fer11]. It is for these reason that it is unlikely that ordered completion can be extended to include arbitrary aggregates in the first-order case since, as was shown in [ALZZ12], ordered completion cannot be extended to disjunctive programs as they are not in the same complexity level. We addressed this problem by only considering the so-called monotone and anti-monotone aggregates. More precisely, we achieved this by restricting some of the domains of the aggregate functions. In particular, since $\text{CARD}$, $\text{MIN}$, and $\text{MAX}$ are already (anti)monotone on the integers, we only had to restrict the domains of $\text{SUM}$ to the non-negative integers, and $\text{PROD}$ to the natural numbers. We emphasize that although we only consider aggregates of this form, it is indeed powerful since it covers almost all the aggregates used in benchmark programs [CIR+11].

For this case of ASP programs with aggregates, we enhanced the so-called modified completion of a predicate $P$ into the following sentence:

$$\forall x \left( \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r \overline{\text{Body}(r)} \rightarrow P(x) \right) \land \quad (5.4)$$

$$\land \forall x \left( P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r)=P(x)} \exists y_r \left[ \overline{\text{Body}(r)} \land \text{Pos}(r) < P(x) \land \overline{\text{PosAgg}(r)} < P(x) \right] \right), \quad (5.5)$$

where $\overline{\text{Pos}(r)} < P(x)$ is defined as in the original definition of the ordered completion and only on the standard (i.e., non-aggregate) atoms of the positive body, and $\overline{\text{PosAgg}(r)} < P(x)$
stands for the sentence\textsuperscript{1}

$$\bigwedge_{\delta \in \text{PosCardSumProd}(r), \leq \in \{=, \geq, >\}} (\text{OP} \langle v : \exists w \text{Body}(\delta) \land \text{Pos}(\delta) < P(x) \rangle \leq N) \land$$

$$\bigwedge_{\delta \in \text{PosMinMax}(r), \leq \in \{<, \leq, =, \geq, >\}} (\text{OP} \langle v : \exists w \text{Body}(\delta) \land \text{Pos}(\delta) < P(x) \rangle \leq N),$$

such that $\text{Pos}(\delta) < P(x)$ is simply the enforcement of the comparison relations into the positive body of $\delta$. Generally speaking, $\text{PosAgg}(r) < P(x)$ is the enhancement part from that of the original definition by considering the occurrences of aggregate atoms in the rule’s body. Then with $\text{MComp}(\Pi)$ again being the conjunctions of all the enhanced modified completion of the intensional predicates of $\Pi$, and with $\text{Trans}(\Pi)$ as in the original definition, we have that the enhance ordered completion of a FO program with (anti)monotone aggregates is again $\text{OC}(\Pi) = \text{MComp}(\Pi) \land \text{Trans}(\Pi)$. Then by Theorem 3, we have that the models of $\text{OC}(\Pi)$ are exactly corresponding to the answer sets (i.e., the stable models) of $\Pi$ on finite structures.

### 5.2 Solver Implementation

The notion of ordered completion also initiated a new direction of ASP solvers by grounding on a program’s ordered completion instead of the program itself. We reported our first implementation of such a solver (see Chapter 3), which compared favorably to other major existing ASP solvers, especially on big problem instances (See Table 3.1).

In the traditional ASP solvers, a 2-step approach is used. First, a grounder is used to transform the FO program into a propositional one. Second, a propositional ASP solver is then called to compute an answer. Differently from the traditional approach, our solver uses a 3-step approach. First, we translate a FO program into its ordered completion. Second, we had implemented a grounder called Groc, to transform the ordered completion together with the extensional database, into a propositional SMT theory. Then finally, we call an SMT solver to compute a model of the SMT theory, which would be an answer set of the

\textsuperscript{1}For the following, we use $\text{PosAgg}(r)$ to denote the aggregate atoms from $\text{Pos}(r)$; $\text{PosCardSumProd}(r)$ to denote the cardinality, sum, and product aggregates from $\text{Pos}(r)$; and $\text{PosMinMax}(r)$ to denote the minimum and maximum aggregate atoms from $\text{Pos}(r)$ respectively.
program based on Theorem 5. It should be noted that since we were grounding directly on a classical FO sentence, we were able to incorporate classical reasoning directly into the grounding technique (see Section 3.4.4). In addition, we also made use of the advantages provided by the interpretations of the comparison atoms by incorporating the notion of “derivation order” as well into our grounding technique. Surprisingly, this method proved to be highly efficient on very big problem instances of the Hamiltonian circuit program (see Section 3.5).

### 5.3 Preferred ASP

The other main contribution of the thesis is the lifting of propositional preferred ASP to the first-order case (see Chapter 4). In particular, we proposed a progression based preference semantics for FO ASP so that for a preferred program \( P = (\Pi, <^P) \), we have that a \( \tau(\Pi) \)-structure \( \mathcal{M} \) is preferred answer set of \( P \) iff \( \lambda_{\mathcal{M}}(\Gamma^\infty(P)_\mathcal{M}) = \mathcal{M} \), i.e., the fixpoint of the operator \( \lambda_{\mathcal{M}}(\Gamma^\infty(P)_\mathcal{M}) \) is \( \mathcal{M} \) (see Definitions 18 and 19).

We then investigated the essential semantical and syntactical properties (see Sections 4.3 and 4.3.2). In particular, the preferred answer sets have the property that all preferred answer sets of a preferred program \( P = (\Pi, <^P) \) are also answer sets of \( \Pi \) (see Proposition 14). In addition, we also showed the syntactical property that preferred programs where all rules have non-empty positive bodies always have an answer set (see Proposition 15), i.e., this mainly being the empty answer set \( \emptyset \). Furthermore, we also showed the interesting property that \( \mathcal{M} \) is a preferred answer set of \( P = (\Pi, <^P) \) iff each \( (r, \eta) \in \Gamma^\infty(P)_\mathcal{M} \) is a generating rule of \( \mathcal{M} \) under \( \eta \) (see Theorem 3). We further derived certain stronger syntactical conditions for which the existence of a preferred answer set can be determined (see Theorem 9). We emphasize that in order to prove these important properties, we also specifically considered the grounding of preferred answer set programs and established its connections to the first-order case (see Section 4.3.1).

We then addressed the expressiveness of preferred FO answer set programming in relation to classical SO logic. More precisely, we proposed a translation of a preferred program \( P = (\Pi, <^P) \) into a SO sentence \( \exists \exists \forall S \varphi_P(\vec{z}, S) \), where

\[
\varphi_P(\vec{z}, S) = \varphi_P^{\text{PREF}}(\vec{z}) \land \varphi_{\Pi}^{\text{PRO}}(\vec{z}, S) \land \varphi_{\Pi}^{\text{COMP}},
\]
such that a $\tau(\Pi)$ structure $M$ is a preferred answer set of $P$ iff $M \models \exists \vec{x} \forall S \varphi_P(\vec{x}, S)$ (see Theorem 11). Intuitively speaking, (5.6) precisely captures the preferred answer sets by "simulating" the progression semantics through encoding a well-ordered relation among the pairs of generating rule and assignment $(r, \eta)$ via the $\varphi^{\text{PRO}}_{\Pi}(\vec{x}, S)$ formula (see Definition 23). This well-ordered encoding was further refined to comply with the preference relations by strengthening the theory with the formula $\varphi^{\text{PREF}}_P(\vec{x})$ (see Definition 24). We then showed that when only finite structures are concerned, the SO formula can be reduced down to an existential SO sentence (see Proposition 18), by using the fact that total-orders are always well-orders on finite sets.

5.4 Future Work

By introducing the notion of the ordered completion, this work enabled us to implement a new direction of ASP solver by first translating a program into its ordered completion, and then to work on finding a model of this FO sentence using, e.g., SAT or SMT solvers. In Section 2.4, we also showed how the notion of the ordered completion can be extended to ASP programs with aggregate constructs. Since our current (in fact first) implementation of the ordered completion (i.e., see Chapter 3) only involves programs without aggregates, we leave the implementation of programs with aggregates as a promising task for our future work. Furthermore, as also mentioned in Chapter 3, since the ordered completion is a classical first-order sentence, a theoretically challenging but practically significant direction is the implementation of directly performing FO reasoning on the ordered completion.

Another interesting direction for the theoretical side of ordered completion is to consider how it relates to more general first-order formulas, i.e., not restricted to the syntax of normal logic programs. A particular promising direction is to consider programs where the heads of the rules are of a single atom (similar to the case of normal programs), but where the bodies are made up of nested expressions [LTT99]. Nested expressions are formulas built up as in classical first-order logic, but where the only connectives used are $\land$, $\lor$, $\neg$, $\exists$, and $\forall$, i.e., anything but the implication ‘$\rightarrow$’. A reason why this is a promising direction is because at the propositional level, programs where all the rules have traditional heads (i.e., head is just an atom), but where the bodies are of nested expressions, are still in NP [Fer11]. An even more general direction is to extend the ordered completion to arbitrary
first-order sentences as a whole, and study how the models of such formulas relates to both its classical and stable models. To gain a general idea of this translation, let us consider the simple case of propositional disjunctive logic programs. A propositional disjunctive program is a set of rules of the form

$$a_1; \ldots; a_k \leftarrow b_1, \ldots, b_l, \neg c_1, \ldots, \neg c_m.$$  \hspace{1cm} (5.7)

Now let $\Pi$ be a propositional logic program with signature $\tau(\Pi)$. Then we define the modified completion of $MComp(\Pi)$ of $\Pi$ as follows

$$\bigwedge_{r \in \Pi} \left( \overline{\text{Body}(r)} \rightarrow \bigvee \overline{\text{Head}(r)} \right)$$
$$\wedge \bigwedge_{a \in \tau(\Pi)} (a \rightarrow \bigvee_{r \in \Pi, \overline{a} \in \text{Head}(r)} (\overline{\text{Body}(r)} \land \text{Pos}(\overline{r}) < a)), \hspace{1cm} (5.8)$$

where for a rule $r \in \Pi$ of the form (5.7):

- $\text{Head}(r) = \{a_1, \ldots, a_k\}$ such that $\bigvee \text{Head}(r)$ stands for the formula $a_1 \lor \cdots \lor a_k$;
- $\overline{\text{Body}(r)} = b_1 \land \cdots \land b_l \land \neg c_1 \land \cdots \land \neg c_m$;
- $\text{Pos}(r) = \{b_1, \ldots, b_l\}$;
- and lastly, $\text{Pos}(\overline{r}) < a$ stands for the following formula:

$$\bigwedge_{b \in \text{Pos}(r)} (\leq b a \land \neg \leq ab),$$

which still encodes that $b$ is used to establish $a$ and not the other way around.

Then with $\text{Trans}(\Pi)$ similarly standing for the formula

$$\bigwedge_{a,b,c \in \tau(\Pi)} (\leq ab \land \leq bc \rightarrow \leq ac), \hspace{1cm} (5.10)$$

we define $OC(\Pi)$ as again $MComp(\Pi) \land \text{Trans}(\Pi)$, it can be shown that on finite structures, if an interpretation $I \subseteq \tau(\Pi)$ is an answer set of $\Pi$, then we have that $I \models OC(\Pi)$. Then we clearly have that $OC(\Pi)$ is weaker than $SM(\Pi)$, i.e., $SM(\Pi) \models OC(\Pi)$. Now
set $MComp'(\Pi)$ to be a minor modification of $MComp(\Pi)$ such that it is defined as the following formula:

$$\bigwedge_{r \in \Pi} (\overset{\sim}{\text{Body}}(r) \rightarrow \bigvee \text{Head}(r))$$

$$\land \bigwedge_{a \in \tau(\Pi)} (a \rightarrow \bigvee_{r \in \Pi, a \in \text{Head}(r)} (\overset{\sim}{\text{Body}}(r) \land \text{Pos}(r) < a \land \neg \text{Head}(r) \setminus \{a\})),$$

where

$$\text{Head}(r) \setminus \{a\} = \bigwedge_{b \in \text{Head}(r) \setminus \{a\}} \neg b,$$

i.e., the conjunctions of the negations of all the atoms in $\text{Head}(r) \setminus \{a\}$. Then with $OC'(\Pi) = MComp'(\Pi) \land \text{Trans}(\Pi)$, it can be shown that $OC'(\Pi) \models SM(\Pi)$, such that we now have the following relationship:

$$OC'(\Pi) \models SM(\Pi) \models OC(\Pi).$$

It would then be interesting if this relationship could be used towards some implementation advantage, as an approximation to the stable models, or alternatively, study if a “tighter” bound exist.

In regards to preferred FO ASP, several issues are also left for our future work. Firstly, it would be an interesting topic to study the application of preferred FO programs. Currently, we are considering to use the proposed formalisms to specify finite structure update, where the entire update process may be represented by a preferred FO program. Another interesting work is to extend the proposed framework to preferred FO disjunctive programs. The recent work of Zhou and Zhang [ZZ11] extends the progression based semantics of FO normal programs to FO disjunctive programs. This new semantics provides for a possibility to develop a similar progression based preference semantics for FO programs.
Appendix A

Proofs

A.1 Proofs for Chapter 2

A.1.1 Proof of Lemma 1

(⇒) If $A$ is an answer of $\Pi$, then $A \models SM(\Pi)$. Then since $A \models \widehat{\Pi}$ (i.e., since $A \models SM(\Pi)$), it will be sufficient to show that all $S \subseteq [P_{int}(\Pi)]^A$ is externally supported. For the sake of contradiction, assume otherwise that there exist such a non-externally supported set $S$. Then with $\sigma_U = \{U_i | 1 \leq i \leq n\}$,\footnote{Recall from Section 1.2.1 that $U = U_1 \ldots U_n$ such that with $P_{int}(\Pi) = \{P_1, \ldots, P_n\}$, each $U_i (1 \leq i \leq n)$ is a new predicate symbol that match the arity of $P_i$.} define the $\tau(\Pi) \cup \sigma_U$-structure $U$ as a structure expanded from $A$ as follows:

- $c^U = c^A$ for each constant $c \in \tau(\Pi)$;
- $P^U = P^A$ for each predicate $P \in \tau(\Pi)$;
- $U_i^U = P_i^A \setminus \{a | P_i(a) \in S\}$ for $1 \leq i \leq n$.

Then clearly, $U \models U < P$ since $S \neq \emptyset$. Moreover, it is also true that $U \models \widehat{\Pi}^*(U)$. Indeed, consider the formula $\forall xy_r(\Body(r)^* \rightarrow P(x)^*)$ for some rule $r \in \Pi$ with local variables $y_r$. Then with a corresponding assignment $xy_r \rightarrow ab_r$ on $xy_r$, either $U \not\models \widehat{\Body(r)}^*[xy_r/ab_r]$ or $U \models \widehat{\Body(r)}^*[xy_r/ab_r]$. We now consider each of these possibilities:
Case: 1 \( \mathcal{U} \not\models Body(r)^*[xy_r/ab_r] \).
Then we clearly have \( \mathcal{U} \models (Body(r)^* \rightarrow P(x)^*)[xy_r/ab_r] \) as well.

Case: 2 \( \mathcal{U} \models Body(r)^*[xy_r/ab_r] \).
Then either \( \mathcal{U} \models P(x)^*[x/a] \) or \( \mathcal{U} \not\models P(x)^*[x/a] \). We now consider each further subcases:

Subcase 1: \( \mathcal{U} \models P(x)^*[x/a] \).
Then we clearly have \( \mathcal{U} \models (Body(r)^* \rightarrow P(x)^*)[xy_r/ab_r] \) as well.

Subcase 2: \( \mathcal{U} \not\models P(x)^*[x/a] \).
Since \( \mathcal{U} < \mathcal{P} \models Body(r)^* \rightarrow Body(r) \) and where \( \mathcal{U} \models U < P \), we have \( \mathcal{U} \models \tau(\Pi) = \mathcal{A} \models Body(r)^* \) as well. Then since \( \mathcal{A} \models \hat{\Pi} \), we must have \( \mathcal{A} \models P(x)[x/a] \) as well. Then since \( \mathcal{U} \not\models P(x)^*[x/a] \), we that \( P(a) \in S \) since \( P(a) \in A \) and \( P(a)^* \not\in \mathcal{U} \). Then since \( S \) is not externally supported, we must have \( Pos(r)[xy_r/ab_r] \cap S \neq \emptyset \). Then since \( \mathcal{U} \) was expanded from \( \mathcal{A} \) by interpreting the \( U_i \) predicates as those of \( P_i \) but omitting those grounded atoms in \( S \), we must have that \( \mathcal{U} \not\models Pos(r)^*[xy_r/ab_r] \), which then implies \( \mathcal{U} \not\models Body(r)^*[xy_r/ab_r] \). This is in contradiction with the assumption \( \mathcal{U} \models Body(r)^*[xy_r/ab_r] \), therefore we cannot have \( \mathcal{U} \not\models P(x)^*[x/a] \).

Therefore, since \( \mathcal{U} \models U < P \land \hat{\Pi}^*(U) \), then this is a contradiction since this implies \( A \not\models SM(\Pi) \).

\((\Leftarrow\Rightarrow)\) Since \( \mathcal{A} \models \hat{\Pi} \), it will be sufficient to show that \( \mathcal{A} \models \forall U(U < P \rightarrow \neg \hat{\Pi}^*(U)) \) to show \( \mathcal{A} \models SM(\Pi) \). For the sake of contradiction, assume \( \mathcal{U} \models U < P \land \hat{\Pi}^*(U) \) for some \( \tau(\Pi) \cup \sigma_U \)-structure \( \mathcal{U} \) expansion of \( \mathcal{A} \). Now let \( S \) be the set of ground atoms \( \{P_i(a) \mid a \in P^U_1 \setminus U^U_1, 1 \leq i \leq n\} \). Then \( S \neq \emptyset \) and \( S \) is not externally supported. For otherwise, assume that \( S \) is externally supported. Then there exist some \( P(a) \in S \), rule \( P(x) \rightarrow Body(r) \in \Pi \) with local variables \( y_r \), and assignment \( xy_r \rightarrow ab_r \) such that:

1. \( \mathcal{A} \models Body(r)[xy_r/ab_r] \) (i.e., the “support” for \( P(a) \));

2. \( Pos(r)[xy_r/ab_r] \cap S = \emptyset \).

\(^2\text{Pos}(r)^* \text{ denotes the conjunctions } \wedge_{Q(y) \in Pos(r)} Q(y)^* \).
Then by the construction of $S$, this implies $U \models \overline{\text{Body}(r)}^*[xy_r/ab_r]$. Then since $U \models \hat{\Pi}^+(U)$, this implies $U \models P(x)^*[x/a]$ as well. This is a contradiction as $P(a) \in S$ and since $P(a) \in S$ implies $P(a)^*$ is not in the $U$ structure. Therefore, this show that $S$ is not externally supported. Then this is again a contradiction since we had that every set $S \subseteq [P_{int}(\Pi)]^A$ is externally supported. This completes the proof of Lemma 1. □

### A.1.2 Proof of Theorem 2

$(\implies)$ To prove this result, we borrow some of the ideas from [ZZ10]. In a similar manner to the “progression stages” in [ZZ10], we define the the $\tau(\Pi) \cup \sigma_U$-structure $U^t(\hat{\Pi}^+(U))$ (or just $U^t$ when clear from context) inductively as follows:

1. $U^0$ is the $\tau(\Pi) \cup \sigma_U$-structure expanded from $A$ as follows:
   - $c^{U^0} = c^A$ for each constant $c \in \tau(\Pi)$;
   - $P^{U^0} = P^A$ for each predicate $P \in \tau(\Pi)$;
   - $U^{U^0} = \emptyset$ for each predicate $U \in \sigma_U$;

2. $U^{t+1}$ is the following $\tau(\Pi) \cup \sigma_U$-structure:

$$U^t \cup \{P(a)^* \mid \text{there exists a rule } P(x) \leftarrow \text{Body}(r) \in \Pi \text{ with local variables } y_r \text{ and assignment } xy_r \rightarrow ab_r \text{ such that } U^t \models \overline{\text{Body}(r)}^*[xy_r/ab_r] \}.$$ 

Then it is not too difficult to show by induction that $U^{t+1}_i \subseteq P^A_i (1 \leq i \leq n)^3$ for all $t \geq 0$ since $A \models \hat{\Pi}$ (i.e., because $A \models SM(\Pi)$). In fact, it is true that the otherway holds as well, as we have by the following claim.

**Claim 1:** $P_i^A \subseteq U_i^{t+\infty} (1 \leq i \leq n)$.

**Proof of Claim 1:** On the contrary, assume $U_i^{t+\infty} \subset P_i^A$ for some $i (1 \leq i \leq n)$. Then let $S$ be the following set of ground atoms:

$$\{P_i(a) \mid a \in P_i^A \setminus U_i^{t+\infty}, 1 \leq i \leq n\}.$$ 

\footnote{Recall from Section 1.2.1 that $U = U_1 \ldots U_n$ corresponds to $P_{int}(\Pi) = \{P_1, \ldots, P_n\}$.}
Then we have $S \neq \emptyset$. Moreover, we even have that $S$ is not externally supported. For otherwise, assume that $S$ is externally supported. Then there is a $P(a) \in S$, rule $P(x) \leftarrow Body(r) \in \Pi$ with local variables $y_r$, and assignment $xy_r \rightarrow ab_r$ such that:

1. $A \models \overline{Body(r)[xy_r/ab_r]}$;
2. $Pos(r)[xy_r/ab_r] \cap S = \emptyset$.

Then with a slight abuse of notation, this implies that $Pos(r)^*|_{\overline{xy_r/ab_r}} \subseteq U^\infty.$

Then from the definition of $\overline{Body(r)^*}$, this implies that $U^t \models \overline{Body(r)^*[xy_r/ab_r]}$ for some $t \geq 1$. Then this is a contradiction since this implies $P(a)^* \in U^{t+1} \subseteq U^\infty$.

Therefore, $S$ cannot be externally supported. But this is again a contradiction since by Lemma 1, we must have that $S$ is externally supported since $A$ is an answer set of $\Pi$. This completes the proof of Claim 1.

Therefore, since $U_{i}^{t_{\infty}} = P_{i}^A (1 \leq i \leq n)$ as implied Claim 1, we can now construct an expansion $A'$ of $A$ on the signature $\sigma_{\leq}$. First, define $\Delta^i$ inductively as follows:

$$
\Delta^0 = \{ P(a)^* | P(a)^* \in U^1, P \in P_{int}(\Pi) \};
\Delta^i = \{ P(a)^* | P(a)^* \in U^{t+1} \setminus U^t, P \in P_{int}(\Pi) \}.
$$

Then clearly, we have $\Delta^i \cap \Delta^j = \emptyset$ if both $\Delta^i$ and $\Delta^j$ are non-empty and $i \neq j$ (i.e., these sets partition $U^\infty$ on the “$U$” interpretations). Moreover, it is also not too difficult to see that $\bigcup_{i \geq 0} \Delta^i = U|_{\sigma_U}$, i.e., $\bigcup_{i \geq 0} \Delta^i$ is exactly the “$U$” interpretations of $U^\infty$. Then we define the expansion $A'$ of $A$ on the signature $\sigma_{\leq}$ by setting

$$
\leq_{P_{Q}}^A = \{(a, b) | P(a)^* \in \Delta^i, Q(b)^* \in \Delta^j, i < j \}
$$

for each pair of predicates $P, Q \in P_{int}(\Pi)$. Now we show that $A' \models OC(\Pi)$. Indeed, since $A'|_{\tau(\Pi)} = A = \tilde{\Pi}$, then it is sufficient to show that $A'$ satisfies

$$
\bigwedge_{P \in P_{int}(\Pi)} \forall x( P(x) \rightarrow \bigvee_{r \in \Pi, \text{Head}(r) = P(x)} \exists y_r( Body(r) \land Pos(r) \subseteq P(x) )) \tag{A.1}
$$

With a slight abuse of notation, $Pos(r)^*|_{\overline{xy_r/ab_r}}$ denotes the following set of ground atoms: $\{ P(a)^* | \beta \in Pos(r), P(a) \text{ corresponds to } \beta[xy_r/ab_r] \}$. 

Hence, assume \( A' \models P(x)[x/a] \) for some assignment \( x \rightarrow a \). Then by Claim 1, \( P(a)^* \in \mathcal{U}' \) for some \( t \geq 1 \). Without loss of generality, assume that \( t \) is the least such stage. Then by the definition of \( \mathcal{U}' \), there exist a rule \( P(x) \leftarrow Bod(y)(r) \in \Pi \) with local variables \( y_r \) and assignment \( xy_r \rightarrow ab_r \) such that \( \mathcal{U}'^{-1} \models Bod(y)(r)^*[xy_r/ab_r] \). Then this implies that \( Pos(r)^*[xy_r/ab_r] \subseteq \mathcal{U}'^{-1} \). Then since \( P(a)^* \in \mathcal{U}' \) and \( P(a)^* \notin \mathcal{U}'^{-1} \) (i.e., since \( t \) is the least such stage), we have from the construction of \( A' \) that \( (b, a) \in \leqQP \) for each \( Q(b)^* \in Pos(r)^*[xy_r/ab_r] \) where \( Q \in \mathcal{P}_{\text{int}}(\Pi) \). Then this implies that \( A' \models Pos(r)^*[xy_r/ab_r] \). Therefore, since \( \mathcal{U}'^{-1} \models Bod(y)(r)^*[xy_r/ab_r] \), and where \( \mathcal{U}'^{-1} \subseteq \mathcal{U} \) and \( \mathcal{U}_i^{\mathcal{U}} = P^A(1 \leq i \leq n) \) as implied by Claim 1, then it follows that \( A' \models (\overline{Bod(y)(r)} \land Pos(r)^* < P(x))[xy_r/ab_r] \).

\((\leftarrow)\) Since \( A' \models OC(\Pi) \), then \( A' \mid \tau(\Pi) = A \models \widehat{\Pi} \). Then by Lemma 1, it will be sufficient to show that all \( S \subseteq [\mathcal{P}_{\text{int}}(\Pi)]^A \) are also externally supported. Indeed, for the sake of contradiction, assume for some \( S \subseteq [\mathcal{P}_{\text{int}}(\Pi)]^A \) that \( S \) is not externally supported. Then we have for all \( P(a) \in S \), rule \( P(x) \leftarrow Bod(y)(r) \in \Pi \) with local variables \( y_r \), and assignment \( xy_r \rightarrow ab_r \) such that \( A' \models Bod(y)(r)[xy_r/ab_r] \), that \( Pos(r)[xy_r/ab_r] \cap S \neq \emptyset \). Now, since \( A' \models OC(\Pi) \), then there exist a \( P(a) \in S \) such that for some rule \( P(x) \leftarrow Bod(y)(r) \in \Pi \) with local variables \( y_r \) and assignment \( xy_r \rightarrow ab_r \), we have \( A' \models (\overline{Bod(y)(r)} \land Pos(r)^* < P(x))[xy_r/ab_r] \). Moreover, due to the finiteness of \( A' \) (and thus, also the finiteness of \( S \)), we can safely assume without loss of generality that for all \( \leqQP \) \( (b, a) \in Pos(r)^* < P(x)[xy_r/ab_r] \), we have \( Q(b) \notin S \). Then this is a contradiction since this implies that \( S \) is externally supported. This ends the proof of Theorem 2. \( \square \)

### A.1.3 Proof of Lemma 2

\((\rightarrow)\) If \( A \) is an answer set of \( \Pi \), then \( A \models SM(\Pi) \). Therefore, since this implies that \( A \models \widehat{\Pi} \), then it will be sufficient to show that every set \( S \subseteq [\mathcal{P}_{\text{int}}(\Pi)]^A \) is also externally supported. So for the sake of contradiction, let us assume that there exist such a set \( S \subseteq [\mathcal{P}_{\text{int}}(\Pi)]^A \) that is not externally supported. Then with \( \sigma_U = \{U_1, \ldots, U_n\} \), set \( \mathcal{U} \) to be the

---

5To validate this assumption, one only has to note that for a chosen \( P(a) \), if \( Pos(r)[xy_r/ab_r] \cap S \neq \emptyset \), then we can choose another \( Q(b) \in Pos(r)[xy_r/ab_r] \cap S \). Then since \( A' \models OC(\Pi) \), there must also exist a rule \( Q(y) \leftarrow Bod(y)(r') \in \Pi \) with local variables \( z_{r'} \) and assignment \( y_{z_{r'}} \rightarrow bc_{r'} \) such that \( A' \models (\overline{Bod(y)(r')} \land Pos(r')^* < Q(y))[yz_{r'}/bc_{r'}] \) and where \( P(a) \notin Pos(r')^*[yz_{r'}/ac_{r'}] \), due to the enforcement of the comparison atoms, i.e., \( Pos(r')^* < Q(y) \). Then due to the finiteness of \( S \), this can be repeated a finite number of times until we reach the desired condition of the assumption.
\(\tau(\Pi) \cup \sigma_U\)-structure as follows:

- \(c^U = c^A\) for each constant \(c \in \tau(\Pi)\);
- \(P^U = P^A\) for each predicate \(P \in \tau(\Pi)\);
- \(U_i^U = P_i^A \{a \mid P_i(a) \in S\}\) for \(1 \leq i \leq n\).

Then clearly, \(U \models U < P\) since \(S \neq \emptyset\). Moreover, it is also true that \(U \models \widehat{\Pi}^*(U)\). Indeed, since \(S\) is non-externally supported, then for all rules \(P(x) \leftarrow \text{Body}(r) \in \Pi\) with local variables \(y_r\) and assignment \(x y_r \rightarrow a b_r\) such that \(A \models \text{Body}(r)[x y_r/ab_r]\) (i.e., a "support" for \(P(a)\)), that there exist some atom \(\beta \in \text{Pos}(r)\) such that either:

- \(\beta = Q(y)\), where \(Q \in \mathcal{P}_{\text{int}}(\Pi)\) and \(Q(b) \in S\) corresponds to \(\beta[x y_r/ab_r]\), or
- \(\beta\) is a aggregate atom \(\delta\) of the form (2.31) and
  \[
  \text{OP}(c_v : A \models \text{Body}(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset) \preceq N
  \]
  does not hold for all assignments \(\alpha\) of the form \(x y_r w v \rightarrow a b_r c_w c_v\) where \(c_w \in \text{Dom}(A)^{|w|}\) and \(c_v \in \text{Dom}(A)^{|v|}\).

Now, to show \(U \models \widehat{\Pi}^*(U)\), consider the formula \(\forall x y_r (\text{Body}(r)^* \rightarrow P(x)^*)\) for some rule \(r \in \Pi\) with local variables \(y_r\). Then with a corresponding assignment \(x y_r \rightarrow a b_r\) on \(x y_r\), either \(U \not\models \text{Body}(r)^*[x y_r/ab_r]\) or \(U \models \text{Body}(r)^*[x y_r/ab_r]\). We now consider each of these possibilities:

**Case 1**: \(U \not\models \text{Body}(r)^*[x y_r/ab_r]\).

Then we clearly have \(U \models (\text{Body}(r)^* \rightarrow P(x)^*)[x y_r/ab_r]\) as well.

**Case 2**: \(U \models \text{Body}(r)^*[x y_r/ab_r]\).

Then either \(U \models P(x)^*[x/a]\) or \(U \not\models P(x)^*[x/a]\). We now consider each further subcases:

**Subcase 1**: \(U \models P(x)^*[x/a]\).

Then we clearly have \(U \models (\text{Body}(r)^* \rightarrow P(x)^*)[x y_r/ab_r]\) as well.
Subcase 2: $\mathcal{U} \not\models P(x)[x/a]$.

Since $\mathcal{U} \subsetneq \mathcal{P} \models \text{Body}(r)^* \rightarrow \text{Body}(r)$ and where $\mathcal{U} \models \mathcal{U} \subsetneq \mathcal{P}$, we have $\mathcal{U} \upharpoonright_{\tau(\Pi)} = \mathcal{A} \models \text{Body}(r)$ as well. Then since $\mathcal{A} \models \hat{\Pi}$, we must have $\mathcal{A} \models P(x)[x/a]$ as well. Then since $\mathcal{U} \not\models P(x)[x/a]$, we that $P(a) \in \mathcal{S}$ since $P(a) \in \mathcal{A}$ and $P(a)^* \notin \mathcal{U}$. Then since $\mathcal{S}$ is not externally supported, we must have that either:

1. $(\text{Pos}(r) \setminus \text{PosAgg}(r))[xy_r/ab_r] \cap \mathcal{S} \neq \emptyset$, or
2. for some aggregate atom $\delta$ of the form (2.31), we have that

$$\text{OP}(c_v : \mathcal{A} \models \text{Body}(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap \mathcal{S} = \emptyset) \leq N$$

does not hold for all assignments $\alpha$ of the form $xy_r, wv \rightarrow ab_r, c_w, c_v$ where $c_w \in \text{Dom}(\mathcal{A})^{\mathcal{w}}$ and $c_v \in \text{Dom}(\mathcal{A})^{\mathcal{v}}$. Note from the construction of the $\tau(\Pi) \cup \sigma_U$-structure $\mathcal{U}$ that

$$\text{OP}(c_v : \mathcal{U} \models \text{Body}(\delta)^*[\alpha]) \leq N$$

does not hold as well.

Then since $\mathcal{U}$ was expanded from $\mathcal{A}$ by interpreting the $\mathcal{U}_i$ predicates as those of $P_i$ but omitting those grounded atoms in $\mathcal{S}$, we must have that $\mathcal{U} \not\models \text{Pos}(r)^*[xy_r/ab_r]$, which then implies $\mathcal{U} \not\models \text{Body}(r)^*[xy_r/ab_r]$. This is in contradiction with the assumption $\mathcal{U} \models \text{Body}(r)^*[xy_r/ab_r]$, therefore we cannot have $\mathcal{U} \not\models P(x)^*[x/a]$.

Hence, since we had shown that $\mathcal{U} \models \hat{\Pi}^*(\mathcal{U})$, then this is a contradiction since $\mathcal{A} \models SM(\Pi)$ and where $\mathcal{U} \models \mathcal{U} \subsetneq \mathcal{P} \land \hat{\Pi}^*(\mathcal{U})$. Therefore, we must have that all $\mathcal{S} \subseteq \{P_{\text{nfl}}(\Pi)\}_A$ that is externally supported.

$(\iff)$ Since $\mathcal{A} \models \hat{\Pi}$, it will be sufficient to show $\mathcal{A} \models \forall \mathcal{U}(\mathcal{U} \subsetneq \mathcal{P} \rightarrow \neg \hat{\Pi}^*(\mathcal{U}))$ to show that $\mathcal{A} \models SM(\Pi)$. For the sake of contradiction, assume $\mathcal{U} \models \mathcal{U} \subsetneq \mathcal{P} \land \hat{\Pi}^*(\mathcal{U})$ for some $\tau(\Pi) \cup \sigma_U$-structure $\mathcal{U}$ expansion of $\mathcal{A}$. Then let $\mathcal{S}$ be the set of ground atoms $\left\{P_i(a) \mid a \in P_d \setminus U_i^\mathcal{U}, 1 \leq i \leq n\right\}$. Then $\mathcal{S} \neq \emptyset$ and $\mathcal{S}$ is not externally supported. For otherwise, assume that $\mathcal{S}$ is externally supported. Then there exist some ground atom $P(a) \in \mathcal{S}$, rule

---

$P_{\text{Pos}}(r)^*$ denotes the conjunction $\bigwedge_{Q(y) \in \text{Pos}(r)} Q(y)^*$. 
\( P(x \leftarrow \text{Body}(r)^* \in \Pi \) with local variables \( y_r \), and assignment \( xy_r \rightarrow ab_r \) such that:

1. \( A \models \text{Body}(r)[xy_r/ab_r]; \)

2. \( (\text{Pos}(r) \setminus \text{PosAgg}(r))[xy_r/ab_r] \cap S = \emptyset; \)

3. For all aggregate atom \( \delta \in \text{PosAgg}(r) \) of the form (2.31),

\[
\text{OP}(c_v : A \models \text{Body}(\delta)[xy_rwv/ab_rc_wc_v], \text{Pos}(\delta)[xy_rwv/ab_rc_wc_v] \cap S = \emptyset, c_w \in \text{Dom}(A)^{|w|} \text{ and } c_v \in \text{Dom}(A)^{|v|} \leq N,)
\]

holds.

Then by the construction of \( S \), this implies \( U \models \text{Body}(r)^*[xy_r/ab_r] \) as well. Then since \( U \models \widehat{\Pi}^*(U) \), this implies \( U \models P(x)^*[x/a] \). Then this is a contradiction because \( P(a) \in S \), and since \( P(a) \in S \) implies \( P(a)^* \) is not in the \( U \) structure. Therefore, this show that \( S \) is not externally supported. But this is again a contradiction since we had that every set \( S \subseteq [P_{\text{int}}(\Pi)]^A \) is externally supported. This completes the proof of Lemma 2. □

### A.1.4 Proof of Theorem 3

\( \implies \) To prove this result, we borrow some of the ideas from [ZZ10]. In a similar manner to the “progression stages” in [ZZ10], we define the the \( \tau(\Pi) \cup \sigma_U \)-structure \( U^t(\widehat{\Pi}^*(U)) \) (or just \( U^t \) when clear from context) inductively as follows:

1. \( U^0 \) is the \( \tau(\Pi) \cup \sigma_U \)-structure expanded from \( A \) as follows:
   - \( c^{U^0} = c^A \) for each constant \( c \in \tau(\Pi); \)
   - \( P^{U^0} = P^A \) for each predicate \( P \in \tau(\Pi); \)
   - \( U^{U^0} = \emptyset \) for each predicate \( U \in \sigma_U; \)

2. \( U^{t+1} \) is the following \( \tau(\Pi) \cup \sigma_U \)-structure:

\[
U^t \cup \{ P(a)^* \mid \text{there exists a rule } P(x) \leftarrow \text{Body}(r) \in \Pi \text{ with local variables } y_r \\
\text{and assignment } xy_r \rightarrow ab_r \text{ such that } U^t \models \text{Body}(r)^*[xy_r/ab_r] \}.
\]
Then it is not too difficult to show by induction that $U_{it}^t \subseteq P_i^A$ ($1 \leq i \leq n$)\(^7\) for all $t \geq 0$ since $A \models \widehat{Π}$ (i.e., because $A \models SM(Π)$). In fact, it is true that the other way holds as well, as we have by the following claim.

**Claim 1:** $P_i^A \subseteq U_{i}^{t \to \infty}$ ($1 \leq i \leq n$).

**Proof of Claim 1:** On the contrary, assume $U_{i}^{t \to \infty} \subset P_i^A$ for some $i$ ($1 \leq i \leq n$). Then let $S$ be the following set of ground atoms:

$$\{P_i(a) \mid a \in P_i^A \setminus U_{i}^{t \to \infty}, 1 \leq i \leq n\}.$$ 

Then we have $S \neq \emptyset$. Moreover, we even have that $S$ is not externally supported. For otherwise, assume that $S$ is externally supported. Then there is a $P(a) \in S$, rule $P(x) \leftarrow Body(r) \in Π$ with local variables $y_r$, and assignment $xy_r \rightarrow ab_r$ such that:

1. $A \models \widehat{Body(r)}[xy_r/ab_r]$;
2. $(Pos(r) \setminus PosAgg(r))[xy_r/ab_r] \cap S = \emptyset$;
3. for each aggregate atom $δ \in PosAgg(r)$ of the form (2.31), we have that

$$\text{OP}(c_v : A \models \widehat{Body(δ)}[α], Pos(δ)[α] \cap S = \emptyset) \leq N, \quad (A.2)$$

holds, where $α$ is the assignment of the form $xy_r wv \rightarrow ab_r c_w c_v$ with $c_w \in \text{Dom} A^{[w]}$ and $c_v \in \text{Dom} A^{[v]}$.

Then with a slight abuse of notation, this implies that $(Pos(r) \setminus PosAgg(r))[xy_r/ab_r] \subseteq U^{\infty}$\(^8\) and $U^{\infty} \models δ^*[x, y_r/ab_r]$ for all aggregate atoms $δ \in PosAgg(r)$. Then from the definition of $\widehat{Body(r)}^*$, this implies that $U^t \models \widehat{Body(r)}^*[xy_r/ab_r]$ for some $t \geq 1$. Then this is a contradiction since this implies $P(a)^* \in U^{t+1} \subseteq U^{\infty}$. Therefore, $S$ cannot be externally supported. But this is again a contradiction since by Lemma 2, we must have that $S$ is externally supported since $A$ is an answer set of $Π$. This completes the proof of Claim 1.

---

\(^7\)Recall from Section 1.2.1 that $U = U_1 \ldots U_n$ corresponds to $P_{int}(Π) = \{P_1, \ldots, P_n\}$.

\(^8\)With a slight abuse of notation, $Pos(r)^*[xy_r/ab_r]$ denotes the following set of ground atoms: $\{P(a)^* \mid β \in Pos(r) \setminus PosAgg(r), P(a) \text{ corresponds to } β[xy_r/ab_r]\}$. 
Therefore, since $U_i^{t\infty} = P_i^A$ ($1 \leq i \leq n$) as implied Claim 1, we can now construct an expansion $\mathcal{A}'$ of $\mathcal{A}$ on the signature $\sigma_\leq$. First, define $\Delta'$ inductively as follows:

$$\Delta^0 = \{ P(a)^* \mid P(a)^* \in U^t, P \in \mathcal{P}_{\text{int}}(\Pi) \};$$

$$\Delta^t = \{ P(a)^* \mid P(a)^* \in U^{t+1} \setminus U^t, P \in \mathcal{P}_{\text{int}}(\Pi) \}.$$

Then clearly, we have $\Delta^i \cap \Delta^j = 0$ if both $\Delta^i$ and $\Delta^j$ are non-empty and $i \neq j$ (i.e., these sets partitions $U^\infty$ on the "$U$" interpretations). Moreover, it is also not too difficult to see that $\bigcup_{i \geq 0} \Delta^i = U_{\sigma_U}$, i.e., $\bigcup_{i \geq 0} \Delta^i$ is exactly the "$U$" interpretations of $U^\infty$. Then we define the expansion $\mathcal{A}'$ of $\mathcal{A}$ on the signature $\sigma_\leq$ by setting

$$\Delta_{PQ}^t = \{(a, b) \mid P(a)^* \in \Delta^i, Q(b)^* \in \Delta^j, i < j\}$$

for each pair of predicates $P, Q \in \mathcal{P}_{\text{int}}(\Pi)$. Now we show that $\mathcal{A}' \models OC(\Pi)$. Indeed, since $\mathcal{A}'|_{\tau(\Pi)} = \mathcal{A} = \hat{\Pi}$, then it is sufficient to show that $\mathcal{A}'$ satisfies

$$\bigwedge_{P \in \mathcal{P}_{\text{int}}(\Pi)} \forall x( P(x) \rightarrow \bigvee_{r \in \Pi, Head(r) = P(x)} \exists y_r[\overline{\text{Body}(r)} \land Pos(r) < P(x) \land PosAgg(r) < P(x)]).$$

(A.3)

Hence, assume $\mathcal{A}' \models P(x)[x/a]$ for some assignment $x \rightarrow a$. Then by Claim 1, $P(a)^* \in U^t$ for some $t \geq 1$. Without loss of generality, assume that $t$ is the least such stage. Then by the definition of $U^t$, there exist a rule $P(x) \leftarrow Body(r) \in \Pi$ with local variables $y_r$ and assignment $xy_r \rightarrow ab_r$ such that $U^{t-1} \models \overline{\text{Body}(r)^*}[xy_r/ab_r]$. Then this implies that:

1. $(Pos(r) \setminus \text{PosAgg}(r))^*[xy_r/ab_r] \subseteq U^{t-1}$;
2. $U^{t-1} \models \delta^*[xy_r/ab_r]$ for each aggregate atom $\delta \in \text{PosAgg}(r)$ of the form (2.31).

Then since $P(a)^* \in U^t$ and $P(a)^* \notin U^{t-1}$ (i.e., since $t$ is the least such stage), we have from the construction of $\mathcal{A}'$ that $(b, a) \in \Delta_{QP}^t$ for each $Q(b)^* \in (Pos(r)\setminus \text{PosAgg}(r))^*[xy_r/ab_r]$ where $Q \in \mathcal{P}_{\text{int}}(\Pi)$, which then implies that $\mathcal{A}' \models Pos(r) < P(x)[xy_r/ab_r]$. In addition, since for each aggregate atom $\delta \in \text{PosAgg}(r)$ we have that

$$\text{OP}(c_{\delta} : U^{t-1} \models \overline{\text{Body}(\delta)^*}[\alpha], Pos(\delta)^*[\alpha] \subseteq U^{t-1}) \leq N$$
holds (where $\alpha : xy_r wv \rightarrow ab_r c_w c_v^9$), then we have that

$$\text{OP}(c_v : A' \models (\text{Body} (\delta) \land \text{Pos} (\delta) < P(x)) [\alpha] \leq N$$

holds as well. Therefore, since $U_{i-1}^{t} \models \text{Body} (r)^* [xy_r/ab_r]$, and where $U_i^{t} \subseteq U^\infty$ and $U_i^{t\infty} = P_i^A (1 \leq i \leq n)$ as implied by Claim 1, then it follows that $A' \models (\text{Body} (r) \land \text{Pos} (r) < P(x) \land \text{PosAgg} (r) < P(x))[xy_r/ab_r]$.

$(\Leftarrow)$ Since $A' \models OC(\Pi)$, then $A' \models A \models \hat{\Pi}$. Then by Lemma 1, it will be sufficient to show that all $S \subseteq [P_{int}(\Pi)]^{A'}$ are also externally supported. Indeed, for the sake of contradiction, assume for some $S \subseteq [P_{int}(\Pi)]^{A'}$ that $S$ is not externally supported. Then we have for all $P(a) \in S$, rule $P(x) \leftarrow \text{Body} (r) \in \Pi$ with local variables $y_r$, and assignment $xy_r \rightarrow ab_r$ such that $A' \models \text{Body} (r)[xy_r/ab_r]$, that there exist some atom $\beta \in \text{Pos} (r)$ such that either:

1. $\beta = Q(y)$, where $Q \in P_{int}(\Pi)$ and $Q(b) \in S$ corresponds to $\beta[xy_r/ab_r]$, or
2. $\beta$ is a aggregate atom $\delta$ of the form (2.31) and

$$\text{OP}(c_v : A \models \text{Body} (\delta)[\alpha], \text{Pos} (\delta)[\alpha] \cap S = \emptyset \leq N$$

does not hold, where $\alpha$ is the assignment of the form $xy_r wv \rightarrow ab_r c_w c_v$, with $c_w \in \text{Dom}(A)^w$ and $c_v \in \text{Dom}(A)^v$.

Now, since $A' \models OC(\Pi)$, then there exist a $P(a) \in S$ such that for some rule $P(x) \leftarrow \text{Body} (r) \in \Pi$ with local variables $y_r$ and assignment $xy_r \rightarrow ab_r$, we have $A' \models (\text{Body} (r) \land \text{Pos} (r) < P(x) \land \text{PosAgg} (r) < P(x))[xy_r/ab_r]$. Moreover, due to the finiteness of $A'$ (and thus, also the finiteness of $S$), we can safely assume without loss of generality that:

1. for all $\leq_{QP} (b, a) \in \text{Pos} (r) < P(x)[xy_r/ab_r]$, we have $Q(b) \notin S$;
2. for all aggregate atoms $\delta \in \text{PosAgg} (r)$ of the form (2.31), we have

$$\text{OP}(c_v : A' \models \text{Body} (\delta)[\alpha], \leq_{QP} (b, a) \in \text{Pos} (\delta) < P(x)[\alpha] \text{ implies } Q(b) \notin S \leq N$$

holds, where $\alpha : xy_r wv \rightarrow ab_r c_w c_v$.

---

9$c_w \in \text{Dom}(A)^w$ and $c_v \in \text{Dom}(A)^v$. 
To validate this assumptions, one only has to note that for a chosen \( P(a) \in S \), if it is the case that either:

- \((\text{Pos}(r) \setminus \text{PosAgg}(r))[xy/ab_r] \cap S \neq \emptyset\), or
- there exist some aggregate atom \( \delta \in \text{PosAgg}(r) \) such that

\[
\text{OP}(c_v : \mathcal{A'} \models \widehat{\text{Body}(\delta)}[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset) \preceq N
\]

does not hold, where \( \alpha \) is the assignment of the form \( xy_r wv \rightarrow ab_r c_w c_v \),

then we can choose another \( Q(b) \in (\text{Pos}(r) \setminus \text{PosAgg}(r))[xy/ab_r] \cap S \) or \( Q(b) \in \text{Pos}(\delta)[\alpha] \cap S \) (for some \( \delta \in \text{PosAgg}(r) \)). Then since \( \mathcal{A'} \models OC(\Pi) \), there must also exists a rule \( Q(y) \leftarrow \text{Body}(r') \in \Pi \) with local variables \( z_{r'} \) and assignment \( yz_{r'} \rightarrow bc_{r'} \) such that \( \mathcal{A'} \models (\text{Body}(r') \wedge \text{Pos}(r') < Q(y) \wedge \text{PosAgg}(r') < Q(y)) [yz_{r'}/bc_{r'}] \), and where:

1. \( P(a) \notin (\text{Pos}(r') \setminus \text{PosAgg}(r'))[yz_{r'}/bc_{r'}] \);
2. and for each aggregate atom \( \delta \in \text{PosAgg}(r) \) of the form (2.31),

\[
\text{OP}(c_v : \mathcal{A'} \models \widehat{\text{Body}(\delta)}[\alpha], P(a) \notin \text{Pos}(\delta)[\alpha]) \preceq N
\]

holds, where \( \alpha \) is the assignment of the form \( xy_r wv \rightarrow ab_r c_w c_v \),

due to the enforcement of the comparison atoms. Then due to the finiteness of \( S \) and the aggregates being either monotone or anti-monotone, this can be repeated a finite number of times until we reach the desired condition of the assumption. Then this is a contradiction since this implies that \( S \) is externally supported. This ends the proof of Theorem 3. \( \square \)

### A.2 Proofs for Chapter 4

#### A.2.1 Proof of Proposition 14

We prove this proposition by again using the result from [ZZ10] that provided a progression based answer set semantics for FO programs. Such progression semantics is a special case of our progression based preferred semantics specified in Definition 18.
APPENDIX A. PROOFS

We again give Zhang and Zhou’s progression semantics as follows. Let $\Pi$ be a FO program and $P_{\text{int}}(\Pi) = \{Q_1, \ldots, Q_n\}$ the set of all the intensional predicates of $\Pi$. Consider a structure $M$ of $\tau(\Pi)$. The $t$-th simultaneous evolution stage of $\Pi$ based on $M$, denoted as $\mathcal{M}^t(\Pi)$, is a structure of $\tau(\Pi)$ defined inductively as follows:

$$\mathcal{M}^0(\Pi) = (\text{Dom}(M), c_1^{\mathcal{M}^0}, \ldots, c_r^{\mathcal{M}^0}, P_1^{\mathcal{M}^0}, \ldots, P_s^{\mathcal{M}^0}, Q_1^{\mathcal{M}^0}, \ldots, Q_n^{\mathcal{M}^0}),$$

where $c_i^{\mathcal{M}^0} = c_i^M$ for each constant symbol $c_i$ ($1 \leq i \leq r$), $P_j^{\mathcal{M}^0} = P_j^M$ for each extensional predicate $P_j$ in $P_{\text{ext}}(\Pi)$ ($1 \leq j \leq s$), and $Q_k^{\mathcal{M}^0} = \emptyset$ for each intensional predicate $Q_k$ in $P_{\text{int}}(\Pi)$ ($1 \leq k \leq n$);

$$\mathcal{M}^{t+1}(\Pi) = \mathcal{M}^t(\Pi) \cup \{ Q_i(x) \eta \mid \text{there exist a rule}$$

$$Q_i(x) \leftarrow \beta_1, \ldots, \beta_l, \text{not } \gamma_1, \ldots, \text{not } \gamma_m \in \Pi$$

$$\text{and an assignment } \eta \text{ such that for all } j \ (1 \leq j \leq l),$$

$$\beta_j \eta \in \mathcal{M}^t(\Pi), \text{ and for all } k \ (1 \leq k \leq m), \gamma_k \eta \notin \mathcal{M}\}.$$

Then by Theorem 1 of [ZZ10], we know that $\mathcal{M}$ is an answer set of $\Pi$ iff $\mathcal{M}^\infty(\Pi) = \mathcal{M}$.

Now to prove this proposition, we will show that under the condition $\lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M) = \mathcal{M}$, $\lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M) \subseteq \mathcal{M}^\infty(\Pi)$ and $\mathcal{M}^\infty(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$. Indeed, it is easy to show the former holds by the definition of $\lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$. Therefore, it is only left to show $\mathcal{M}^\infty(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$. We show by induction on $t$ that $\mathcal{M}^t(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$.

**Basis** Clearly, since $\lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M) = \mathcal{M}^0(\Pi) \cup \{\text{Head}(r)\eta \mid (r, \eta) \in \Gamma^\infty(P)_M\}$, then it immediately follows that $\mathcal{M}^0(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$.

**Step** Assume for $0 \leq t' \leq t$ we have $\mathcal{M}^{t'}(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$.

We show $\mathcal{M}^{t+1}(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$. On the contrary, suppose $\mathcal{M}^{t+1}(\Pi) \not\subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$. Then there exist a rule $r \in \Pi$ and assignment $\eta$ such that $\text{Pos}(r)\eta \subseteq \mathcal{M}^t(\Pi)$ and $\text{Neg}(r)\eta \cap M = \emptyset$, and were $\text{Head}(r)\eta \in \mathcal{M}^{t+1}(\Pi)$ and $\text{Head}(r)\eta \notin \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$. Now, since $\mathcal{M}^t(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(P)_M)$ (i.e., by the inductive assumption), then for some $n \geq 0$, we have $\mathcal{M}^t(\Pi) \subseteq \lambda_{\mathcal{M}^0}(\Gamma^n(P)_M)$. Furthermore, since $\text{Head}(r)\eta \notin \lambda_{\mathcal{M}^0}(\Gamma^n(P)_M)$, then we also have $\text{Head}(r)\eta \notin \lambda_{\mathcal{M}^0}(\Gamma^{n+1}(P)_M)$, and hence, that $(r, \eta) \notin \Gamma^{n+1}(P)_M$. Then by the definition of $\Gamma^{n+1}(P)_M$, there must
exists a rule $r' \prec^P r$ and an assignment $\eta'$ such that:

(a) $\text{Pos}(r')\eta' \subseteq \mathcal{M}$ and $\text{Neg}(r')\eta' \cap \lambda_{\mathcal{M}}(\Gamma^n(\mathcal{P})_{\mathcal{M}}) = \emptyset$;

(b) $(r', \eta') \notin \Gamma^n(\mathcal{P})_{\mathcal{M}}$,

i.e., a rule blocking $r$ from being applied at stage $n + 1$. Now, for a $k \geq 0$ and (arbitrary) rule $r^* \in \Pi$ and corresponding assignment $\eta^*$, set $B^k(r^*, \eta^*)$ to be such that

$B^0(r^*, \eta^*) = \{(r', \eta') \mid r' \prec^P r^*, \text{Pos}(r')\eta' \subseteq \mathcal{M} \text{ and } \text{Neg}(r')\eta' \cap \lambda_{\mathcal{M}}(\Gamma^0(\mathcal{P})_{\mathcal{M}}) = \emptyset\}$

and

$B^k(r^*, \eta^*) = \{(r', \eta') \mid (a) r' \prec^P r^*;
\quad \text{(b) } \text{Pos}(r')\eta' \subseteq \mathcal{M} \text{ and } \text{Neg}(r')\eta' \cap \lambda_{\mathcal{M}}(\Gamma^k(\mathcal{P})_{\mathcal{M}}) = \emptyset;
\quad \text{(c) } (r', \eta') \notin \Gamma^k(\mathcal{P})_{\mathcal{M}}\}$

for $k \geq 1$. Intuitively, $B^k(r^*, \eta^*)$ comprises the pairs $(r', \eta')$ that blocks $(r^*, \eta^*)$ from being applied at stage $k + 1$. Now, the following Claims 1, 2 and 3 reveal important properties of the set $B^k(r^*, \eta^*)$.

**Claim 1** For all $k' \geq k \geq 1$, we have $B^k(r^*, \eta^*) \supseteq B^{k'}(r^*, \eta^*)$. In other words, we do not gain anymore pairs $(r', \eta')$ blocking $(r^*, \eta^*)$ from being applied as we progress along the stages of $\Gamma^k(\mathcal{P})_{\mathcal{M}}$.

**Proof of Claim 1** For simplicity we assume that $k \geq 1$ (the case where we allow $k = 0$ immediately follows). Set $k'$ to be such that $k' \geq k$ and let $(r'', \eta'') \in B^{k'}(r^*, \eta^*)$. Then by the definition of $B^{k'}(r^*, \eta^*)$, we have:

(a) $r'' \prec^P r^*$;

(b) $\text{Pos}(r'')\eta'' \subseteq \mathcal{M}$ and $\text{Neg}(r'')\eta'' \cap \lambda_{\mathcal{M}}(\Gamma^{k'}(\mathcal{P})_{\mathcal{M}}) = \emptyset$;

(c) $(r'', \eta'') \notin \Gamma^{k'}(\mathcal{P})_{\mathcal{M}}$.

Then by the monotonicity of $\Gamma^{k'}(\mathcal{P})_{\mathcal{M}}$ for $k' \geq k$ (i.e., $\Gamma^k(\mathcal{P})_{\mathcal{M}} \subseteq \Gamma^{k'}(\mathcal{P})_{\mathcal{M}}$),
we also have:

(a) \( r'' <^P r^* \);
(b) \( \text{Pos}(r'')\eta'' \subseteq \mathcal{M} \) and \( \text{Neg}(r'')\eta'' \cap \lambda_{\mathcal{M}^\circ}(\Gamma^k(\mathcal{P}),\mathcal{M}) = \emptyset \);
(c) \( (r'', \eta'') \notin \Gamma^k(\mathcal{P},\mathcal{M}) \),

and hence, that \( (r'', \eta'') \in B^k(r^*, \eta^*) \). This ends the proof of Claim 1.

**Claim 2** For each \( (r', \eta') \in B^k(r^*, \eta^*) \), \( B^k(r', \eta') \subseteq B^k(r^*, \eta^*) \). That is, for each pairs \( (r', \eta') \in B^k(r^*, \eta^*) \) blocking \( (r^*, \eta^*) \) from being applied at stage \( k + 1 \), the pairs \( (r'', \eta'') \) blocking \( (r', \eta') \) in turn at stage \( k + 1 \), are themselves in \( B^k(r^*, \eta^*) \). Intuitively, this implies \( B^k(r^*, \eta^*) \) satisfies some form of closure.

**Proof of Claim 2** Set \( (r', \eta') \in B^k(r^*, \eta^*) \) and let \( (r'', \eta'') \in B^k(r', \eta') \). Then we have \( r' <^P r^* \) and \( r'' <^P r' \) by the definitions of \( B^k(r^*, \eta^*) \) and \( B^k(r', \eta') \) respectively. Then by transitivity, we also have \( r'' <^P r^* \). Hence, by the definition of \( B^k(r^*, \eta^*) \), we also have \( (r'', \eta'') \in B^k(r^*, \eta^*) \). This ends the proof of Claim 2.

**Claim 3** \( B^k(r^*, \eta^*) = \emptyset \) for some ordinal \( k \geq 1 \). That is, eventually, there will be some stage \( k \geq 1 \) such that there will be no more pairs \( (r', \eta') \) blocking \( (r^*, \eta^*) \) from being applied. First, we provide the intuition of the proof of Claim 3.

Basically, under the assumption \( \Gamma^\infty(\mathcal{P}),\mathcal{M} = \mathcal{M} \), we prove by showing that for all the pairs \( (r', \eta') \in B^k(r^*, \eta^*) \), there will be some \( l > 0 \) such that either \( \Gamma^{k+l}(\mathcal{P}),\mathcal{M} \) will “defeat” \( (r', \eta') \) (i.e., as in \( \text{Neg}(r')\eta' \cap \Gamma^{k+l}(\mathcal{P}),\mathcal{M} \neq \emptyset \)) or gets “eaten up” by it (i.e., as in \( (r', \eta') \in \Gamma^{k+l}(\mathcal{P}),\mathcal{M} \)).

**Proof of Claim 3** First, we show for all \( k \geq 1 \), \( B^k(r^*, \eta^*) \neq \emptyset \) implies there exist some \( l > 0 \) such that \( B^k(r^*, \eta^*) \supseteq B^{k+l}(r^*, \eta^*) \). Thus, for a \( k \geq 1 \), let \( (r', \eta') \in B^k(r^*, \eta^*) \). Moreover, without loss of generality, assume that for all \( (r'', \eta'') \in B^k(r^*, \eta^*) \), we have \( r'' <^P r' \) (i.e., note that due to the finiteness of \( \Pi \), we will always have these pairs \( (r', \eta') \in B^k(r^*, \eta^*) \)). Then we have \( B^k(r', \eta') = \emptyset \). This is because if \( B^k(r', \eta') \neq \emptyset \) and let \( (r'', \eta'') \in B^k(r', \eta') \), we will have \( r'' <^P r' \) by the definition of \( B^k(r', \eta') \), which contradicts the initial assumption about the pair \( (r', \eta') \). Now, as \( (r', \eta') \in B^k(r^*, \eta^*) \), then by the definition of
$B^k(r^*, \eta^*)$, we have $\text{Pos}(r^*) \eta^r \subseteq \mathcal{M}$ and $\text{Neg}(r^*) \eta^r \cap \lambda_{\mathcal{M}}(\Gamma^k(\mathcal{P}), \mathcal{M}) = \emptyset$.

Also, since $\lambda_{\mathcal{M}}(\Gamma^{\infty}(\mathcal{P}), \mathcal{M}) = \mathcal{M}$ by assumption, then for some $l \geq 0$, we have $\text{Pos}(r^*) \eta^r \subseteq \lambda_{\mathcal{M}}(\Gamma^{k+l}(\mathcal{P}), \mathcal{M})$. Now there can only be two possibilities:

**Case 1** $\text{Neg}(r^*) \eta^r \cap \mathcal{M} = \emptyset$.

Then since we also have $B^{k+l}(r^*, \eta^r) = \emptyset$ by Claim 1 (i.e., since $B^k(r^*, \eta^r) = \emptyset$), then we have $(r^*, \eta^r) \in \mathcal{M}^0(\Gamma^{k+l+1}(\mathcal{P}), \mathcal{M})$ by the definition of $\Gamma^{k+l+1}(\mathcal{P}), \mathcal{M}$ since $(r^*, \eta^r)$ will now be applicable at this stage (i.e., since there are no pairs $(r'', \eta'')$ blocking $(r^*, \eta^r)$ from application at stage $k + l + 1$ since $B^{k+l}(r^*, \eta^r) = \emptyset$). Then by the definition of $B^{k+l+1}(r^*, \eta^r)$, we have $(r^*, \eta^r) \notin B^{k+l+1}(r^*, \eta^r)$, which implies $B^k(r^*, \eta^r) \neq B^{k+l+1}(r^*, \eta^r)$.

Then since by Claim 1, we have $B^k(r^*, \eta^r) \supseteq B^{k+l+1}(r^*, \eta^r)$, then we must have $B^k(r^*, \eta^r) \supseteq B^{k+l+1}(r^*, \eta^r)$.

**Case 2** $\text{Neg}(r^*) \eta^r \cap \mathcal{M} \neq \emptyset$.

Then there could be two further possibilities:

**Subcase 1** $\text{Neg}(r^*) \eta^r \cap \lambda_{\mathcal{M}}(\Gamma^{k+l}(\mathcal{P}), \mathcal{M}) \neq \emptyset$.

Then we have $(r^*, \eta^r) \notin B^{k+l}(r^*, \eta^r)$ by the definition of $B^{k+l}(r^*, \eta^r)$. Moreover, since $(r^*, \eta^r) \in B^k(r^*, \eta^r)$, then it must be that $k + l > k$, which implies $l > 0$. Hence, similarly to above, we have also that $B^k(r^*, \eta^r) \supseteq B^{k+l}(r^*, \eta^r)$ with $l > 0$.

**Subcase 2** $\text{Neg}(r^*) \eta^r \cap \lambda_{\mathcal{M}}(\Gamma^{k+l}(\mathcal{P}), \mathcal{M}) = \emptyset$.

Then since $\lambda_{\mathcal{M}}(\Gamma^{\infty}(\mathcal{P}), \mathcal{M}) = \mathcal{M}$ by assumption, there exist some $m > 0$ such that $\text{Neg}(r^*) \eta^r \cap \lambda_{\mathcal{M}}(\Gamma^{k+l+m}(\mathcal{P}), \mathcal{M}) \neq \emptyset$, which implies $(r^*, \eta^r) \notin B^{k+l+m}(r^*, \eta^r)$ by the definition of $B^{k+l+m}(r^*, \eta^r)$. Hence, in a similar manner to above, we have $B^k(r^*, \eta^r) \supseteq B^{k+l+m}(r^*, \eta^r)$ with $m > 0$.

Therefore, we have that for all $k \geq 1$, $B^k(r^*, \eta^r) \neq \emptyset$ implies there exist some $l > 0$ such that $B^k(r^*, \eta^r) \supseteq B^{k+l}(r^*, \eta^r)$. Now we show for all $k \geq 1$ and $l > 0$, $B^k(r^*, \eta^r) \supseteq B^{k+l}(r^*, \eta^r)$ implies $\Gamma^k(\mathcal{P}), \mathcal{M} \subseteq \Gamma^{k+l}(\mathcal{P}), \mathcal{M}$. Hence, assume that $B^k(r^*, \eta^r) \supseteq B^{k+l}(r^*, \eta^r)$ and let $(r^*, \eta^r) \in B^k(r^*, \eta^r)$ where $(r^*, \eta^r) \notin B^{k+l}(r^*, \eta^r)$. Then by the definition of $B^k(r^*, \eta^r)$ we have:

1. $r^r \notin \mathcal{P} r^*$;
2. $\text{Pos}(r^*) \eta^r \subseteq \mathcal{M}$ and $\text{Neg}(r^*) \eta^r \cap \lambda_{\mathcal{M}}(\Gamma^k(\mathcal{P}), \mathcal{M}) = \emptyset$;
APPENDIX A. PROOFS

150

3. \((r', \eta') \notin \Gamma^k(\mathcal{P})_M\).

Also, since \((r', \eta') \notin B^{k+l}(r^*, \eta^*)\), then by the definition of \(B^{k+l}(r^*, \eta^*)\), we have either:

1. \(\text{Neg}(r')\eta' \cap \lambda_{M^0}(\Gamma^{k+l}(\mathcal{P})_M) \neq \emptyset\) or
2. \((r', \eta') \in \Gamma^{k+l}(\mathcal{P})_M\).

In either of these two aforementioned possibilities, it can be seen that \(\Gamma^k(\mathcal{P})_M \neq \Gamma^{k+l}(\mathcal{P})_M\). Therefore, since \(\Gamma^k(\mathcal{P})_M \subseteq \Gamma^{k+l}(\mathcal{P})_M\), then we must have \(\Gamma^k(\mathcal{P})_M \subset \Gamma^{k+l}(\mathcal{P})_M\). Hence, from this, we obtain the equivalent true statement (i.e., its contrapositive): if for all \(k \geq 1\) and \(l > 0\) we have \(\Gamma^k(\mathcal{P})_M = \Gamma^{k+l}(\mathcal{P})_M\), then \(B^k(r^*, \eta^*) = B^{k+l}(r^*, \eta^*)\). Now, by the monotonicity of \(\Gamma^k(\mathcal{P})_M\) for all \(k \geq 1\) and its guaranteed convergence, it follows that there exist some \(K \geq 1\) such that for all \(l > 0\), \(\Gamma^K(\mathcal{P})_M = \Gamma^{K+l}(\mathcal{P})_M\) (i.e., reached its convergent point). Then from above, we also have \(B^K(r^*, \eta^*) = B^{K+l}(r^*, \eta^*)\).

Then since this holds for all \(l > 0\), we must have \(B^K(r^*, \eta^*) = \emptyset\) for if \(B^K(r^*, \eta^*) \neq \emptyset\), then this will contradict the above result that there exist some \(l > 0\) for which \(B^K(r^*, \eta^*) \supset B^{K+l}(r^*, \eta^*)\) (i.e., which also means \(B^K(r^*, \eta^*) \neq B^{K+l}(r^*, \eta^*)\)). This ends the proof of Claim 3.

Therefore, by Claim 3, there exist some ordinal \(n' > n\) for which \(B^{n'}(r, \eta) = \emptyset\). Then at this stage (i.e., \(n')\), there would be no other pairs \((r', \eta')\) blocking \((r, \eta)\) from being applied. Now, by the monotonicity of \(\Gamma^n(\mathcal{P})_M\) for all \(n \geq 0\), we have \(\text{Pos}(r)\eta \subseteq \lambda_{M^0}(\Gamma^n(\mathcal{P})_M)\) and \(\text{Neg}(r)\eta \cap \mathcal{M} = \emptyset\). Then this implies \((r, \eta) \in \Gamma^{n+1}(\mathcal{P})_M\), which further implies \(\text{Head}(r)\eta \in \lambda_{M^0}(\Gamma^\infty(\mathcal{P})_M)\). This then contradicts the initial assumption \(\text{Head}(r)\eta \notin \lambda_{M^0}(\Gamma^\infty(\mathcal{P})_M)\). Therefore, we must have \(\mathcal{M}^{l+1}(\Pi) \subseteq \lambda_{M^0}(\Gamma^\infty(\mathcal{P})_M)\).

This completes our proof of \(\mathcal{M}^\infty(\Pi) \subseteq \lambda_{M^0}(\Gamma^\infty(\mathcal{P})_M)\). □

A.2.2 Proof of Theorem 6

(\(\Rightarrow\)) Suppose \(\mathcal{M}\) is a preferred answer set of \(\mathcal{P}\). According to Definitions 20 and 21, we construct a set \(S\) of propositional atoms from the expansion \(\mathcal{M}'\) of \(\mathcal{M}\) as described earlier, and show that \(S\) is a preferred answer set for the grounded preferred program \(\mathcal{P}^*\).
and \( S \cap \mathcal{M} = \mathcal{M} \). Let \( \mathcal{P}' = (\Pi', <^\mathcal{P}') \) be the tagged preferred program obtained from \( \mathcal{P} \), then the construction of \( S \) is as follows:

1. For each predicate \( P \in \tau(\Pi') \) other than \( = \), if \( P(a) \in P_{\mathcal{M}'} \), then \( P(a) \in S \) (here \( P(a) \) is treated as a propositional atom);

2. For each element \( a \in \text{Dom}(\mathcal{M}') \), propositional atom \( a = a \) is in \( S \) (note that \( \text{Dom}(\mathcal{M}) = \text{Dom}(\mathcal{M}') \));

3. \( S \) does not contain any other atoms.

Clearly, if we view all elements occurring \( \mathcal{M} \)'s relations as propositional atoms, we have \( S \cap \mathcal{M} = \mathcal{M}' \cap \mathcal{M} = \mathcal{M} \).

Now we show that \( S \) is a preferred answer set of \( \mathcal{P}^* \). For each rule \( r \in \Pi \), we denote \( r^T \in \Pi' \) to be the tagged rule obtained from \( r \) by adding the corresponding atom \( \text{Tag}(x) \) to \( r \)'s positive body (see earlier definition). Also note that since \( \text{Dom}(\mathcal{M}) = \text{Dom}(\mathcal{M}') \), each assignment \( \eta \) of \( \mathcal{M} \) is also an assignment of \( \mathcal{M}' \). Then from the definition of \( \mathcal{P}^* \) and where \( \mathcal{P}^* = (\text{Ground}(\Pi)_\mathcal{M}, <^{\mathcal{P}^*}) \), we can see that for each \( r \in \Pi \), assignment \( \eta \) of \( \mathcal{M} \) and \( t (t = 0, 1, \cdots) \), \( (r, \eta) \in \Gamma^i(\mathcal{P})_\mathcal{M} \) iff \( r^T \eta \in \Delta^i(\mathcal{P}^*)_S \). On the other hand, since the only extra rules in \( \Delta^\infty(\mathcal{P}^*)_S \) are those of the forms \( \text{Tag}_i(a) \leftarrow a = a \leftarrow \), and while \( S \) contains all atoms of the forms \( \text{Tag}_i(a) \) and \( a = a \), then \( S \) is exactly the set \( \{ \text{Head}(r^T \eta) | r^T \eta \in \Delta^\infty(\mathcal{P}^*)_S \} \cup \{ \text{Tag}_i(a), a = a \mid \text{ for all corresponding } i \text{'s, a's and a's} \}. \) This means that \( S \) is a preferred answer set of \( \mathcal{P}^* = (\text{Ground}(\Pi)_\mathcal{M}, <^{\mathcal{P}^*}) \).

The other direction can be proved with similar arguments. \( \square \)

### A.2.3 Proof of Lemma 3

(\( \implies \)) Suppose \( S \) is a preferred answer set of \( \mathcal{P} \). If \( r \in \Delta^\infty(\mathcal{P})_S \), then according to Definition 21, we know that \( \text{Pos}(r) \subseteq \text{Head}(\Delta^\infty(\mathcal{P})_S) \) and \( \text{Neg}(r) \cap S = \emptyset \). Since \( \text{Head}(\Delta^\infty(\mathcal{P})_S) = S \), this implies that \( r \) is a generating rule of \( S \). Now we assume that \( r \) is a generating rule of \( S \). We will show that \( r \in \Delta^\infty(\mathcal{P})_S \). We prove this result by induction on the sequence of rules under ordering \( <^\mathcal{P} \). Firstly, consider any rule \( r \in S \) where there does not exist any other rules \( r' \) in \( \Pi \) such that \( r' <^\mathcal{P} r \). According to Definition 21, \( r \in \Delta^0(\mathcal{P})_S \) if \( \text{Pos}(r) = \emptyset \), otherwise \( r \in \Delta^1(\mathcal{P})_S \). Now we assume that for all generating rules \( r' \) of \( S \) such that \( r' <^\mathcal{P} r \), \( r' \in \Delta^\infty(\mathcal{P})_S \). Suppose
that \( r \notin \Delta^\infty(\mathcal{P})_S \). Then for all \( t \), we have that \( r \notin \Delta^t(\mathcal{P})_S \). That is, for all \( t \), either (1) \( \text{Pos}(r) \not\subseteq \text{Head}(\Delta^{t-1}(\mathcal{P})_S) \) or \( \text{Neg}(r) \cap S \neq \emptyset \); or (2) there exist some \( r' \in \Pi \) such that \( r' <^P r \), \( r' \notin \Delta^{t-1}(\mathcal{P})_S \), \( \text{Pos}(r') \subseteq S \) and \( \text{Neg}(r') \cap \text{Head}(\Delta^{t-1}(\mathcal{P})_S) \). Since \( r \) is a generating rule of \( S \) and \( S = \text{Head}(\Delta^\infty(\mathcal{P})_S) \), it is obvious that case (1) cannot occur. So it has to be case (2). In this case, we can select a sufficient big \( t \) such that for any other \( t' \) where \( t' > (t - 1) \), no more rules from \( \Pi \) can be added into \( \Delta^t(\mathcal{P})_S \), i.e., \( \Delta^{t-1}(\mathcal{P})_S = \Delta^t(\mathcal{P})_S \) for all \( t' > (t - 1) \). Therefore, for this particular \( t \), we can find some \( r' \in \Pi \) such that \( r' <^P r \) and \( r' \notin \Delta^{t-1}(\mathcal{P})_S \). Since \( \text{Pos}(r') \subseteq S \) and \( \text{Neg}(r') \cap \text{Head}(\Delta^{t-1}(\mathcal{P})_S) = \text{Neg}(r') \cap \text{Head}(\Delta^\infty(\mathcal{P})_S) = \text{Neg}(r') \cap S = \emptyset \), it follows that \( r' \) is a generating rule of \( S \). Since according to our inductive assumption, we had that \( r' \in \Delta^\infty(\mathcal{P})_S \), then this is a contradiction.

\[ (\iff) \text{For each } r \in \Delta^\infty(\mathcal{P})_S, r \text{ is a generating rule of } S, \text{ so } \text{Pos}(r) \subseteq S \text{ and } \text{Neg}(r) \cap S = \emptyset. \text{ Also, since } S \text{ is an answer set of } \Pi, \text{ this follows } \text{Head}(r) \in S. \text{ So } \text{Head}(\Delta^\infty(\mathcal{P})_S) \subseteq S. \text{ Now we show } S \subseteq \text{Head}(\Delta^\infty(\mathcal{P})_S). \text{ Suppose this is not the case. Then there must exist some } \text{Head}(r) \in S \text{ such that } r \notin \Delta^\infty(\mathcal{P})_S. \text{ From the condition, we know that } r \text{ is a generating rule of } S. \text{ So there is a rule } r \in \Pi \text{ where } \text{Pos}(r) \subseteq S, \text{ Neg}(r) \cap S = \emptyset \text{ but } \text{Head}(r) \notin S. \text{ This is in contradiction with the fact that } S \text{ is an answer set of } \Pi. \text{ Therefore, we have } S = \text{Head}(\Delta^\infty(\mathcal{P})_S). \text{ So } S \text{ is an answer set of } \mathcal{P} = (\Pi, <^P). \square \]

### A.2.4 Proof of Lemma 4

We show that \( S \) is a preferred answer set of \( \mathcal{P} = (\Pi, <^P) \). We first prove that if \( r \in \Delta^\infty(\mathcal{P})_S \), then \( r \) is a generating rule of \( S \). We prove this by induction on \( t \). Consider \( \Delta^t(\mathcal{P})_S \). Since for all \( r \in \Delta^0(\mathcal{P})_S \), \( \text{Pos}(r) = \emptyset \) and \( \text{Neg}(r) \cap S = \emptyset \). This means that \( r \) is a generating rule of \( S \). Since \( S \) is an answer set of \( \Pi \), this implies \( \text{Head}(r) \in S \). Suppose for all \( t \) that \( r \in \Delta^t(\mathcal{P})_S \) implies \( r \) is a generating rule of \( S \), which implies \( \text{Head}(\Delta^t(\mathcal{P})_S) \subseteq S \). Now we consider \( \Delta^{t+1}(\mathcal{P})_S \). According to the definition, if \( r \notin \Delta^t(\mathcal{P})_S \) but \( r \in \Delta^{t+1}(\mathcal{P})_S \), then \( \text{Pos}(r) \subseteq \text{Head}(\Delta^t(\mathcal{P})_S) \) and \( \text{Neg}(r) \cap S = \emptyset \). From the induction assumption, we have \( \text{Pos}(r) \subseteq \text{Head}(\Delta^t(\mathcal{P})_S) \subseteq S \). So \( r \) is also a generating rule of \( S \), and hence \( \text{Head}(r) \in S \).

Now we show that under the given conditions of this lemma, if \( r \) is a generating rule
of \( S \), then \( r \in \Delta^\infty(\mathcal{P})_S \). We prove this by induction on \(<^\mathcal{P} \). Firstly, suppose there does not exist a rule \( r^* \in \Pi \) such that \( r^* <^\mathcal{P} r \) and \( r^* \) is a generating rule of \( S \). Since \( r \) is a generating rule and \( S \) is an answer set of \( \Pi \), it must be the case that \( \text{Pos}(r) = \emptyset \) and hence, that \( r \in \Delta^0(\mathcal{P})_S \). Now we assume that for all generating rules \( r^* \) such that \( r^* <^\mathcal{P} r \) and those \( r^* \) satisfy the condition of this lemma, that \( r^* \in \Delta^\infty(\mathcal{P})_S \). Now we consider \( r \).

Since for any \( r' \) where \( r <^\mathcal{P} r' \) and \( \text{Head}(r') \cap (\text{Pos}(r') \cup \text{Neg}(r')) \neq \emptyset \) we have that \( r' \) is not a generating rule of \( S \), then there must exist generating rules \( r_1, \ldots, r_k \) of \( S \) such that \( r_i <^\mathcal{P} r \) (\( i = 1, \ldots, k \)), \( \text{Pos}(r) \subseteq \bigcup_{i=1}^{k} \text{Head}(r_i) \) and \( \text{Neg}(r) \cap \bigcup_{i=1}^{k} \text{Head}(r_i) = \emptyset \).

Now, according to the induction assumption, these \( r_1, \ldots, r_k \) are in \( \Delta^\infty(\mathcal{P})_S \). So there exist some certain \( t \) for which we have \( \text{Pos}(r) \subseteq \text{Head}(\Delta^t(\mathcal{P})_S) \) and \( \text{Neg}(r) \cap S = \emptyset \) (this is due to the fact that \( r \) is a generating rule of \( S \)). Therefore, from the definition of \( \Delta^\infty(\mathcal{P})_S \) (see Definition 21), we know that if \( r \notin \Delta^\infty(\mathcal{P})_S \), then for all \( t \), there must exist a rule \( r' \) such that \( r' <^\mathcal{P} r \), \( r' \notin \Delta^{t-1}(\mathcal{P})_S \) and \( \text{Pos}(r') \subseteq S \) and \( \text{Neg}(r') \cap \text{Head}(\Delta^{t-1}(\mathcal{P})_S) \).

By selecting a sufficient big \( t \), we would have \( \Delta^t(\mathcal{P})_S = \Delta^\infty(\mathcal{P})_S \). This implies that there exist some \( r' \) such that \( r' <^\mathcal{P} r \), \( r' \notin \Delta^\infty(\mathcal{P})_S \) and \( r' \) is a generating rule of \( S \). This is in contradiction with our inductive assumption. So \( r \) must be in \( \Delta^t(\mathcal{P})_S \) for some \( t \). That is, \( r \in \Delta^\infty(\mathcal{P})_S \).

Finally, from Lemma 3, we know that \( S \) is also a preferred answer set of \( \mathcal{P} = (\Pi, <^\mathcal{P}) \).

This completes our proof. \( \square \)

### A.2.5 Proof of Theorem 10

To prove this theorem, we will use a result from [ZZ10] that \( \mathcal{M}^\infty(\Pi) = \mathcal{M} \) iff \( \mathcal{M} \) is an answer set of \( \Pi \). Thus, we prove the equivalent statement: \( \mathcal{M}^\infty(\Pi) = \mathcal{M} \) iff \( \mathcal{M} \models \exists \exists \forall S \varphi_{\Pi}(\exists \exists S, S) \).

\((\implies)\) Assume \( \mathcal{M}^\infty(\Pi) = \mathcal{M} \). For \( t \geq 0 \), define the operator

\[
(\mathcal{M}^*)^t(\Pi) : 2^{\Sigma(\Pi)_{\mathcal{M}}} \rightarrow 2^{\Sigma(\Pi)_{\mathcal{M}}}
\]

as

\[
(\mathcal{M}^*)^t(\Pi) = \{(r, \eta) \mid \text{Pos}(r) \eta \subseteq \mathcal{M}^{t+1}(\Pi) \text{ and } \text{Neg}(r) \eta \cap \mathcal{M} = \emptyset\}
\]
Then since $M^\infty(\Pi) = M$ by assumption, we also have that $(M^*)^\infty(\Pi) = \Gamma(\Pi)_M$. Now, for $t \geq 0$, set $\Lambda^t(\Pi)_M : 2^{\Sigma(\Pi)}_M \rightarrow 2^{\Sigma(\Pi)}_M$ to be an operator defined inductively as follows:

$$\Lambda^0(\Pi)_M = (M^*)^0(\Pi);$$
$$\Lambda^{t+1}(\Pi)_M = (M^*)^{t+1}(\Pi) \setminus (M^*)^t(\Pi).$$

Then clearly, we also have $\bigcup_{t \geq 0} \Lambda^t(\Pi)_M = (M^*)^\infty(\Pi)$. Moreover, by the definition of $\Lambda^t(\Pi)_M$, it is not difficult to see that the sets $\Lambda^0(\Pi)_M, \Lambda^1(\Pi)_M, \cdots, \Lambda^\infty(\Pi)_M$ partitions $(M^*)^\infty(\Pi)$. Now we construct a well-ordered relation $W = (\Gamma(\Pi)_M, \mathcal{V}_W)$ on the set $\Gamma(\Pi)_M$ as follows:

1. For each $\Lambda^t(\Pi)_M$, by the well-ordering theorem (every set can be well-ordered), there exist a well-ordering on the elements of $\Lambda^t(\Pi)_M$. Set such a well-ordering as $W_t = (\Lambda^t(\Pi)_M, \mathcal{V}_W)$.
2. Then we define the well-order $W = (\Gamma(\Pi)_M, \mathcal{V}_W)$ on $\Gamma(\Pi)_M$ by setting

$$\mathcal{V}_W = \bigcup_{t \geq 0} \mathcal{V}_W_t \cup \{(r_1, \eta_1), (r_2, \eta_2) | (r_1, \eta_1) \in \Lambda^t_1(\Pi)_M, (r_2, \eta_2) \in \Lambda^t_2(\Pi)_M, t_1 < t_2\}.$$ 

Incidentally, this simply follows from the fact that the sum and products of well-ordered types are themselves well-ordered [End77].

Now we denote by $\forall S \varphi_{\Pi}(S)$ as the sentence obtained from $\forall S \varphi_{\Pi}(\mathcal{V}, S)$ by treating the existentially quantified predicate variables in $\mathcal{V}$ as predicate constants. Then to show $\mathcal{M} \models \exists \mathcal{V} \forall S \varphi_{\Pi}(\mathcal{V}, S)$, we show that there exist an expansion $\mathcal{M}'$ of $\mathcal{M}$ such that $\mathcal{M}' \models \forall S \varphi_{\Pi}(S)$. In the following, we denote the treatment of each predicate variables, $\mathcal{V}_{r_{1\prime}r_2}$, as a predicate constant by simply denoting it as $\mathcal{V}_{r_{1\prime}r_2}$ (just removing the ‘\prime’ accent). Now set $\mathcal{M}'$ to be an expansion of $\mathcal{M}$ such that $\mathcal{M}'$ is of the extended signature $\tau(\Pi) \cup \{<_{r_{1\prime}r_2}\}$

\footnote{Note that $(M^*)^t(\Pi) \subseteq \Sigma(\Pi)_M$ and is not an actual $\tau$-structure like $M^t(\Pi)$.}

\footnote{Here $\infty$ denotes an arbitrary order-type.}
Proof of Claim 1: To prove this claim, it is sufficient to show the following:

\[
\langle \eta_1(u_1), \ldots, \eta_1(u_k), \eta_2(v_1), \ldots, \eta_2(v_l) \rangle \mid \langle u_1, \ldots, u_k \rangle = x_{r_1}, \langle v_1, \ldots, v_l \rangle = x_{r_2}, \text{and } (r_1, \eta_1) <^W (r_2, \eta_2) \}
\]

with \(W\) the well-order on \(\Gamma(\Pi)\), as defined above. Now, through the following Claims 1 and 2, we show \(\mathcal{M}' = \forall S \varphi_\Pi^{\text{PRO}}(S)\) and \(\mathcal{M}' = \varphi_\Pi^{\text{COMP}}\) respectively, so that \(\mathcal{M}' = \forall S \varphi_\Pi(S)\).

Claim 1: \(\mathcal{M}' = \forall S \varphi_\Pi^{\text{PRO}}(S)\).

Proof of Claim 1: To prove this claim, it is sufficient to show the following:

1. \(\mathcal{M}' = \bigwedge_{r_1, r_2, r_3 \in \Pi} \forall x_{r_1} x_{r_2} x_{r_3} (\langle r_1, r_2, r_3 \rangle (x_{r_1}, x_{r_2}) \wedge \langle r_2, r_3 \rangle (x_{r_2}, x_{r_3}) \rightarrow \langle r_1, r_3 \rangle (x_{r_1}, x_{r_3}))\);
2. \(\mathcal{M}' = \bigwedge_{r_1, r_2 \in \Pi} \forall x_{r_1} x_{r_2} (\langle r_1, r_2 \rangle (x_{r_1}, x_{r_2}) \rightarrow \neg \langle r_2, r_1 \rangle (x_{r_2}, x_{r_1}))\);
3. \(\mathcal{M}' = \bigwedge_{r_1, r_2 \in \Pi} \forall x_{r_1} x_{r_2} (\langle r_1, r_2 \rangle (x_{r_1}, x_{r_2}) \rightarrow \varphi_{r_1}^{\text{GEN}}(x_{r_1}) \wedge \varphi_{r_2}^{\text{GEN}}(x_{r_2}))\);
4. \(\mathcal{M}' = \bigwedge_{r \in \Pi} \forall x_r (\varphi_r^{\text{GEN}}(x_r) \rightarrow \varphi_r^{\text{SUP}}(x_r))\);
5. \(\mathcal{M}' = \forall S \varphi_\Pi^{\text{WELLOr}}(S)\),

Now we show each of the above statements.

1. \(\mathcal{M}' = \bigwedge_{r_1, r_2, r_3 \in \Pi} \forall x_{r_1} x_{r_2} x_{r_3} (\langle r_1, r_2, r_3 \rangle (x_{r_1}, x_{r_2}) \wedge \langle r_2, r_3 \rangle (x_{r_2}, x_{r_3}) \rightarrow \langle r_1, r_3 \rangle (x_{r_1}, x_{r_3}))\):

Suppose for some assignment \(u\), we have \(\mathcal{M}' \models \langle r_1, r_2 \rangle (a_{r_1}, a_{r_2}) \wedge \langle r_1, r_2 \rangle (a_{r_2}, a_{r_3})\) such that \(a_{r_1}, a_{r_2}\) and \(a_{r_3}\) denotes the tuples obtained from \(x_{r_1}\), \(x_{r_2}\) and \(x_{r_3}\) respectively by replacing each of the variable \(x\) in \(x_r\) with \(u(x)\). We now show \(\mathcal{M}' \models \langle r_1, r_2 \rangle (a_{r_1}, a_{r_3})\). From the definition of \(\langle r_1, r_2 \rangle\), we have that there exist \((r_1, \eta_1) \in \Gamma(\Pi)\) and \((r_2, \eta_2) \in \Gamma(\Pi)\) with \((r_1, \eta_1) <^W (r_2, \eta_2)\) such that \(\langle u_1, \ldots, u_k \rangle = x_{r_1}\) and \(\langle v_1, \ldots, v_l \rangle = x_{r_2}\), then \(\langle \eta_1(u_1), \ldots, \eta_1(u_k) \rangle = a_{r_1}\) and \(\langle \eta_2(v_1), \ldots, \eta_2(v_l) \rangle = a_{r_2}\). Similarly, by the definition of \(\langle r_2, r_3 \rangle\), there exist \((r_2, \eta_2') \in \Gamma(\Pi)\) and \((r_3, \eta_3) \in \Gamma(\Pi)\) with \((r_2, \eta_2') <^W (r_3, \eta_3)\) such that \(\langle w_1, \ldots, w_m \rangle = x_{r_3}\) then \(\langle \eta_2'(v_1), \ldots, \eta_2'(v_l) \rangle = a_{r_2}\) and \(\langle \eta_3(w_1), \ldots, \eta_3(w_m) \rangle = a_{r_3}\). Then, since \(\langle \eta_2'(v_1), \ldots, \eta_2'(v_l) \rangle = \langle \eta_2(v_1), \ldots, \eta_2(v_l) \rangle\), we must have...
\[ \eta'_2 = \eta_2. \] Then we have \((r_1, \eta_1) \prec_W (r_3, \eta_3)\) by transitivity since \((r_2, \eta_2) = (r_2, \eta'_2)\). Then from the definition of \(<_{r_1r_3}^{M'}\), it follows that \(\langle a_{r_1}, a_{r_3} \rangle \in <_{r_1r_3}^{M'}\), and hence, that \(M' \models <_{r_1r_3} \langle a_{r_1}, a_{r_3} \rangle\).

2. \(M' \models \bigwedge_{r_1,r_2 \in \Pi} \forall x_{r_1} x_{r_2} (<_{r_1r_2} \langle x_{r_1}, x_{r_2} \rangle \rightarrow \varphi^{\text{GEN}}_{r_1}(x_{r_1}) \land \varphi^{\text{GEN}}_{r_2}(x_{r_2}))\):

On the contrary, assume for some assignment \(\nu\), we have \(M' \models <_{r_1r_2} \langle a_{r_1}, a_{r_2} \rangle\) \land \(<_{r_2r_1} \langle a_{r_2}, a_{r_1} \rangle\). Then we also have \(<_{r_1r_1} \langle a_{r_1}, a_{r_1} \rangle\) by the transitivity axiom (which was already shown above to be satisfied by \(M'\)). Then by the definition of \(<_{r_1r_1}^{M'}\), there exist \((r_1, \eta_1) \in \Lambda^1(\Pi)_M\) and \((r_1, \eta_2) \in \Lambda^2(\Pi)_M\) with \(t_1 < t_2\) such that if \(\langle u_1, \ldots, u_k \rangle = x_{r_1}\), we have \(\langle \eta_1(u_1), \ldots, \eta_1(u_k) \rangle = a_{r_1}\) and \(\langle \eta_2(u_1), \ldots, \eta_2(u_k) \rangle = a_{r_1}\). Then this implies \(\eta_1 = \eta_2\), and since there is a unique \(t\) for which \((r_1, \eta) \in \Lambda^t(\Pi)_M\) for each \((r_1, \eta)\), then it must also be that \(t_1 = t_2\). Then this is a contradiction since \(t_1 < t_2\).

3. \(M' \models \bigwedge_{r_1, r_2 \in \Pi} \forall x_r (\varphi^{\text{GEN}}_r(x_r) \rightarrow \varphi^{\text{SUP}}_r(x_r))\):

This follows from the interpretations \(<_{r_1r_2}^{M'}\) of the predicates \(<_{r_1r_2}\) (for \(r_1, r_2 \in \Pi\)) and where it is a “representation” of the well-order \(W = (\Gamma(\Pi)_M, \prec_W)\) on \(\Gamma(\Pi)_M\).

4. \(M' \models \bigwedge_{r \in \Pi} \forall x_r (\varphi^{\text{GEN}}_r(x_r) \rightarrow \varphi^{\text{SUP}}_r(x_r))\):

Suppose for some assignment \(\nu\), we have \(M' \models \varphi^{\text{GEN}}_r(a_r)\) such that \(a_r\) is the tuple obtained from \(x_r\) via \(\nu\) as above. Then we show

\[ M' \models \bigwedge_{P(a_P) \in \text{Pos}(r), \ P \in \mathcal{P}_{\text{int}(\Pi)}} \bigvee_{r' \in \Pi, \ H_{\text{head}}(r') = P(y_p)} \exists x_{r'} (<_{r'r'} (x_{r'}, a_r) \land a_P = y_P) \]

Now, since \(M' \models \varphi^{\text{GEN}}_r(a_r)\), then by the definition of \(M'\), it also follows that \(M' \models \varphi^{\text{GEN}}_r(a_r)\) (since \(\varphi^{\text{GEN}}_r(x_r)\) only involves those symbols occurring in \(\tau(\Pi)\)). Then there exist an assignment \(\eta\) such that with
\[ \langle u_1, \ldots, u_k \rangle = x_r, \text{ we have } \langle \eta(u_1), \ldots, \eta(u_k) \rangle = a_r \text{ and where } \text{Pos}(r) \eta \subseteq M \text{ and } \text{Neg}(r) \cap M = \emptyset. \text{ Moreover, there must be the least stage } t \text{ for which } \text{Pos}(r) \eta \subseteq M'(\Pi). \text{ Now let } P(a_p) \in \text{Pos}(r) \eta \text{ where } P \in \mathcal{P}_{\text{int}}(\Pi). \text{ Then there must also be some least stage } t' \text{ such that for some rule } r' \text{ and corresponding assignment } \eta', \text{ we have Head}(r') \eta' = P(a_p), \text{ Pos}(r') \eta' \subseteq M'(\Pi) \text{ and } \text{Neg}(r') \eta' \cap M = \emptyset (i.e., the least stage that derives } P(a_p)). \text{ Moreover, since } P(a_p) \in M'(\Pi) \text{ (i.e., since } P(a_p) \in \text{Pos}(r) \eta \subseteq M'(\Pi)), \text{ then } t' < t \text{ for if } t = t', \text{ this will contradict the assumption that } P(a_p) \in M'(\Pi) \text{ since this implies } P(a_p) \notin M'(\Pi) \text{ (i.e., as } t' \text{ is the least stage that derives } P(a_p)). \text{ Then by the definitions of } \Lambda^{t+1}_M(\Pi) \text{ and } \Lambda^{t+1}_M(\Pi), \text{ we have } \langle r', \eta' \rangle \in \Lambda^{t+1}_M(\Pi) \text{ and } (r, \eta) \in \Lambda^{t+1}_M(\Pi). \text{ Then as } t' + 1 < t + 1, \text{ by the definition of } <_W, \text{ we also have } \langle r', \eta' \rangle <_W (r, \eta). \text{ Then by the interpretation of } <_M, \text{ if we let } a_{r'} = \langle \eta'(v_1), \ldots, \eta'(v_l) \rangle \text{ such that } \langle v_1, \ldots, v_l \rangle = x_r', \text{ then } \langle a_{r'}, a_r \rangle \in <_M. \text{ Then this implies that } M' \models <_{r_r} (a_{r'}, a_r) \wedge a_p = a_p \text{ (where } a_p \text{ is the projection under } y_p \text{ from the tuple } a_{r'}). \]

5. \( M' \models \forall S^{\text{WELLOR}}(\Pi) \). That is, we show that \( M' \) satisfies the SO sentence

\[
\forall S(( \bigwedge_{r \in \Pi} \forall x_r(\tilde{S}_r(x_r) \rightarrow \varphi^\text{GEN}_r(x_r))) \wedge ( \bigvee_{r \in \Pi} \exists x_r \tilde{S}_r(x_r)) \rightarrow \\
\bigvee_{r' \in \Pi} \exists x_{r'}(\tilde{S}_{r'}(x_{r'}) \wedge \forall y_{r'}(\tilde{S}_{r'}(y_{r'}) \wedge y_{r'} \neq x_{r'} \rightarrow <_{r_{r'}} (x_{r'}, y_{r'})) \\
\wedge \bigwedge_{r'' \notin \Pi, r' \neq r''} \forall x_{r''}(\tilde{S}_{r''}(x_{r''}) \rightarrow <_{r_{r''}} (x_{r''}, x_{r''})))):
\]

This simply follows from the assumption that \( W \) is a well-ordering on \( \Gamma(\Pi)_M \) since a well-order implies that each non-empty subset contains a least element. Indeed, \( \bigwedge_{r \in \Pi} \forall x_r(\tilde{S}_r(x_r) \rightarrow \varphi^\text{GEN}_r(x_r)) \) encodes the condition that we only consider tuples corresponding to the generating rules in the well-order, \( \bigvee_{r \in \Pi} \exists x_r \tilde{S}_r(x_r) \) encodes the condition that the subset of \( \Gamma(\Pi)_M \) that we are
APPENDIX A. PROOFS

considering is non-empty, and lastly,

\[ \bigvee_{r' \in \Pi} \exists x_{r'}(\overline{S}_{r'}(x_{r'}) \land \forall y_{r'}(\overline{S}_{r'}(y_{r'}) \land y_{r'} \neq x_{r'} \rightarrow \rho_{r',r'}(x_{r'}, y_{r'})) \]

\[ \land \bigwedge_{r'' \in \Pi, r'' \neq r'} \forall x_{r''}(\overline{S}_{r''}(x_{r''}) \rightarrow \rho_{r'',r'}(x_{r'}, x_{r''}))) \]

encodes the condition that for such a (non-empty) subset, a least element exists.

This ends the proof of Claim 1.

Claim 2: \( \mathcal{M}' \models \varphi_{\Pi}^{\text{COMP}} \).

Proof of Claim 2: From [ZZ10], \( \mathcal{M}^{\infty}(\Pi) = \mathcal{M} \models SM(\Pi) \), where \( SM(\Pi) \) is the SO sentence (1.5). Moreover, if \( \mathcal{M} \models SM(\Pi) \), then \( \mathcal{M} \models \varphi_{\Pi}^{\text{COMP}} \). This ends the proof of Claim 2.

So we have shown that \( \mathcal{M} \models \exists \overline{\Sigma} \forall S_{\varphi_{\Pi}}(\overline{\Sigma}, S) \).

(\( \iff \)) Now let us assume \( \mathcal{M} \models \exists \overline{\Sigma} \forall S_{\varphi_{\Pi}}(\overline{\Sigma}, S) \). Then for some expansion \( \mathcal{M}' \) of \( \mathcal{M} \) of the extended signature \( \tau(\Pi) \cup \{\rho_{r_1,r_2} \mid r_1, r_2 \in \Pi\} \), we have \( \mathcal{M}' \models \forall S_{\varphi_{\Pi}}(S) \) where \( \forall S_{\varphi_{\Pi}}(S) \) is the sentence obtained from \( \forall S_{\varphi_{\Pi}}(\overline{\Sigma}, S) \) by simply treating the predicate variables in \( \overline{\Sigma} \) as predicate constants. To show \( \mathcal{M}^{\infty}(\Pi) = \mathcal{M} \), we show \( \mathcal{M}^{\infty}(\Pi) \subseteq \mathcal{M} \) and \( \mathcal{M} \subseteq \mathcal{M}^{\infty}(\Pi) \). First we show \( \mathcal{M}^{\infty}(\Pi) \subseteq \mathcal{M} \) by induction.

Basis Clearly, \( \mathcal{M}^0(\Pi) \subseteq \mathcal{M} \) by the definition of \( \mathcal{M}^0(\Pi) \) (i.e., only considering the interpretations of the external predicates of \( \Pi \)).

Step Assume that for \( t' \leq t \), we have \( \mathcal{M}^{t'}(\Pi) \subseteq \mathcal{M} \).

Then let \( P(a_P) \in \mathcal{M}^{t+1}(\Pi) \) such that \( P(a_P) \notin \mathcal{M}^t(\Pi) \) (i.e., for if \( P(a_P) \in \mathcal{M}^t(\Pi) \) then the result is clear by the inductive hypothesis). We will now show \( P(a_P) \in \mathcal{M} \).

Indeed, since \( P(a_P) \in \mathcal{M}^{t+1}(\Pi) \) with \( P(a_P) \notin \mathcal{M}^t(\Pi) \), then there exist a rule \( r \in \Pi \) and an assignment \( \eta \) such that \( Head(r)\eta = P(a_P) \), \( Pos(r)\eta \subseteq \mathcal{M}^t(\Pi) \) and \( Neg(r)\eta \cap \mathcal{M} = \emptyset \). Then since \( \mathcal{M}^t(\Pi) \subseteq \mathcal{M} \) by assumption, \( \mathcal{M}^{t}(\Pi) = \mathcal{M} \) and \( \mathcal{M} \models \varphi_{\Pi}^{\text{COMP}} \) (i.e., since \( \varphi_{\Pi}^{\text{COMP}} \) only involves those symbols in \( \tau(\Pi) \)), it follows that \( P(a_P) \in \mathcal{M} \).
Thus we have shown $\mathcal{M}^\infty(\Pi) \subseteq \mathcal{M}$. Next we show $\mathcal{M} \subseteq \mathcal{M}^\infty(\Pi)$. Indeed, since $\mathcal{M}' \models \forall S \varphi_{\Pi}^{\text{PRO}}(S)$, set the well-order $\mathcal{W} = (\text{Dom}(\mathcal{W}), <^\mathcal{W})$ such that:

\[
\text{Dom}(\mathcal{W}) = \Gamma(\Pi), \quad <^\mathcal{W} = \{(r_1, \eta_1), (r_2, \eta_2) \mid r_1, r_2 \in \Pi, \eta_1 \text{ and } \eta_2 \text{ are assignments such that if } x_{r_1} = \langle u_1, \ldots, u_k \rangle \text{ and } x_{r_2} = \langle v_1, \ldots, v_l \rangle \text{ then } \langle \eta_1(u_1), \ldots, \eta_1(u_k), \eta_2(v_1), \ldots, \eta_2(v_l) \rangle <^\mathcal{W} \}
\]

Now, for an element $e \in \text{Dom}(\mathcal{W})$, we define the operator $\mathcal{W}^e(\Pi)$ inductively as follows:

\[
\mathcal{W}^\text{BOT}(\mathcal{W})(\Pi) = \{\text{Head}(r)\eta \mid (r, \eta) = \text{BOT}(\mathcal{W})\};
\]

\[
\mathcal{W}^\text{SUCC}(\mathcal{W})(\Pi) = \mathcal{W}^e(\Pi) \cup \{\text{Head}(r)\eta \mid (r, \eta) = \text{SUCC}(e)\},
\]

where: (1) $\text{BOT}(\mathcal{W})$ denotes the least element of $\text{Dom}(\mathcal{W})$ under the well-order $\mathcal{W}$; (2) $\text{SUCC}(e)$ the successor element of $e$ under $\mathcal{W}$; (3) and $\text{ORD}(\mathcal{W})$, the order type of $\mathcal{W}$ (i.e., that is, $\text{ORD}(\mathcal{W})$ is equal to the size of $\text{Dom}(\mathcal{W})$). As $\mathcal{W}$ is a well-order on $\text{Dom}(\mathcal{W})$, we use transfinite induction [End77] on the set $\text{Dom}(\mathcal{W})$ to show that $\mathcal{W}^\text{ORD}(\mathcal{W})(\Pi) \subseteq \mathcal{M}^\infty(\Pi)$.

**Basis** Without loss of generality, assume $\text{BOT}(\mathcal{W}) = (r, \eta)$. Then since $\mathcal{M}' \models \forall S \varphi_{\Pi}^{\text{PRO}}(S)$, there can only be two possibilities:

**Case 1** $\text{Pred}(\text{Pos}(r)) \cap \mathcal{P}_{\text{int}}(\Pi) = \emptyset$ (i.e., no intensional predicates).

In this case, since $(r, \eta) \in \text{Dom}(\mathcal{W}) = \Gamma(\Pi), \mathcal{M}$ (a generating rule) and $\mathcal{M} \models \varphi_{\Pi}^{\text{COMP}}$ (which implies that $\mathcal{M}' \upharpoonright \tau(\Pi) = \mathcal{M}$ is logically closed under $\Pi$), then it follows from the definition of $\mathcal{M}^1(\Pi)$ that $\text{Head}(r)\eta \in \mathcal{M}^1(\Pi) \subseteq \mathcal{M}^\infty(\Pi)$.

Hence, by the definition of $\mathcal{W}^\text{BOT}(\mathcal{W})(\Pi)$, we have $\mathcal{W}^\text{BOT}(\mathcal{W})(\Pi) = \{\text{Head}(r)\eta\} \subseteq \mathcal{M}^\infty(\Pi)$.

**Case 2** $\text{Pred}(\text{Pos}(r)) \cap \mathcal{P}_{\text{int}}(\Pi) \neq \emptyset$.

Then since $(r, \eta) \in \Gamma(\Pi), \mathcal{M}$ and

\[
\mathcal{M}' \models \bigwedge_{r \in \Pi} \forall x_r (\varphi_r^{\text{GEN}}(x_r) \rightarrow \varphi_r^{\text{SUP}}(x_r))
\]
APPENDIX A. PROOFS

160

(i.e., which obeys the notion of a support for each intensional predicate instance in the positive body from a preceding element), then this contradicts the assumption \((r, \eta)\) is the bottom element of \(\operatorname{Dom}(W)\) under \(<^W\). Therefore, we cannot have \(\operatorname{Pred}(\operatorname{Pos}(r)) \cap \mathcal{P}_{\text{int}}(\Pi) \neq \emptyset\).

**Step** Assume for \(\text{BOT}(W) \leq^W e' \leq^W e\), we have \(\mathcal{W}^{e'}(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\).

We show \(\mathcal{W}^{\text{SUCC}(e)}(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\). Hence, assume that \(\text{SUCC}(e) = (r, \eta)\). In a similar manner to the base case, there can only be two possibilities:

**Case 1** \(\operatorname{Pred}(\operatorname{Pos}(r)) \cap \mathcal{P}_{\text{int}}(\Pi) = \emptyset\).

Then in a similar manner to Case 1 of the Basis above, it follows that \(\text{Head}(r) \eta \in \mathcal{M}^1(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\), which implies \(\mathcal{W}^{\text{SUCC}(e)}(\Pi) = \mathcal{W}^e(\Pi) \cup \{\text{Head}(r) \eta\} \subseteq \mathcal{M}^\infty(\Pi)\).

**Case 2** \(\operatorname{Pred}(\operatorname{Pos}(r)) \cap \mathcal{P}_{\text{int}}(\Pi) \neq \emptyset\).

Then as \(\mathcal{M}' |\begin{array}{c}
\bigwedge_{r \in \Pi} \forall x_r (\varphi^\text{GEN}_r(x_r) \rightarrow \varphi^\text{SUP}_r(x_r))
\end{array}\)

(each intensional predicate instances in the positive body is supported by a preceding element), we have \(\operatorname{Pos}(r) \eta \subseteq \mathcal{W}^e(\Pi)\) by the definition of \(\mathcal{W}^e(\Pi)\). Hence, as \(\mathcal{W}^e(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\) by assumption, then \(\operatorname{Pos}(r) \eta \subseteq \mathcal{M}^\infty(\Pi)\). Thus, there must be the least stage \(t\) for which \(\operatorname{Pos}(r) \eta \subseteq \mathcal{M}^t(\Pi)\). Then by the definition of \(\mathcal{M}^{t+1}(\Pi)\) and as \((r, \eta) \in \Gamma(\Pi), M\) (i.e., which implies \(\text{Neg}(r) \eta \cap \mathcal{M} = \emptyset\) as \((r, \eta)\) is a generating rule under \(\mathcal{M}\), it follows that \(\text{Head}(r) \eta \in \mathcal{M}^{t+1}(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\) and hence, that \(\mathcal{W}^{\text{SUCC}(e)}(\Pi) = \mathcal{W}^e(\Pi) \cup \{\text{Head}(r) \eta\} \subseteq \mathcal{M}^\infty(\Pi)\).

Thus, to show \(\mathcal{M} \subseteq \mathcal{M}^\infty(\Pi)\), it will now be sufficient to only show that \(\mathcal{M} \subseteq \mathcal{W}^{\text{ORD}(W)}(\Pi) \cup \mathcal{M}^1(\Pi)\) holds since \(\mathcal{W}^{\text{ORD}(W)}(\Pi) \cup \mathcal{M}^1(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\) (i.e., since we had already verified that \(\mathcal{W}^{\text{ORD}(W)}(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\) and where \(\mathcal{M}^1(\Pi) \subseteq \mathcal{M}^\infty(\Pi)\)). Thus, let \(P(a_P) \in \mathcal{M}\) such that \(P \in \mathcal{P}_{\text{int}}(\Pi)\) (i.e., for if \(P \notin \mathcal{P}_{\text{int}}(\Pi)\), then it immediately follows that \(P(a_P) \in \mathcal{M}^0(\Pi) \subseteq \mathcal{M}^1(\Pi)\)). As \(\mathcal{M} \models \varphi^\text{COMP}\) implies \(\mathcal{M} \models \varphi^\text{COMP}\) (i.e., since \(\varphi^\text{COMP}\) only involves the symbols occurring in \(\tau(\Pi)\)), then for some rule \(r \in \Pi\) and assignment \(\eta\), we have \(\text{Head}(r) \eta = P(a_P), \operatorname{Pos}(r) \eta \subseteq \mathcal{M}\) and \(\operatorname{Neg}(r) \eta \cap \mathcal{M} = \emptyset\). Now, about the rule \(r\), there can only be two possibilities:
Case 1 \( \text{Pred}(\text{Pos}(r)) \cap \mathcal{P}_{\text{int}}(\Pi) = \emptyset \).

Then by the definition of \( \mathcal{M}^1(\Pi) \), we have \( \text{Head}(r)\eta = P(a_P) \in \mathcal{M}^1(\Pi) \subseteq \mathcal{W}^{\text{ORD}(W)}(\Pi) \cup \mathcal{M}^1(\Pi) \).

Case 2 \( \text{Pred}(\text{Pos}(r)) \cap \mathcal{P}_{\text{int}}(\Pi) \neq \emptyset \).

Then as \((r, \eta) \in \Gamma(\Pi)_{\mathcal{M}} \) and since \( \text{Dom}(W) = \Gamma(\Pi)_{\mathcal{M}} \), it is clear that \((r, \eta) \in \text{Dom}(W)\). Hence, by the definition of \( \mathcal{W}^{\text{ORD}(W)}(\Pi) \), we have \( P(a_P) \in \mathcal{W}^{\text{ORD}(W)}(\Pi) \subseteq \mathcal{W}^{\text{ORD}(W)}(\Pi) \cup \mathcal{M}^1(\Pi) \).

Therefore, we have \( \mathcal{M} \subseteq \mathcal{M}^\infty(\Pi) \). This finally completes the proof of this theorem. \( \square \)

A.2.6 Proof of Theorem 11

(\( \implies \)) Assume \( \lambda_{\mathcal{M}^0}(\Gamma^\infty(\mathcal{P})_{\mathcal{M}}) = \mathcal{M} \) (i.e., \( \mathcal{M} \) is a preferred answer set of \( \mathcal{P} = (\Pi, <^P) \)).

We show \( \mathcal{M} \models \exists \exists \forall S \varphi_P(z, S) \) in two steps:

1. Given \( \lambda_{\mathcal{M}^0}(\Gamma^\infty(\mathcal{P})_{\mathcal{M}}) = \mathcal{M} \), we show there exist a well-order \( W = (\Gamma(\Pi)_{\mathcal{M}}, <^W) \) on the set \( \Gamma(\Pi)_{\mathcal{M}} \) called the preference preserving well-order, satisfying the following conditions. For each \((r, \eta) \in \Gamma(\Pi)_{\mathcal{M}}\):

   a. \( \text{Pos}(r) \eta \subseteq \mathcal{M}^0(\Pi) \cup \{ \text{Head}(r')\eta' | (r', \eta') <^W (r, \eta) \} \);

   b. for each rule \( r' <^P r \) and assignment \( \eta' \):

      i. \( (r', \eta') \in \Gamma(\Pi)_{\mathcal{M}} \) implies \( (r', \eta') <^W (r, \eta) \);

      ii. \( (r', \eta') \notin \Gamma(\Pi)_{\mathcal{M}} \) implies that either:

         A. \( \text{Pos}(r')\eta' \not\subseteq \mathcal{M} \) or

         B. \( \text{Neg}(r')\eta' \cap (\mathcal{M}^0(\Pi) \cup \{ \text{Head}(r'')\eta'' | (r'', \eta'') <^W (r, \eta) \}) \neq \emptyset \);

2. Based on the well-order \( W \) on \( \Gamma(\Pi)_{\mathcal{M}} \), we construct an expansion \( \mathcal{M}' \) of \( \mathcal{M} \) and show that \( \mathcal{M}' \models \forall S \varphi_P(S) \) where \( \forall S \varphi_P(S) \) is the sentence obtained from \( \mathcal{M} \models \forall S \varphi_P(z, S) \) by treating the predicate variables in \( z \) as constants.

Now for the first part, we start with the following claim.

**Claim 1**: \( \Gamma^\infty(\mathcal{P})_{\mathcal{M}} = \Gamma(\Pi)_{\mathcal{M}} \). That is, \( \Gamma^\infty(\mathcal{P})_{\mathcal{M}} \) contains exactly the generating rules.
Proof of Claim 1: Since \( M \) is a preferred answer set of \( P \), then by Theorem 7, \((r, \eta) \in \Gamma^\infty(P)_M\) iff \( r \) is a generating rule of \( M \) under \( \eta \). This completes the proof of Claim 1.

Now, based on \( \Gamma^\infty(P)_M \), we show by induction on \( t \) for \( t \geq 0 \) that there exist a well-order \( \mathcal{W} = (\Gamma^t(P)_M, <^\mathcal{W}) \) on \( \Gamma^t(P)_M \) with the following properties. For each \((r, \eta) \in \Gamma^t(P)_M\):

1. \( Pos(r) \eta \subseteq M^0(\Pi) \cup \{ Head(r') \eta' \mid (r', \eta') <^W (r, \eta) \} \);

2. for each rule \( r' <^P r \) and assignment \( \eta' \):
   a. \( (r', \eta') \in \Gamma^t(P)_M \) implies \( (r', \eta') <^W (r, \eta) \);
   b. \( (r', \eta') \notin \Gamma^t(P)_M \) implies that either:
      i. \( Pos(r') \eta' \not\subseteq M \) or
      ii. \( Neg(r') \eta' \cap (M^0(\Pi) \cup \{ Head(r'') \eta'' \mid (r'', \eta'') <^W (r, \eta) \}) \neq \emptyset \).

Thus ultimately, since \( \Gamma^\infty(P)_M = \Gamma(\Pi)_M \) by Claim 1, then we would have showed the first part.

Basis By the well-ordering theorem [End77] (which states that every set can be well-ordered), there exist a well-order \( \mathcal{W} = (\Gamma^0(P)_M, <^\mathcal{W}) \) on \( \Gamma^0(P)_M \). Moreover, due to the finiteness of \( \Pi \), it should not be too difficult to see that we can make the well-order \( \mathcal{W} \) in such a way that for each \((r, \eta), (r', \eta') \in \Gamma^0(P)_M \), \( r <^P r' \) implies \( (r, \eta) <^W (r', \eta') \). To see this, note that we can partition \( \Gamma^0(\Pi)_M \) into a sequence of sets \( S_{r_1}, \ldots, S_{r_n} \) where \( r_i <^P r_j \) implies \( i < j \) and such that \((r, \eta) \in S_{r_i} \) implies \( r = r_i \). Then by the well-ordering theorem, there exist a well-order \( \mathcal{W}_i = (S_{r_i}, <^{W_i}) \) of each of the set \( S_{r_i} \). Thus, we can just define the well-order \( \mathcal{W} = (\Gamma^0(P)_M, <^W) \) by setting \( <^W = (\bigcup_{1 \leq i \leq n} <^{W_i}) \cup \{(r_i, \eta_i), (r_j, \eta_j) \mid (r_i, \eta_i) \in S_{r_i}, (r_j, \eta_j) \in S_{r_j}, i < j \} \). This simply follows from the fact that the sum and products of well-ordered types are also well-ordered [End77]. Hence, assume \( \mathcal{W} \) to be such a well-order. We now make the following claims:

Claim 2 \((r, \eta) \in \Gamma^0(P)_M \) implies \( Pos(r) \eta \subseteq M^0(\Pi) \cup \{ Head(r') \eta' \mid (r', \eta') <^W (r, \eta) \} \).
Proof of Claim 2 Clearly by the definition of $\Gamma^0(\mathcal{P})_\mathcal{M}$, $(r, \eta) \in \Gamma^0(\mathcal{P})_\mathcal{M}$ implies $\text{Pos}(r)\eta \subseteq \mathcal{M}^0(\Pi)$. This completes the proof of Claim 2.

Claim 3 If $(r, \eta) \in \Gamma^0(\mathcal{P})_\mathcal{M}$, then $(r', \eta') \in \Gamma^0(\mathcal{P})_\mathcal{M}$, where $r' \prec \prec \eta$ implies $(r', \eta') \prec^W (r, \eta)$.

Proof of Claim 3 Follows from the description of $\mathcal{W}$. This ends the proof of Claim 3.

Claim 4 If $(r, \eta) \in \Gamma^0(\mathcal{P})_\mathcal{M}$, then $(r', \eta') \notin \Gamma(\Pi)_\mathcal{M}$, where $r' \prec \prec \eta$ implies that either:

1. $\text{Pos}(r')\eta' \notin \mathcal{M}$ or
2. $\text{Neg}(r')\eta' \cap (\mathcal{M}^0(\Pi) \cup \{\text{Head}(r'')\eta'' \mid (r'', \eta'') \prec^W (r, \eta)\}) \neq \emptyset$.

Proof of Claim 4 On the contrary, assume that $\text{Pos}(r')\eta' \subseteq \mathcal{M}$ and $\text{Neg}(r')\eta' \cap (\mathcal{M}^0(\Pi) \cup \{\text{Head}(r'')\eta'' \mid (r'', \eta'') \prec^W (r, \eta)\}) = \emptyset$. Then by the latter, we also have $\text{Neg}(r')\eta' \cap \mathcal{M}^0(\Pi) = \emptyset$. Then this contradicts the assumption $(r, \eta) \in \Gamma^0(\mathcal{P})_\mathcal{M}$ since by the definition of $\Gamma^0(\mathcal{P})_\mathcal{M}$, we have $(r', \eta')$ will be blocking $(r, \eta)$ from being applied. This completes the proof of Claim 4.

Step Assume for $1 \leq t' \leq t$ the hypothesis holds.

We will now show it also holds for $t + 1$. Indeed, by the inductive hypothesis, there exist a well-order $\mathcal{W} = (\Gamma^t(\mathcal{P})_\mathcal{M}, \prec^W)$ on $\Gamma^t(\mathcal{P})_\mathcal{M}$ satisfying the conditions of the preference preserving well-order. Moreover, by the well-ordering theorem, there also exists a well-order $\mathcal{W}' = (\Lambda^{t+1}(\mathcal{P})_\mathcal{M}, \prec^W)$ on $\Lambda^{t+1}(\mathcal{P})_\mathcal{M}$ such that $\Lambda^{t+1}(\mathcal{P})_\mathcal{M} = \Gamma^{t+1}(\mathcal{P})_\mathcal{M} \setminus \Gamma^t(\mathcal{P})_\mathcal{M}$. Furthermore, due to the finiteness of $\Pi$, it is not difficult to see that we can further define $\mathcal{W}'$ in such a way that for each $(r, \eta), (r', \eta') \in \Lambda^{t+1}(\mathcal{P})_\mathcal{M}$, $r \prec \prec \eta$ implies $(r, \eta) \prec^W (r', \eta')$ (i.e., same argument as above). Now we specify $\mathcal{W}'' = (\Gamma^{t+1}(\mathcal{P})_\mathcal{M}, \prec^W)$ to be a well-order on $\Gamma^{t+1}(\mathcal{P})_\mathcal{M}$ by setting $\prec^W \subseteq \prec^W \cup \prec^W \cup (\Gamma^t(\mathcal{P})_\mathcal{M} \times \Lambda^{t+1}(\mathcal{P})_\mathcal{M})$.

Claim 5 $(r, \eta) \in \Gamma^{t+1}(\mathcal{P})_\mathcal{M}$ implies $\text{Pos}(r)\eta \subseteq \mathcal{M}^0(\Pi) \cup \{\text{Head}(r')\eta' \mid (r', \eta') \prec^W (r, \eta)\})$.

Proof of Claim 5 We consider the possibilities:

Case 1 $(r, \eta) \in \Gamma^t(\mathcal{P})_\mathcal{M}$.

Then by the inductive hypothesis, we have $\text{Pos}(r)\eta \subseteq \mathcal{M}^0(\Pi) \cup \{\text{Head}(r')\eta' \mid (r', \eta') \prec^W (r, \eta)\})$. 


If Claim 6

On the contrary, assume

Proof of Claim 6

(Case 1)

Then as

Then this is a contradiction, since by the ind. hyp., we must have that

(Case 2)

Then as

Then by the definition of \( \Lambda^{t+1}(P)_M \), we have:

1. \( Pos(r') \eta' \subseteq \lambda_{M^0}(\Gamma'(P)_M) \);

2. \( Neg(r') \eta' \cap M = \emptyset. \)

Then since \( \lambda_{M^0}(\Gamma'(P)_M) \subseteq M \) (i.e., since \( \lambda_{M^0}(\Gamma^\infty(P)_M) = M \) by assumption), it follows that \( (r', \eta') \) is also a pair such that:

1. \( r' <^P r \) (i.e., by the assumption);

2. \( Pos(r') \eta' \subseteq M \);

3. \( Neg(r') \eta' \cap \lambda_{M^0}(\Gamma'(P)_M) = \emptyset; \)

4. \( (r', \eta') \notin \Gamma'(P)_M \) (i.e., by the definition of \( \Lambda^{t+1}(P)_M \) and as \( (r', \eta') \in \Lambda^{t+1}(P)_M \) for all \( 1 \leq t' \leq t. \)

Then by the definition of \( \Gamma'(P)_M \), this contradicts the assumption \( (r, \eta) \in \Gamma'(P)_M \) because the pair \( (r', \eta') \) will always be blocking \( (r, \eta) \) from being applied at all stages \( 0 \leq t' \leq t. \).
APPENDIX A. PROOFS

165

Proof of Claim 7

We again consider the possibilities:

Claim 7

If either:

Case 2: (r, η') ∈ \(\Gamma^t(\mathcal{P})_\mathcal{M}\) and \(r, \eta) \in \Lambda^{t+1}(\mathcal{P})_\mathcal{M}\).

Then by the construction of \(\mathcal{W}'\), we have that \((r', \eta') <_{\mathcal{W}'} (r, \eta)\), which contradicts the assumption \((r, \eta) <_{\mathcal{W}'} (r', \eta')\). Therefore, we cannot have this possibility.

Case 4: \((r, \eta), (r', \eta') \in \Lambda^{t+1}(\mathcal{P})_\mathcal{M}\).

Then since \(<_{\mathcal{W}} \subseteq <_{\mathcal{W}'}\) and by the construction of the well-order \(\mathcal{W}\) on \(\Lambda^{t+1}(\mathcal{P})_\mathcal{M}\), then we must have \((r', \eta') <_{\mathcal{W}'} (r, \eta)\) since \(r' <_P r\) by assumption. This is contrary to the initial assumption \((r, \eta) <_{\mathcal{W}'} (r', \eta')\).

This ends the proof of Claim 6.

Claim 7 If \((r, \eta) \in \Gamma^{t+1}(\mathcal{P})_\mathcal{M}\), then \((r', \eta') \notin \Gamma(\Pi)_\mathcal{M}\) where \(r' <_P r\) implies that either:

1. \(Pos(r')\eta' \subseteq \mathcal{M}\) or
2. \(Neg(r')\eta' \cap (\mathcal{M}^0(\Pi) \cup \{Head(r'')\eta'' | (r'', \eta'') <_{\mathcal{W}''} (r, \eta)\}) \neq \emptyset\).

Proof of Claim 7 We again consider the possibilities:

Case 1: \((r, \eta) \in \Gamma^t(\mathcal{P})_\mathcal{M}\).

Then it immediately follows by the inductive hypothesis that Claim 7 holds.

Case 2: \((r, \eta) \in \Lambda^{t+1}(\mathcal{P})_\mathcal{M}\).

For the sake of contradiction, assume \(Pos(r')\eta' \subseteq \mathcal{M}\) and \(Neg(r')\eta' \cap (\mathcal{M}^0(\Pi) \cup \{Head(r'')\eta'' | (r'', \eta'') <_{\mathcal{W}''} (r, \eta)\}) = \emptyset\). Then by the construction of \(\mathcal{W}''\), it also follows that \(Neg(r')\eta' \cap \lambda_{\mathcal{M}^0}(\Gamma^t(\mathcal{P})_\mathcal{M}) = \emptyset\). Then this implies \((r', \eta')\) is a pair such that:

1. \(r' <_P r\) (i.e., by assumption);
2. \(Pos(r')\eta' \subseteq \mathcal{M}\) and \(Neg(r')\eta' \cap \lambda_{\mathcal{M}^0}(\Gamma^t(\mathcal{P})_\mathcal{M}) = \emptyset\);
3. \((r', \eta') \notin \Gamma^t(\mathcal{P})_\mathcal{M}\) (i.e., since \((r', \eta') \notin \Gamma(\Pi)_\mathcal{M}\) by assumption and where \(\Gamma^\infty(\mathcal{P})_\mathcal{M} = \Gamma(\Pi)_\mathcal{M}\) by Claim 1).

Then this contradicts the assumption \((r, \eta) \in \Lambda^{t+1}(\mathcal{P})_\mathcal{M} = \Gamma^{t+1}(\mathcal{P})_\mathcal{M} \setminus \Gamma^t(\mathcal{P})_\mathcal{M}\) (i.e., \((r, \eta)\) is applied in \(\Gamma^{t+1}(\mathcal{P})_\mathcal{M}\)) since by the definition of \(\Gamma^{t+1}(\mathcal{P})_\mathcal{M}\), we have \((r', \eta')\) is blocking \((r, \eta)\) from being applied at stage \(t + 1\).

This ends the proof of Claim 7.
Thus, by Claims 5, 6 and 7, we have the hypothesis also holds for $t + 1$.

Now we prove the second part by showing $\mathcal{M} \models \exists \exists \forall \mathcal{S} \varphi_{p}(\mathcal{S}, S)$. As mentioned in the beginning, based on the result of the first part (i.e., that there exist a preference preserving well-order $W = (\Gamma(\Pi)_{\mathcal{M}}, <^{W})$ on the set $\Gamma(\Pi)_{\mathcal{M}}$), we now construct an expansion $\mathcal{M}'$ of $\mathcal{M}$ based on the well-order $W$. Thus, set the structure $\mathcal{M}'$ to be an expansion of $\mathcal{M}$ on the signature $\tau(\Pi) \cup \{<_{r_{1}r_{2}} | r_{1}, r_{2} \in \Pi\}$ (i.e., where $\mathcal{M}$ is of the signature $\tau(\Pi)$) such that for each (new) predicate symbol $<_{r_{1}r_{2}}$ (i.e., where $r_{1}, r_{2} \in \Pi$ and $r_{1}$ and $r_{2}$ could be the same), we have

$$<_{r_{1}r_{2}}^{\mathcal{M}'} = \{\langle \eta_{1}(u_{1}), \ldots, \eta_{1}(u_{k}), \eta_{2}(v_{1}), \ldots, \eta_{2}(v_{l}) \rangle | \langle u_{1}, \ldots, u_{k} \rangle = x_{r_{1}}, \langle v_{1}, \ldots, v_{l} \rangle = x_{r_{2}}, (r_{1}, \eta_{1}) <^{W} (r_{2}, \eta_{2})\}.$$

Then in a similar manner to the proof of Theorem 10, it can be shown (although tedious) that $\mathcal{M}' \models \forall \mathcal{S} \varphi_{p}(S)$.

$(\Leftarrow)$ Assume $\mathcal{M} \models \exists \exists \forall \mathcal{S} \varphi_{p}(\mathcal{S}, S)$. We show $\lambda_{\mathcal{M}^{0}}(\Gamma^{\infty}(P)_{\mathcal{M}}) = \mathcal{M}$ by showing $\lambda_{\mathcal{M}^{0}}(\Gamma^{\infty}(P)_{\mathcal{M}}) \subseteq \mathcal{M}$ and $\mathcal{M} \subseteq \lambda_{\mathcal{M}^{0}}(\Gamma^{\infty}(P)_{\mathcal{M}})$. Indeed, since $\mathcal{M} \models \exists \exists \forall \mathcal{S} \varphi_{p}(\mathcal{S}, S)$ implies $\mathcal{M} \models \exists \exists \forall \mathcal{S} \varphi_{p}^{\text{pro}}(\mathcal{S}, S) \wedge \varphi_{p}^{\text{comp}}$, then $\mathcal{M}$ is an answer set of $\Pi$ by Theorem 10. Thus, from [ZZ10], we have $\mathcal{M}^{\infty}(\Pi) = \mathcal{M}$. Then since $\lambda_{\mathcal{M}^{0}}(\Gamma^{\infty}(P)_{\mathcal{M}}) \subseteq \mathcal{M}^{\infty}(\Pi)$, it follows that $\lambda_{\mathcal{M}^{0}}(\Gamma^{\infty}(P)_{\mathcal{M}}) \subseteq \mathcal{M}$. Therefore, it is only left for us to show that $\mathcal{M} \subseteq \lambda_{\mathcal{M}^{0}}(\Gamma^{\infty}(P)_{\mathcal{M}})$. Now, since $\mathcal{M} \models \exists \exists \forall \mathcal{S} \varphi_{p}(\mathcal{S})$, then there exist an expansion $\mathcal{M}'$ of $\mathcal{M}$, on the signature $\tau(\Pi) \cup \{<_{r_{1}r_{2}} | r_{1}, r_{2} \in \Pi\}$, such that $\mathcal{M}' \models \forall \mathcal{S} \varphi_{p}(S)$ with $\forall \mathcal{S} \varphi_{p}(S)$ the sentence obtained from $\forall \mathcal{S} \varphi_{p}(\mathcal{S})$ by treating the predicate variables in $\mathcal{S}$ as constants. Then since $\mathcal{M}' \models \forall \mathcal{S} \varphi_{p}(S)$ (i.e., which implies a well-order on $\Gamma(\Pi)_{\mathcal{M}}$), we construct a well-order $W = (\Gamma(\Pi)_{\mathcal{M}}, <^{W})$ of $\Gamma(\Pi)_{\mathcal{M}}$ by setting

$$<^{W} = \{\langle (r_{1}, \eta_{1}), (r_{2}, \eta_{2}) \rangle | r_{1}, r_{2} \in \Pi, x_{r_{1}} = \langle u_{1}, \ldots, u_{k} \rangle, x_{r_{2}} = \langle v_{1}, \ldots, v_{l} \rangle, \langle \eta_{1}(u_{1}), \ldots, \eta_{1}(u_{k}), \eta_{2}(v_{1}), \ldots, \eta_{2}(v_{l}) \rangle \in <^{\mathcal{M}'}_{r_{1}r_{2}}\}.$$

Hence, for an $e \in \Gamma(\Pi)_{\mathcal{M}}$, define $W^{e}(\Pi)$ inductively as follows:

$$W^{\text{BOT}(W)}(\Pi) = \{\text{BOT}(W)\};$$

$$W^{\text{SUCC}(e)}(\Pi) = W^{e}(\Pi) \cup \{\text{SUCC}(e)\};$$
where $\text{BOT}(W)$, $\text{SUCC}(e)$ and $\text{ORD}(W)$ denotes the bottom element, the successor element of $e$ and the order type of $\Gamma(\Pi)_M$ under $W$ respectively. Intuitively, $W^e(\Pi)$ represents the “gradual” collection of the pairs $(r, \eta)$ of $\Gamma(\Pi)_M$ under the well-order $W$ up to and including the element $e$.

**Claim 1:** $W^{\text{ORD}(W)}(\Pi) \subseteq \Gamma^\infty(\Pi)_M$.

**Proof of Claim 1:** We prove by induction on $e$ for $e \geq \text{BOT}(W)$.

**Basis:** Since $W$ is a preference preserving well-order on $\Gamma(\Pi)_M$, then we have that $\text{BOT}(W) = (r, \eta)$ is such that:

1. $\text{Pred}(\text{Pos}(r)) \cap \mathcal{P}_\text{int}(\Pi) = \emptyset$ (i.e., since $(r, \eta)$ is the least element under $W$ of $\Gamma(\Pi)_M$ and that the well-order $W$ satisfies the notion of support since $M' | = \forall x_r (\varphi^\text{GEN}_r(x_r) \rightarrow \varphi^\text{SUP}_r(x_r)))$;
2. Any rule $r' <^P r$ and assignment $\eta'$ implies $(r', \eta') \notin \Gamma(\Pi)_M$. For assume $(r', \eta') \in \Gamma(\Pi)_M$. Then we must have $(r', \eta) <^W (r, \eta)$ since $r' <^P r$ and where $(r, \eta)$ and $(r', \eta')$ are both in $\Gamma(\Pi)_M$, and were

$$M' | = \forall x_r (\varphi^\text{GEN}_r(x_r) \rightarrow \bigwedge_{r' <^P r} \forall x_{r'} (\varphi^\text{GEN}_{r'}(x_{r'}) \rightarrow <^r_{r'} (x_{r'}, x_r)))$$

(i.e., since $M' | = \varphi^\text{PREF}_p$). This is absurd since $(r, \eta)$ is the least most under $W$;
3. For each rule $r' <^P r$ and assignment $\eta'$ with $(r', \eta') \notin \Gamma(\Pi)_M$, we have either:
   (a) $\text{Pos}(r')\eta' \not\subseteq M$
   (b) $\text{Neg}(r')\eta' \cap (M^0(\Pi) \cup \{\text{Head}(r'')\eta' | (r'', \eta'') <^W (r, \eta)\}) \neq \emptyset$ since

$$M' | = \forall x_r (\varphi^\text{GEN}_r(x_r) \rightarrow \bigwedge_{r' <^P r} \forall x_{r'} (\neg \varphi^\text{GEN}_{r'}(x_{r'}) \rightarrow ($$$$

(\Phi^\text{POS}_{r'}(x_{r'}) \lor \Phi^\text{DEF}_{r'}(x_r, x_{r'}))))).$$

Then since $(r, \eta) = \text{BOT}(W)$ (i.e., the least element under $W$, which means that $\{\text{Head}(r'')\eta'' | (r'', \eta'') <^W (r, \eta)\} = \emptyset$), it must be that $\text{Pos}(r')\eta' \not\subseteq$
\[ M \text{ or } \text{Neg}(r')\eta' \cap M^0(\Pi) \neq \emptyset. \]

Then by 2 and 3 above, we know that there does not exist a rule \( r' <^p r \) and assignment \( \eta' \) with \( \text{Pos}(r')\eta' \subseteq M \) and \( \text{Neg}(r')\eta' \cap M^0(\Pi) = \emptyset \) since we will always have \( \text{Pos}(r')\eta' \not\subseteq M \) or \( \text{Neg}(r')\eta' \cap M^0(\Pi) \neq \emptyset \). Hence, as \((r, \eta) \in \Gamma(\Pi)_M \) and where \( Pos(r)\eta \subseteq M^0(\Pi) \) by 1 above, then this implies \((r, \eta) \in \Gamma(\Pi)_M \) by the definition of \( \Gamma(\Pi)_M \). Thus, we have \( W^{\text{bot}(W)}(\Pi) \subseteq \Gamma(\Pi)_M \subseteq \Gamma(\Pi)_M \).

**Step:** Assume for \( \text{bot}(W) \leq^W e' \leq^W e \), we have \( W^{e'}(\Pi) \subseteq \Gamma(\Pi)_M \). We will show that \( W^{\text{succ}(e)}(\Pi) \subseteq \Gamma(\Pi)_M \). Thus, assume \( \text{succ}(e) = (r, \eta) \). Then by the ind. hyp., there exist some certain \( t \) for which \( W^{e'}(\Pi) \subseteq \Gamma_t(\Pi)_M \) (i.e., since \( W^{e'}(\Pi) \subseteq \Gamma(\Pi)_M \)). Moreover, since \( M' = \bigwedge_{r \in \Pi} \forall x_r(\varphi_r^{\text{gen}}(x_r) \rightarrow \varphi_r^{\text{sup}}(x_r)) \) (i.e., obeys the notion of “support”), then we also have \( Pos(r)\eta \subseteq \lambda_0(W^{e'}(\Pi)) \), and hence, that \( Pos(r)\eta \subseteq \lambda_0(\Gamma_t(\Pi)_M) \), since \( W^{e'}(\Pi) \subseteq \Gamma_t(\Pi)_M \).

**Subclaim 1:** There does not exist a rule \( r' <^p r \) and assignment \( \eta' \) such that

1. \( Pos(r')\eta' \subseteq M \) and \( \text{Neg}(r')\eta' \cap \lambda_0(\Gamma_t(\Pi)_M) = \emptyset; \)
2. \( (r', \eta') \notin \Gamma_t(\Pi)_M. \)

**Proof of Subclaim 1:** On the contrary, assume that there exist such a rule \( r' \) and assignment \( \eta' \). Then there can only be two possibilities:

**Case 1:** \((r', \eta') \in \Gamma(\Pi)_M. \)

Then as

\[ M' = \bigwedge_{r' \in \text{pr}} \forall x_r(\varphi_r^{\text{gen}}(x_r) \rightarrow \bigwedge_{r' < r} \forall x_{r'}(\varphi_r^{\text{gen}}(x_{r'}) \rightarrow <_{r'} ((r', \eta') , (x_{r'}, x_r)))) \]

and since \( r' <^p r \), then we must have \((r', \eta') <^W (r, \eta) = \text{succ}(e), \) or in other words, that \((r', \eta') \in W^{e'}(\Pi) \) (i.e., by the definition of \( W^{e'}(\Pi) \)).

Then since \( W^{e'}(\Pi) \subseteq \Gamma_t(\Pi)_M \) by assumption (i.e., ind. hyp.), this contradicts the assumption \((r', \eta') \notin \Gamma_t(\Pi)_M. \)

**Case 2:** \((r', \eta') \notin \Gamma(\Pi)_M. \)
Then since
\[ \mathcal{M}' \models \forall x_r (\varphi_r^{\text{GEN}}(x_r) \rightarrow \bigwedge_{r' < r} \forall x_{r'} (\neg \varphi_{r'}^{\text{GEN}}(x_{r'}) \rightarrow (\Phi_{r'}^{\text{POS}}(x_{r'}) \lor \Phi_{r'}^{\text{DEF}}(x_r, x_{r'})))) , \]

there can only be two possibilities:

**Subcase 1:** \( Pos(r') \eta' \not\subseteq \mathcal{M} \).

Then this contradicts the assumption \( Pos(r') \eta' \subseteq \mathcal{M} \).

**Subcase 2:** \( Neg(r') \eta' \cap \lambda_{\mathcal{M}^0}(\mathcal{W}^e(\Pi)) \neq \emptyset \).

Then this contradicts the assumption \( Neg(r') \eta' \cap \lambda_{\mathcal{M}^0}(\Gamma^t(\mathcal{P}), \mathcal{M}) = \emptyset \) since \( \mathcal{W}^e(\Pi) \subseteq \Gamma^t(\mathcal{P}), \mathcal{M} \), by the ind. hyp.

This completes the proof of Subclaim 1.

Hence, by Subclaim 1, we must have \((r, \eta) \in \Gamma^{t+1}(\mathcal{P}), \mathcal{M}\) (i.e., since there is no pair \((r', \eta')\) blocking \((r, \eta)\) from being applied at stage \(t+1\)), which then implies \( \mathcal{W}^{\text{SUCC}(e)}(\Pi) \subseteq \Gamma^\infty(\mathcal{P}), \mathcal{M} \).

This completes the proof of Claim 1.

Therefore, using the fact that \( \mathcal{W}^{\text{ORD}(\mathcal{W})}(\Pi) \subseteq \Gamma^\infty(\mathcal{P}), \mathcal{M} \) by Claim 1, it is now sufficient to only show that \( \mathcal{M} \subseteq \lambda_{\mathcal{M}^0}(\mathcal{W}^{\text{ORD}(\mathcal{W})}(\Pi)) \) to show \( \mathcal{M} \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(\mathcal{P}), \mathcal{M}) \). Hence, let \( P(a_P) \in \mathcal{M} \) such that \( P(a_P) \notin \mathcal{M}^0(\Pi) \) (i.e., for if \( P(a_P) \in \mathcal{M}^0(\Pi) \) then the result is clear). We will show that \( P(a_P) \in \{ \text{Head}(r) \eta \mid (r, \eta) \in \mathcal{W}^{\text{ORD}(\mathcal{W})}(\Pi) \} \). Indeed, since \( \mathcal{M}^\infty(\Pi) = \mathcal{M} \) (i.e., since \( \mathcal{M} \models \exists \exists \exists S(\varphi_\Pi^{\text{PRO}}(\preceq, S) \land \varphi_\Pi^{\text{COMP}}) \) and by Theorem 10 and [ZZ10]), we have for some \( t > 1 \), rule \( r \), and corresponding assignment \( \eta \):

1. \( \text{Head}(r) \eta = P(a_P) \);
2. \( Pos(r) \eta \subseteq \mathcal{M}^t(\Pi) \subseteq \mathcal{M}^{\infty}(\Pi) = \mathcal{M} \) and \( Neg(r) \eta \cap \mathcal{M} = \emptyset \).

Then we have that \((r, \eta) \in \Gamma(\Pi), \mathcal{M}\), which further implies \((r, \eta) \in \mathcal{W}^{\text{ORD}(\mathcal{W})}(\Pi) \) (i.e., since \( \text{Dom}(\mathcal{W}) = \Gamma(\Pi), \mathcal{M}\)). Therefore, we have \( P(a_P) \in \{ \text{Head}(r) \eta \mid (r, \eta) \in \mathcal{W}^{\text{ORD}(\mathcal{W})}(\Pi) \} \subseteq \lambda_{\mathcal{M}^0}(\mathcal{W}^{\text{ORD}(\mathcal{W})}(\Pi)) \). Hence, we had shown that \( \mathcal{M} \subseteq \lambda_{\mathcal{M}^0}(\Gamma^\infty(\mathcal{P}), \mathcal{M}) \).

This completes the proof of the theorem. \( \square \)
References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


