Ordered Completion for First-Order Logic Programs on Finite Structures

Vernon Asuncion\textsuperscript{a}, Fangzhen Lin\textsuperscript{b}, Yan Zhang\textsuperscript{a}, Yi Zhou\textsuperscript{a,∗}

\textsuperscript{a}Intelligent Systems Lab, School of Computing and Mathematics, University of Western Sydney, Penrith South DC, NSW 1797, Australia
\textsuperscript{b}Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

Abstract

In this paper, we propose a translation from normal first-order logic programs under the stable model semantics to first-order sentences on finite structures. Specifically, we introduce ordered completions which are modifications of Clark’s completions with some auxiliary predicates added to keep track of the derivation order. We show that, on finite structures, classical models of the ordered completion of a normal logic program correspond exactly to the stable models of the program. We also extend this result to normal programs with constraints and choice rules.

From a theoretical viewpoint, this work is an important step to clarify the relationships between normal logic programming under the stable model semantics and classical first-order logic. It follows that, on finite structures, every normal program can be defined by a first-order sentence with auxiliary predicates. Interestingly, such a translation does not exist if infinite structures are considered or auxiliary predicates are prohibited. Also, there is no translation from disjunctive programs to first-order logic assuming $NP \neq coNP$.

From a practical viewpoint, this work leads to a new type of ASP solver by grounding on a program’s ordered completion instead of the program itself. We report on a first implementation of such a solver based on several optimization techniques, which compares favorably to other major ASP solvers

∗Corresponding author

Email addresses: vernon@scm.uws.edu.au (Vernon Asuncion), flin@cse.ust.hk (Fangzhen Lin), yan@scm.uws.edu.au (Yan Zhang), yzhou@scm.uws.edu.au (Yi Zhou)
according to the experimental results, especially on large domains.

**Keywords:** answer set programming, the ordered completion, knowledge representation, nonmonotonic reasoning

---

1. **Introduction**

This work is about translating logic programs under the stable model (answer set) semantics [13] to first-order logic. Viewed in the context of formalizing the semantics of logic programs in classical logic, work in this direction goes back to that of Clark [5] who gave us what is now called Clark’s completion semantics, on which our work, like most other work in this direction, is based.

In terms of the stable model semantics, Clark’s completion semantics is too weak in the sense that not all models of Clark’s completion are stable models, unless the programs are “tight” [9]. Various ways to remedy this have been proposed, particularly in the propositional case given the recent interest in Answer Set Programming (ASP) and the prospect of using SAT solvers to compute answer sets [20]. This paper considers first-order logic programs, and the prospect of capturing the answer sets of these programs in first-order logic.

A crucial consideration in work of this kind is whether auxiliary symbols (in the propositional case) or predicates (in the first-order case) can be used. For propositional logic programs, Ben-Eliyahu and Dechter’s translation [2] is polynomial in space but uses $O(n^2)$ auxiliary variables, while Lin and Zhao’s translation [20] using loop formulas is exponential in the worst case but does not use any auxiliary variables. Chen et al. [3] extended loops and loop formulas to first-order case and showed that for finite domains, the answer sets of a first-order normal logic program can be captured by its completion and all its first-order loop formulas. However, in general, a program may have an infinite number of loops and loop formulas. But this seems to be the best that one can hope for if no auxiliary predicates are used: it is well-known that transitive closure, which can be easily written as a first-order logic program, cannot be captured by any finite first-order theory on finite structures [6].

The situation is different if we introduce auxiliary predicates. Our main technical result of this paper is that by using some additional predicates that keep track of the derivation order from bodies to heads in a program, we
can modify Clark’s completion into what we call the *ordered completion* that captures exactly the answer set semantics on finite structures.

The rest of the paper is organized as follows. We recall some background knowledge in the next section. In section 3, we define our ordered completions for first-order normal logic programs, and show that they capture exactly the stable models of the programs on finite structures. We then show that this result can be extended to normal programs with constraints and choice rules. However, it cannot be extended to disjunctive logic programs and does not hold on arbitrary structures. Specifically, in section 3 we show that, on arbitrary structures, there exist some normal programs that cannot be captured by any first-order theories, and on finite structures, there exist some disjunctive programs that cannot be captured by any first-order sentences provided $NP \neq coNP$. We then present some techniques for optimizing the ordered completion in section 4, based on which, we have implemented a first-order solver and we report the implementation as well as the experimental results in section 5. Then, we discuss some related work in section 6. Finally, we conclude the paper in section 7.

2. Preliminaries

We assume readers are familiar with some basic notions and notations of classical first-order logic. Here, we consider a finite first-order language without function symbols but with equality. In particular, an atom is called an *equality atom* if it is of the form $t_1 = t_2$, and a *proper atom* otherwise.

Let $\sigma$ and $\sigma_1$ be two signatures such that $\sigma \subseteq \sigma_1$. Given a structure $\mathcal{A}$ of signature $\sigma_1$, we say that the *reduct* of $\mathcal{A}$ on $\sigma$, denoted by $\mathcal{A} \uparrow \sigma$, is the $\sigma$-structure that agrees with $\mathcal{A}$ on all interpretations of predicates and constants in $\sigma$. Conversely, we say that $\mathcal{A}$ is an *expansion* of $\mathcal{A} \uparrow \sigma$ to $\sigma_1$.

A *normal logic program* (program for short) is a finite set of *rules* of the following form

$$\alpha \leftarrow \beta_1, \ldots, \beta_k, \text{not } \gamma_1, \ldots, \text{not } \gamma_l,$$

(1)

where $\alpha$ is a proper atom, $0 \leq k \leq l$, and $\beta_i$ $(1 \leq i \leq k)$, $\gamma_j$ $(1 \leq j \leq l)$ are atoms. Given a rule $r$ of form (1), we call $\alpha$ the *head* of $r$, denoted by $Head(r)$, and $\{\beta_1, \ldots, \beta_k, \text{not } \gamma_1, \ldots, \text{not } \gamma_l\}$ the *body* of $r$, denoted by $Body(r)$. In particular, we call $\{\beta_1, \ldots, \beta_k\}$ the *positive body* of $r$, denoted by $Pos(r)$, and $\{\gamma_1, \ldots, \gamma_l\}$ the *negative body* of $r$, denoted by $Neg(r)$. We call a variable in a rule a *body variable* if it occurs in the body but not the head of the rule.
Given a program $\Pi$, a predicate is called \textit{intensional} if it occurs in the head of some rule in $\Pi$, and \textit{extensional} otherwise. The \textit{signature} of $\Pi$ contains all intensional predicates, extensional predicates and constants occurring in $\Pi$.

For convenience and without loss of generality, in the following we assume that programs are \textit{normalized} in the sense that for each intensional predicate $P$, there is a tuple $\overline{\alpha}$ of distinct variables matching the arity of $P$ such that for each rule, if its head mentions $P$, then the head must be $P(\overline{\alpha})$. So all the rules with $P$ occurring in the heads in a program can be enumerated as:

$$P(\overline{\alpha}) \leftarrow \text{Body}_1, \ldots, P(\overline{\alpha}) \leftarrow \text{Body}_k.$$  

2.1. \textit{Clark’s completion}

Our following definition of Clark’s completion is standard except that we do not make completions for extensional predicates.

Given a program $\Pi$, and a predicate $P$ in it, Clark’s Completion of $P$ in $\Pi$ is the following first-order sentence [5]:

$$\forall \overline{\alpha} \ (P(\overline{\alpha}) \leftrightarrow \bigvee_{1 \leq i \leq k} \exists \overline{y}_i \overline{\text{Body}}_i),$$  

(2)

where

- $P(\overline{\alpha}) \leftarrow \text{Body}_1, \ldots, P(\overline{\alpha}) \leftarrow \text{Body}_k$ are all the rules whose heads mention the predicate $P$;

- $\overline{y}_i$ is the tuple of body variables in $P(\overline{\alpha}) \leftarrow \text{Body}_i$;

- $\overline{\text{Body}}_i$ is the conjunction of elements in $\text{Body}_i$ by simultaneously replacing the occurrences of \textit{not} by $\neg$.

Clark’s Completion (completion for short if clear from the context) of $\Pi$, denoted by $\text{Comp}(\Pi)$, is then the set of Clark’s completions of all intensional predicates in $\Pi$.

\textbf{Example 1.} [Transitive Closure (TC)] The following normal logic program TC computes the transitive closure of a given graph:

$$S(x, y) \leftarrow E(x, y)$$

$$S(x, y) \leftarrow E(x, z), S(z, y),$$
where \( E \) is the only extensional predicate of TC, representing the edges of a graph, and \( S \) is the only intensional predicate of TC. Ideally, the intensional predicate computes the transitive closure (i.e. all the paths) of a given graph. The Clark’s Completion of TC is the following first-order sentence:

\[
\forall xy (S(x, y) \leftrightarrow (E(x, y) \lor \exists z E(x, z) \land S(z, y))).
\]

### 2.2. The stable model semantics for propositional programs

In the propositional case, the stable model semantics for normal propositional programs was proposed by Gelfond and Lifschitz [13], and later extended to become answer set semantics for propositional programs that can have classical negation, constraints, disjunctions, and other operators [7, 12, 14, 18, 22, 23]. Several equivalent characterizations are proposed, including the Gelfond-Lifschitz transformation [13], the logic of GK [21], loop formulas [20], equilibrium logic [27], general reduction [12], and so on. Here, we briefly review the standard Gelfond-Lifschitz transformation semantics [13] and the loop formula characterization in the propositional case [20], as they are needed in the proof of our main theorem.

Let \( \Pi \) be a propositional program and \( \mathcal{A} \) a set of atoms. We say that \( \mathcal{A} \) satisfies a rule \( r \) in \( \Pi \) if \( \text{Head}(r) \in \mathcal{A} \) whenever \( \text{Pos}(r) \subseteq \mathcal{A} \) and \( \text{Neg}(r) \cap \mathcal{A} = \emptyset \). Then, \( \mathcal{A} \) satisfies \( \Pi \) if it satisfies all rules in \( \Pi \). The reduct of \( \Pi \) relative to \( \mathcal{A} \), denoted by \( \Pi^\mathcal{A} \), is the program obtained from \( \Pi \) by (i) deleting every rule \( r \) in \( \Pi \) whose negative body is not satisfied by \( \mathcal{A} \) (i.e. not disjoint with \( \mathcal{A} \)), and (ii) deleting all negative atoms in other rules. Then, \( \mathcal{A} \) is said to be a stable model (or an answer set) of \( \Pi \) iff \( \mathcal{A} \) satisfies \( \Pi \) and there does not exist \( \mathcal{A}' \subset \mathcal{A} \) such that \( \mathcal{A}' \) satisfies \( \Pi^\mathcal{A} \).

An equivalent characterization of the stable model semantics is the loop formula approach [20]. Let \( \Pi \) be a program. The (positive) dependency graph of \( \Pi \), denoted by \( G_\Pi \), is the finite graph \((V, E)\), where \( V \) is the set of atoms occurring in \( \Pi \), and \((x, y)\) is an edge in \( E \) iff there exists a rule \( r \in \Pi \) whose head is \( x \) and its positive body contains \( y \). A set of atoms \( L \) is said to be a loop if there exists a cycle in \( G_\Pi \) that goes through only and all the nodes in \( L \). Let \( x \) be a propositional atom. If there exists a rule \( r \in \Pi \) such that its head is \( x \), then we say that the body of \( r \) is a support of \( x \) in \( \Pi \). Let \( L \) be a loop, and \( x \) an atom in \( L \). We say that a support \( \text{Body} \) of \( x \) in \( \Pi \) is an external support with respect to \( L \) if the positive part of \( \text{Body} \) contains no atoms from \( L \). Given a loop \( L \), we use \( ES(L, \Pi) \) to denote the set of all external supports of some element in \( L \) with respect to \( L \) in \( \Pi \). Then, the
loop formula of $L$ in $\Pi$, denoted by $LF(L, \Pi)$, is the following formula

$$\bigvee_{x \in L} x \rightarrow \bigvee_{\text{Body} \in ES(L, \Pi)} \text{Body}.$$ 

Lin and Zhao [20] showed that, in the propositional case, a set of atoms is an answer set (stable model) of a finite program if and only if it is a model of the Clark’s completion together with all loop formulas of the program.

2.3. The stable model semantics for first-order programs

In Gelfond and Lifschitz’s seminal work [13], the stable model/answer set semantics for first-order logic programs (i.e. programs with variables) is defined via grounding on Herbrand structures. Recently, there has been interest in defining the stable model semantics for first-order programs directly on a first-order level [3, 4, 11, 21, 26, 28]. Such semantics usually consider arbitrary structures instead of Herbrand structures. Nevertheless, for normal logic programs, all of them coincide with Gelfond and Lifschitz’s original semantics [13] when only considering Herbrand structures [4, 11, 17, 21, 28].

We briefly review the translational semantics [11, 21] by defining the stable model semantics of logic programs in second-order logic. Under our context, we only consider normal first-order logic programs.

Given a normal logic program $\Pi$, let $\Omega_{\Pi} = \{Q_1, \ldots, Q_n\}$ be the set of all intentional predicates of $\Pi$. Let $\Omega^*_\Pi = \{Q^*_1, \ldots, Q^*_n\}$ be a new set of predicates corresponding to $\Omega_{\Pi}$, where each $Q^*_i$ in $\Omega^*_\Pi$ has the same arity of predicate $Q_i$ in $\Omega_{\Pi}$. Given a rule $r$ in $\Pi$ of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not} \, \gamma_1, \ldots, \text{not} \, \gamma_l,$$

by $\hat{r}$, we denote the universal closure of the following formula

$$\beta_1 \land \cdots \land \beta_m \land \neg \gamma_1 \land \cdots \land \neg \gamma_l \rightarrow \alpha;$$

by $r^*$, we denote the universal closure of the following formula

$$\beta^*_1 \land \cdots \land \beta^*_m \land \neg \gamma_1 \land \cdots \land \neg \gamma_l \rightarrow \alpha^*,$$

where $\alpha^* = Q^*(\overline{x})$ if $\alpha = Q(\overline{x})$ and $\beta^*_i$, $(1 \leq i \leq m) = \left\{ \begin{array}{ll} Q^*_i(\overline{T}_j) & \text{if } \beta_i = Q_j(\overline{T}_j) \text{ and } Q_j \in \Omega_{\Pi} \\ \beta_i & \text{otherwise.} \end{array} \right.$
By \( \hat{\Pi} \), we denote the first-order sentence \( \bigwedge_{r \in \Pi} \hat{r} \); by \( \Pi^* \), we denote the first-order sentence \( \bigwedge_{r \in \Pi} r^* \). Let \( \Pi \) be a normal logic program. By \( SM(\Pi) \), we denote the following second-order sentence:

\[
\hat{\Pi} \land \neg \exists \Omega^*_\Pi((\Omega^*_\Pi < \Omega^*_\Pi) \land \Pi^*),
\]

where \( \Omega^*_\Pi < \Omega^*_\Pi \) is the abbreviation of the formula

\[
\bigwedge_{1 \leq i \leq n} \forall \overline{x}(Q^*_i(\overline{x}) \rightarrow Q_i(\overline{x})) \land \neg \bigwedge_{1 \leq i \leq n} \forall \overline{x}(Q_i(\overline{x}) \rightarrow Q^*_i(\overline{x})).
\]

This second-order sentence is used to capture the stable models of \( \Pi \). We call this the translational semantics.

Here, for our purposes, we present an alternative characterization of this semantics by grounding. Similar to Gelfond and Lifschitz’s grounding approach, we define the stable model semantics by grounding into the propositional case. However, we consider grounding on arbitrary structures instead of Herbrand structures. It is worth mentioning that, on arbitrary structures, the unique name assumption (i.e. distinct constants must be interpreted differently) does not necessarily hold. Consequently, we will not assume it in our grounding procedure.

Given a program \( \Pi \), and a first-order structure \( M \) of the signature used in \( \Pi \), we use the interpretations of \( M \) on the constants and extensional predicates to ground \( \Pi \).

**Definition 1.** The grounding of a program \( \Pi \) on a structure \( M \), written \( \Pi_M \) below, is the union of the following three sets:

1. The set of all instances of the rules in \( \Pi \) under \( M \), here an instance of a rule under \( M \) is the result of simultaneously replacing every constant in the rule by its interpretation in \( M \), and every variable \( x \) in the rule by a domain object \( d \) in \( M \);
2. \( EQ_M = \{ u = u \mid u \text{ is a domain object in } M \} \); \(^1\)
3. \( Ext_M = \{ P(\overline{u}) \mid P \text{ is an extensional predicate and } \overline{u} \in P^M \} \), here \( P^M \) is the interpretation of \( P \) in \( M \).

We now have the following definition:

\(^1\)Note that here \( u = u \) is a propositional atom but not an equality atom.
Definition 2. Let $\Pi$ be a normal logic program and $\mathcal{M}$ a structure. We say that $\mathcal{M}$ is a stable model (or an answer set) of $\Pi$ if the following set

$$EQ_{\mathcal{M}} \cup Ext_{\mathcal{M}} \cup Int_{\mathcal{M}}$$

is an answer set of $\Pi_{\mathcal{M}}$ in the propositional case, where $Int_{\mathcal{M}}$ is the following set

$$\{ P(\overline{u}) \mid P \text{ is an intensional predicate, and } \overline{u} \in P^M \}.$$  

Example 2. Consider the following program $\Pi_0$

$$P(x) \leftarrow \text{not } Q(x),$$
$$P(a_1),$$
$$Q(a_2).$$

According to Gelfond and Lifschitz’s original stable model semantics [13], the unique stable model/answer set of $\Pi_0$ is $X = \{ P(a_1), Q(a_2) \}$.

Now, let us reconsider the program $\Pi_0$ under the new semantics (i.e. Definition 2). Notice that both $P$ and $Q$ are intensional predicates. According to Definition 2, the following structure $\mathcal{M}_0$, where

$$\mathcal{M}_0 = \{ d_1, d_2 \}, a_1^{\mathcal{M}_0} = d_1, a_2^{\mathcal{M}_0} = d_2, P^{\mathcal{M}_0} = \{ d_1 \}, Q^{\mathcal{M}_0} = \{ d_2 \}$$

is a stable model of $\Pi_0$. In fact, this structure $\mathcal{M}_0$ corresponds to the unique stable model $X$ of $\Pi_0$ under the original semantics. Notice that, similar to classical first-order logic, we distinguish from $a_1$ and $d_1$ here (also $a_2$ and $d_2$) because the former is a constant in the language while the latter is a domain element.

However, under the new semantics, $\mathcal{M}_0$ is not the only stable model of $\Pi_0$. The following structure $\mathcal{M}_1$, where

$$\mathcal{M}_1 = \{ d_1 \}, a_1^{\mathcal{M}_1} = d_1, a_2^{\mathcal{M}_1} = d_1, P^{\mathcal{M}_1} = Q^{\mathcal{M}_1} = \{ d_1 \}$$

is also a stable model of $\Pi_0$. Here, both $a_1$ and $a_2$ are mapped to the same domain element $d_1$. This is allowed as the unique name assumption (i.e. distinct constants must be interpreted to different domain elements) is not assumed in the new semantics.

Also, the new semantics may allow new objects in the stable models. For instance, the following structure $\mathcal{M}_2$, where

$$\mathcal{M}_2 = \{ d_1, d_2, d_3 \}, a_1^{\mathcal{M}_2} = d_1, a_2^{\mathcal{M}_2} = d_2, P^{\mathcal{M}_2} = \{ d_1, d_3 \}, Q^{\mathcal{M}_2} = \{ d_2 \}$$

8
is a stable model of $\Pi_0$ as well. Notice that $d_3 \in P^{M_2}$ although $d_3$ is not mapped from any constants in the program.

In fact, this definition of stable model (answer set) is nothing new. The grounding technique in Definition 1 is basically the same as the standard one in [13] except that here it is done on an arbitrary structure instead of the Herbrand structure. The only reason we propose this alternative definition is that it is more suitable for understanding the proof of our main theorem (see Theorem 2). Also, the following theorem shows that this semantics by grounding on arbitrary structures actually coincides with the translational semantics mentioned previously in this section.

**Theorem 1.** Let $\Pi$ be a normal logic program whose signature is $\sigma$, and $A$ a finite $\sigma$-structure. Then, $A$ is a stable model of $\Pi$ (under Definition 2) iff $A$ is a model of $SM(\Pi)$.

**Proof:** For convenience, we use $Gr(A)$ to denote $EQ_A \cup Ext_A \cup Int_A$. First, $Gr(A)$ satisfies $\Pi_A$ iff $Gr(A)$ satisfies all instances of the rules in $\Pi$ under $M$ iff $Gr(A)$ satisfies every $r\eta$, where $r \in \Pi$ and $\eta$ is an assignment iff $A$ satisfies $\hat{r}$ for all $r \in \Pi$ iff $A \models \hat{\Pi}$.

Second, we show that $A \models \neg \exists \Omega_{\Pi}^*((\Omega_{\Pi}^* < \Omega_{\Pi}) \wedge \Pi^*)$ iff there does not exist $X \subset Gr(A)$ such that $X$ satisfies $\Pi_M^{Gr(A)}$. We have $A \models \exists \Omega_{\Pi}^*((\Omega_{\Pi}^* < \Omega_{\Pi}) \wedge \Pi^*)$ iff there exists a structure $A'$ such that $A'$ has the same domain as $A$, interprets all constants and extensional predicates the same as $A$, but

- for all intensional predicates $Q$ of $\Pi$, $Q^{A'} \subseteq Q^A$, and for some intensional predicates $\bar{Q}$, $Q^{A'} \subset Q^A$;
- for every assignment $\eta$ and every rule $r$ of form (1) in $\Pi$, if for all $i$ ($1 \leq i \leq k$), $\beta_i \eta \in A'$ and for all $j$ ($1 \leq j \leq l$), $\gamma_j \eta \notin A'$, then $\alpha \eta \in A'$ iff there exists $X \subset Gr(A)$ satisfying $(r\eta)^{Gr(A)}$ for every pair $r, \eta$, where $r \in \Pi$ and $\eta$ is an assignment (let $X = EQ_A \cup Ext_A \cup Int_A$) iff there exists $X \subset Gr(A)$ satisfying $\Pi_M^{Gr(A)}$.

This shows that the original assertion holds. $lacksquare$

As a consequence of the above Theorem and Theorem 1 in [11], Definition 2 basically coincides with Gelfond and Lifschitz’s original semantics on Herbrand structures. The main difference is that we consider not only Herbrand
structures but also arbitrary ones. For the latter, we do not need the unique name assumption (see Example 2).

Notice that the above definitions can be applied to infinite structures as well since the stable model (answer set) semantics for infinite propositional programs is well defined by using Gelfond-Lifschitz transformation [13]. However, in this paper, we are mainly concerned with finite structures unless stated otherwise (particularly in Section 3.5).

3. The Ordered Completion

It is well-known that Clark’s completion does not fully capture the answer set semantics because of positive cycles. As a simple example, the following program

\[ p \leftarrow q \]
\[ q \leftarrow p \]

has one answer set \( \emptyset \), but its Clark’s completion \( p \leftrightarrow q \) has two models \( \{ p, q \} \) and \( \emptyset \). Here, we propose a modification of Clark's completion to address this issue. The main technical property of our new translation is that for each finite first-order logic program, our translation yields a finite first-order theory that captures exactly the finite stable models of the program. The ideas behind such translation can be best illustrated by simple propositional programs. Consider the program mentioned above. We introduce four auxiliary symbols \( T_{pq}, T_{pp}, T_{qq}, T_{qp} \) (read, e.g. \( T_{pq} \) as from \( p \) to \( q \)), and translate this program into the following theory

\[ (p \rightarrow q) \land (q \rightarrow p), \]
\[ p \rightarrow (q \land T_{qp} \land \neg T_{pq}), \]
\[ q \rightarrow (p \land T_{pq} \land \neg T_{qp}), \]
\[ T_{pq} \land T_{qp} \rightarrow T_{pp}, \]
\[ T_{qp} \land T_{pq} \rightarrow T_{qq}. \]

The first sentence is the direct encoding of the two rules. The second one is similar to the other side of Clark’s completion for \( p \) except that we add \( T_{qp} \) and \( \neg T_{pq} \): for \( p \) to be true, \( q \) must be true and it must be the case that \( q \) is used to derive \( p \) but not the other way around. The third sentence is similar, and the last two sentences are about the transitivity of the \( T \) atoms. It can be checked that in all models of the above sentences, both \( p \) and \( q \) must be false.
3.1. Definition of the ordered completion

In general, let \( \Pi \) be a first-order normal logic program, and \( \Omega_\Pi \) its set of intensional predicates. For each pair of predicates \((P, Q)\) (\(P\) and \(Q\) might be the same) in \( \Omega_\Pi \), we introduce a new predicate \( \leq_{PQ} \), called the comparison predicate, whose arity is the sum of the arities of \(P\) and \(Q\). The intuitive meaning of \( \leq_{PQ} (\overline{x}, \overline{y}) \), read as from \(P(\overline{x})\) to \(Q(\overline{y})\), is that \(P(\overline{x})\) is used for deriving \(Q(\overline{y})\). In the following, we use infix notation for \( \leq_{PQ} \) and write \( \leq_{PQ} (\overline{x}, \overline{y}) \) as \( \overline{x} \leq_{PQ} \overline{y} \).

**Definition 3 (Ordered Completion).** Let \( \Pi \) be a normal logic program. The ordered completion of \( \Pi \), denoted by \(OC(\Pi)\), is the set of following sentences:

- For each intensional predicate \(P\), the following sentences:
  \[
  \forall \overline{x} \left( \bigvee_{1 \leq i \leq k} \exists \overline{y}_i (\text{Body}_i \rightarrow P(\overline{x})) \right), \tag{3}
  \]
  \[
  \forall \overline{x} (P(\overline{x}) \rightarrow \bigvee \exists y_i (\text{Body}_i \land \bigwedge Q(z) \in \text{Pos}_i, Q \in \Omega_\Pi \rightarrow z \leq_{QP} x \land \neg x \leq_{PQ} z)), \tag{4}
  \]
  where we have borrowed the notations used in the definition of Clark’s completion, and further assume that \(\text{Pos}_i\) is the positive part of \(\text{Body}_i\), and \(Q(\overline{z})\) ranges over all the intensional atoms in the positive part of \(\text{Body}_i\);

- For each triple of intensional predicates \(P\), \(Q\), and \(R\) (two or all of them might be the same) the following sentence:
  \[
  \bigwedge_{P, Q, R \in \Omega_\Pi} \forall \overline{x} \overline{y} \overline{z} (\overline{x} \leq_{PQ} \overline{y} \land \overline{y} \leq_{QR} \overline{z} \rightarrow \overline{x} \leq_{PR} \overline{z}). \tag{5}
  \]

In the following, we use \(M\text{Comp}(\Pi)\) to denote the set of the formulas (3) and (4), and \(\text{Trans}(\Pi)\) the set of formulas (5). So \(OC(\Pi) = M\text{Comp}(\Pi) \cup \text{Trans}(\Pi)\).
Clearly, for finite programs, $OC(\Pi)$ is finite, and the predicates occurring in $OC(\Pi)$ are all the predicates occurring in $\Pi$ together with all the comparison predicates $\{\leq_{PQ} \mid P, Q \in \Omega_\Pi\}$.

Notice that the Clark’s completion of a predicate can be rewritten as two parts:

$$\forall \overline{x} (\bigvee_{1 \leq i \leq k} \exists \overline{y}_i \overline{Body}_i \rightarrow P(\overline{x})), \quad \forall \overline{x} (P(\overline{x}) \rightarrow \bigvee_{1 \leq i \leq k} \exists \overline{y}_i \overline{Body}_i).$$

Thus, the only difference between $MComp(\Pi)$ and $Comp(\Pi)$ is that the former introduces some assertions on the comparison predicates, which intuitively mean that there exist derivation paths from the intensional atoms in the body to head but not the other way around (see Equation (4)). In addition, $Trans(\Pi)$ simply means that the comparison predicates satisfy “transitivity.”

**Proposition 1.** Let $\Pi$ be a normal logic program. Then, $OC(\Pi)$ introduces $m^2$ new predicates whose arities are no more than $2s$, and the size of $OC(\Pi)$ is $O(s \times m^3 + s \times n)$, where $m$ is the number of intensional predicates of $\Pi$, $s$ the maximal arity of the intensional predicates of $\Pi$ and $n$ the length of $\Pi$.

**Example 3.** [Transitive Closure continued] Recall the Transitive Closure program TC presented in Example 1. In this case, since the only intensional predicate is $S$, we only need to introduce one additional predicate $\leq_{SS}$, whose arity is 4. The ordered completion of TC consists of the following sentences:

$$\forall xy ((E(x, y) \lor \exists z(E(x, z) \land S(z, y))) \rightarrow S(x, y)), \quad \forall xy (S(x, y) \rightarrow (E(x, y) \lor \exists z(E(x, z) \land S(z, y)$$

$$\land \overline{xy} \leq_{SS} \overline{yz} \land \neg \overline{xy} \leq_{SS} \overline{zy})), \quad \forall xyuvzw (\overline{xy} \leq_{SS} \overline{uw} \land \overline{uv} \leq_{SS} \overline{zw} \rightarrow \overline{xy} \leq_{SS} \overline{zw}).$$

Intuitively, one can understand $\overline{xy} \leq_{SS} \overline{uw}$ to mean that $S(x, y)$ is used to establish $S(u, v)$. So the sentence (7) means that for $S(x, y)$ to be true, either $E(x, y)$ (the base case), or inductively, for some $z$, $E(x, z)$ is true and $S(z, y)$ is used to establish $S(x, y)$ but not the other way around.

To see how these axioms work, consider the graph in Figure 1 with four vertices $a, b, c, d$, with $E$ representing the edge relation: $E(a, b), E(b, a), E(c, d), E(d, c)$.
$E(c, a), E(c, d)$. Clearly, if there is a path from $x$ to $y$, then $S(x, y)$ (by sentence (6)). We want to show that if there is no path from $x$ to $y$, then $\neg S(x, y)$. Consider $S(d, a)$. If it is true, then since $\neg E(d, a)$, there must be an $x$ such that

$$E(d, x) \land S(x, a) \land \overrightarrow{x a} \leq SS \overrightarrow{da} \land \neg \overrightarrow{da} \leq SS \overrightarrow{xa}.$$ 

This is false as there is no edge going out of $d$.

Now consider $S(a, c)$. If it is true, then there must be an $z$ such that

$$E(a, z) \land S(z, c) \land \overrightarrow{zc} \leq SS \overrightarrow{ac} \land \neg \overrightarrow{ac} \leq SS \overrightarrow{zc}.$$ 

So $z$ must be $b$, and

$$S(b, c) \land \overrightarrow{bc} \leq SS \overrightarrow{ac} \land \neg \overrightarrow{ac} \leq SS \overrightarrow{bc}.$$ 

Since $S(b, c)$ is true and there is no edge from $b$ to $c$, there must be a $y$ such that

$$E(b, y) \land S(y, c) \land \overrightarrow{yc} \leq SS \overrightarrow{bc} \land \neg \overrightarrow{bc} \leq SS \overrightarrow{yc}.$$ 

So $y$ must be $a$, and

$$S(a, c) \land \overrightarrow{ac} \leq SS \overrightarrow{bc} \land \neg \overrightarrow{bc} \leq SS \overrightarrow{ac}.$$ 

However, this contradicts with (9).

Notice that the Clark’s completion of TC, i.e. $\forall xy(S(x, y) \leftrightarrow (E(x, y) \lor \exists zE(x, z) \land S(z, y)))$ (see Example 1), does not assure that $S$ is the transitive closure of the edge relation. For instance, for the above, the following interpretation on $S$

$$S(a, b), S(b, a), S(a, c), S(c, a), S(b, c), S(c, b), S(a, d), S(b, d), S(c, d)$$

satisfies the Clark’s completion of TC. However, there is no path from, e.g. $a$ to $c$, in the graph shown in Figure 1.

![Figure 1: An example graph](image-url)
3.2. The main theorem

In order to introduce the main theorem, we first consider a notion called derivation order, which is a reformulation of Fages’ “well-supportedness” under our context [8]. Roughly speaking, a derivation order is a sequence of ground atoms, in which the anterior ones are used to derive the posterior ones.

Formally, a derivation order of a finite structure $A$ on a program $\Pi$ is a sequence of ground atoms $P_1(\overline{a_1}), \ldots, P_k(\overline{a_k})$ such that $\{P_1(\overline{a_1}), \ldots, P_k(\overline{a_k})\} = \text{Int}_A$ (see Definition 2), and for all $i$ ($1 \leq i \leq k$), there exists a rule $r \in \Pi$ and an assignment $\eta$ such that

- $\text{Head}(r)\eta = P_i(\overline{a_i})$,
- for all intensional atoms $Q(\overline{t})$ in $\text{Pos}(r)$, $Q(\overline{t})\eta \in \{P_1(\overline{a_1}), \ldots, P_{i-1}(\overline{a_{i-1}})\}$,
- for all intensional atoms $Q(\overline{t})$ in $\text{Neg}(r)$, $Q(\overline{t})\eta \notin \text{Int}_A$,
- for all extensional atoms $Q(\overline{t})$ in $\text{Pos}(r)$ (or $\text{Neg}(r)$ resp.), $Q(\overline{t})\eta \in \text{Ext}_A$ (or $Q(\overline{t})\eta \notin \text{Ext}_A$ resp.).

Example 4. Consider the following program $\Pi_1$

\[
\begin{align*}
p_1 & \leftarrow p_2, \\
p_2 & \leftarrow p_1, \\
p_1 & \leftarrow \text{not } p_3.
\end{align*}
\]

Clearly, $X = \{p_1, p_2\}$ is an answer set of $\Pi_1$. According to the definition, $p_1, p_2$ is a derivation order of $X$ on $\Pi_1$ but $p_2, p_1$ is not. This is because $p_1$ is used to derive $p_2$ but not the other way around.

We show that derivation orders and answer sets are corresponded.

Lemma 2. Let $\Pi$ be a normal logic program whose signature is $\sigma$, and $A$ a finite $\sigma$-structure. Then, $A$ is an answer set of $\Pi$ iff

(i) $A \models \overline{\Pi}$;
(ii) there exists a (possibly many) derivation order of $A$ on $\Pi$. 

14
Proof: According to the grounding definition (see Definition 1), it suffices to prove this assertion in the propositional case.

On one hand, consider a finite propositional program Π and a set \( \mathcal{A} \) of atoms. If \( \mathcal{A} \) is an answer set of Π, then \( \mathcal{A} \) satisfies Π according to the Gelfond-Lifschitz transformation semantics. In addition, we construct a sequence of atoms \( \{p_1, p_2, \ldots, p_k\} \subseteq \mathcal{A} \) as follows:

- for any \( i \) (\( 1 \leq i \leq k \)), there exists a rule \( r \) in Π such that \( \text{Head}(r) = p_i \), \( \text{Pos}(r) \subseteq \{p_1, \ldots, p_{i-1}\} \) and \( \text{Neg}(r) \cap \mathcal{A} = \emptyset \),
- there does not exist \( r \) in Π such that \( \text{Pos}(r) \subseteq \{p_1, \ldots, p_k\} \), \( \text{Neg}(r) \cap \mathcal{A} = \emptyset \) and \( \text{Head}(r) \not\in \{p_1, \ldots, p_k\} \).

Then, \( \{p_1, p_2, \ldots, p_k\} = \mathcal{A} \). Otherwise, \( \{p_1, p_2, \ldots, p_k\} \models \Pi^{A} \) but \( \{p_1, p_2, \ldots, p_k\} \subset \mathcal{A} \), a contradiction. Hence, \( \{p_1, p_2, \ldots, p_k\} \) is a derivation order of \( \mathcal{A} \) on Π.

On the other hand, suppose that \( \mathcal{A} \models \Pi \) and there exists a derivation order of \( \mathcal{A} \) on Π. To prove that \( \mathcal{A} \) is an answer set of Π, it suffices to show that there does not exist \( \mathcal{A}' \subset \mathcal{A} \) such that \( \mathcal{A}' \) satisfies \( \Pi^{A} \). Otherwise, let \( p \) be the atom in \( \mathcal{A} \setminus \mathcal{A}' \) with the least ordinal in the derivation order. Then, according to the definition, there exists a rule \( r \) such that \( \text{Head}(r) = p \), \( \text{Pos}(r) \subseteq \mathcal{A}' \) (since \( p \) has the least ordinal in the derivation order) and \( \text{Neg}(r) \cap \mathcal{A} = \emptyset \).

Therefore, \( \text{Neg}(r) \cap \mathcal{A}' = \emptyset \). It follows that \( \mathcal{A}' \models \text{Pos}(r) \) but \( \mathcal{A}' \not\models \text{Head}(r) \).

Hence, \( \mathcal{A}' \) does not satisfy \( \Pi^{A} \), a contradiction. \( \blacksquare \)

Now we are able to present the following main theorem.

**Theorem 2.** Let Π be a normal logic program whose signature is \( \sigma \), and \( \mathcal{A} \) a finite \( \sigma \)-structure. Then, \( \mathcal{A} \) is an answer set of Π if and only if there exists a model \( \mathcal{M} \) of \( \text{OC}(\Pi) \) such that \( \mathcal{A} \) is the reduct of \( \mathcal{M} \) on \( \sigma \).

**Proof:** On one hand, we show that every finite answer set \( \mathcal{A} \) of Π can be expanded to a model of \( \text{OC}(\Pi) \). By Lemma 2, there exists a derivation order of \( \mathcal{A} \) on Π. Then, based on it, we construct a finite structure \( \mathcal{M} \) by expanding \( \mathcal{A} \) with the following interpretations on \( \leq_{PQ} \) for each pair \( (P, Q) \) of intensional predicates in Π:

\[
\overrightarrow{a} \leq_{PQ} \overrightarrow{b} \text{ iff there exists a path from } Q(\overrightarrow{b}) \text{ to } P(\overrightarrow{a}) \text{ in the dependency graph of the ground program } \Pi_{\mathcal{A}} \text{ and the position of } P(\overrightarrow{a}) \text{ is before the position of } Q(\overrightarrow{b}) \text{ in the derivation order,}
\]

15
where \( \overrightarrow{a} \) and \( \overrightarrow{b} \) are two tuples of elements in the domain of \( A \) that match the arities of \( P \) and \( Q \) respectively. In this case, we say that there is a derivation path from \( P(\overrightarrow{a}) \) to \( Q(\overrightarrow{b}) \) (with respect to the derivation order).

We need to prove that \( M \) is a model of both \( \text{TranS}(\Pi) \) and \( \text{MComp}(\Pi) \). Firstly, it is easy to see that \( M \) is a model of \( \text{TranS}(\Pi) \) by contradiction. Secondly, we show that \( M \) is a model of \( \text{MComp}(\Pi) \) as well. Clearly, \( M \) is a model of the first part of \( \text{MComp}(\Pi) \), i.e. Equation (3) since \( M \) is expanded from \( A \), a model of the Clark's completion of \( \Pi \). For the second part of \( \text{MComp}(\Pi) \), i.e. Equation (4), we prove it by contradiction. Suppose that \( M \) is not a model of Equation (4). Then,

\[
M \models \bigvee_{P \in \Omega_{\Pi}} (P(\overrightarrow{x}) \land \bigwedge_{1 \leq i \leq k} \forall \overrightarrow{y}_i (\overrightarrow{\text{Body}}_i \rightarrow \\
\bigwedge_{Q(\overrightarrow{z}) \in \text{Pos}_i \cap \Omega_{\Pi}} \neg \overrightarrow{z} \leq \overrightarrow{QP} - \overrightarrow{x} \lor - \overrightarrow{x} \leq \overrightarrow{PQ} \overrightarrow{z})).
\]

Therefore, there exists \( P(\overrightarrow{a}) \in M \) such that for all assignments \( \eta \) and all rules \( r \) whose head mentions \( P \), if \( M \models \overrightarrow{\text{Body}}(r) \eta \), then there exists an intensional atom \( Q(\overrightarrow{z}) \) in the positive body of \( r \) such that \( M \models \neg \overrightarrow{z} \leq \overrightarrow{QP} - \overrightarrow{x} \eta \) or \( M \models \overrightarrow{z} \leq \overrightarrow{PQ} - \overrightarrow{x} \eta \).

Now, consider the position of \( P(\overrightarrow{a}) \) in the derivation order. There exists a rule \( r \) and an assignment \( \eta \) satisfying the conditions mentioned above. Hence, according to the conditions, for all intensional atom \( Q(\overrightarrow{t}) \) in the positive body of \( r \), we have that the ordinal of \( Q(\overrightarrow{t}) \eta \) in the derivation order is less than the ordinal of \( P(\overrightarrow{a}) \) in the derivation order. Hence, \( M \models \overrightarrow{t} \leq \overrightarrow{QP} - \overrightarrow{x} \eta \) and \( M \not\models \overrightarrow{t} \leq \overrightarrow{PQ} - \overrightarrow{x} \eta \), a contradiction.

This shows that \( M \), expanded from \( A \), is a model of \( \text{OC}(\Pi) \).

On the other hand, we prove that the reduct of any finite model \( M \) of \( \text{OC}(\Pi) \) on \( \sigma \) must be an answer set of \( \Pi \). Clearly, \( M \uparrow \sigma \) is a model of \( \text{Comp}(\Pi) \) since \( M\text{Comp}(\Pi) \models \text{Comp}(\Pi) \). Hence, according to the loop formula characterization of answer set semantics in the propositional case [20], it suffices to show that for all loops \( L \) of the ground program \( \Pi_{M\uparrow \sigma} \), the set of ground atoms \( EQ_{M\uparrow \sigma} \cup Ext_{M\uparrow \sigma} \cup Int_{M\uparrow \sigma} \) is a model of its loop formula.

We prove this by contradiction. Suppose that there exists a loop \( L \) of the ground program \( \Pi_{M\uparrow \sigma} \) such that the above set of ground atoms is not a model of \( LF(L, \Pi_{M\uparrow \sigma}) \). Then, there exists a ground atom \( P_0(\overrightarrow{a}_0) \in L \) and
holds. Hence, we can get a sequence of ground atoms $\overline{a_0} \rightarrow \overline{b}$ in the positive body of $\text{Body}_0$, $\overline{b} \leq_{QP} \overline{a_0} \land \neg \overline{a_0} \leq_{P_0} \overline{b}$ holds, where $Q$ is an intensional predicate of $\Pi$ and $\overline{b}$ a tuple of elements in $M$ that matches the arity of $Q$. If every $Q(\overline{b})$ is not in $L$, then $\text{Body}_0$ is an external support of $P_0(\overline{a_0})$ with respect to $L$, a contradiction. Hence, there exists $P_1(\overline{a_1})$ in the positive body of $\text{Body}_0$ such that $P_1(\overline{a_1}) \in \text{Int}_{\Pi|\sigma}$, $P_1(\overline{a_1}) \in L$ and $\overline{a_1} \leq_{P_0} \overline{a_0} \land \neg \overline{a_0} \leq_{P_0} \overline{a_1}$ holds.

Again, following the same procedure described above, there exists a support $\text{Body}_1$ of $P_1(\overline{a_1})$, and a ground atom $P_2(\overline{a_2})$ in the positive body of $\text{Body}_1$ such that $P_2(\overline{a_2}) \in \text{Int}_{\Pi|\sigma}$, $P_2(\overline{a_2}) \in L$ and $\overline{a_2} \leq_{P_2, P_1} \overline{a_1} \land \neg \overline{a_1} \leq_{P_2, P_1} \overline{a_2}$ holds. Hence, we can get a sequence of ground atoms $P_0(\overline{a_0}), P_1(\overline{a_1}), \ldots, P_i(\overline{a_i}), \ldots$ such that for all $i$, $P_i(\overline{a_i}) \in \text{Int}_{\Pi|\sigma}$, $P_i(\overline{a_i}) \in L$. In addition, both $\overline{a_{i+1}} \leq_{P_i, P_{i+1}} \overline{a_i}$ and $\neg \overline{a_i} \leq_{P_i, P_{i+1}} \overline{a_{i+1}}$ hold.

Next, we show that this sequence cannot be infinite due to the finiteness of the structure $M$ and the fact that the new comparison predicates satisfy transitivity. Since $M$ is finite, there exists $k < l$ such that $P_k(\overline{a_k}) = P_l(\overline{a_l})$ in the above sequence. However, since for all $i$, $\overline{a_{i+1}} \leq_{P_i, P_{i+1}} \overline{a_i}$ holds, we have that $\overline{a_i} \leq_{P_k, P_{k+1}} \overline{a_{k+1}}$ holds as well according to the transitivity axioms. Hence, $\overline{a_k} \leq_{P_k, P_{k+1}} \overline{a_{k+1}}$ holds since $P_k(\overline{a_k}) = P_l(\overline{a_l})$. This contradicts to the fact that $\neg \overline{a_{k+1}} \leq_{P_k, P_{k+1}} \overline{a_k}$ holds.

This shows that $M \uparrow \sigma$ is an answer set of $\Pi$. ■

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

**downgrading** every ground atom $P(\overline{a})$ in $A$ has some supports from earlier stages. The “support” part is ensured by Clark’s completion, and the “earlier stages” part is ensured by adding some assertions on the comparison predicates (see Equation (4));

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

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

Together with the above three conditions, each ground atom \( P(\overline{a}) \) 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.

### 3.3. Normal logic program with constraints

Recall that we have required the head of a rule to be a proper atom. If we allow the head to be empty, then we have so-called constraints:

\[
\leftarrow \beta_1, \ldots, \beta_k, \text{not} \gamma_1, \ldots, \text{not} \gamma_l,
\]

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

\[
\forall \overrightarrow{y} \neg(\beta_1 \land \cdots \land \beta_k \land \neg\gamma_1 \land \cdots \land \neg\gamma_l),
\]

where \( \overrightarrow{y} \) is the tuple of all variables occurring in (10). In the following, if \( c \) is a constraint of form (10), then we use \( \hat{c} \) to denote its corresponding formula above.

A normal logic program with constraints is then a finite set of rules and constraints. The stable model (answer set) semantics can be extended to normal logic programs with constraints: a model is an answer set of the set of the program and satisfies all the constraints.

Both Clark’s completion and our ordered completion can be extended to normal logic programs with constraints: one simply adds the sentences corresponding to the constraints to the respective completions.

**Proposition 3.** Let \( \Pi \) be a normal logic program whose signature is \( \sigma \), \( C \) a set of constraints, and \( A \) a finite \( \sigma \)-structure. Then, \( A \) is an answer set of \( \Pi \cup C \) iff there exists a model \( M \) of \( OC(\Pi) \cup \{\hat{c} \mid c \in C\} \), such that \( A \) is the reduct of \( M \) on \( \sigma \).
**Proof:** \( \mathcal{A} \) is an answer set of \( \Pi \cup C \) iff \( \mathcal{A} \) is an answer set of \( \Pi \) and \( \mathcal{A} \) is a model of \( \{ \hat{c} \mid c \in C \} \) iff there exists \( \mathcal{M} \), which is a model of both \( OC(\Pi) \) and \( \{ \hat{c} \mid c \in C \} \) and whose reduct on \( \sigma \) is \( \mathcal{A} \) iff there exists \( \mathcal{M} \), which is a model of \( OC(\Pi) \cup \{ \hat{c} \mid c \in C \} \) such that \( \mathcal{A} \) is the reduct of \( \mathcal{M} \) on \( \sigma \).

**Example 5.** The following program checks whether all the nodes of a given graph can be reached from a given initial node.

\[
\begin{align*}
R(a) \\
R(x) & \leftarrow R(y), E(y, x) \\
& \leftarrow \text{not } R(x),
\end{align*}
\]

where \( E \) is the only extensional predicate representing the edges of the graph; \( a \) is a constant representing the initial node; and \( R \) is the only intensional predicate representing whether a node can be reached from \( a \). The program has a stable model iff all the nodes in the graph can be reached from \( a \). According to Proposition 3, this program can be captured by the following sentence:

\[
R(a) \land \forall xy(E(y, x) \land R(y) \rightarrow R(x)) \\
\land \forall x(R(x) \rightarrow x = a \lor \exists y(R(y) \land E(y, x) \land y \leq RR x \land \neg x \leq RR y)) \\
\land \forall xyz(x \leq RR y \land y \leq RR z \rightarrow x \leq RR z) \\
\land \forall xR(x).
\]

### 3.4. Adding choice rules

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

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

where \( P \) is a predicate and \( \overrightarrow{x} \) 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.

The stable model (answer set) semantics of normal logic programs with choice rules (possibly with constraints as well) can be defined similarly by grounding. More precisely, the set of ground rules of a choice rule of form (11) on a structure \( \mathcal{M} \) contains all rules of the form:

\[
\{ P(\overrightarrow{w}) \},
\]

19
where \( \overline{u} \) is a tuple of domain elements in \( M \) that matches the arity of \( P \). The set of ground constraints of a constraint of form (10) on a structure \( M \) contains all the instances of the constraint under \( M \).

The answer set semantics for propositional programs with choice rules and constraints can be defined by Gelfond-Lifschitz transformation as well [23]. Let \( p \) be a propositional atom and \( A \) a set of atoms. The reduct of the choice rule \( \{p\} \) relative to \( A \) is \( p \) itself if \( p \in A \), and empty (i.e. \( \top \)) otherwise. Again, a set \( A \) of propositional atoms is said to be an answer set of a propositional program \( \Pi \) with choice rules and constraints if \( A \models \Pi \) and \( A \) is the minimal model of \( \Pi \).

Then, a structure \( A \) is said to be a stable model of a first-order normal program \( \Pi \) with choice rules and constraints if \( EQ_A \cup Ext_A \cup Int_A \) is an answer set of the ground program \( \Pi_A \) in the propositional case.

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

**Proposition 4.** Let \( \Pi \) be a normal logic program whose signature is \( \sigma \), \( \Omega \subseteq \sigma \) a set of predicates in \( \sigma \), \( C \) a set of constraints, \( \text{Choice}(\Omega) \) the set of choice rules for every predicate in \( \Omega \), and \( A \) a finite \( \sigma \)-structure. Then, \( A \) is an answer set of \( \Pi \cup C \cup \text{Choice}(\Omega) \) iff there exists a model \( M \) of the following set of sentences such that \( A \) is the reduct of \( M \) on \( \sigma \):

- For each intensional predicate \( P \), the following sentence:
  \[
  \forall \overline{x} \left( \bigwedge_{1 \leq i \leq k} \exists \overline{y}_i (\text{Body}_i \rightarrow P(\overline{x})) \right).
  \]

- For each intensional predicate \( P \) not in \( \Omega \), the following sentence:
  \[
  \forall \overline{x} (P(\overline{x}) \rightarrow \bigwedge_{1 \leq i \leq k} \exists \overline{y}_i (\text{Body}_i \wedge \bigwedge_{Q(\overline{z}) \in Pos_i, Q \in \Pi \setminus \Omega} (\overline{z} \leq_{QP} \overline{x} \wedge \neg \overline{x} \leq_{PQ} \overline{z})))\).

\(^2\)Firstly, this means that \( A \) satisfies all constraints. Secondly, \( A \) is regarded as a model of every choice rule.
• For each triple of intensional predicates $P$, $Q$, and $R$ not in $\Omega$, the following sentence:

$$\bigwedge_{P,Q,R \in \Omega \setminus \Omega} \forall \overrightarrow{x} \overrightarrow{y} \overrightarrow{z} (\overrightarrow{x} \leq_{PQ} \overrightarrow{y} \wedge \overrightarrow{y} \leq_{QR} \overrightarrow{z} \rightarrow \overrightarrow{x} \leq_{PR} \overrightarrow{z}).$$

• Finally, for each $c \in C$, the sentence $\widehat{c}$.

Proof: This assertion follows directly from Proposition 3 and the following fact: the answer sets of $\Pi \cup C \cup \text{Choice}(\Omega)$ are exactly the same as the answer sets of $\Pi^* \cup C$, where $\Pi^*$ is the program obtained from $\Pi$ by rewriting each rule of form (1), whose head mentions predicates from $\Omega$, to the following constraint

$$\leftarrow \beta_1, \ldots, \beta_k, \text{not } \gamma_1, \ldots, \text{not } \gamma_l, \text{not } \alpha.$$

Example 6. Consider the following normal program with constraints and choice rules for computing all Hamiltonian circuits of a graph:

\begin{align*}
\{hc(x, y)\} \\
\leftarrow hc(x, y), \text{not } 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 \text{not } 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; and $R(x)$ is an intensional predicate to check that all vertices are in the Hamiltonian circuit. In particular, the first rule of the program is a choice rule to guess a possible Hamiltonian circuit.
According to Proposition 4, this program can be captured by the following sentence:

\[ \forall x (hc(a, x) \rightarrow R(x)) \land \forall x y (hc(y, x) \land R(y) \rightarrow R(x)) \land \forall x y z (x \leq_R y \land y \leq_R z \rightarrow x \leq_R z) \land \forall x y (hc(x, y) \land \neg E(x, y)) \land \forall x y (hc(x, y) \land hc(x, z) \land y \neq z) \land \forall x y (hc(y, x) \land hc(z, x) \land y \neq z) \land \forall x R(x). \]

3.5. Arbitrary structures

It is worth mentioning that the correspondence between classical first-order models of ordered completions and stable models of a logic program holds only on finite structures. In general, the result does not hold if infinite structures are allowed. For instance, the ordered completion of Transitive Closure (TC) in Example 3 on finite structures does not capture TC on some infinite structures.

Example 7. [Transitive Closure continued] Consider the graph that contains an infinite chain and an individual vertex. Let \( a_1, a_2, \ldots \) be an infinite chain such that \( E(a_i, a_{i+1}) \) for all \( i \), and \( b \) a node different from all \( a_i \). Consider the structure \( M \) of the signature \( \{ E, S, \leq_{SS} \} \) such that

- \( S(a_i, a_j) \) for all \( i < j \);
- \( S(a_i, b) \) for all \( i \);
- \( \overline{a_i a_j} \leq_{SS} \overline{a_k a_l} \) for all \( j - i \leq l - k \), where \( i < j \) and \( k < l \);
- \( \overline{a_i b} \leq_{SS} \overline{a_j b} \) for all \( i < j \).

It can be checked that \( M \) is a model of the ordered completion of TC (see Example 3). However, clearly, \( S \) is not the transitive closure of the graph given in this example.

It still remains the question whether or not Transitive Closure can be captured by other first-order theories (even infinite) with auxiliary predicates. Unfortunately, the answer is negative either.
Proposition 5. There does not exist a first-order theory whose signature contains the signature of TC, and the reducts of all its models are exactly corresponding to the stable models of TC on arbitrary structures.

Proof: We prove this assertion by contradiction. Let $\sigma$ be the signature of TC. We assume that $\phi$ is a first-order theory, whose vocabulary is $\sigma_1$ such that $\sigma \subseteq \sigma_1$, and the reducts of the models of $\phi$ on $\sigma$ are exactly the stable models of TC. It is well known that TC can be defined by a universal second-order theory [6]. Therefore, the complement of TC can be defined by an existential second-order theory. Hence, there exist a first-order theory whose signature contains $\sigma$, and the reducts of all its models on $\sigma$ are exactly the complement of the class of stable models of TC on arbitrary structures.

Let $\psi$ be such as a theory of signature $\sigma_2$. Without loss of generality, we can assume that $\sigma_1 \cap \sigma_2 = \sigma$. Then, $\phi \models \neg \psi$ in the first-order language $\sigma_1 \cup \sigma_2$. Thus, according to Craig’s Interpolation Theorem, there exists a theory $\phi_0$ of the signature $\sigma_1 \cap \sigma_2$ (namely $\sigma$) such that $\phi \models \phi_0$ and $\phi_0 \models \neg \psi$. This shows that TC is exactly captured by the theory $\phi_0$, whose signature is $\sigma$. This contradicts to the well-known result that TC is not first-order definable on arbitrary structures [6].

3.6. Disjunctive logic programs

Disjunctive logic programs is a very important extension of normal programs for dealing with incomplete information [7, 14]. A disjunctive logic program is a finite set of disjunctive rules of the following form

$$\alpha_1; \ldots; \alpha_n \leftarrow \beta_1, \ldots, \beta_k, \text{not } \gamma_1, \ldots, \text{not } \gamma_l.$$ (12)

Similar to normal programs, we can distinguish intensional and extensional predicates here. The answer set semantics for disjunctive logic programs can be defined similarly by grounding [7, 14].

A natural question arises as whether the ordered completion can be extended to first-order disjunctive programs. Unfortunately, the answer is negative provided some well-recognized assumptions in the computational complexity theory are true. Actually, our following proposition shows a stronger result that there exist disjunctive programs that cannot be captured by any first-order sentences with a larger signature.
Proposition 6. There exist a disjunctive program $\Pi$ such that it cannot be captured by any first-order sentence with the same or a larger signature unless $NP = coNP$. That is, there is no first-order sentence $\phi$ whose signature contains the signature of $\Pi$, and the reducts of all its finite models are exactly the finite stable models of $\Pi$.

Proof: We show that the following program 3-UNCOLOR (originated from Example 2 in [7]) cannot be captured by any first-order sentences on finite structures if $NP \neq coNP$:

$$
\begin{align*}
R(x); G(x); B(x) & \leftarrow , \\
NC & \leftarrow E(x, y), R(x), R(y), \\
NC & \leftarrow E(x, y), G(x), G(y), \\
NC & \leftarrow E(x, y), B(x), B(y), \\
R(x) & \leftarrow NC, \\
G(x) & \leftarrow NC, \\
B(x) & \leftarrow NC, \\
NC & \leftarrow \text{not } NC,
\end{align*}
$$

where $E$ is the only extensional predicate to represent a graph; $R$, $G$ and $B$ are three different colors respectively, and NC is a 0-ary predicate to claim that this graph cannot be colored. It is not difficult to check that the program has answer sets iff the graph, represented by $E$, cannot be colored by the three colors. In addition, in this case, there is a unique answer set that contains the given graph, $NC$ and the full interpretation for $R$, $G$ and $B$.

Assuming that $NP \neq coNP$, no $coNP$ complete problem is in $NP$. Hence, the problem of 3-uncolorability (a well-known $coNP$ complete problem) is not in $NP$. By Fagin’s theorem [10], the Boolean query of 3-uncolorability for a given graph cannot be defined by an existential second-order sentence. On the other hand, assume that there exists a first-order sentence $\phi$ whose signature contains the signature of 3-UNCOLOR, and the reducts of all its finite models are exactly corresponding to the finite stable models of 3-UNCOLOR. Then, the Boolean query of 3-uncolorability can be defined by the following existential second-order sentence

$$
\exists \mathcal{P} \phi,
$$
where \( \mathcal{P} \) is a set of predicates including \( NC, R, G, B \) and all the other predicates in \( \phi \) but not in 3-UNCOLOR, a contradiction. This shows that 3-UNCOLOR cannot be captured by any first-order sentences with a larger signature. ■

Following from the proof of Proposition 6 and Theorem 2, another negative result is that, most likely, first-order disjunctive logic programs cannot be reduced to normal programs on finite structures, even with new predicates.

**Corollary 7.** Unless \( NP \neq coNP \), there is no normal program whose signature contains the signature of 3-UNCOLOR, and the reducts of all its finite stable models are exactly corresponding to the finite stable models of 3-UNCOLOR.

**Proof:** Otherwise, according to Theorem 2, the program 3-UNCOLOR can be captured by a first-order sentence with a larger signature. ■

In fact, the above two results coincide with the result presented in [7], stating that in terms of brave reasoning, disjunctive logic programs with the stable model semantics exactly capture the complexity class \( \Sigma^P_2 \), while normal programs only capture \( NP \).

4. **Optimizations**

In this section, we present several techniques to optimize ordered completion introduced in Section 3. The goal of these techniques is for simplifying the translation, including reducing the number of new predicates, the arities of new predicates and the overall length of the ordered completion. As we will see, the techniques presented below can be combined together. For the sake of clarity, we will introduce them step-by-step.

4.1. **Exploiting maximal predicate loops**

In the definition of our ordered completion, we introduce a comparison predicate between each pair of predicates. This is not necessary. We only need to do so for pairs of predicates that belong to a same strongly connected component in the predicate dependency graph of the program.

Formally, the *predicate dependency graph* of a first-order program \( \Pi \) is a finite graph \( PG_\Pi = (V, E) \), where \( V \) is the set of all intensional predicates
of Π and \((P, Q) \in E\) iff there is a rule whose head mentions \(P\) and whose positive body contains \(Q\). Maximal predicate loops of the program Π are then strongly connected components of \(PG_Π\). Since \(PG_Π\) can be constructed easily, all the maximal predicate loops of Π can be identified in polynomial time with respect to the size of Π.

The ordered completions on maximal predicate loops are the same as the ordered completions except that the comparison predicates \(\leq_{PQ}\) are defined only when \(P\) and \(Q\) belong to a same maximal predicate loop. More precisely, the ordered completion of Π on maximal predicate loops, denoted by \(OC^1_1(Π)\), is of the similar form as the ordered completion of Π (see Definition 3), except that

- \(Q(\overline{a})\) in Equation (4) ranges over all the intensional atoms in the positive part of \(Body_i\) such that \(Q\) and \(P\) are in the same maximal predicate loop of Π;
- \(P, Q\) and \(R\) in Equation (5) are intensional predicates that belong to the same maximal predicate loop of Π.

The following proposition is a refinement of the main theorem.

**Proposition 8.** Let Π be a normal logic program whose signature is \(σ\), and \(A\) a finite \(σ\)-structure. Then, \(A\) is an answer set of Π if and only if there exists a model \(M\) of \(OC^1_1(Π)\) such that \(A\) is the reduct of \(M\) on \(σ\).

**Proof:** The “only if” part can be proved by using a similar construction except that the comparison predicates \(\leq_{PQ}\) are only defined over those pairs of \((P, Q)\) in a same maximal predicate loop. For the “if” part, we can prove it by contradiction again. Similarly, we can get a same sequence of ground atoms \(P_0(\overline{a}_0), P_1(\overline{a}_1), \ldots, P_i(\overline{a}_i), \ldots\) such that for all \(i, P_i(\overline{a}_i) \in Int(M^σ), P_i(\overline{a}_i) \in L,\) and both \(\overline{a}_{i+1} \leq_{P_i+1} \overline{a}_i\) and \(\neg \overline{a}_i \leq_{P_{i+1}} \overline{a}_{i+1}\) hold. Notice that for all \(i (i \geq 0),\) there exists an edge from \(P_i\) to \(P_{i+1}\) in \(PG_Π\). Again, there exists \(k < l\) such that \(P_k(\overline{a}_k) = P_l(\overline{a}_l).\) Hence, \(P_k, P_{k+1}, \ldots, P_l\) must be in a maximal predicate loop of Π, say \(L.\) Hence, according to the transitivity axioms with respect to \(L, \overline{a}_i \leq_{P_{l+1}} \overline{a}_{l+1}\) holds. This contradicts to the facts that \(P_k(\overline{a}_k) = P_l(\overline{a}_l)\) and \(\neg \overline{a}_k \leq_{P_{k+1}} \overline{a}_{k+1}\) holds.

In many cases, restricting comparison predicates on maximal predicate loops results in a much smaller ordered completion. As many benchmark logic programs have no predicate loops, in these cases, \(OC^1_1(Π)\) is exactly the
Clark’s completion of Π. Even for those programs with predicate loops, this optimization technique will also significantly simplify the ordered completion since, in many cases, not all predicates are in a same predicate loop. Let us consider the following program for computing all Hamiltonian circuits.

**Example 8.** The following program $HC$ is another encoding for computing all Hamiltonian circuits of a given graph [23]:

\[
\begin{align*}
hc(x, y) &\leftarrow \text{arc}(x, y), \neg \text{otherroute}(x, y), \\
\text{otherroute}(x, y) &\leftarrow \text{arc}(x, y), \text{arc}(x, z), \hc(x, z), y \neq z, \\
\text{otherroute}(x, y) &\leftarrow \text{arc}(x, y), \text{arc}(z, y), \hc(z, y), x \neq z, \\
\text{reached}(y) &\leftarrow \text{arc}(x, y), \hc(x, y), \text{reached}(x), \neg \text{init}(x), \\
\text{reached}(y) &\leftarrow \text{arc}(x, y), \hc(x, y), \text{init}(x), \\
&\leftarrow \text{vertex}(x), \neg \text{reached}(x).
\end{align*}
\]

This program has three intensional predicates: $hc$, $otherroute$ and $reached$. According to the original version of the ordered completion (see Definition 3), we need to introduce 9 comparison predicates, and the maximal arity is 4.

However, by using maximal predicate loops, only one auxiliary predicate is needed since $HC$ has only one maximal predicate loop, namely \{reached\}. The only comparison predicate needed is $\leq_{RR} (x, y)$, which is binary. Hence, $OC_1(HC)$ is the following set of sentences:

\[
\begin{align*}
\forall xy & (hc(x, y) \leftrightarrow \text{arc}(x, y) \land \neg \text{otherroute}(x, y)), \\
\forall xy & (\text{otherroute}(x, y) \leftrightarrow \exists z(\text{arc}(x, y) \land \text{arc}(x, z) \land \hc(x, z) \land y \neq z) \lor \\
& \exists z(\text{arc}(x, y) \land \text{arc}(z, y) \land \hc(z, y) \land x \neq z)), \\
\forall y((\exists x(\text{arc}(x, y) \land \hc(x, y) \land \text{reached}(x) \land \neg \text{init}(x))) \lor \\
& \exists x(\text{arc}(x, y) \land \hc(x, y) \land \text{init}(x))) \rightarrow \text{reached}(y)), \\
\forall y & (\text{reached}(y) \rightarrow (\exists x(\text{arc}(x, y) \land \hc(x, y) \land \text{init}(x))) \lor \\
& \exists x(\text{arc}(x, y) \land \hc(x, y) \land \text{reached}(x) \land \neg \text{init}(x) \land x \leq_{RR} y \land \neg y \leq_{RR} x)), \\
\forall x & (\neg (\text{vertex}(x) \land \neg \text{reached}(x))), \\
\forall xyz & (x \leq_{RR} y \land y \leq_{RR} z \rightarrow x \leq_{RR} z).
\end{align*}
\]

**4.2. Folding reverse comparison predicates**

In the definition of the ordered completion, for a pair of intensional predicates \((P, Q)\), we introduce two new comparison predicates $\leq_{PQ}$ and $\leq_{QP}$. 27
This can be simplified by just introducing one of them because if there is a
derivation path from one ground atom to another, then there is none from
the other way around.

Formally, we abbreviate \( \overrightarrow{x} \leq_{QP} \overrightarrow{y} \land \neg \overrightarrow{x} \leq_{QP} \overrightarrow{y} \) as \( \overrightarrow{x} <_{QP} \overrightarrow{y} \), meaning
that there is a derivation path from \( P(\overrightarrow{x}) \) to \( Q(\overrightarrow{y}) \) but not the other way
around. We rank all the intensional predicates occurring in \( \Pi \) as \( \{P_1, \ldots, P_n\} \),
and define \( \text{Rank}(P_i) = i \) (1 \( \leq \) i \( \leq \) n). For every maximal predicate loop \( L \) of
\( \Pi \), and for every two predicates \( P_i, P_j \in L \), we introduce a new comparison
predicate \( <_{P_iP_j} \) if \( \text{Rank}(P_i) \leq \text{Rank}(P_j) \). This method reduces almost half
of the comparison predicates introduced.

The new version of the ordered completion by folding reverse comparison
predicates, denoted by \( OC_2(\Pi) \), is defined based on \( OC_1(\Pi) \) except that

- \( \overrightarrow{x} \leq_{QP} \overrightarrow{y} \land \neg \overrightarrow{x} \leq_{QP} \overrightarrow{y} \) in Equation (4) is replaced by \( \overrightarrow{x} <_{QP} \overrightarrow{y} \) if
  \( \text{Rank}(Q) \leq \text{Rank}(P) \), and \( \neg \overrightarrow{x} <_{QP} \overrightarrow{y} \) if \( \text{Rank}(Q) > \text{Rank}(P) \);

- the transitivity axioms, i.e. Equation (5) with respect to a predicate
  loop \( L \) are replaced as follows:

  - for any \( P \in L \), the sentence \( \forall \overrightarrow{x} \neg \overrightarrow{x} <_{QP} \overrightarrow{y} \), and
  - for any \( P_i, P_j, P_k \in L \) such that \( \text{Rank}(P_i) \leq \text{Rank}(P_j) \leq \text{Rank}(P_k) \),
    the following two sentences:

\[
\forall \overrightarrow{x} \forall \overrightarrow{y} \forall \overrightarrow{z} (\overrightarrow{x} <_{P_iP_j} \overrightarrow{y} \lor \overrightarrow{y} <_{P_jP_k} \overrightarrow{z} \lor \neg \overrightarrow{x} <_{P_iP_k} \overrightarrow{z}),
\]
\[
\forall \overrightarrow{x} \forall \overrightarrow{y} \forall \overrightarrow{z} (\neg \overrightarrow{x} <_{P_iP_j} \overrightarrow{y} \lor \neg \overrightarrow{y} <_{P_jP_k} \overrightarrow{z} \lor \overrightarrow{x} <_{P_iP_k} \overrightarrow{z}).
\]

**Proposition 9.** Let \( \Pi \) be a normal logic program whose signature is \( \sigma \), and
\( \mathcal{A} \) a finite \( \sigma \)-structure. Then, \( \mathcal{A} \) is an answer set of \( \Pi \) if and only if there
exists a model \( \mathcal{M} \) of \( OC_2(\Pi) \) such that \( \mathcal{A} \) is the reduct of \( \mathcal{M} \) on \( \sigma \).

**Proof:** Again, the “only if” part is easy. Now we prove the “if” part by con-
tradiction. Similarly, we can get a sequence of ground atoms \( P_0(\overrightarrow{a_0}), P_1(\overrightarrow{a_1}), \ldots, P_l(\overrightarrow{a_l}) \), \ldots \) such that for all \( i \), \( P_i(\overrightarrow{a_i}) \in \text{Int}_{\mathcal{M}\sigma} \), all \( P_l(\overrightarrow{a_l}) \) are in a ground
loop \( L \). In addition, \( \overrightarrow{a_{i+1}} <_{P_{i+1}P_i} \overrightarrow{a_i} \) holds if \( \text{Rank}(P_{i+1}) \leq \text{Rank}(P_i) \), and
\( \neg \overrightarrow{a_i} <_{P_{i+1}P_l} \overrightarrow{a_{i+1}} \) holds if \( \text{Rank}(P_{i+1}) > \text{Rank}(P_i) \). (Note that here \( i \) is not
the rank of \( P_i \).) Since all the \( <_{PQ} \), where \( \text{Rank}(P) \leq \text{Rank}(Q) \), satisfy the
new transitivity axioms, by induction, it can be proved that for all \( i \leq j \),
\( \overrightarrow{a_j} <_{P_{j+1}P_j} \overrightarrow{a_i} \) holds if \( \text{Rank}(P_j) \leq \text{Rank}(P_i) \), and \( \neg \overrightarrow{a_i} <_{P_{j+1}P_j} \overrightarrow{a_j} \) holds if
\( \text{Rank}(P_j) > \text{Rank}(P_i) \). Again, we can find \( k < l \) such that \( P_k(\overrightarrow{a_k}) = P_l(\overrightarrow{a_l}) \).

28
in the sequence. Hence, $\overrightarrow{a_i} <_{p_1p_{k+1}} \overrightarrow{a_{k+1}}$ holds if $\text{Rank}(P_i) \leq \text{Rank}(P_{k+1})$, and $\overrightarrow{a_{k+1}} <_{p_{k+1}p_i} \overrightarrow{a_i}$ holds if $\text{Rank}(P_i) > \text{Rank}(P_{k+1})$. There are three cases:

**Case 1:** $\text{Rank}(P_k) = \text{Rank}(P_i) < \text{Rank}(P_{k+1})$. Then, $\overrightarrow{a_i} <_{p_1p_{k+1}} \overrightarrow{a_{k+1}}$ holds. Therefore $\overrightarrow{a_k} <_{p_kp_{k+1}} \overrightarrow{a_{k+1}}$ holds since $P_k(\overrightarrow{a_k}) = P_i(\overrightarrow{a_i})$, a contradiction.

**Case 2:** $\text{Rank}(P_k) = \text{Rank}(P_i) > \text{Rank}(P_{k+1})$. Then, $\overrightarrow{a_{k+1}} <_{p_{k+1}p_i} \overrightarrow{a_i}$ holds. Therefore $\overrightarrow{a_{k+1}} <_{p_{k+1}p_k} \overrightarrow{a_k}$ holds since $P_k(\overrightarrow{a_k}) = P_i(\overrightarrow{a_i})$, a contradiction.

**Case 3:** $\text{Rank}(P_k) = \text{Rank}(P_i) = \text{Rank}(P_{k+1})$. Then, $\overrightarrow{a_i} <_{p_1p_{k+1}} \overrightarrow{a_{k+1}}$ holds. Therefore $\overrightarrow{a_k} <_{p_kp_{k+1}} \overrightarrow{a_{k+1}}$ holds. Also, $\overrightarrow{a_{k+1}} <_{p_{k+1}p_k} \overrightarrow{a_k}$ holds. Then, according to the transitivity axioms, $\overrightarrow{a_k} <_{p_kp_k} \overrightarrow{a_k}$ holds, a contradiction.

---

**Example 9.** Recall the program $\Pi_1$ in Example 4

\[
\begin{align*}
p_1 &\leftarrow p_2, \\
p_2 &\leftarrow p_1, \\
p_1 &\leftarrow \neg p_3.
\end{align*}
\]

Then, according to the definition, $OC_2(\Pi_1)$ is

\[
(p_2 \lor \neg p_3 \rightarrow p_1) \land (p_1 \rightarrow p_2)
\]
\[
\land (p_1 \rightarrow p_2 \land \neg <_{p_1p_2} \lor \neg p_3)
\]
\[
\land (p_2 \rightarrow p_1 \land <_{p_1p_2})
\]
\[
\land (<_{p_1p_1} \lor <_{p_1p_1} \lor <_{p_1p_1}) \land (\neg <_{p_1p_1} \lor \neg <_{p_1p_1} \lor <_{p_1p_1})
\]
\[
\land (<_{p_1p_1} \lor <_{p_1p_2} \lor <_{p_1p_2} \lor <_{p_1p_2} \lor <_{p_1p_2} \lor <_{p_1p_2})
\]
\[
\land (<_{p_2p_2} \lor <_{p_2p_2} \lor <_{p_2p_2} \lor <_{p_2p_2} \lor <_{p_2p_2} \lor <_{p_2p_2})
\]
\[
\land \neg <_{p_1p_1} \land \neg <_{p_2p_2},
\]

which is equivalent to

\[
(p_2 \lor \neg p_3 \rightarrow p_1) \land (p_1 \rightarrow p_2) \land (p_1 \rightarrow p_2 \land \neg <_{p_1p_2} \lor \neg p_3)
\]
\[
\land (p_2 \rightarrow p_1 \land <_{p_1p_2}) \land \neg <_{p_1p_1} \land \neg <_{p_2p_2}.
\]
Then, $OC_2(\Pi_1)$ has three models, namely $\{p_3, <_{p_1p_2}\}$, $\{p_3\}$, and $\{p_1, p_2, <_{p_1p_2}\}$. Hence, $\Pi_1$ has two answer sets: $\{p_3\}$ and $\{p_1, p_2\}$.

4.3. Simplifying transitivity axioms

Now we consider to simplify the transitivity axioms. In the definition of the ordered completion and its variations presented previously, we need to introduce the transitivity axioms for every three intensional predicates $P$, $Q$ and $R$ in a maximal predicate loop such that $\text{Rank}(P) \leq \text{Rank}(Q) \leq \text{Rank}(R)$. In other words, we expand the maximal predicate loop to a complete graph and introduce the transitivity axioms for every three vertices. In fact, this can be reduced by expanding the maximal predicate loop conservatively.

Let $\Pi$ be a program and $L = \{P_1, \ldots, P_n\}$ be a maximal predicate loop in $PG_\Pi$. The following procedure generates a set of triples among (the undirected version of) $L$:

1. pick up the vertex $P$ in the undirected version of $L$ which has the least number of edges;
2. for every two predicates $Q$ and $R$ connected to $P$ in the undirected version of the graph, add an edge between $Q$ and $R$, and then select the triples related to the three predicates $P$, $Q$ and $R$;
3. delete the vertex $P$ in $L$;
4. go to step 1 and repeat this procedure till all triples generated.

Example 10. Consider the following program $\Pi_2$

$$
P_1(x_1) \leftarrow P_2(x_2),
P_2(x_2) \leftarrow P_3(x_3),
P_3(x_3) \leftarrow P_4(x_4),
P_4(x_4) \leftarrow P_1(x_1),
P_1(x_1) \leftarrow \text{not } P_5(x_5).
$$

The predicate dependency graph of $\Pi_2$ contains four nodes, i.e. $P_1, P_2, P_3, P_4$. In $OC_2(\Pi)$, for the transitivity axioms, we need to consider every combination of three predicates, which counts $4^3$ groups.

According to the procedure above, we can reduce the number to $2 \times 3^3$ groups. First, we pick up a vertex, which had the least number of edges, say $P_1$. Then, we need to connect $P_2$ and $P_4$ as they are connected to $P_1$ in the
predicate dependency graph. Now, one triple is selected, namely \( P_1, P_2, P_4 \). Then, we can delete the node \( P_1 \) from the dependency graph. The rest is a triple now, namely \( P_2, P_3, P_4 \). Hence, we need \( 2 \times 3^3 \) groups as there are two triples generated.

It can be expected that the bigger the predicate dependency graph is, the more transitivity axioms can be reduced.

Then, the ordered completion by simplifying transitivity axioms, denoted by \( OC_3(\Pi) \), is defined based upon \( OC_2(\Pi) \) by replacing the transitivity axioms only for the triples of intensional predicates generated according to the above procedure.

**Proposition 10.** Let \( \Pi \) be a normal logic program whose signature is \( \sigma \), and \( A \) a finite \( \sigma \)-structure. Then, \( A \) is an answer set of \( \Pi \) if and only if there exists a model \( M \) of \( OC_3(\Pi) \) such that \( A \) is the reduct of \( M \) on \( \sigma \).

**Proof:** The “only if” can be proved similarly. For the “if” part, we prove it by contradiction. Otherwise, following again the same proof techniques in Proposition 9, we can get a sequence of ground atoms \( P_k(\overline{a}_k), P_{k+1}(\overline{a}_{k+1}), \ldots, P_l(\overline{a}_l) \) such that

- for all \( k \leq i \leq l \), \( P_i(\overline{a}_i) \in Int_{M|\sigma} \);
- for all \( k \leq i \leq l \), \( P_i(\overline{a}_i) \) are in a ground loop of \( \Pi_{M|\sigma} \);
- for all \( k \leq i \leq l-1 \), \( \overline{a}_{i+1} <_{P_{i+1}P_i} \overline{a}_i \) holds if \( \text{Rank}(P_{i+1}) \leq \text{Rank}(P_i) \), and \( \neg \overline{a}_i <_{P_{i+1}P_i} \overline{a}_{i+1} \) holds if \( \text{Rank}(P_{i+1}) > \text{Rank}(P_i) \);
- \( P_k(\overline{a}_k) = P_l(\overline{a}_l) \).

Assume that \( L \) is a sequence of ground atoms that satisfies the above conditions and has the least number of ground atoms. Select the ground atom \( P_i(\overline{a}_i) \) \( k \leq i \leq l \) in \( L \) such that \( P_i \) is the first predicate calculated according to the procedure, among all predicates occurred in the ground loop \( L \). Now we consider \( P_{i-1}(\overline{a}_{i-1}) \) and \( P_{i+1}(\overline{a}_{i+1}) \). According to the procedure, the triple \( \langle P_{i-1}, P_i, P_{i+1} \rangle \) must be selected. Then, there are several cases about the order of ranks among the three predicates \( P_{i-1}, P_i \) and \( P_{i+1} \). It can be checked that, no matter the order of rank is, \( \overline{a}_{i+1} <_{P_{i+1}P_{i-1}} \overline{a}_{i-1} \) holds if \( \text{Rank}(P_{i+1}) \leq \text{Rank}(P_{i-1}) \), and \( \neg \overline{a}_{i-1} <_{P_{i-1}P_{i+1}} \overline{a}_{i+1} \) holds if \( \text{Rank}(P_{i+1}) > \text{Rank}(P_{i-1}) \) since \( \langle P_{i-1}, P_i, P_{i+1} \rangle \) satisfy the new transitivity axioms. Hence, \( L \setminus \{P_i(\overline{a}_i)\} \)
also satisfies the above conditions. This contradicts to our assumption that \( L \) has the least number of ground atoms satisfying the conditions. 

5. Implementation and Experimental Results

To the best of our knowledge, ordered completion provides for the first time a translation from first-order normal logic programs under the stable model semantics to classical first-order 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.

5.1. Implementation

In this section, we report on a first implementation of such a solver. In order to be consistent with existing ASP solvers, we consider Hebrand structures but not arbitrary structures at this stage. As stated in Section 2.3, under this context, our proposed semantics coincides with Gelfond and Lifschitz’s original semantics [13]. Following the same as the existing approaches of answer set solving, the input is divided into two parts:

1. a first-order normal logic program, and
2. an interpretation of all extensional predicates of the program, which is usually called the set of ground facts\(^3\) in existing approaches. We also call this an extensional database.

Our goal is to compute one answer set of the program based on the ground facts, i.e. an answer set in which the interpretations of the extensional predicates coincide with the extensional database. It is worth mentioning that while the program remains the same, the extensional database could be vary.

For instance, for the transitivity closure program TC (see Example 1), the extensional database can be any graph, whose edges are represented by \( E \) (e.g. the graph in Figure 1). Our goal is to computer one answer set of the program TC based on the given graph. What we need to do is to compute the interpretation of the intensional predicates, namely \( S \) in this example, which represents the transitive closure of the graph. Again, while the TC

\(^3\)Notice that the set of ground facts contains no information about the intensional predicates.
program is always the same, the extensional database can be any graph, and
the purpose of TC is to compute the transitive closure of it.

It would be useful to review briefly how an answer set is computed in
existing ASP solvers. Usually, this contains two steps (see Figure 2).

1. A grounder, such as lparse\(^4\) and gringo\(^5\), transforms a first-order logic
program together with a set of ground facts into a propositional pro-
gram.

2. A propositional ASP solver, such as clasp\(^6\), cmodels\(^7\) and lp2diff\(^8\), com-
putes the answer sets of the propositional program, which correspond
to the answer sets of the original first-order program based on those
ground facts.

![Figure 2: Traditional ASP solvers](image)

With the work of ordered completion, one can do so in a different way
(see Figure 3).

1’ A translator translates a first-order logic program to its ordered com-
pletion (see Definition 3).

2’ A grounder transforms this ordered completion together with a set of
ground facts to a propositional theory.

3’ A SAT solver is called to compute the models of the propositional
theory, which correspond to the answer sets of the original first-order
program based on those ground facts (by Theorem 2).

\(^4\)http://www.tcs.hut.fi/Software/smodels/
\(^5\)http://sourceforge.net/projects/potassco/files/gringo/
\(^6\)http://www.cs.uni-potsdam.de/clasp/
\(^7\)http://www.cs.utexas.edu/~tag/cmodels/
\(^8\)http://www.tcs.hut.fi/Software/lp2diff/
Compared our new approach to the traditional one, it has the following potential benefits.

- Grounding on ordered completion (i.e. a first-order sentence) is based on the semantics of classical logic, which is in general simpler than the stable model semantics. Therefore, many simplification techniques in classical first-order logic can be used, and some of which might not be applicable based on the stable model semantics. Thus, ideally, grounding on ordered completion of a program can utilize more optimization and simplification techniques. It can be expected that this is particularly good for large extensional databases.

- The translator part can utilize some techniques (maybe inspired from similar techniques in the area of theorem proving) to simplify the first-order sentence as much as possible. More importantly, this part can be considered as “off line” since ground facts are not involved yet. In other words, the time used by translator can be exempted from the whole ASP solving as it only needs to be done once for each program, and the result of translator can be used repeatedly when new extensional databases come.

- The SAT solver part is used as a black box. Any SAT solver can be used here. Hence, this part can be benefited from the area of SAT solving.

Following the above ideas, we have implemented a new ASP solver based on the notion of ordered completion. Firstly, a translator is used to translate
a given first-order program to its ordered completion. During this procedure, some optimization techniques presented in Section 4 are used. Then, we have implemented a grounder, called GRounding on Ordered Completion (GROC), to transform the ordered completion together with a set of ground facts (the extensional database) to a propositional theory.

Supposedly, the last step is to call an SAT solver to compute the models of the propositional theory obtained above. According to Theorem 2, these models should correspond to the answer sets of the original program together with the ground facts. However, this might suffer from the transitivity formulas, for which, even with the optimization techniques used in Section 4, we need to introduce $O(n^2)$ new predicates and $O(m^3)$ new proportional clauses in worst case, where $n$ is the number of intensional predicates and $m$ is the number of intensional ground atoms. As we have tested, this turns out to be an unbearable burden for ordered completion based ASP solvers.

To address this issue, we propose an alternative solution. Inspired by Niemelä’s translation [24], we use Satisfiability Modulo Theories (SMT) [25] instead of classical first-order logic as the host language. That is, we translate every normal logic program under the stable model semantics to a sentence (again, its ordered completion) in SMT rather than in classical first-order logic.

Roughly speaking, Satisfiability Modulo Theories [25] are first-order theories together with some background theories, such as the theory of real numbers and theory of data structures. For our purpose, we need the theory of partial orders to eliminate the transitivity axioms. The ordered completion in SMT (with the theory of partial orders as the background theory) is basically the same as the version in classical first-order logic except that we do not need the transitivity axioms (i.e. $\text{Trans}(\Pi)$). This is because the comparison predicates $\leq_{PQ}$ can be regarded as built-in predicates in the theory of partial orders, which naturally satisfy transitivity. In this sense, the ordered completion of $\Pi$ in SMT (with the theory of partial orders as the background theory) is simply the modified Clark’s completion of $\Pi$, namely $\text{MComp}(\Pi)$.

However, the theory of partial orders is not well-supported in many modern SMT solvers. To address this issue, we show that, for capturing ordered completion, we can use linear arithmetic as the background theory in SMT as well. The main reason to do so is that linear arithmetic is well supported
by some modern SMT solvers, e.g. Z3\(^9\), so that they can be used as the black box in step 3′ of our implementation (see Figure 3).

Formally, the ordered completion in SMT (with linear arithmetic as the background theory) is defined as follows.

**Definition 4 (Ordered Completion in SMT).** Let \( \Pi \) be a normal logic program. The SMT-ordered completion of \( \Pi \), denoted by \( OC'(\Pi) \), is the following set of sentences for each intensional predicate \( P \)

\[
\forall \bar{x} \left( \bigvee_{1 \leq i \leq k} \exists \bar{y}_i \left( \text{Body}_i \rightarrow P(\bar{x}) \right) \right),
\]

\[
\forall \bar{x} \left( P(\bar{x}) \rightarrow \bigvee_{1 \leq i \leq k} \exists \bar{y}_i \left( \text{Body}_i \land \bigwedge_{Q(\bar{z}) \in \text{Pos}_i, Q \in \Omega(\Pi)} n_Q(\bar{z}) < n_P(\bar{x}) \right) \right),
\]

where the notations used are borrowed from Definition 3. In addition,

- for each intensional predicate \( P \) of arity \( n \), \( n_P \) is a function from domain tuples to integers, i.e. \( n_P : D^n \rightarrow \mathbb{N} \);
- \( < \) is a built-in predicate in linear arithmetic, meaning “less than”.

We show that the stable models of a program can be equivalently captured by its SMT-ordered completion as well.

**Theorem 3.** Let \( \Pi \) be a normal logic program whose signature is \( \sigma \), and \( A \) a finite \( \sigma \)-structure. Then the following statements are equivalent:

1. \( A \) is an answer set of \( \Pi \);
2. there exists a model \( \mathcal{M} \) of \( OC(\Pi) \) such that \( A \) is the reduct of \( \mathcal{M} \) on \( \sigma \);
3. there exists a model \( \mathcal{M}' \) of \( OC'(\Pi) \) such that \( A \) is the reduct of \( \mathcal{M}' \) on \( \sigma \).

**Proof:** Theorem 2 proves 1 \( \Leftrightarrow \) 2. Now we show 2 \( \Rightarrow \) 3. Suppose that we have a structure \( \mathcal{M} \) of the signature \( \sigma \cup \{ \leq_{PQ} \mid P, Q \in \Omega(\Pi) \} \), which is a model of \( OC(\Pi) \), i.e. \( M\text{Comp}(\Pi) \cup \text{TranS}(\Pi) \). Since \( \mathcal{M} \) satisfies \( \text{TranS}(\Pi) \), we can define an order \( \preceq \) on the set \( \text{Int}_\mathcal{M} \) of intensional ground atoms, i.e. \( \{ P(\bar{u}) \mid P \in \Omega(\Pi), \bar{u} \in P^\mathcal{M} \} \), of \( \Pi \) as follows:

\[^9\text{http://research.microsoft.com/en-us/um/redmond/projects/z3/}\]
• for every $P(\overline{u}) \in \text{Int}_\mathcal{M}$, $P(\overline{u}) \preceq P(\overline{v})$;

• for every pair $P(\overline{u}), Q(\overline{v}) \in \text{Int}_\mathcal{M}$, $P(\overline{u}) \preceq Q(\overline{v})$ iff $(\overline{u}, \overline{v}) \in \leq_{PQ}^{\mathcal{M}}$;

• for every pair $P(\overline{u}), Q(\overline{v}) \in \text{Int}_\mathcal{M}$, we write $P(\overline{u}) = Q(\overline{v})$ iff $P(\overline{u}) \preceq Q(\overline{v})$ and $Q(\overline{v}) \preceq P(\overline{u})$.

Clearly, $\preceq$ is a partial order on $\text{Int}_\mathcal{M}$ as the comparison predicates $\leq_{PQ}$ satisfy the transitivity axioms, i.e. $\text{Trans}(\Pi)$. Therefore, $\preceq$ can be extended to a total order (also called linear order) $\preceq'$ on $\text{Int}_\mathcal{M}$. We can construct a mapping $f$ from $\text{Int}_\mathcal{M}$ to natural numbers, i.e. $f : \text{Int}_\mathcal{M} \rightarrow \mathbb{N}$, such that for every $P(\overline{u}) \in \text{Int}_\mathcal{M}$, $f(P(\overline{u}))$ is the position of $P(\overline{u})$ in this total order $\preceq'$. That is, $f(P(\overline{u})) = t$ iff there exists $t$ elements $E_1, \ldots, E_t \in \text{Int}_\mathcal{M}$ such that $E_i \neq E_j$ for all $1 \leq i < j \leq t$ and $E_i \preceq' P(\overline{u})$ for all $1 \leq i \leq t$. Now we construct $\mathcal{M}'$ based on $\mathcal{M}$ and $f$ as follows:

• $\mathcal{M}'$ has the same domain $M$ and constant interpretations as $\mathcal{M}$;

• for all $P \in \sigma$, $P^{\mathcal{M}'} = P^{\mathcal{M}}$;

• for all $Q \in \Omega(\Pi)$ and $\overline{u} \in M^n$, $n_Q(\overline{u}) = f(Q(\overline{u}))$.

It can be shown that $\mathcal{M}'$ is a model of $OC'(\Pi)$ since $\mathcal{M}$ is a model of $M\text{Comp}(\Pi)$ and for any two intensional ground atoms $P(\overline{u}), Q(\overline{v}) \in \text{Int}_\mathcal{M}$, $\overline{u} \preceq_{PQ} \overline{v}$ (i.e. $(\overline{u}, \overline{v}) \in \leq_{PQ}^{\mathcal{M}}$) iff $n_P(\overline{u}) < n_Q(\overline{v})$ or $n_P(\overline{u}) = n_Q(\overline{v})$.

For $3 \Rightarrow 2$, suppose that we have a structure $\mathcal{M}'$ of $\sigma \cup \{n_P \mid P \in \Omega(\Pi)\}$, which is a model of $OC'(\Pi)$. We can construct a structure $\mathcal{M}$ of $\sigma \cup \{\leq_{PQ} \mid P, Q \in \Omega(\Pi)\}$ such that $\mathcal{M}$ agrees everything the same on the signature $\sigma$ and $(\overline{u}, \overline{v}) \in \leq_{PQ}^{\mathcal{M}}$ iff $n_P(\overline{u}) < n_Q(\overline{v})$ or $n_P(\overline{u}) = n_Q(\overline{v})$. Following the similar arguments, it can be shown that $\mathcal{M}$ is a model of $M\text{Comp}(\Pi)$. In addition, $\mathcal{M}$ is a model of $\text{Trans}(\Pi)$ as well because the functions $n_P$ naturally yield a total order, thus a partial order, on $\text{Int}_\mathcal{M}$ (the same as $\text{Int}_\mathcal{M}'$). Hence, $\mathcal{M}$ is a model of $OC'(\Pi)$.

Based on Theorem 3, we have implemented a prototype of a new ASP solver, which contains three steps. First, we translate a program $\Pi$ to its SMT-ordered completion, i.e. $OC'(\Pi)$. Notice that some of the optimization techniques mentioned in Section 4 can still be used here. Second, we use our grounder $GROC$ to ground $OC'(\Pi)$ into a propositional SMT theory. Finally,
we call the SMT solver Z3 to compute a model of the SMT theory, which, by Theorem 3, should be corresponding to an answer set of II. Next, we report on some preliminary experimental results of our solver on the Hamiltonian Circuit program (see Example 8), compared to other major modern ASP solvers.

5.2. Experimental results

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 (see Figure 3) with a number of existing ASP solvers based on the traditional approach (see Figure 2).

Again, the input is divided into two parts: a first-order 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.

As mentioned earlier, in order to solve this problem, existing ASP solvers normally use a 2-step approach. First, a grounder is used to transform the first-order program together with the extensional database to a propositional program. Here, we consider to use gringo (version 3.03) as the grounder. Second, a propositional ASP solver is called to compute an answer set. In this paper, we consider three different propositional ASP solvers, including clasp (version 2.0.1), cmodels (version 3.81) and lp2diff (version 1.27) with Z3 (version 3.2.18).

Differently, our solver needs 3 steps. First, we translate a first-order program to its SMT-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, to transform the ordered completion together with the extensional database to a proportional 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 3. We use Z3 (version 3.2.18) as the SMT solver in this step.

We consider the Hamiltonian Circuit benchmark program by Niemelä [23] (also see Example 8). The current benchmark graph instances for the HC program normally contain no more than 150 nodes. Here, instead, we 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
rand_nodes_edges_number, where rand means this is a random graph, nodes
represents the number of nodes in this graph, edges represents the number
of edges in this graph, and number is the code of this graph in this category.
For instance, rand_node200_edge2000_node6 is a random graph with 200 nodes and 2000
edges, and is the 6-th graph instance in this category.

Table 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 extensional databases, and 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 1, the first column specifies the graph instances. In the second
column, “y” (“n”) means that the corresponding graph has a (no) Hamil-
tonian Circuit, while “?” means that this problem instance is not solved by
any approaches within limited time. The rest four columns record the overall
time in seconds used by the four different approaches. It is worth mention-
ing that, normally, the grounding time (i.e. for gringo and groc) is much less
than the solving time.

Table 1 shows that our solver compares favorably to the others on the
Hamiltonian Circuit benchmark program, especially for those big graph in-
stances. 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 goes bigger, our advantages emerge. For those 400
node and 600 node graphs, our solver clearly outperforms the rest approaches.
Moreover, for 1000 node random graphs, our solver is the only one capable of
<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>
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 immediately identified.

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 first-order formula ordered completion. As mentioned earlier, this step can be regarded as off line 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 to the future.

6. Related Work and Discussions

In this section, we discuss some related work, mostly other existing or possible translations from logic 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.

6.1. First-order case

Other first-order translations As the focus of this paper is to consider the first-order case, we first review the existing work about translating first-order logic programs under the stable model semantics to standard first-order logic. To the best of our knowledge, the only such translation is the loop formula approach [3, 17]. From a syntactical viewpoint, the main difference between this approach and ours is that the ordered completion results in a finite first-order theory (which can be represented as a single first-order sentence) but uses auxiliary predicates, while the loop formula approach does not use any auxiliary predicates but in general results in an infinite first-order theory.

From a semantical viewpoint, both approaches share some similar ideas. First of all, both of them are extended from Clark’s completion, and the extended parts play a similar role to eliminate those structures which are models of the Clark’s completion but not stable models of the logic 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 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 translations in Datalog** Another related work [16] is in the area of finite model theory and fixed-point logic. Although fixed-point logic and normal logic programming are not comparable, they have a common fragment, namely Datalog. Kolaitis [16] 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 first-order 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 logic 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 first-order sentences using some new predicates, then it must also be possible for normal logic 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.

### 6.2. Propositional 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 [2, 15, 19, 20, 24].
An early attempt is due to Ben-Eliyahu and Dechter [2], 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 [24], 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 $x_q - 1 \geq x_p$ 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 [15], who proposed a simplified translation by level numbering as well. Different from the above approaches, Lin and Zhao [19] translated an arbitrary normal logic program equivalently to a tight program first by adding some new atoms, and then 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 [20] yields another translation from propositional normal logic 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 [2], which turns out to be rather complicated. This is not an issue for Niemelä’s work [24] because the level numbering \(x_p\) associated with 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 \(\leq pq \wedge \neg \leq pq\), which are further specified by the transitivity formulas.

The similarities and differences among the three approaches can be illustrated by the following example.

**Example 11.** Recall the program \(\Pi_1\) in Example 4:

\[
\begin{align*}
p_1 & \leftarrow p_2, \\
p_2 & \leftarrow p_1, \\
p_1 & \leftarrow \text{not} p_3. 
\end{align*}
\]

According to the definitions, the modified completion part of \(\Pi_1\) 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

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

and finally, for Niemelä’s work, this is \(^{10}\)

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

\(^{10}\)It can be observed that the new atoms \(bd_a^i\) in Niemelä’s work are not necessary.
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 $x_p - 1 \geq x_q$ (e.g. $x_{p_1} - 1 \geq x_{p_2}$) with its corresponding counterpart $\leq_{q_p} \land \neg \leq_{p_q}$ 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 the built-in predicate $\geq$. However, for the other two approaches, more work need to be done.

In the ordered completion, we use additionally transitivity formulas among new atoms $\leq_{pq}$ for this purpose. For instance, for the program $\Pi_1$, the transitivity formulas is: \footnote{All the other transitive formulas are trivially true.}

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

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 $\Pi_1$, the encoding of each index, e.g. $\#p_1$, is:

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

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

$$p_1 = 1 \land p_2 = 2.$$
A program is *(first-order) definable* (on finite structures) iff 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 1, both on arbitrary structures and on finite structures [6].

A weaker notion of first-order definability is to allow new predicates. A program is *(first-order) weakly definable* (on finite structures) iff there exists a first-order 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 see that a program is weakly definable (on finite structures) if and only if it is defined by an existential second-order sentence (on finite structures).

The following result immediately follows from Theorem 2.

**Corollary 11.** Every normal logic 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 5, 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 5 can be extended to the following result.

**Proposition 12.** On arbitrary structures, if a normal logic program is not definable, then it is not weakly definable.

7. Conclusion

The main contribution of this paper is introducing a notion of ordered completion that captures exactly the answer set semantics of first-order normal logic programs with constraints and choice rules on finite structures (see Theorem 2, Propositions 3 and 4). It can be summarized as follows:

For first-order normal logic programs on finite structures,

\[
\text{Answer Set} = \text{Clark’s Completion} + \text{Derivation Order} = \text{Ordered Completion}
\]
This seems to be a very tight result. First of all, as we have seen, this result cannot be extended to disjunctive logic programs unless \( NP = coNP \) (see Proposition 6). For normal logic programs, with this result, we now have a rather complete picture of mappings from logic programs to first-order logic which is summarized by Table 2.

<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>

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 logic programs under the stable model semantics to first-order sentences. Also, it initiates a new direction of ASP solvers by grounding on a program’s ordered completion instead of the program itself. We report our first implementation of such a solver (see Section 5), which compares favorably to other major existing ASP solvers, especially on big problem instances (See Table 1).

References


