Artificial Intelligence 177–179 (2012) 1–24

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Ordered completion for first-order logic programs on finite structures

Vernon Asuncion [a], Fangzhen Lin [b], Yan Zhang [a], Yi Zhou [a,*]

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

ARTICLE INFO

Article history:
Received 20 October 2010
Received in revised form 16 November 2011
Accepted 17 November 2011
Available online 25 November 2011

Keywords:
Answer set programming
Ordered completion
Knowledge representation
Nonmonotonic reasoning

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. The translation is done through, what we call, ordered completion which is a modification of Clark's completion 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 clarifies 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 if new predicates are allowed. This is a tight result as not every normal logic program can be defined by a first-order sentence if no extra predicates are allowed or when infinite structures are considered. Furthermore, we show that the result cannot be extended to disjunctive logic programs, assuming that 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. Our experimental results show that our solver compares favorably to other major ASP solvers on the Hamiltonian Circuit program, especially on large domains.

1. Introduction

This work is about translating logic programs under the stable model (answer set) semantics [12] 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 [4] 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" [8]. 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 [19]. 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 for 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 [1] is polynomial in space but uses $O(n^2)$ auxiliary variables, while Lin and Zhao's translation [19] using loop formulas is exponential in the

* Corresponding author.
E-mail addresses: vernon@scm.uws.edu.au (V. Asuncion), flin@cse.ust.hk (F. Lin), yan@scm.uws.edu.au (Y. Zhang), yzhou@scm.uws.edu.au (Y. Zhou).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2011.11.001

worst case but does not use any auxiliary variables. Chen et al. [2] 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 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 [5].

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 logic programs that cannot be captured by any first-order theory, and on finite structures, there exists some disjunctive logic program that cannot be captured by any first-order sentence provided that NP $\neq$ coNP. We then present some techniques for optimizing ordered completions in Section 4. These techniques are used in our implementation of a first-order solver. We describe this solver and some of the experimental results in Section 5. We discuss some related work in Section 6, and conclude the paper in Section 7.

2. Preliminaries

We assume that 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 σ and σ_1 be two signatures such that $\sigma \subseteq \sigma_1$. Given a structure $\mathcal{A}$ of signature σ_1, we say that the *reduct* of $\mathcal{A}$ on σ, denoted by $\mathcal{A} \upharpoonright \sigma$, is the σ-structure that agrees with $\mathcal{A}$ on all interpretations of predicates and constants in σ. Conversely, we say that $\mathcal{A}$ is an *expansion* of $\mathcal{A} \upharpoonright \sigma$ to σ_1.

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

$$\alpha \leftarrow \beta_1, \ldots, \beta_k, \mathsf{not}\, \gamma_1, \ldots, \mathsf{not}\, \gamma_l, \tag{1}$$

where α is a proper atom, $0 \leqslant k \leqslant l$, and β_i $(1 \leqslant i \leqslant k)$, γ_j $(1 \leqslant j \leqslant l)$ are atoms. Given a rule r of form (1), we call α the *head* of r, denoted by $Head(r)$, and $\{\beta_1, \ldots, \beta_k, \mathsf{not}\, \gamma_1, \ldots, \mathsf{not}\, \gamma_l\}$ 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 Π, a predicate is called *intensional* if it occurs in the head of some rule in Π, and *extensional* otherwise. The *signature* of Π contains all intensional predicates, extensional predicates and constants occurring in Π.

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

$$P(\vec{x}) \leftarrow Body_1, \ldots, P(\vec{x}) \leftarrow Body_k.$$

2.1. 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 Π, and a predicate P in it, *Clark's Completion* of P in Π is the following first-order sentence [4]:

$$\forall \vec{x}\left(P(\vec{x}) \leftrightarrow \bigvee_{1 \leqslant i \leqslant k} \exists \vec{y}_i \widehat{Body_i} \right), \tag{2}$$

where

- $P(\vec{x}) \leftarrow Body_1, \ldots, P(\vec{x}) \leftarrow Body_k$ are all the rules whose heads mention the predicate P;
- $\vec{y}_i$ is the tuple of body variables in $P(\vec{x}) \leftarrow Body_i$;
- $\widehat{Body_i}$ is the conjunction of elements in $Body_i$ by simultaneously replacing the occurrences of not by $\neg$.

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

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:

$$\forall xy\big(S(x, y) \leftrightarrow \big(E(x, y) \vee \exists z E(x, z) \wedge S(z, y)\big)\big).$$

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 [12], and later extended to become answer set semantics for propositional programs that can have classical negation, constraints, disjunctions, and other operators [6,11,13,17,21,22]. Several equivalent characterizations are proposed, including the Gelfond–Lifschitz transformation [12], the logic of GK [20], loop formulas [19], equilibrium logic [26], general reduction [11], and so on. Here, we briefly review the standard Gelfond–Lifschitz transformation semantics [12] and the loop formula characterization in the propositional case [19], as they are needed in the proof of our main theorem.

Let Π be a propositional program and $\mathcal{A}$ a set of atoms. We say that $\mathcal{A}$ *satisfies* a rule r in Π if $Head(r) \in \mathcal{A}$ whenever $Pos(r) \subseteq \mathcal{A}$ and $Neg(r) \cap \mathcal{A} = \emptyset$. Then, $\mathcal{A}$ *satisfies* Π if it satisfies all rules in Π. The *reduct* of Π relative to $\mathcal{A}$, denoted by $\Pi^{\mathcal{A}}$, is the program obtained from Π by (i) deleting every rule r in Π 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 Π iff $\mathcal{A}$ satisfies Π 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 [19]. Let Π be a program. The (*positive*) *dependency graph* of Π, denoted by G_Π, is the finite graph (V, E), where V is the set of atoms occurring in Π, 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_Π 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 Π. Let L be a loop, and x an atom in L. We say that a support *Body* of x in Π is an *external support* with respect to L if the positive part of *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 Π. Then, the *loop formula* of L in Π, denoted by $LF(L, \Pi)$, is the following formula

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

Lin and Zhao [19] 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 [12], 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 [2,3,10,20,25,27]. 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 [12] when only considering Herbrand structures [3,10,16,20,27].

We briefly review the translational semantics [10] 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 Π, let $\Omega_\Pi = \{Q_1, \ldots, Q_n\}$ be the set of all intentional predicates of Π. Let $\Omega_\Pi^* = \{Q_1^*, \ldots, Q_n^*\}$ be a new set of predicates corresponding to Ω_Π, where each Q_i^* in Ω_Π^* has the same arity of predicate Q_i in Ω_Π. Given a rule r in Π of the form

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

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

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

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

$$\beta_1^* \wedge \cdots \wedge \beta_m^* \wedge \neg\gamma_1 \wedge \cdots \wedge \neg\gamma_l \rightarrow \alpha^*,$$

where $\alpha^* = Q^*(\vec{x})$ if $\alpha = Q(\vec{x})$ and

$$\beta_i^*, (1 \leqslant i \leqslant m) = \begin{cases} Q_j^*(\vec{t}_j) & \text{if } \beta_i = Q_j(\vec{t}_j) \text{ and } Q_j \in \Omega_\Pi, \\ \beta_i & \text{otherwise.} \end{cases}$$

By $\widehat{\Pi}$, we denote the first-order sentence $\bigwedge_{r\in\Pi} \hat{r}$; by Π^*, we denote the first-order sentence $\bigwedge_{r\in\Pi} r^*$. Let Π be a normal logic program. By $SM(\Pi)$, we denote the following second-order sentence:

$$\widehat{\Pi} \wedge \neg\exists\Omega_\Pi^*\big(\big(\Omega_\Pi^* < \Omega_\Pi\big) \wedge \Pi^*\big),$$

where $\Omega_\Pi^* < \Omega_\Pi$ is the abbreviation of the formula

$$\bigwedge_{1\leqslant i\leqslant n} \forall\vec{x}\big(Q_i^*(\vec{x}) \rightarrow Q_i(\vec{x})\big) \wedge \neg \bigwedge_{1\leqslant i\leqslant n} \forall\vec{x}\big(Q_i(\vec{x}) \rightarrow Q_i^*(\vec{x})\big).$$

This second-order sentence is used to capture the stable models of Π. 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.

Definition 1. The grounding of a program Π on a structure $\mathcal{M}$, written $\Pi_\mathcal{M}$ below, is the union of the following three sets:

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

We now have the following definition:

Definition 2. Let Π be a normal logic program and $\mathcal{M}$ a structure. We say that $\mathcal{M}$ is a *stable model* (or an *answer set*) of Π 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

$$\big\{P(\vec{u}) \;\big|\; P \text{ is an intensional predicate, and } \vec{u} \in P^\mathcal{M}\big\}.$$

Example 2. Consider the following program Π_0

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

According to Gelfond and Lifschitz's original stable model semantics [12], the unique stable model/answer set of Π_0 is $X = \{P(a_1), Q(a_2)\}$.

Now, let us reconsider the program Π_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

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

is a stable model of Π_0. In fact, this structure $\mathcal{M}_0$ corresponds to the unique stable model X of Π_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 Π_0. The following structure $\mathcal{M}_1$, where

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

[1] Note that here $u = u$ is a propositional atom but not an equality atom.

is also a stable model of Π_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

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

is a stable model of Π_0 as well. Notice that $d_3 \in P^{\mathcal{M}_2}$ 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 [12] 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 Π be a normal logic program whose signature is σ, and $\mathcal{A}$ a finite σ-structure. Then, $\mathcal{A}$ is a stable model of Π (under Definition 2) iff $\mathcal{A}$ is a model of $SM(\Pi)$.*

Proof. For convenience, we use $Gr(\mathcal{A})$ to denote $EQ_{\mathcal{A}} \cup Ext_{\mathcal{A}} \cup Int_{\mathcal{A}}$. First, $Gr(\mathcal{A})$ satisfies $\Pi_{\mathcal{A}}$ iff $Gr(\mathcal{A})$ satisfies all instances of the rules in Π under $\mathcal{A}$ iff $Gr(\mathcal{A})$ satisfies every $r\eta$, where $r \in \Pi$ and η is an assignment iff $\mathcal{A}$ satisfies $\widehat{r}$ for all $r \in \Pi$ iff $\mathcal{A} \models \widehat{\Pi}$.

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

- for all intensional predicates Q of Π, $Q^{\mathcal{A}'} \subseteq Q^{\mathcal{A}}$, and for some intensional predicates Q, $Q^{\mathcal{A}'} \subset Q^{\mathcal{A}}$;
- for every assignment η and every rule r of form (1) in Π, if for all i $(1 \leqslant i \leqslant k)$, $\beta_i\eta \in \mathcal{A}'$ and for all j $(1 \leqslant j \leqslant l)$, $\gamma_j\eta \notin \mathcal{A}$, then $\alpha\eta \in \mathcal{A}'$

iff there exists $X \subset Gr(A)$ satisfying $(r\eta)^{Gr(\mathcal{A})}$ for every pair r, η, where $r \in \Pi$ and η is an assignment (let $X = EQ_{\mathcal{A}} \cup Ext_{\mathcal{A}} \cup Int_{\mathcal{A}'}$) iff there exists $X \subset Gr(A)$ satisfying ${\Pi_{\mathcal{A}}}^{Gr(\mathcal{A})}$.

This shows that the original assertion holds. □

As a consequence of the above theorem and Theorem 1 in [10], 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 [12]. 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

$$(p \rightarrow q) \wedge (q \rightarrow p),$$
$$p \rightarrow (q \wedge T_{qp} \wedge \neg T_{pq}),$$
$$q \rightarrow (p \wedge T_{pq} \wedge \neg T_{qp}),$$
$$T_{pq} \wedge T_{qp} \rightarrow T_{pp},$$
$$T_{qp} \wedge T_{pq} \rightarrow T_{qq}.$$

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

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

- For each intensional predicate P, the following sentences:

$$\forall \vec{x}\Big(\bigvee_{1\leqslant i\leqslant k} \exists \vec{y}_i \widehat{Body_i} \rightarrow P(\vec{x})\Big), \tag{3}$$

$$\forall \vec{x}\Big(P(\vec{x}) \rightarrow \bigvee_{1\leqslant i\leqslant k} \exists \vec{y}_i \Big(\widehat{Body_i} \wedge \bigwedge_{Q(\vec{z})\in Pos_i, Q\in\Omega_\Pi} \vec{z} \leqslant_{QP} \vec{x} \wedge \neg \vec{x} \leqslant_{PQ} \vec{z}\Big)\Big),$$

  where we have borrowed the notations used in the definition of Clark's completion, and further assume that Pos_i is the positive part of $Body_i$ and $Q(\vec{z})$ ranges over all the intensional atoms in the positive part of $Body_i$;
- For each triple of intensional predicates P, Q, and R (two or all of them might be the same) the following sentence:

$$\bigwedge_{P,Q,R\in\Omega_\Pi} \forall \vec{x}\vec{y}\vec{z}(\vec{x} \leqslant_{PQ} \vec{y} \wedge \vec{y} \leqslant_{QR} \vec{z} \rightarrow \vec{x} \leqslant_{PR} \vec{z}). \tag{4}$$

In the following, we use $MComp(\Pi)$ to denote the set of the formulas (3) and (4), and $TranS(\Pi)$ the set of formulas (4). So $OC(\Pi) = MComp(\Pi) \cup TranS(\Pi)$.

Clearly, for finite programs, $OC(\Pi)$ is finite, and the predicates occurring in $OC(\Pi)$ are all the predicates occurring in Π together with all the comparison predicates $\{\leqslant_{PQ} | P, Q \in \Omega_\Pi\}$.

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

$$\forall \vec{x}\Big(\bigvee_{1\leqslant i\leqslant k} \exists \vec{y}_i \widehat{Body_i} \rightarrow P(\vec{x})\Big),$$

$$\forall \vec{x}\Big(P(\vec{x}) \rightarrow \bigvee_{1\leqslant i\leqslant k} \exists \vec{y}_i \widehat{Body_i}\Big).$$

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 Eq. (4)). In addition, $TranS(\Pi)$ simply means that the comparison predicates satisfy "transitivity".

Proposition 1. *Let Π 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 Π, s the maximal arity of the intensional predicates of Π and n the length of Π.*

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 $\leqslant_{SS}$, whose arity is 4. The ordered completion of TC consists of the following sentences:

$$\forall xy\big(\big(E(x, y) \vee \exists z\big(E(x, z) \wedge S(z, y)\big)\big) \rightarrow S(x, y)\big), \tag{5}$$

$$\forall xy\big(S(x, y) \rightarrow \big(E(x, y) \vee \exists z\big(E(x, z) \wedge S(z, y) \wedge \vec{zy} \leqslant_{SS} \vec{xy} \wedge \neg \vec{xy} \leqslant_{SS} \vec{zy}\big)\big)\big), \tag{6}$$

$$\forall xyuvzw(\vec{xy} \leqslant_{SS} \vec{uv} \wedge \vec{uv} \leqslant_{SS} \vec{zw} \rightarrow \vec{xy} \leqslant_{SS} \vec{zw}). \tag{7}$$

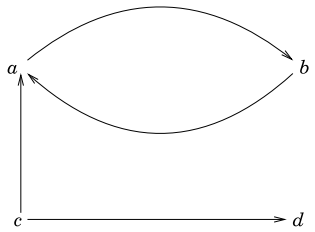

Fig. 1. An example graph.

Intuitively, one can understand $\overrightarrow{xy} \leqslant_{SS} \overrightarrow{uv}$ to mean that $S(x, y)$ is used to establish $S(u, v)$. So the sentence (6) 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 Fig. 1 with four vertices a, b, c, d, with E representing the edge relation: $E(a, b)$, $E(b, a)$, $E(c, a)$, $E(c, d)$. Clearly, if there is a path from x to y, then $S(x, y)$ (by sentence (5)). 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

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

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

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

So z must be b, and

$$S(b, c) \wedge \overrightarrow{bc} \leqslant_{SS} \overrightarrow{ac} \wedge \neg\overrightarrow{ac} \leqslant_{SS} \overrightarrow{bc}. \tag{8}$$

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

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

So y must be a, and

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

However, this contradicts with (8).

Notice that the Clark's completion of TC, i.e. $\forall xy(S(x, y) \leftrightarrow (E(x, y) \vee \exists z E(x, z) \wedge S(z, y)))$ (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 Fig. 1.

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 [7]. 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 $\mathcal{A}$ on a program Π is a sequence of ground atoms $P_1(\vec{a}_1), \ldots, P_k(\vec{a}_k)$ such that $\{P_1(\vec{a}_1), \ldots, P_k(\vec{a}_k)\} = Int_{\mathcal{A}}$ (see Definition 2), and for all i $(1 \leqslant i \leqslant k)$, there exists a rule $r \in \Pi$ and an assignment η such that

- $Head(r)\eta = P_i(\vec{a}_i)$,
- for all intensional atoms $Q(\vec{t})$ in $Pos(r)$, $Q(\vec{t})\eta \in \{P_1(\vec{a}_1), \ldots, P_{i-1}(\overrightarrow{a_{i-1}})\}$,
- for all intensional atoms $Q(\vec{t})$ in $Neg(r)$, $Q(\vec{t})\eta \notin Int_{\mathcal{A}}$,
- for all extensional atoms $Q(\vec{t})$ in $Pos(r)$ ($Neg(r)$ resp.), $Q(\vec{t})\eta \in Ext_{\mathcal{A}}$ ($Q(\vec{t})\eta \notin Ext_{\mathcal{A}}$ resp.).

Example 4. Consider the following program Π_1

$$p_1 \leftarrow p_2,$$
$$p_2 \leftarrow p_1,$$
$$p_1 \leftarrow \text{not } p_3.$$

Clearly, $X = \{p_1, p_2\}$ is an answer set of Π_1. According to the definition, p_1, p_2 is a derivation order of X on Π_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 Π be a normal logic program whose signature is σ, and $\mathcal{A}$ a finite σ-structure. Then, $\mathcal{A}$ is an answer set of Π iff*

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

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

Then, $\{p_1, p_2, \ldots, p_k\} = \mathcal{A}$. Otherwise, $\{p_1, p_2, \ldots, p_k\} \models \Pi^{\mathcal{A}}$ 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 $\mathcal{A} \models \widehat{\Pi}$ 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 $\Pi^{\mathcal{A}}$. Otherwise, let p be the atom in $\mathcal{A} \backslash \mathcal{A}'$ with the least ordinal in the derivation order. Then, according to the definition, there exists a rule r such that $Head(r) = p$, $Pos(r) \subseteq \mathcal{A}'$ (since p has the least ordinal in the derivation order) and $Neg(r) \cap \mathcal{A} = \emptyset$. Therefore, $Neg(r) \cap \mathcal{A}' = \emptyset$. It follows that $\mathcal{A}' \models Pos(r)$ but $\mathcal{A}' \not\models Head(r)$. Hence, $\mathcal{A}'$ does not satisfy $r^{\mathcal{A}}$, a contradiction. □

Now we are able to present the following main theorem.

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

Proof. On one hand, we show that every finite answer set $\mathcal{A}$ of Π can be expanded to a model of $OC(\Pi)$. 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 $\leqslant_{PQ}$ for each pair (P, Q) of intensional predicates in Π:

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

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

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

$$\mathcal{M} \models \bigvee_{P \in \Omega_\Pi} \exists \vec{x} \Bigg(P(\vec{x}) \wedge \bigwedge_{1 \leqslant i \leqslant k} \forall \vec{y}_i \Big(\widehat{Body_i} \rightarrow \bigvee_{Q(\vec{z}) \in Pos_i \cap \Omega_\Pi} \neg \vec{z} \leqslant_{QP} \vec{x} \vee \vec{x} \leqslant_{PQ} \vec{z} \Big) \Bigg).$$

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

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

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

On the other hand, we prove that the reduct of any finite model $\mathcal{M}$ of $OC(\Pi)$ on σ must be an answer set of Π. Clearly, $\mathcal{M} \upharpoonright \sigma$ is a model of $Comp(\Pi)$ since $MComp(\Pi) \models Comp(\Pi)$. Hence, according to the loop formula characterization of answer set semantics in the propositional case [19], it suffices to show that for all loops L of the ground program $\Pi_{\mathcal{M}\upharpoonright\sigma}$, the set of ground atoms $EQ_{\mathcal{M}\upharpoonright\sigma} \cup Ext_{\mathcal{M}\upharpoonright\sigma} \cup Int_{\mathcal{M}\upharpoonright\sigma}$ is a model of its loop formula.

We prove this by contradiction. Suppose that there exists a loop L of the ground program $\Pi_{\mathcal{M}\uparrow\sigma}$ such that the above set of ground atoms is not a model of $LF(L, \Pi_{\mathcal{M}\uparrow\sigma})$. Then, there exists a ground atom $P_0(\vec{a}_0) \in L$ and $P_0(\vec{a}_0) \in Int_{\mathcal{M}\uparrow\sigma}$, and for every $Q(\vec{b}) \in L$, $Q(\vec{b})$ has no external support with respect to L in the ground program $\Pi_{\mathcal{M}\uparrow\sigma}$.

Since $\mathcal{M}$ is a model of $MComp(\Pi)$, there exists a support $Body_0$ of $P_0(\vec{a}_0)$ in the ground program $\Pi_{\mathcal{M}\uparrow\sigma}$ such that for all ground atoms $Q(\vec{b})$ in the positive body of $Body_0$, $\vec{b} \leqslant_{Q\,P_0} \vec{a}_0 \wedge \neg\vec{a}_0 \leqslant_{P_0\,Q} \vec{b}$ holds, where Q is an intensional predicate of Π and $\vec{b}$ a tuple of elements in M that matches the arity of Q. If every $Q(\vec{b})$ is not in L, then $Body_0$ is an external support of $P_0(\vec{a}_0)$ with respect to L, a contradiction. Hence, there exists $P_1(\vec{a}_1)$ in the positive body of $Body_0$ such that $P_1(\vec{a}_1) \in Int_{\mathcal{M}\uparrow\sigma}$, $P_1(\vec{a}_1) \in L$ and $\vec{a}_1 \leqslant_{P_1P_0} \vec{a}_0 \wedge \neg\vec{a}_0 \leqslant_{P_0P_1} \vec{a}_1$ holds.

Again, following the same procedure described above, there exists a support $Body_1$ of $P_1(\vec{a}_1)$, and a ground atom $P_2(\vec{a}_2)$ in the positive body of $Body_1$ such that $P_2(\vec{a}_2) \in Int_{\mathcal{M}\uparrow\sigma}$, $P_2(\vec{a}_2) \in L$ and $\vec{a}_2 \leqslant_{P_2P_1} \vec{a}_1 \wedge \neg\vec{a}_1 \leqslant_{P_1P_2} \vec{a}_2$ holds. Hence, we can get a sequence of ground atoms $P_0(\vec{a}_0), P_1(\vec{a}_1), \ldots, P_i(\vec{a}_i), \ldots$ such that for all i, $P_i(\vec{a}_i) \in Int_{\mathcal{M}\uparrow\sigma}$, $P_i(\vec{a}_i) \in L$. In addition, both $\overrightarrow{a_{i+1}} \leqslant_{P_{i+1}P_i} \vec{a}_i$ and $\neg\vec{a}_i \leqslant_{P_iP_{i+1}} \overrightarrow{a_{i+1}}$ hold.

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

This shows that $\mathcal{M} \uparrow \sigma$ is an answer set of Π. □

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 $\mathcal{A}$ is an answer set of a program Π iff it is a model of Π and satisfies the following conditions:

downgrading every ground atom $P(\vec{a})$ in $\mathcal{A}$ 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 Eq. (4));

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

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

Together with the above three conditions, each ground atom $P(\vec{a})$ in a finite answer set $\mathcal{A}$ 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*:

$$\leftarrow \beta_1, \ldots, \beta_k, \mathsf{not}\,\gamma_1, \ldots, \mathsf{not}\,\gamma_l, \tag{9}$$

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

$$\forall\vec{y}\neg(\beta_1 \wedge \cdots \wedge \beta_k \wedge \neg\gamma_1 \wedge \cdots \wedge \neg\gamma_l),$$

where $\vec{y}$ is the tuple of all variables occurring in (9). In the following, if c is a constraint of form (9), 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 a program with constraints if it 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 Π be a normal logic program whose signature is σ, C a set of constraints, and $\mathcal{A}$ a finite σ-structure. Then, $\mathcal{A}$ is an answer set of $\Pi \cup C$ iff there exists a model $\mathcal{M}$ of $OC(\Pi) \cup \{\hat{c} \mid c \in C\}$, such that $\mathcal{A}$ is the reduct of $\mathcal{M}$ on σ.*

Proof. $\mathcal{A}$ is an answer set of $\Pi \cup C$ *iff* $\mathcal{A}$ is an answer set of Π 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 σ 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 σ. □

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

$$
\begin{aligned}
&R(a)\\
&R(x) \leftarrow R(y), E(y,x)\\
&\qquad \leftarrow \mathsf{not}\, R(x),
\end{aligned}
$$

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:

$$
\begin{aligned}
&R(a) \wedge \forall xy\big(E(y,x) \wedge R(y) \rightarrow R(x)\big)\\
&\wedge \forall x\big(R(x) \rightarrow x = a \vee \exists y\big(R(y) \wedge E(y,x) \wedge y \leqslant_{RR} x \wedge \neg x \leqslant_{RR} y\big)\big)\\
&\wedge \forall xyz(x \leqslant_{RR} y \wedge y \leqslant_{RR} z \rightarrow x \leqslant_{RR} z)\\
&\wedge \forall x R(x).
\end{aligned}
$$

3.4. Adding choice rules

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

$$\big\{P(\vec{x})\big\}, \tag{10}$$

where P is a predicate and $\vec{x}$ 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 (10) on a structure $\mathcal{M}$ contains all rules of the form:

$$\big\{P(\vec{u})\big\},$$

where $\vec{u}$ is a tuple of domain elements in $\mathcal{M}$ that matches the arity of P. The set of ground constraints of a constraint of form (9) on a structure $\mathcal{M}$ contains all the instances of the constraint under $\mathcal{M}$.

The answer set semantics for propositional programs with choice rules and constraints can be defined by Gelfond–Lifschitz transformation as well [22]. Let p be a propositional atom and $\mathcal{A}$ a set of atoms. The *reduct* of the choice rule $\{p\}$ relative to $\mathcal{A}$ is p itself if $p \in \mathcal{A}$, and empty (i.e. $\top$) otherwise. Again, a set $\mathcal{A}$ of propositional atoms is said to be an *answer set* of a propositional program Π with choice rules and constraints if $\mathcal{A} \models \Pi^2$ and $\mathcal{A}$ is the minimal model of $\Pi^{\mathcal{A}}$.

Then, a structure $\mathcal{A}$ is said to be a *stable model* of a first-order normal program Π with choice rules and constraints if $EQ_{\mathcal{A}} \cup Ext_{\mathcal{A}} \cup Int_{\mathcal{A}}$ is an answer set of the ground program $\Pi_{\mathcal{A}}$ in the propositional case.

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

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

- *For each intensional predicate P, the following sentence*:

$$\forall \vec{x}\Big(\bigvee_{1 \leqslant i \leqslant k} \exists \vec{y}_i \widehat{Body_i} \rightarrow P(\vec{x})\Big).$$

- *For each intensional predicate P not in Ω, the following sentence*:

$$\forall \vec{x}\Big(P(\vec{x}) \rightarrow \bigvee_{1 \leqslant i \leqslant k} \exists \vec{y}_i \Big(\widehat{Body_i} \wedge \bigwedge_{Q(\vec{z}) \in Pos_i,\, Q \in \Omega_\Pi \setminus \Omega} (\vec{z} \leqslant_{QP} \vec{x} \wedge \neg \vec{x} \leqslant_{PQ} \vec{z})\Big)\Big).$$

- *For each triple of intensional predicates P, Q, and R not in Ω, the following sentence*:

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

- *Finally, for each $c \in C$, the sentence $\hat{c}$.*

[2] Firstly, this means that $\mathcal{A}$ satisfies all constraints. Secondly, $\mathcal{A}$ is regarded as a model of every choice rule.

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

$$\leftarrow \beta_1, \ldots, \beta_k, \mathsf{not}\, \gamma_1, \ldots, \mathsf{not}\, \gamma_l, \mathsf{not}\, \alpha. \qquad \square$$

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

$$\begin{aligned}
&\{hc(x, y)\} \\
&\quad \leftarrow hc(x, y), \mathsf{not}\, E(x, y) \\
&\quad \leftarrow hc(x, y), hc(x, z), y \neq z \\
&\quad \leftarrow hc(y, x), hc(z, x), y \neq z \\
&R(x) \leftarrow hc(a, x) \\
&R(x) \leftarrow R(y), hc(y, x) \\
&\quad \leftarrow \mathsf{not}\, R(x),
\end{aligned}$$

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:

$$\begin{aligned}
&\forall x\big(hc(a, x) \to R(x)\big) \wedge \forall xy\big(hc(y, x) \wedge R(y) \to R(x)\big) \\
&\wedge \forall x\big(R(x) \to hc(a, x) \vee \exists y\big(R(y) \wedge hc(y, x) \wedge y \leqslant_{RR} x \wedge \neg x \leqslant_{RR} y\big)\big) \\
&\wedge \forall xyz(x \leqslant_{RR} y \wedge y \leqslant_{RR} z \to x \leqslant_{RR} z) \\
&\wedge \forall xy\neg\big(hc(x, y) \wedge \neg E(x, y)\big) \\
&\wedge \forall xyz\neg\big(hc(x, y) \wedge hc(x, z) \wedge y \neq z\big) \\
&\wedge \forall xyz\neg\big(hc(y, x) \wedge hc(z, x) \wedge y \neq z\big) \\
&\wedge \forall xR(x).
\end{aligned}$$

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 $\mathcal{M}$ of the signature $\{E, S, \leqslant_{SS}\}$ such that

- $S(a_i, a_j)$ for all $i < j$;
- $S(a_i, b)$ for all i;
- $\overrightarrow{a_i a_j} \leqslant_{SS} \overrightarrow{a_k a_l}$ for all $j - i \leqslant l - k$, where $i < j$ and $k < l$;
- $\overrightarrow{a_i b} \leqslant_{SS} \overrightarrow{a_j b}$ for all $i < j$.

It can be checked that $\mathcal{M}$ 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 σ be the signature of TC. We assume that ϕ is a first-order theory, whose vocabulary is σ_1 such that $\sigma \subseteq \sigma_1$, and the reducts of the models of ϕ on σ are exactly the stable models of TC. It is well known that TC can be defined by a universal second-order theory [5]. Therefore, the complement of TC can be defined by an existential second-order theory. Hence, there exists a first-order theory whose signature contains σ, and the reducts of all its models on σ are exactly the complement of the class of stable models of TC on arbitrary structures. Let ψ be such as a theory of signature σ_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 ϕ_0 of the signature $\sigma_1 \cap \sigma_2$ (namely σ) such that $\phi \models \phi_0$ and $\phi_0 \models \neg\psi$. This shows that TC is exactly captured by the theory ϕ_0, whose signature is σ. This contradicts to the well-known result that TC is not first-order definable on arbitrary structures [5]. □

3.6. Disjunctive logic programs

Disjunctive logic programs is a very important extension of normal programs for dealing with incomplete information [6,13]. A *disjunctive logic program* is a finite set of *disjunctive rules* of the following form

$$\alpha_1; \ldots; \alpha_n \leftarrow \beta_1, \ldots, \beta_k, \text{not}\, \gamma_1, \ldots, \text{not}\, \gamma_l. \tag{11}$$

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 [6,13].

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 exists a disjunctive program Π 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 ϕ whose signature contains the signature of Π, and the reducts of all its finite models are exactly the finite stable models of Π.*

Proof. We show that the following program 3-UNCOLOR (originated from Example 2 in [6]) cannot be captured by any first-order sentences on finite structures if NP $\neq$ coNP:

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

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 [9], 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 ϕ 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 ϕ 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 [6], stating that in terms of brave reasoning, disjunctive logic programs with the stable model semantics exactly capture the complexity class Σ_2^P, 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 the same strongly connected component in the predicate dependency graph of the program.

Formally, the *predicate dependency graph* of a first-order program Π is a finite graph $PG_\Pi = \langle V, E\rangle$, 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 $\leqslant_{PQ}$ are defined only when P and Q belong to the same maximal predicate loop. More precisely, the ordered completion of Π on maximal predicate loops, denoted by $OC_1(\Pi)$, is of the similar form as the ordered completion of Π (see Definition 3), except that

- $Q(\vec{z})$ in Eq. (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 Eq. (4) 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 $\mathcal{A}$ a finite σ-structure. Then, $\mathcal{A}$ is an answer set of Π if and only if there exists a model $\mathcal{M}$ of $OC_1(\Pi)$ such that $\mathcal{A}$ is the reduct of $\mathcal{M}$ on σ.*

Proof. The "only if" part can be proved by using a similar construction except that the comparison predicates $\leqslant_{PQ}$ are only defined over those pairs of (P, Q) in the same maximal predicate loop. For the "if" part, we can prove it by contradiction again. Similarly, we can get the same sequence of ground atoms $P_0(\vec{a}_0), P_1(\vec{a}_1), \ldots, P_i(\vec{a}_i), \ldots$ such that for all i, $P_i(\vec{a}_i) \in Int_{\mathcal{M}\uparrow\sigma}$, $P_i(\vec{a}_i) \in L$, and both $\overrightarrow{a_{i+1}} \leqslant_{P_{i+1}P_i} \vec{a}_i$ and $\neg\vec{a}_i \leqslant_{P_iP_{i+1}} \overrightarrow{a_{i+1}}$ hold. Notice that for all i ($i \geqslant 0$), there exists an edge from P_i to P_{i+1} in PG_Π. Again, there exists $k < l$ such that $P_k(\vec{a}_k) = P_l(\vec{a}_l)$. 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, $\vec{a}_l \leqslant_{P_lP_{k+1}} \overrightarrow{a_{k+1}}$ holds. This contradicts to the facts that $P_k(\vec{a}_k) = P_l(\vec{a}_l)$ and $\neg\vec{a}_k \leqslant_{P_kP_{k+1}} \overrightarrow{a_{k+1}}$ 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, $OC_1(\Pi)$ 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 the 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 [22]:

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

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 $\leqslant_{RR}(x, y)$, which is binary. Hence, $OC_1(HC)$ is the following set of sentences:

$$\forall xy\big(hc(x, y) \leftrightarrow arc(x, y) \wedge \neg otherroute(x, y)\big),$$
$$\forall xy\big(otherroute(x, y) \leftrightarrow \exists z\big(arc(x, y) \wedge arc(x, z) \wedge hc(x, z) \wedge y \neq z\big) \vee \exists z\big(arc(x, y) \wedge arc(z, y) \wedge hc(z, y) \wedge x \neq z\big)\big),$$
$$\forall y\big(\big(\exists x\big(arc(x, y) \wedge hc(x, y) \wedge reached(x) \wedge \neg init(x)\big) \vee \exists x\big(arc(x, y) \wedge hc(x, y) \wedge init(x)\big)\big) \rightarrow reached(y)\big),$$
$$\forall y\big(reached(y) \rightarrow \big(\exists x\big(arc(x, y) \wedge hc(x, y) \wedge init(x)\big) \vee \exists x\big(arc(x, y) \wedge hc(x, y) \wedge reached(x) \wedge \neg init(x) \wedge x \leqslant_{RR} y \wedge \neg y \leqslant_{RR} x\big)\big)\big),$$
$$\forall x\neg\big(vertex(x) \wedge \neg reached(x)\big),$$
$$\forall xyz(x \leqslant_{RR} y \wedge y \leqslant_{RR} z \rightarrow x \leqslant_{RR} z).$$

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 $\leqslant_{PQ}$ and $\leqslant_{QP}$. 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 $\vec{x} \leqslant_{PQ} \vec{y} \wedge \neg\vec{y} \leqslant_{QP} \vec{x}$ as $\vec{x} <_{PQ} \vec{y}$, meaning that there is a derivation path from $P(\vec{x})$ to $Q(\vec{y})$ but not the other way around. We rank all the intensional predicates occurring in Π as $\{P_1, \ldots, P_n\}$, and define $Rank(P_i) = i$ $(1 \leqslant i \leqslant n)$. For every maximal predicate loop L of Π, and for every two predicates $P_i, P_j \in L$, we introduce a new comparison predicate $<_{P_iP_j}$ if $Rank(P_i) \leqslant Rank(P_j)$. 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

- $\vec{z} \leqslant_{QP} \vec{x} \wedge \neg\vec{x} \leqslant_{PQ} \vec{z}$ in Eq. (4) is replaced by $\vec{z} <_{QP} \vec{x}$ if $Rank(Q) \leqslant Rank(P)$, and $\neg\vec{x} <_{PQ} \vec{z}$ if $Rank(Q) > Rank(P)$;
- the transitivity axioms, i.e. Eq. (4) with respect to a predicate loop L are replaced as follows:
  - for any $P \in L$, the sentence $\forall\vec{x}\neg\vec{x} <_{PP} \vec{x}$, and
  - for any $P_i, P_j, P_k \in L$ such that $Rank(P_i) \leqslant Rank(P_j) \leqslant Rank(P_k)$, the following two sentences:

$$\forall\vec{x}\vec{y}\vec{z}(\vec{x} <_{P_iP_j} \vec{y} \vee \vec{y} <_{P_jP_k} \vec{z} \vee \neg\vec{x} <_{P_iP_k} \vec{z}),$$
$$\forall\vec{x}\vec{y}\vec{z}(\neg\vec{x} <_{P_iP_j} \vec{y} \vee \neg\vec{y} <_{P_jP_k} \vec{z} \vee \vec{x} <_{P_iP_k} \vec{z}).$$

Proposition 9. *Let Π be a normal logic program whose signature is σ, and $\mathcal{A}$ a finite σ-structure. Then, $\mathcal{A}$ is an answer set of Π 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 σ.*

Proof. Again, the "only if" part is easy. Now we prove the "if" part by contradiction. Similarly, we can get a sequence of ground atoms $P_0(\vec{a}_0), P_1(\vec{a}_1), \ldots, P_i(\vec{a}_i), \ldots$ such that for all i, $P_i(\vec{a}_i) \in Int_{\mathcal{M}\uparrow\sigma}$, all $P_i(\vec{a}_i)$ are in a ground loop L. In addition, $\overrightarrow{a_{i+1}} <_{P_{i+1}P_i} \vec{a}_i$ holds if $Rank(P_{i+1}) \leqslant Rank(P_i)$, and $\neg\vec{a}_i <_{P_iP_{i+1}} \overrightarrow{a_{i+1}}$ holds if $Rank(P_{i+1}) > Rank(P_i)$. (Note that here i is not the rank of P_i.) Since all the $<_{PQ}$, where $Rank(P) \leqslant Rank(Q)$, satisfy the new transitivity axioms, by induction, it can be proved that for all $i \leqslant j$, $\overrightarrow{a_j} <_{P_jP_i} \vec{a}_i$ holds if $Rank(P_j) \leqslant Rank(P_i)$, and $\neg\overrightarrow{a_i} <_{P_iP_j} \vec{a}_j$ holds if $Rank(P_j) > Rank(P_i)$. Again, we can find $k < l$ such that $P_k(\vec{a}_k) = P_l(\vec{a}_l)$ in the sequence. Hence, $\vec{a}_l <_{P_lP_{k+1}} \overrightarrow{a_{k+1}}$ holds if $Rank(P_l) \leqslant Rank(P_{k+1})$, and $\neg\overrightarrow{a_{k+1}} <_{P_{k+1}P_l} \vec{a}_l$ holds if $Rank(P_l) > Rank(P_{k+1})$. There are three cases:

Case 1: $Rank(P_k) = Rank(P_l) < Rank(P_{k+1})$. Then, $\vec{a}_l <_{P_l P_{k+1}} \overrightarrow{a_{k+1}}$ holds. Therefore $\vec{a}_k <_{P_k P_{k+1}} \overrightarrow{a_{k+1}}$ holds since $P_k(\vec{a}_k) = P_l(\vec{a}_l)$, a contradiction.

Case 2: $Rank(P_k) = Rank(P_l) > Rank(P_{k+1})$. Then, $\neg\overrightarrow{a_{k+1}} <_{P_{k+1} P_l} \vec{a}_l$ holds. Therefore $\neg\overrightarrow{a_{k+1}} <_{P_{k+1} P_k} \vec{a}_k$ holds since $P_k(\vec{a}_k) = P_l(\vec{a}_l)$, a contradiction.

Case 3: $Rank(P_k) = Rank(P_l) = Rank(P_{k+1})$. Then, $\vec{a}_l <_{P_l P_{k+1}} \overrightarrow{a_{k+1}}$ holds. Therefore $\vec{a}_k <_{P_k P_{k+1}} \overrightarrow{a_{k+1}}$ holds. Also, $\overrightarrow{a_{k+1}} <_{P_{k+1} P_k} \vec{a}_k$ holds. Then, according to the transitivity axioms, $\vec{a}_k <_{P_k P_k} \vec{a}_k$ holds, a contradiction. □

Example 9. Recall the program Π_1 in Example 4

$$p_1 \leftarrow p_2,$$
$$p_2 \leftarrow p_1,$$
$$p_1 \leftarrow \text{not } p_3.$$

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

$$\begin{aligned}
&(p_2 \vee \neg p_3 \rightarrow p_1) \wedge (p_1 \rightarrow p_2) \\
&\wedge (p_1 \rightarrow p_2 \wedge \neg <_{p_1 p_2} \vee \neg p_3) \\
&\wedge (p_2 \rightarrow p_1 \wedge <_{p_1 p_2}) \\
&\wedge (<_{p_1 p_1} \vee <_{p_1 p_1} \vee \neg <_{p_1 p_1}) \wedge (\neg <_{p_1 p_1} \vee \neg <_{p_1 p_1} \vee <_{p_1 p_1}) \\
&\wedge (<_{p_1 p_1} \vee <_{p_1 p_2} \vee \neg <_{p_1 p_2}) \wedge (\neg <_{p_1 p_1} \vee \neg <_{p_1 p_2} \vee <_{p_1 p_2}) \\
&\wedge (<_{p_1 p_2} \vee <_{p_2 p_2} \vee \neg <_{p_1 p_2}) \wedge (\neg <_{p_1 p_2} \vee \neg <_{p_2 p_2} \vee <_{p_1 p_2}) \\
&\wedge (<_{p_2 p_2} \vee <_{p_2 p_2} \vee \neg <_{p_2 p_2}) \wedge (\neg <_{p_2 p_2} \vee \neg <_{p_2 p_2} \vee <_{p_2 p_2}) \\
&\wedge \neg <_{p_1 p_1} \wedge \neg <_{p_2 p_2},
\end{aligned}$$

which is equivalent to

$$\begin{aligned}
&(p_2 \vee \neg p_3 \rightarrow p_1) \wedge (p_1 \rightarrow p_2) \wedge (p_1 \rightarrow p_2 \wedge \neg <_{p_1 p_2} \vee \neg p_3) \\
&\wedge (p_2 \rightarrow p_1 \wedge <_{p_1 p_2}) \wedge \neg <_{p_1 p_1} \wedge \neg <_{p_2 p_2}.
\end{aligned}$$

Then, $OC_2(\Pi_1)$ has three models, namely $\{p_3, <_{p_1 p_2}\}$, $\{p_3\}$, and $\{p_1, p_2, <_{p_1 p_2}\}$. Hence, Π_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 $Rank(P) \leqslant Rank(Q) \leqslant Rank(R)$. 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 Π be a program and $L = \{P_1, \ldots, P_n\}$ be a maximal predicate loop in PG_Π. 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 Π_2

$$P_1(\vec{x}_1) \leftarrow P_2(\vec{x}_2),$$
$$P_2(\vec{x}_2) \leftarrow P_3(\vec{x}_3),$$
$$P_3(\vec{x}_3) \leftarrow P_4(\vec{x}_4),$$
$$P_4(\vec{x}_4) \leftarrow P_1(\vec{x}_1),$$
$$P_1(\vec{x}_1) \leftarrow \text{not } P_5(\vec{x}_5).$$

The predicate dependency graph of Π_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 2×3^3 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 2×3^3 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 Π be a normal logic program whose signature is σ, and $\mathcal{A}$ a finite σ-structure. Then, $\mathcal{A}$ is an answer set of Π if and only if there exists a model $\mathcal{M}$ of $OC_3(\Pi)$ such that $\mathcal{A}$ is the reduct of $\mathcal{M}$ on σ.*

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 $P_k(\vec{a}_k), P_{k+1}(\overrightarrow{a_{k+1}}), \ldots, P_l(\vec{a}_l)$ such that

- for all i $(k \leqslant i \leqslant l)$, $P_i(\vec{a}_i) \in Int_{\mathcal{M}\uparrow\sigma}$;
- for all i $(k \leqslant i \leqslant l)$, $P_i(\vec{a}_i)$ are in a ground loop of $\Pi_{\mathcal{M}\uparrow\sigma}$;
- for all i $(k \leqslant i \leqslant l-1)$, $\overrightarrow{a_{i+1}} <_{P_{i+1}P_i} \vec{a}_i$ holds if $Rank(P_{i+1}) \leqslant Rank(P_i)$, and $\neg\vec{a}_i <_{P_iP_{i+1}} \overrightarrow{a_{i+1}} <_{P_iP_{i+1}} (\vec{a}_i, \overrightarrow{a_{i+1}})$ holds if $Rank(P_{i+1}) > Rank(P_i)$;
- $P_k(\vec{a}_k) = P_l(\vec{a}_l)$.

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 $P_i(\vec{a}_i)$ $(k \leqslant i \leqslant l)$ 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 $P_{i-1}(\overrightarrow{a_{i-1}})$ and $P_{i+1}(\overrightarrow{a_{i+1}})$. 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, $\overrightarrow{a_{i+1}} <_{P_{i+1}P_{i-1}} \overrightarrow{a_{i-1}}$ holds if $Rank(P_{i+1}) \leqslant Rank(P_{i-1})$, and $\neg\overrightarrow{a_{i-1}} <_{P_{i-1}P_{i+1}} \overrightarrow{a_{i+1}}$ holds if $Rank(P_{i+1}) > Rank(P_{i-1})$ since $\langle P_{i-1}, P_i, P_{i+1}\rangle$ satisfy the new transitivity axioms. Hence, $L\backslash\{P_i(\vec{a}_i)\}$ also satisfies the above conditions. This contradicts to our assumption that L has the least number of ground atoms satisfying the conditions. □

4.4. Ordered completion in SMT

In worst case, the transitive formulas in ordered completion need to introduce $O(m^2)$ new predicates and $O(s \times m^3)$ new formulas, where m is the number of intensional predicates and s is the maximal arity of the intensional predicates of Π (e.g. consider the program whose predicated dependency graph is complete). This is sometimes a heavy burden for implementation.

To address this issue, we propose an alternative solution. Inspired by Niemelä's translation [23], we use Satisfiability Modulo Theories (SMT) [24] 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 [24] 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. $TranS(\Pi)$). This is because the comparison predicates $\leqslant_{PQ}$ can be regarded as built-in predicates in the theory of partial orders, which naturally satisfy transitivity. In this sense, the ordered completion of Π in SMT (with the theory of partial orders as the background theory) is simply the modified Clark's completion of Π, namely $MComp(\Pi)$.

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.[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 Π be a normal logic program. The *SMT-ordered completion* of Π, denoted by $OC'(\Pi)$, is the following set of sentences for each intensional predicate P

[3] http://research.microsoft.com/en-us/um/redmond/projects/z3/

$$\forall\vec{x}\Big(\bigvee_{1\leqslant i\leqslant k}\exists\vec{y}_i\widehat{Body_i}\rightarrow P(\vec{x})\Big),$$

$$\forall\vec{x}\Big(P(\vec{x})\rightarrow\bigvee_{1\leqslant i\leqslant k}\exists\vec{y}_i\Big(\widehat{Body_i}\wedge\bigwedge_{Q(\vec{z})\in Pos_i,\,Q\in\Omega_\Pi}n_Q(\vec{z})<n_P(\vec{x})\Big)\Big),$$

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 Π be a normal logic program whose signature is σ, and $\mathcal{A}$ a finite σ-structure. Then the following statements are equivalent*:

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

Proof. Theorem 2 proves $1 \Leftrightarrow 2$. Now we show $2 \Rightarrow 3$. Suppose that we have a structure $\mathcal{M}$ of the signature $\sigma \cup \{\leqslant_{PQ} | P, Q \in \Omega(\Pi)\}$, which is a model of $OC(\Pi)$, i.e. $MComp(\Pi) \cup TranS(\Pi)$. Since $\mathcal{M}$ satisfies $TranS(\Pi)$, we can define an order $\preccurlyeq$ on the set $Int_{\mathcal{M}}$ of intensional ground atoms, i.e. $\{P(\vec{u}) \mid P \in \Omega(\Pi), \vec{u} \in P^{\mathcal{M}}\}$, of Π as follows:

- for every $P(\vec{u}) \in Int_{\mathcal{M}}$, $P(\vec{u}) \preccurlyeq P(\vec{u})$;
- for every pair $P(\vec{u}), Q(\vec{v}) \in Int_{\mathcal{M}}$, $P(\vec{u}) \preccurlyeq Q(\vec{v})$ iff $(\vec{u}, \vec{v}) \in \leqslant_{PQ}{}^{\mathcal{M}}$;
- for every pair $P(\vec{u}), Q(\vec{v}) \in Int_{\mathcal{M}}$, we write $P(\vec{u}) = Q(\vec{v})$ iff $P(\vec{u}) \preccurlyeq Q(\vec{v})$ and $Q(\vec{v}) \preccurlyeq P(\vec{u})$.

Clearly, $\preccurlyeq$ is a partial order on $Int_{\mathcal{M}}$ as the comparison predicates $\leqslant_{PQ}$ satisfy the transitivity axioms, i.e. $TranS(\Pi)$. Therefore, $\preccurlyeq$ can be extended to a total order (also called linear order) $\preccurlyeq'$ on $Int_{\mathcal{M}}$. We can construct a mapping f from $Int_{\mathcal{M}}$ to natural numbers, i.e. $f : Int_{\mathcal{M}} \rightarrow \mathbb{N}$, such that for every $P(\vec{u}) \in Int_{\mathcal{M}}$, $f(P(\vec{u}))$ is the position of $P(\vec{u})$ in this total order $\preccurlyeq'$. That is, $f(P(\vec{u})) = t$ iff there exist t elements $E_1, \ldots E_t \in Int_{\mathcal{M}}$ such that $E_i \neq E_j$ for all $1 \leqslant i < j \leqslant t$ and $E_i \preccurlyeq' P(\vec{u})$ for all $1 \leqslant i \leqslant t$. 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 $\vec{u} \in M^n$, $n_Q(\vec{u}) = f(Q(\vec{u}))$.

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

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 $\sigma \cup \{\leqslant_{PQ} | P, Q \in \Omega(\Pi)\}$ such that $\mathcal{M}$ agrees everything the same on the signature σ and $(\vec{u}, \vec{v}) \in \leqslant_{PQ}{}^{\mathcal{M}}$ iff $n_P(\vec{u}) < n_Q(\vec{v})$ or $n_P(\vec{u}) = n_Q(\vec{v})$. Following the similar arguments, it can be shown that $\mathcal{M}$ is a model of $MComp(\Pi)$. In addition, $\mathcal{M}$ is a model of $TranS(\Pi)$ as well because the functions n_P naturally yield a total order, thus a partial order, on $Int_{\mathcal{M}}$ (the same as $Int_{\mathcal{M}'}$). Hence, $\mathcal{M}$ is a model of $OC(\Pi)$. □

5. Implementation and experimental results

To the best of our knowledge, the ordered completions provide for the first time a translation from first-order normal logic programs under the stable model semantics to classical first-order logic on finite structures. Significantly, this translation enables us to develop a new kind 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 Herbrand 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 [12]. Following the current practice, the input program is divided into two parts:

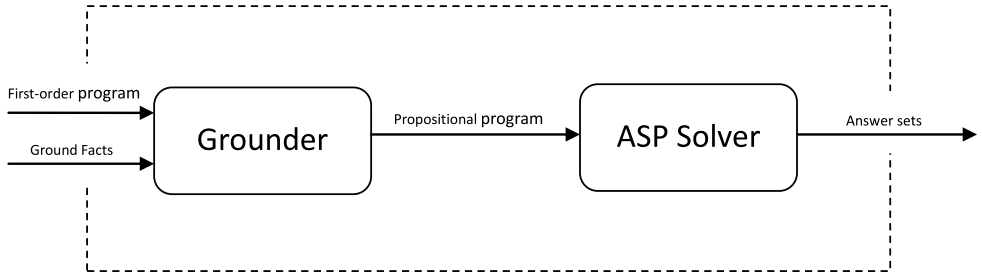

Fig. 2. Traditional ASP solvers.

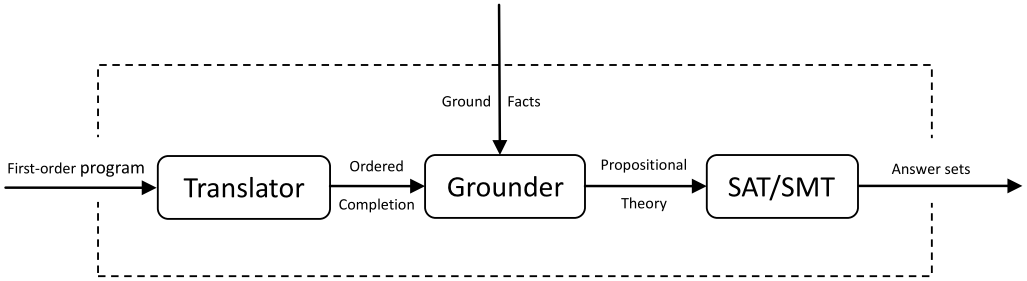

Fig. 3. Our new ASP solver.

1. a first-order normal logic program, and
2. a set of ground facts that defines the extensional predicates.[4] We also call this set an *extensional database*.

Our goal is to compute answer sets of the program based on the given set of 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 for the same first-order logic program, different extensional databases can be provided.

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 Fig. 1). Our goal is to compute 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 program is always the same, the extensional database can be any graph, and the purpose of TC is to compute the transitive closure of the given graph.

Typically, existing ASP solvers have two components (see Fig. 2):

1. A *grounder*, such as lparse[5] and gringo,[6] transforms a first-order logic program together with a set of ground facts into a propositional program.
2. A propositional *ASP solver*, such as clasp,[7] cmodels[8] and lp2diff,[9] computes the answer sets of the propositional program, which correspond to the answer sets of the original first-order program based on those ground facts.

Using our ordered completion, we can do it differently (see Fig. 3):

1′. A *translator* translates a first-order logic program to its ordered completion (see Definition 3).
2′. A *grounder* transforms this ordered completion together with a set of ground facts to a propositional theory.
3′. A *SAT/SMT 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 2/3).

We can see the following potential benefits for our approach:

- Grounding an ordered completion, which is a first-order sentence, is based on the semantics of classical logic. Therefore, many simplification techniques in classical first-order logic can be used that may not be available for logic programs

[4] Notice that the set of ground facts contains no information about the intensional predicates.
[5] http://www.tcs.hut.fi/Software/smodels/.
[6] http://sourceforge.net/projects/potassco/files/gringo/.
[7] http://www.cs.uni-potsdam.de/clasp/.
[8] http://www.cs.utexas.edu/~tag/cmodels/.
[9] http://www.tcs.hut.fi/Software/lp2diff/.

under the stable semantics. These simplification techniques may be crucial to scale up the grounding process to deal with large extensional databases.
- In addition to the simplification technique used in run time when doing grounding, the same or similar techniques can be used to simplify the ordered completion of the "first-order" part of the program, before the extensional database is given. Importantly, this can be done "off line", and done once and for all for each first-order logic program.
- The SAT solver part is used as a black box. Any SAT solver can be used here. Hence, we can benefit from any progress in SAT.

Based on Theorem 3, we have implemented a prototype of a new ASP solver, which contains three steps. First, we translate a program Π 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 Π. 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

The goal of our experiments is to compare our new ASP solver to the existing ASP solvers.

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. In our experiments, we use gringo (version 3.03) as the grounder. Secondly, a propositional ASP solver is called to compute an answer set. In this paper, we consider three different propositional ASP solvers: clasp (version 2.0.1), cmodels (version 3.81), and lp2diff (version 1.27) with Z3 (version 3.2.18).

Our solver, however, needs 3 steps. First, we translate a first-order program to its SMT-ordered completion. As this step is normally very fast and can be done off line, we do not count the time used in this step. Secondly, we implemented a first-order theory grounder called groc, and use it 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 [22] (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*_200_2000_6 is a random graph with 200 nodes and 2000 edges, and is the 6th 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 2 GB of memory and AMD Phenom 9950 Quad-Core processor running at 2.6 GHz. 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) Hamiltonian 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 mentioning 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 instances. For 200-node random graphs, our solver seems not as good as gringo + classp 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 other approaches. Moreover, for 1000-node random graphs, our solver is the only one capable of solving the problems within the time threshold. Also, it is interesting to take a closer

Table 1
Experimental results about the Hamiltonian Circuit program.

Instances		gringo + clasp	gringo + cmodels	gringo + lp2diff + Z3	groc + Z3
rand_200_2000_1	y	0.325	3.130	6.954	1.79
rand_200_2000_2	y	0.604	3.310	10.185	1.95
rand_200_2000_3	n	0.175	0.150	2.507	0.00
rand_200_2000_4	y	1.453	7.960	18.412	1.66
rand_200_2000_5	y	0.329	7.600	8.899	15.24
rand_400_4000_1	y	–	–	49.506	5.08
rand_400_4000_2	y	24.110	–	–	59.31
rand_400_4000_3	?	–	–	–	–
rand_400_4000_4	y	–	–	46.938	8.10
rand_400_4000_5	y	–	–	162.277	8.00
rand_600_6000_1	y	140.830	–	114.973	12.16
rand_600_6000_2	y	–	–	203.500	38.41
rand_600_6000_3	y	–	–	340.219	45.84
rand_600_6000_4	y	–	–	83.650	52.13
rand_600_6000_5	y	–	–	403.075	9.20
rand_1000_10000_1	y	–	–	–	324.22
rand_1000_10000_2	y	–	–	–	133.66
rand_1000_10000_3	y	–	–	–	99.32
rand_1000_10000_4	y	–	–	–	256.91
rand_1000_10000_5	y	–		–	295.89

look at the only instance with no answer sets, i.e. *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. In particular, it is interesting to see if some technique can be developed to simplify the ordered completion. As mentioned earlier, this can be done off line since it only needs to be done once for each program. We have observed that for some logic programs, their ordered completion can indeed be simplified to yield a first-order sentence that is significantly smaller in size. To us, this is one of the most important future work.

6. Related work and discussions

In this section, we discuss and compare our translation to other translations from logic programs under the stable model (answer set) semantics to classical logic. In general, one can say that the intuitions behind most of the translations are similar. The main differences are in the ways that 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 [2,16]. 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 [15] 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 [15] 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 [1,14,18,19,23].

An early attempt is due to Ben-Eliyahu and Dechter [1], 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 [23], 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 $x_q - 1 \geqslant x_p$ 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 [14], who proposed a simplified translation by level numbering as well. Different from the above approaches, Lin and Zhao [18] 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 [19] 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 $\leqslant_{pq} \wedge \neg \leqslant_{qp}$ in the ordered completion, $\#p < \#q$ in Ben-Eliyahu and Dechter's translation and $x_q - 1 \geqslant x_p$ 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 [1], which turns out to be rather complicated. This is not an issue for Niemelä's work [23] because the level numbering x_p associated with atoms and the comparisons $x_q - 1 \geqslant x_p$ 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 $\leqslant_{pq}$ in classical propositional logic to explicitly represent the comparisons $\leqslant_{pq} \wedge \neg \leqslant_{pq}$, 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 Π_1 in Example 4:

$$p_1 \leftarrow p_2,$$
$$p_2 \leftarrow p_1,$$
$$p_1 \leftarrow \mathsf{not}\, p_3.$$

According to the definitions, the modified completion part of Π_1 for the ordered completion is

$$\begin{aligned}&(p_2 \vee \neg p_3 \rightarrow p_1) \wedge (p_1 \rightarrow p_2)\\ &\wedge \big(p_1 \rightarrow p_2 \wedge [\leqslant_{p_2 p_1} \wedge \neg \leqslant_{p_1 p_2}] \vee \neg p_3\big)\\ &\wedge \big(p_2 \rightarrow p_1 \wedge [\leqslant_{p_1 p_2} \wedge \neg \leqslant_{p_2 p_1}]\big),\end{aligned}$$

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

$$\begin{aligned}&(p_2 \vee \neg p_3 \rightarrow p_1) \wedge (p_1 \rightarrow p_2)\\ &\wedge \big(p_1 \rightarrow p_2 \wedge [\#p_2 < \#p_1] \vee \neg p_3\big)\\ &\wedge \big(p_2 \rightarrow p_1 \wedge [\#p_1 < \#p_2]\big),\end{aligned}$$

and finally, for Niemelä's work, this is[10]

$$\begin{aligned}&(p_2 \vee \neg p_3 \rightarrow p_1) \wedge (p_1 \rightarrow p_2)\\ \wedge\ &\big(p_1 \rightarrow p_2 \wedge [x_{p_1} - 1 \geqslant x_{p_2}] \vee \neg p_3\big)\\ \wedge\ &\big(p_2 \rightarrow p_1 \wedge [x_{p_2} - 1 \geqslant x_{p_1}]\big).\end{aligned}$$

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 $x_p - 1 \geqslant x_q$ (e.g. $x_{p_1} - 1 \geqslant x_{p_2}$) with its corresponding counterpart $\leqslant_{qp} \wedge \neg \leqslant_{pq}$ in the ordered completion ($\leqslant_{p_2p_1} \wedge \neg \leqslant_{p_1p_2}$ 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. $x_{p_1} - 1 \geqslant x_{p_2}$, can be naturally represented in the language of difference logic with the built-in predicate $\geqslant$. However, for the other two approaches, more work need to be done.

In the ordered completion, we use additionally transitivity formulas among new atoms $\leqslant_{pq}$ for this purpose. For instance, for the program Π_1, the transitivity formulas is[11]:

$$\begin{aligned}&\leqslant_{p_1p_2} \wedge \leqslant_{p_2p_1} \rightarrow \leqslant_{p_1p_1}\\ \wedge\ &\leqslant_{p_2p_1} \wedge \leqslant_{p_1p_2} \rightarrow \leqslant_{p_2p_2}.\end{aligned}$$

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 Π_1, the encoding of each index, e.g. $\#p_1$, is:

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

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

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

6.3. First-order definability and weak definability

Since the ordered completion is about translating logic programs to first-order logic, it is closely related to the concepts of (first-order) definability for answer set programming [16,27].

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 [5].

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.*

[10] It can be observed that the new atoms bd_a^i in Niemelä's work are not necessary.

[11] All the other transitive formulas are trivially true.

Table 2
From normal ASP to FOL.

Structures	New predicates	Resulting theory	Translation
Arbitrary	Allowed	No restriction	Does not exist
Finite	Not allowed	Finite	Does not exist
Finite	Not allowed	No restriction	Loop formula
Finite	Allowed	Finite	Ordered completion

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,

$$\begin{aligned}\text{Answer set} &= \text{Clark's completion} + \text{Derivation order}\\ &= \text{Ordered completion}\end{aligned}$$

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.

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. Furthermore, it makes it possible to implement a new ASP solver by grounding a first-order theory instead of the program itself, an idea that motivated this and the work of [2] as well. We report our first implementation of such a solver (Section 5), and did some experiments on the Hamiltonian Circuit problems, which are by far the best known benchmark logic programs that have loops. Our results clearly showed that our new solver is competitive, with the edge over others on large problems (see Table 1). We are still working on improving our solver, and believe that we will have some new results to report in the near future.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments. This research is supported in part by ARC Discovery project grant DP0988396, HK RGC GRF 616909 and NSFC grants 90718009 and 60963009.

References

[1] Ben-Eliyahu Rachel, Rina Dechter, Propositional semantics for disjunctive logic programs, Annals of Mathematics and Artificial Intelligence 12 (1–2) (1994) 53–87.
[2] Yin Chen, Fangzhen Lin, Yisong Wang, Mingyi Zhang, First-order loop formulas for normal logic programs, in: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06), 2006, pp. 298–307.
[3] Yin Chen, Fangzhen Lin, Yan Zhang, Yi Zhou, Loop-separable programs and their first-order definability, Artificial Intelligence 175 (3–4) (2011) 890–913.
[4] Keith L. Clark, Negation as failure, in: H. Gallaire, J. Minker (Eds.), Logics and Databases, Plenum Press, New York, 1978, pp. 293–322.
[5] Heinz-Dieter Ebbinghaus, Jörg Flum, Finite Model Theory, Springer-Verlag, 1995.
[6] Thomas Eiter, Georg Gottlob, Heikki Mannila, Disjunctive Datalog, ACM Transactions on Database Systems 22 (3) (1997) 364–418.
[7] Franćois Fages, A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics, New Generation Computing 9 (3–4) (1991) 425–443.
[8] Franćois Fages, Consistency of Clark's completion and existence of stable models, Journal of Methods of Logic in Computer Science 1 (1994) 51–60.
[9] Ronald Fagin, Contributions to the model theory of finite structures, PhD Thesis, UC Berkeley, 1973.
[10] Paolo Ferraris, Joohyung Lee, Vladimir Lifschitz, A new perspective on stable models, Artificial Intelligence 175 (1) (2011) 236–263.
[11] Paolo Ferraris, Answer sets for propositional theories, in: Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05), 2005, pp. 119–131.
[12] Michael Gelfond, Vladimir Lifschitz, The stable model semantics for logic programming, in: Proceedings of the Fifth International Conference and Symposium (ICLP'88), 1988, pp. 1070–1080.
[13] Michael Gelfond, Vladimir Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (3–4) (1991) 365–386.
[14] Tomi Janhunen, Representing normal programs with clauses, in: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04), 2004, pp. 358–362.
[15] Phokion G. Kolaitis, Implicit definability on finite structures and unambiguous computations (preliminary report), in: Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS'90), 1990, pp. 168–180.
[16] Joohyung Lee, Yunsong Meng, On loop formulas with variables, in: Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08), 2008, pp. 444–453.
[17] Vladimir Lifschitz, R. Tang Lappoon, Hudson Turner, Nested expressions in logic programs, Annals of Mathematics and Artificial Intelligence 25 (3–4) (1999) 369–389.
[18] Fangzhen Lin and Jicheng Zhao, On tight logic programs and yet another translation from normal logic programs to propositional logic, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), 2003, pp. 853–858.

[19] Fangzhen Lin, Yuting Zhao, ASSAT: computing answer sets of a logic program by SAT solvers, Artificial Intelligence 157 (1–2) (2004) 115–137.
[20] Fangzhen Lin, Yi Zhou, From answer set logic programming to circumscription via logic of GK, Artificial Intelligence 175 (1) (2011) 264–277.
[21] Victor W. Marek, Miroslaw Truszczynski, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: A 25-Year Perspective, Springer-Verlag, 1999, pp. 375–398.
[22] Ilkka Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (3–4) (1999) 241–273.
[23] Ilkka Niemelä, Stable models and difference logic, Annals of Mathematics and Artificial Intelligence 53 (1–4) (2008) 313–329.
[24] Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli, Solving SAT and SAT modulo theories: from an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T), Journal of the ACM 53 (6) (2006) 937–977.
[25] David Pearce, Agustín Valverde, Towards a first order equilibrium logic for nonmonotonic reasoning, in: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA'04), 2004, pp. 147–160.
[26] David Pearce, A new logical characterisation of stable models and answer sets, in: Non-Monotonic Extensions of Logic Programming (NMELP'96), 1996, pp. 57–70.
[27] Yan Zhang, Yi Zhou, On the progression semantics and boundedness of answer set programs, in: Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR'10), 2010, pp. 518–527.