A Progression Semantics for First-Order Logic Programs

Yi Zhou, Yan Zhang

Artificial Intelligence Research Group (AIRG), School of Computing, Engineering and Mathematics, University of Western Sydney, Australia

Abstract

In this paper, we propose a progression semantics for first-order normal logic programs, and show that it coincides with the stale model (answer set) semantics. The progression semantics sheds new insights into Answer Set Programming (ASP), for instance, its relationships to Datalog, First-Order Logic (FOL) and Satisfiability Modulo Theories (SMT). It can be further evident from the progression semantics that Datalog is exactly the monotonic counterpart of ASP, and many important Datalog techniques can be applied to ASP as well. As an example, we extend the notion of boundedness for ASP, and show that it coincides with the notions of recursion-free and loop-free under program equivalence. In addition, we prove that boundedness precisely captures first-order definability for normal logic programs on arbitrary structures. Finally, we show that the progression semantics suggests an alternative translation from ASP to SMT, which yields a new way of implementing first-order ASP.

1. Introduction

Answer Set Programming (ASP) has emerged as a predominant approach for nonmonotonic reasoning in the area of knowledge representation and reasoning due to its simplicity, expressive power and computational advantage [5, 20, 30, 31]. At its beginning, the stable model (answer set) semantics for first-order logic programs is defined only on Herbrand Structures by grounding into propositional programs [21, 22]. In recent years, a number of approaches have been developed to release this restriction by directly defining the stable model semantics on arbitrary structures [3, 4, 6, 8, 10, 15, 18, 25, 26, 28, 33, 35, 37, 38].

A typical approach on this research line is to use a translation to another host language, e.g. second-order language [18] or circumscription [28]. For this purpose, second-order is inevitable as the class of the stable models of some logic programs, e.g. transitive closure, cannot be captured in first-order logic [16]. Under this backdrop, a first-order logic program Π is transformed to a corresponding second-order sentence $SM(\Pi)$, and the stable models of Π are defined as the models of $SM(\Pi)$ [18]. While this semantics provides a precise mathematical representation and also generalizes the traditional propositional ASP, it, however, does not reveal much information about the expressiveness of first-order answer set programming. For instance, it is unclear whether we can provide a complete characterization of first-order definability for first-order ASP.

In this paper, we propose a progression semantics for first-order normal logic programs. Intuitively, this semantics may be viewed as a generalization of the Gelfond-Lifschitz transformation [5] to the first-order case as well as a generalization of the progression semantics for Datalog [1]. Also, it shares some fundamental ideas with Reiter's semantics for default logic [34]. Simply enough, in the progression semantics, a first-order structure $\mathcal{M}$ is a stable model of a first-order normal logic program Π if and only if it is the fixed point of the progression of Π with respect to $\mathcal{M}$. More precisely, $\mathcal{M}$ coincides with the structure obtained by recursively applying the rules in Π, where the negative parts are fixed by $\mathcal{M}$ itself. We show that, for normal logic programs, this progression semantics coincides with the general stable model semantics defined by $SM(\Pi)$.

The progression semantics sheds new insights into Answer Set Programming (ASP), for instance, its relationships to Datalog, First-Order Logic (FOL) and Satisfiability Modulo Theories (SMT). It can be further evident from the progression semantics that Datalog is exactly the monotonic counterpart of ASP, and many important Datalog techniques can be applied to ASP as well. Based on the proposed progression semantics, we are able to define the notion of boundedness for first-order answer set programs. We prove that our notion of boundedness coincides with the notions of first-order definability for logic programs, and is also equivalent to recursion-free and loop-free under program equivalence. We believe that results in this aspect will establish a foundation for the further study of the expressiveness and related properties of first-order ASP.

The progression semantics is not only of theoretical interests but also of practical relevance as it directly yields a new translation from first-order ASP to Satisfiability Modulo Theories (SMT). Comparing this translation to the one obtained from ordered completion [3, 4], it is logically stronger as it has less models.

This paper is organized as follows. Section 2 introduces necessary backgrounds. Section 3 proposes the progression semantics and shows that it coincides with the translation semantics. Based on the progression semantics, Section 4 shows that Datalog is exactly the monotonic counterpart of first-order ASP. Then, Section 5 extends the notion of boundedness in Datalog for ASP and shows that it is equivalent to the notions of recursion-free and loop-free under program equivalence. Section 6 further shows that boundedness exactly captures first-order definability of ASP. Section 7 reports a natural translation from first-order ASP to SMT based on the progression semantics. Finally, Section 9 discusses some related and ongoing works and Section 10 concludes the paper respectively.

2. Preliminaries

We start with necessary logical notions and notations. We consider a second-order language without function symbols but with equality. A *vocabulary* τ is a set that consists of *relation symbols* (or *predicates*) including the *equality* symbol $=$ and *constant symbols* (or *constants*). Each predicate is associated with a natural number, called its *arity*. Given a vocabulary, *term*, *atom*, *substitution*, (first-order and second-order) *formula* and (first-order and second-order) *sentence* are defined as usual. In particular, an atom is called an *equality* atom if it has the form $t_1 = t_2$, where t_1 and t_2 are terms. Otherwise, it is called a *proper atom*.

A *structure* $\mathcal{A}$ of vocabulary τ (or a τ*-structure*) is a tuple $\mathcal{A} = (A, c_1^{\mathcal{A}}, \cdots, c_m^{\mathcal{A}}, P_1^{\mathcal{A}}, \cdots, P_n^{\mathcal{A}})$, where A is a nonempty set called the *domain* of $\mathcal{A}$, $c_i^{\mathcal{A}}$ $(1 \leq i \leq m)$ is an element in A for every constant c_i in τ, and $P_j^{\mathcal{A}}$ $(1 \leq j \leq n)$ is a k-ary *relation* over A for every k-ary predicate P_j in τ. $P_j^{\mathcal{A}}$ is also called the *interpretations* of P_j in $\mathcal{A}$. A structure is *finite* if its domain is a finite set. In this paper, we consider both finite and infinite structures.

Let $\mathcal{A}$ be a structure of τ. An *assignment* in $\mathcal{A}$ is a function η from the set of variables to A. An assignment can be extended to a corresponding function from the set of terms to A by mapping $\eta(c)$ to $c^{\mathcal{A}}$, where c is an arbitrary constant. Let $P(\overrightarrow{x})$ be an atom, η an assignment in structure $\mathcal{A}$. We also use $P(\overrightarrow{x})\eta \in \mathcal{A}$ to denote $\eta(\overrightarrow{x}) \in P^{\mathcal{A}}$. The *satisfaction relation* $\models$ between a structure $\mathcal{A}$ and a formula ϕ associated with an assignment η, denoted by $\mathcal{A} \models \phi[\eta]$, is defined as usual. Let $\overrightarrow{x}$ be the set of free variables occurring in a formula ϕ. Then, the satisfaction relation only relies on the assignment of $\overrightarrow{x}$. In this case, we write $\mathcal{A} \models \phi(\overrightarrow{x}/\overrightarrow{a})$ to denote the satisfaction relation, where $\overrightarrow{a}$ is a tuple of elements in A. In particular, if ϕ is a sentence, then the satisfaction relation is independent from the assignment. In this case, we simply write $\mathcal{A} \models \phi$ for short. A *ground atom* on A is of the form $P(\overrightarrow{a})$, where P is a predicate and $\overrightarrow{a}$ a tuple of elements that matches the arity of P. For convenience, we also use $P(\overrightarrow{a}) \in \mathcal{M}$, or $\mathcal{M} \models P(\overrightarrow{a})$, to denote $\overrightarrow{a} \in P^{\mathcal{M}}$.

Given a structure $\mathcal{A}$ of τ and an assignment η in $\mathcal{A}$, if Q is a predicate in τ, then we use $\mathcal{A} \cup \{Q(\overrightarrow{x})\eta\}$ to denote a new structure of τ which is obtained from $\mathcal{A}$ by expanding the interpretation of predicate Q in $\mathcal{A}$ (i.e. $Q^{\mathcal{A}}$) to $Q^{\mathcal{A}} \cup \{\eta(\overrightarrow{x})\}$. This can be done similarly for expanding a structure with a set of such values.

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two structures of τ sharing the same domain, and for each constant c in τ, $c^{\mathcal{A}_1} = c^{\mathcal{A}_2}$. By $\mathcal{A}_1 \subseteq \mathcal{A}_2$, we simply mean that for each predicate $P \in \tau$, $P^{\mathcal{A}_1} \subseteq P^{\mathcal{A}_2}$. By $\mathcal{A}_1 \subset \mathcal{A}_2$, we mean that $\mathcal{A}_1 \subseteq \mathcal{A}_2$ but not $\mathcal{A}_2 \subseteq \mathcal{A}_1$. We write $\mathcal{A}_1 \cup \mathcal{A}_2$ to denote the structure of τ where the domain of $\mathcal{A}_1 \cup \mathcal{A}_2$ is the same as $\mathcal{A}_1$ and $\mathcal{A}_2$'s domain, each constant c is interpreted in the same way as in $\mathcal{A}_1$ and $\mathcal{A}_2$, and for each predicate P in τ, $P^{\mathcal{A}_1 \cup \mathcal{A}_2} = P^{\mathcal{A}_1} \cup P^{\mathcal{A}_2}$.

2.1. First-order normal logic program

A *rule* r is of the following form:

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

where α is a proper atom, β_i $(0 \leq i \leq m)$, and γ_j $(0 \leq j \leq l)$ are atoms. We say that α is the *head* of r, denoted by $Head(r)$; $\{\beta_1, \ldots, \beta_m\}$ the *positive body* of r, denoted by $Pos(r)$; and $\{\mathsf{not}\,\gamma_1, \ldots, \mathsf{not}\,\gamma_l\}$ the *negative body* of r, denoted by $Neg(r)$. In addition, we use $Body(r)$ to denote $Pos(r) \cup Neg(r)$.

A *normal logic program* (*program* for short) is a finite set of rules. Given a program Π, predicates that occur in the head of some rules in Π are said to be *intensional*; all other predicates are said to be *extensional*. For a given program Π, we use $\tau(\Pi)$ to denote the vocabulary of Π; $\tau_{ext}(\Pi)$ to denote all the extensional predicates in Π together with all the constants in Π; $\tau_{int}(\Pi)$ to denote all the intensional predicates of Π. Clearly, $\tau(\Pi) = \tau_{ext}(\Pi) \cup \tau_{int}(\Pi)$. In addition, $\tau_{int}(\Pi)$ contains no constants. We also use Ω_Π to denote the set of all intensional predicates of Π. Although Ω_Π is the same as $\tau_{int}(\Pi)$, we use two notations to make a difference because the former denotes a set of predicates whilst the latter presents a vocabulary.

Example 1. Consider the following program Φ about determining whether a certain customer is eligible for discount in a shop.

$$\begin{aligned}
Discount(x) &\leftarrow Staff(x), \mathsf{not}\,BlackList(x), && (2)\\
Discount(x) &\leftarrow VIP(x), \mathsf{not}\,BlackList(x), && (3)\\
BlackList(x) &\leftarrow Purchase(x,y), Discount(x), \mathsf{not}\,Discount(y), && (4)\\
Discount(x) &\leftarrow Delegated(x,y), Discount(y), \mathsf{not}\,BlackList(x). && (5)
\end{aligned}$$

Rules (2) and (3) state that all staff guests and all VIP guests are eligible for discount if there is no evidence that she/he is in the black list. Rule (4) states that if someone uses the discount to purchase something on behalf of another person who has no evidence to be offered the discount, then the former will be put into a black list. Rule (5) states that if someone is delegated from another customer who is eligible for discount and there is no evident that she/he is in the black list, then she/he should be eligible for discount as well. In the program Φ, $Discount$ and $BlackList$ are intensional predicates while $Staff$, VIP, $Purchase$ and $Delegated$ are extensional predicates.

Let us modify the program Φ into a new program $\widetilde{\Phi}$ as follows.

$$\begin{array}{rcl} Discount(x) & \leftarrow & Staff(x), \mathsf{not}\, BlackList(x), \\ Discount(x) & \leftarrow & VIP(x), \mathsf{not}\, BlackList(x), \\ BlackList(x) & \leftarrow & Purchased(x,y), Discount(x), \mathsf{not}\, Discount(y), \\ Discount(x) & \leftarrow & Delegate(x,y), VIP(y), \mathsf{not}\, BlackList(x). \end{array} \tag{6}$$

The only thing changed is the last rule, i.e., rule (5) in Φ is replaced by rule (6) in $\widetilde{\Phi}$, which states that only the VIP guests, instead of all people who are eligible for the discount, have the right to delegate the discount offer to another person.

Without loss of generality, we may assume that all rules are presented in a *normalized form*. That is, each intensional predicate Q is associated with a tuple of distinguishable variables $\overrightarrow{x_Q}$ so that the head of each rule is of the form $Q(\overrightarrow{x_Q})$. For instance, if for some rule with an intensional predicate Q of its head, there is a constant c occurring in Q, i.e. $Q(x_1, \cdots, x_{i-1}, c, x_{i+1}, \cdots, x_n)$, we simply introduce a new variable x_i to replace c: $Q(x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_n)$, and add atom $x_i = c$ in the body of this rule. We say that a variable x is a *local variable* of a rule r if it does not occur in the head of r. For convenience in our proofs, we assume that the sets of local variables in rules are pairwise disjoint.

Let $\mathcal{M}$ be a structure, r a rule of the form (1) and η an assignment. We say that $\mathcal{M}$ satisfies the positive body of r under η, namely $Pos(r)$, written $\mathcal{M} \models Pos(r)\eta$, if for all atoms $P(\overrightarrow{t}) \in Pos(r)$, $\mathcal{M} \models P(\overrightarrow{t})\eta$; $\mathcal{M}$ satisfies the negative body of r under η, namely $Neg(r)$, written $\mathcal{M} \models Neg(r)\eta$, if for all atoms $\mathsf{not}\, P(\overrightarrow{t}) \in Neg(r)$, $\mathcal{M} \not\models P(\overrightarrow{t})\eta$; $\mathcal{M}$ satisfies the body of r under η, namely $Body(r)$, written $\mathcal{M} \models Body(r)\eta$ if $\mathcal{M} \models Pos(r)\eta$ and $\mathcal{M} \models Neg(r)\eta$; and finally, $\mathcal{M}$ satisfies the rule r under η, written $\mathcal{M} \models r\eta$ if $\mathcal{M} \models Head(r)\eta$ whenever $\mathcal{M} \models Body(r)\eta$.

2.2. The translation stable model semantics

Let Π be a program and Ω_Π the set of intensional predicates in Π. We introduce $\Omega_\Pi^* = \{Q_1^*, \ldots, Q_n^*\}$ to 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. Let r be a rule 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

$$\widehat{Body(r)} \rightarrow \alpha,$$

where $\widehat{Body(r)}$ is the conjunction of all elements in $Body(r)$ by replacing each occurrence of not with $\neg$, i.e.

$$\widehat{Body(r)} = \beta_1 \wedge \cdots \wedge \beta_m \wedge \neg\gamma_1 \wedge \cdots \wedge \neg\gamma_l.$$

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

$$\beta_i^*, (1 \leq i \leq m) = \begin{cases} Q_j^*(\overrightarrow{t_j}) & \text{if } \beta_i = Q_j(\overrightarrow{t_j}) \text{ and } Q_j \in \Omega_\Pi \\ \beta_i & \text{otherwise.} \end{cases}$$

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:

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

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

$$\bigwedge_{1\leq i\leq n} \forall\overrightarrow{x}(Q_i^*(\overrightarrow{x}) \rightarrow Q_i(\overrightarrow{x})) \wedge \neg \bigwedge_{1\leq i\leq n} \forall\overrightarrow{x}(Q_i(\overrightarrow{x}) \rightarrow Q_i^*(\overrightarrow{x})).$$

Definition 1 (Translation stable model semantics). *Let* Π *be a program and* $\mathcal{A}$ *a* $\tau(\Pi)$*-structure. We say that* $\mathcal{A}$ *is a(n)* stable model *(*answer set*) of* Π *if* $\mathcal{A}$ *is a model of* $SM(\Pi)$.

We refer this semantics to the *translation semantics*.

Notice that Definition 1 is slightly different from the general stable model semantics [18]. The main reason here is that we are focused on normal logic programs but not arbitrary ones. Nevertheless, it is not difficult to observe that these two are actually the same when restricted into normal logic programs.

2.3. Clark's completion and ordered completion

Answer Set Programming is a rule-based formalism for dealing with iterative reasoning (recursion) and nonmonotonic reasoning, which is significant different from the classic first-order logic. However, these two types of formalisms are closely related. The relationships between answer set programming and classic logics have been one of the central topics in this area since its origin, and have attracted many attentions in the literature [3, 4, 13].

Among them, one influential work is the completion approaches [13], which intend to use first-order sentences directly to capture the stable model (answer set) semantics of logic programs.

Definition 2 (Clark's completion). *Let* Π *be a program and* P *an intensional predicate in* Π. *The Clark's Completion (completion for short if clear from the context) of* Π, *denoted by* $Comp(\Pi)$, *is the following first-order sentence:*

$$\bigwedge_{P \in \tau_{int}(\Pi)} \forall \overrightarrow{x}(P(\overrightarrow{x}) \leftrightarrow \bigvee_{1 \le i \le k} \exists \overrightarrow{y_i} \widehat{Body_i}), \tag{7}$$

where

- P *ranges over all intensional predicates in* Π*;*
- $P(\overrightarrow{x}) \leftarrow Body_1, \ldots, P(\overrightarrow{x}) \leftarrow Body_k$ *are all the rules whose heads mention the predicate* P*;*
- $\overrightarrow{y_i}$ *is the tuple of body variables in* $P(\overrightarrow{x}) \leftarrow Body_i$*;*
- $\widehat{Body_i}$ *is the conjunction of elements in* $Body_i$ *by simultaneously replacing the occurrences of* not *by* $\neg$.

It is shown that any stable model of a program Π must be a classical model of its completion, i.e. $Comp(\Pi)$. However, the converse does not hold in general. In this sense, Clark's completion fails to capture the stable model semantics.

The gap has been bridged recently. Asuncion et al. showed that the stable models semantics for a normal logic program can be exactly captured by a modification of Clark's completion, namely the ordered completion [3, 4]. Roughly speaking, ordered completion enhances Clark's completion by adding some comparison predicates to keep track of the derivation order.

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

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

$$\forall \overrightarrow{x}(\bigvee_{1\leq i\leq k} \exists \overrightarrow{y_i}\widehat{Body_i} \rightarrow P(\overrightarrow{x})), \tag{8}$$

$$\begin{aligned}&\forall \overrightarrow{x}(P(\overrightarrow{x}) \rightarrow \bigvee_{1\leq i\leq k} \exists \overrightarrow{y_i}(\widehat{Body_i} \wedge \\ &\qquad \bigwedge_{Q(\overrightarrow{z})\in Pos_i, Q\in\Omega_\Pi} \leq_{QP} (\overrightarrow{z}, \overrightarrow{x}) \wedge \neg \leq_{PQ} (\overrightarrow{x}, \overrightarrow{z}))),\end{aligned} \tag{9}$$

  where

  - *some basic notations are borrowed from Definition 2;*
  - *Pos_i is the positive part of $Body_i$ so that $Q(\overrightarrow{z})$ ranges over all the intensional atoms in the positive part of $Body_i$;*
  - *$\leq_{QP}$ ($\leq_{PQ}$) is a new predicate, and $\leq_{QP} (\overrightarrow{z}, \overrightarrow{x})$ intuitively means that the level of $Q(\overrightarrow{z})$ is less or equal than the level of $P(\overrightarrow{x})$;*

- *for each triple of intensional predicates P, Q, and R (two or all of them might be the same) the following sentence:*

$$\bigwedge_{P,Q,R\in\Omega_\Pi} \forall \overrightarrow{x}\,\overrightarrow{y}\,\overrightarrow{z}(\leq_{PQ} (\overrightarrow{x}, \overrightarrow{y}) \wedge \leq_{QR} (\overrightarrow{y}, \overrightarrow{z}) \rightarrow \leq_{PR} (\overrightarrow{x}, \overrightarrow{z})). \tag{10}$$

The following theorem states that the stable models of a normal program are corresponding to the classical models of its ordered completion on finite structures.

Proposition 1 (Theorem 1, [3]). *Let Π be a program. Then, a finite $\tau(\Pi)$ structure is a stable model of Π if and only if it can be expanded to a model of $OC(\Pi)$.*

One can further eliminate the transitive formulas (i.e., formula (10)) by using Satisfiability Modulo Theories (SMT), more precisely, first-order logic augmented with a background theory about the comparison operator $<$ of integers. For every predicate P, we introduce an integer predicate n_P with the same arity. Then, the SMT version of ordered completion, written $OC'(\Pi)$, is the conjunction of formula (8) and formula (9), where the second the line of formula (9) is replaced by

$$\bigwedge_{Q(\overrightarrow{z})\in Pos_i, Q\in\Omega(\Pi)} n_Q(\overrightarrow{z}) < n_P(\overrightarrow{x}))).$$

In the SMT version of ordered completion, there is no need for formula (10) as it is implied by the nature of the built-in comparison operator $<$.

2.4. The progression semantics for Datalog

A program is called a *datalog* program if every predicate occurred in the negative part of some rule in the program is extensional. That is, the negative parts of rules in the program mention no intensional predicates, thus their values are fixed.

The semantics for datalog programs is usually defined in a progressional style as follows.

Definition 4 (Datalog evaluation stage). *Let* Π *be a datalog program and* $\mathcal{D}$ *a structure of* $\tau_{ext}(\Pi)$ *(called the* extensional database*). Let* $\Omega_\Pi = \{Q_1, \ldots, Q_n\}$ *be the set of all intensional predicates of* Π*. The* t-th simultaneous evaluation stage *of* Π*, denoted as* $\{Q_1^t, \ldots, Q_n^t\}$*, is defined inductively as follows:*

- *for any* $i, 1 \leq i \leq n$, $Q_i^0 = \emptyset$*;*
- *for any* $i, 1 \leq i \leq n$, $Q_i^{k+1} = Q_i^k \cup \{Head(r)\eta \mid$ *there exists a rule* $r = Q_i(\overrightarrow{x}) \leftarrow Body \in \Pi$ *and an assignment* η *such that* $\mathcal{D} \cup Q_1^k \cup \cdots \cup Q_n^k \models \widehat{Body}[\eta]\}$.

The underlying intuition behind Definition 4 is quite clear. The evaluation stage for a datalog program is defined step-by-step. At the beginning, all interpretations of intensional predicates are set to be empty. At each stage k, the value of an intensional predicate Q_i (i.e. Q_i^{k+1}) is expanded from the previous one (i.e. Q_i^k) with all values computed at this stage by the datalog program Π. More precisely, Q_i^{k+1} is expanded from Q_i^k by the head of all applicable rules associated with Q_i at stage k, where a rule in Π is associated with Q_i if its head mentions Q_i, and is applicable at stage k if its body is satisfied by the current evaluation, i.e., $\mathcal{D} \cup Q_1^k \cup \cdots \cup Q_n^k$.

Clearly, for any Q_i, the sequence $Q_0, Q_1, \ldots, Q_k, \ldots$ is monotonic in the sense that $Q_k \subseteq Q_{k+1}$ for any k. Hence, a convergence always exists. This is called the intended value of Q_i on $\mathcal{D}$ for Π.

Definition 5 (Intended value). *Let* Π *be a datalog program and* $\mathcal{D}$ *a structure of* $\tau_{ext}(\Pi)$*. Let* $Q \in \Omega_\Pi$ *be an intensional predicate. The* intended value of Q on $\mathcal{D}$ for Π*, denoted by* $Q^\infty(\Pi, \mathcal{D})$*, is*

$$\bigcup_{0 \leq j} Q^j.$$

Notice that Definition 4 can be extended for structures with arbitrary cardinality. For an arbitrary cardinal number ϵ, we define

- $Q_i^\epsilon = \bigcup_{\xi<\epsilon} Q_i^\xi \cup \{Head(r)\eta \mid$ there exists a rule $r = Q_i(\overrightarrow{x}) \leftarrow Body \in \Pi$ and an assignment η such that $\mathcal{D} \cup Q_1^\xi \cup \dots \cup Q_n^\xi \models \widehat{Body}[\eta]\}$.

Nevertheless, for simplicity and clarity, we mainly use the notion of evaluation stage proposed in Definition 4 unless stated otherwise. This should not affect the major conclusions drawn in this paper.

3. A Progression Semantics for Normal Logic Programs

In this section, we propose a progression semantics for first-order normal logic programs and show that it coincides with the translation stable model semantics.

3.1. The progression semantics

First of all, we define the evaluation stage for normal logic programs with respect to a structure.

Definition 6 (Evaluation stage). *Let* Π *be a (normal) program and* $\Omega_\Pi = \{Q_1, \dots, Q_n\}$ *the set of all the intensional predicates of* Π. *Consider a structure* $\mathcal{M}$ *of* $\tau(\Pi)$. *The* t-th simultaneous evaluation stage *of* Π *with respect to* $\mathcal{M}$*, denoted by* $\mathcal{M}^t(\Pi)$*, is a structure of* $\tau(\Pi)$ *defined inductively as follows:*

- $\mathcal{M}^0(\Pi) = \mathcal{M}|\tau_{ext}(\Pi) \cup \mathcal{E}_{\tau_{int}(\Pi)}$*, where* $\mathcal{M}|\tau_{ext(\Pi)}$ *is the reduct*[1] *of* $\mathcal{M}$ *on* $\tau_{ext}(\Pi)$*, and* $\mathcal{E}_{\tau_{int}(\Pi)}$ *is the structure defined on* $\tau_{int}(\Pi)$ *such that all interpretations of predicates are empty;*
- $\mathcal{M}^{k+1}(\Pi) = \mathcal{M}^k(\Pi) \cup \{Head(r)\eta |$*there exists* $r = Q(\overrightarrow{x}) \leftarrow \beta_1, \dots, \beta_m, \text{not}\,\gamma_1, \dots, \text{not}\,\gamma_l \in \Pi$ *and an assignment* η *such that for all* i $(1 \le i \le m)$, $\beta_i\eta \in \mathcal{M}^k(\Pi)$, *and for all* j $(1 \le j \le l)$, $\gamma_j\eta \notin \mathcal{M}\}$.

Although Definition 6 looks a little complicated, the underlying idea is quite simple. At each step, we expand the structure by adding those heads of rules that are applicable. Here, a rule r is applicable at step k if $Pos(r)$ is satisfied by $\mathcal{M}^k(\Pi)$ and $Neg(r)$ is satisfied by $\mathcal{M}$.

Let us take a closer look at Definition 6. Clearly, $\mathcal{M}^0(\Pi)$ just takes all extensional relations as the initial input, while all relations corresponding to intensional

[1]Let σ and σ_1 be two signatures such that $\sigma \subseteq \sigma_1$, and $\mathcal{M}$ a structure of σ_1. The *reduct* of $\mathcal{M}$ on σ, denoted by $\mathcal{M}|\sigma$, is a σ-structure that agrees everything the same as $\mathcal{M}$. That is, for every constant $c \in \sigma$ (every predicate $P \in \sigma$), $c^{\mathcal{M}|\sigma} = c^{\mathcal{M}}$ $(P^{\mathcal{M}|\sigma} = P^{\mathcal{M}})$.

predicates in $\tau_{int}(\Pi)$ are set to be empty. Then, $\mathcal{M}^{t+1}(\Pi)$ is obtained from $\mathcal{M}^t(\Pi)$ by adding all derivable intensional values from $\mathcal{M}^t(\Pi)$ by fixing $\mathcal{M}$. Here, an intensional value is derivable from $\mathcal{M}^t(\Pi)$ by fixing $\mathcal{M}$ if there exists a rule applying on an assignment whose head is exactly the intensional value, whose positive body can be derived from $\mathcal{M}^t(\Pi)$ and whose negative body is consistent with $\mathcal{M}$. It is important to emphasize that, in Definition 6, the negative part is fixed by $\mathcal{M}$ (i.e. the original structure) but not $\mathcal{M}^t(\Pi)$ (i.e. the t-th evaluation stage).

For each intensional predicate $Q \in \Omega_\Pi$, we use $Q^i(\Pi, \mathcal{M})$ to denote $Q^{\mathcal{M}^i(\Pi)}$ for simplicity. Then, it is easy to see that the sequence $Q^0(\Pi, \mathcal{M})$, $Q^1(\Pi, \mathcal{M})$, $Q^2(\Pi, \mathcal{M})$, $\cdots$, always increases, that is, $Q^j(\Pi, \mathcal{M}) \subseteq Q^i(\Pi, \mathcal{M})$ for $j < i$. So a convergence for the sequence of $Q^0(\Pi, \mathcal{M})$, $Q^1(\Pi, \mathcal{M})$, $Q^2(\Pi, \mathcal{M})$, $\cdots$, always exists. We call $Q^\infty(\Pi, \mathcal{M}) = \bigcup_{1 \le j \le \infty} Q^j(\Pi, \mathcal{M})$ the *intended value* of Q on $\mathcal{M}$ for Π. Consequently, the convergence of the sequence $\mathcal{M}^0(\Pi)$, $\mathcal{M}^1(\Pi)$, $\mathcal{M}^2(\Pi)$, $\cdots$, also exists:

$$\mathcal{M}^\infty(\Pi) = \bigcup_{1 \le j \le \infty} \mathcal{M}^j(\Pi).$$

If $Q(a_1, \dots, a_n) \in \mathcal{M}^\infty(\Pi)$, then we say that $Q(a_1, \dots, a_n)$ is a *link* of $\mathcal{M}$ with respect to Π. In addition, the *evaluation time* of $Q(a_1, \dots, a_n)$ on $\mathcal{M}$ with respect to Π is the least number t such that $Q(a_1, \dots, a_n) \in \mathcal{M}^t(\Pi)$. In particular, if $Q(a_1, \dots, a_n)$ is not a link of $\mathcal{M}$, we treat the evaluation time of $Q(a_1, \dots, a_n)$ as ∞.

Similarly, Definition 6 can be extended for structures with arbitrary cardinality. For simplicity and clarity, we mainly use the notion and notations in Definition 6. Again, this should not affect the major conclusions drawn in this paper.

Based on the definition of evaluation stage, we are able to present a new stable model semantics for first-order normal logic programs.

Definition 7 (Progression stable model semantics). *Let* Π *be a normal program and* $\mathcal{M}$ *a structure of* $\tau(\Pi)$. $\mathcal{M}$ *is called a* stable model *(or an* answer set*) of* Π *iff* $\mathcal{M}^\infty(\Pi) = \mathcal{M}$.

We refer this semantics to the *progression semantics*. Intuitively, a structure $\mathcal{M}$ is a stable model of a program Π iff it is the fixed point of the progression of Π with respect to $\mathcal{M}$. More precisely, $\mathcal{M}$ coincides with the structure obtained by recursively applying the rules in Π, where the negative parts are fixed by $\mathcal{M}$ itself.

For convenience, we use $AS(\Pi)$ to denote the collection of all answer sets (stable models) of Π. Two programs are said to be *equivalent* if they have the same set of stable models.

Example 2. Consider the following program Π_G:

$$\begin{aligned}&GoShopping(x, y) \leftarrow Friends(x, y),\\&GoShopping(x, y) \leftarrow GoShopping(x, z), Likes(z, y), \mathsf{not}\, Hate(x, y).\end{aligned}$$

Note that *GoShopping* is the only intensional predicate in program Π_G. We consider a finite structure $\mathcal{M}$, where

$$\begin{aligned}&\mathsf{Dom}(\mathcal{M}) = \{alice, carol, jane, sue\},\\&Friends^{\mathcal{M}} = \{(alice, carol), (jane, sue)\},\\&Likes^{\mathcal{M}} = \{(carol, sue)\},\\&Hate^{\mathcal{M}} = \{(alice, jane), (jane, alice)\},\\&GoShopping^{\mathcal{M}} = \{(alice, carol), (jane, sue), (alice, sue)\}.\end{aligned}$$

Then, from Definition 6, we obtain the following sequence:

$$\begin{aligned}&GoShopping^0(\Pi_G, \mathcal{M}) = \emptyset,\\&GoShopping^1(\Pi_G, \mathcal{M}) = \{(alice, carol), (jane, sue)\},\\&GoShopping^2(\Pi_G, \mathcal{M}) = \{(alice, carol), (jane, sue), (alice, sue)\},\\&GoShopping^3(\Pi_G, \mathcal{M}) = GoShopping^2(\Pi_G, \mathcal{M}).\end{aligned}$$

So $GoShopping^{\infty}(\Pi_G, \mathcal{M}) = \{(alice, carol), (jane, sue), (alice, sue)\}$. From Definition 7, we can see that $\mathcal{M}$ is a stable model of Π_G. $\Box$

The progression semantics for answer set programs may be viewed as a generalization of the Gelfond-Lifschitz transformation [21, 22] to the first-order case. First, we guess a first-order structure $\mathcal{M}$. Then, we evaluate the intended values of all intensional predicates with respect to the candidate structure. Finally, if all the intended values are the same as the ones specified in the candidate structure $\mathcal{M}$, then $\mathcal{M}$ is a stable model (answer set) of the underlying program. One major difference is that the progression semantics is defined directly on a first-order level, while the Gelfond-Lifschitz transformation is defined essentially on a propositional level by grounding on the Herbrand structures. Another important difference is that the progression semantics does not require a notion of "reduct". This cannot be done as the "reduct" of a first-order rule with respect to a first-order structure may be corresponding to many different rules under different assignments.

On the other hand, the progression semantics for normal programs can be viewed as an extension of the progression semantics for Datalog [1]. From a syntactic point of view, datalog program is a special case of normal program, where the negative bodies mention no intensional predicates. To address this difference semantically, one needs to handle the occurrences of intensional predicates in the negative bodies. For this purpose, we use several major techniques. First, we guess a candidate structure on the signature of the program but not just use a structure of the extensional signature (i.e., the extensional database) to start with the progression. Second, we fix the negative parts of the program by the guessed structure. In this sense, the evaluation process (i.e., the progression) follows similarly to Datalog. Finally, the guessed structure is considered to be a stable model if it coincides with the progressed structure.

Our progression semantics also shares some fundamental ideas with Reiter's semantics for default logic [34]. Recall Reiter's definition of extensions. First, a candidate theory T is guessed; then an iterative process is applied to compute the result $\Gamma(T)$ of applying default rules with respect to this guessed theory T, in which the negative parts of default rules are fixed by T; finally, T is an extension if it coincides with $\Gamma(T)$. Nevertheless, there are two major differences. First of all, in Reiter's default logic, what we guess is a theory, but in our progress semantics, what we guess is a first-order structure. Also, Reiter's semantics is essentially propositional (or can only be applied to closed first-order logic) as it requires the closure property.

3.2. Progression semantics = translation semantics

We show that the progression semantics (i.e. Definition 7) indeed coincides with the translation semantics (i.e. Definition 1).

Theorem 1. *Let* Π *be a program and* $\mathcal{M}$ *a structure of* $\tau(\Pi)$*. Then,* $\mathcal{M}$ *is a model of* $SM(\Pi)$ *iff* $\mathcal{M}^{\infty}(\Pi) = \mathcal{M}$*.*

Proof: In order to prove this theorem, we introduce an alternative semantics for first-order logic programs and show that it is equivalent to both the progression semantics and the translation semantics described above.

Let Π be a program and $\mathcal{M}$ a structure of $\tau(\Pi)$. We say that $\mathcal{M}$ is a *stable model* of Π iff

1. for every assignment η and every rule r of form (1) in Π, if for all i ($1 \leq i \leq m$), $\beta_i\eta \in \mathcal{M}$ and for all j ($1 \leq j \leq l$), $\gamma_j\eta \notin \mathcal{M}$, then $\alpha\eta \in \mathcal{M}$.
2. there does not exist a structure $\mathcal{M}'$ of $\tau(\Pi)$ such that

(a) $\mathsf{Dom}(\mathcal{M}') = \mathsf{Dom}(\mathcal{M})$,
(b) for each constant c in $\tau(\Pi)$, $c^{\mathcal{M}'} = c^{\mathcal{M}}$,
(c) for each $P \in \tau_{ext}(\Pi)$, $P^{\mathcal{M}'} = P^{\mathcal{M}}$,
(d) for all $Q \in \tau_{int}(\Pi)$, $Q^{\mathcal{M}'} \subseteq Q^{\mathcal{M}}$, and for some $Q \in \tau_{int}(\Pi)$, $Q^{\mathcal{M}'} \subset Q^{\mathcal{M}}$,
(e) for every assignment η and every rule r of form (1) in Π, if for all i $(1 \leq i \leq m)$, $\beta_i\eta \in \mathcal{M}'$ and for all j $(1 \leq j \leq l)$, $\gamma_j\eta \notin \mathcal{M}$, then $\alpha\eta \in \mathcal{M}'$.

We first show that this semantics coincides with the translation semantics. It is not difficult to verify that Condition 1 holds iff $\mathcal{M} \models \widehat{\Pi}$. Now we prove that Condition 2 does not hold iff $\mathcal{M} \models \exists\Omega^*_\Pi((\Omega^*_\Pi < \Omega_\Pi) \wedge \Pi^*)$. On the one hand, suppose that there exists such an $\mathcal{M}'$, we construct n new relations in $\mathcal{M}$ on predicates $\Omega^*_\Pi = \{Q^*_1, \ldots, Q^*_n\}$ corresponding to $\Omega_\Pi = \{Q_1, \ldots, Q_n\}$ such that each $Q^* \in \Omega^*_\Pi$ and its corresponding $Q \in \Omega_\Pi$, $Q^{*\mathcal{M}} = Q^{\mathcal{M}'}$. Therefore, $\mathcal{M} \models \Omega^* < \Omega$ according to Condition 2(d). In addition, from Condition 2(e), it is easy to see that $\mathcal{M}$ satisfies Π^* where for each $Q^* \in \Omega^*_\Pi$, $Q^{*\mathcal{M}} = Q^{\mathcal{M}'}$ as specified above, here Q is Q^*'s corresponding predicate in Ω_Π. Hence, $\mathcal{M} \models \exists\Omega^*_\Pi((\Omega^*_\Pi < \Omega_\Pi) \wedge \Pi^*)$. On the other hand, suppose that $\mathcal{M} \models \exists\Omega^*_\Pi((\Omega^*_\Pi < \Omega_\Pi) \wedge \Pi^*)$. We can always construct $\mathcal{M}'$ in such a way: (1) $\mathsf{Dom}(\mathcal{M}') = \mathsf{Dom}(\mathcal{M})$; (2) for each constant c in $\tau(\Pi)$, $c^{\mathcal{M}'} = c^{\mathcal{M}}$; (3) for each $P \in \tau_{ext}(\Pi)$, $P^{\mathcal{M}'} = P^{\mathcal{M}}$; and (4) for each $Q \in \Omega_\Pi$ and its corresponding $Q^* \in \Omega^*_\Pi$, $Q^{\mathcal{M}'} = Q^{*\mathcal{M}}$. Then it is not difficult to observe that $\mathcal{M}'$ satisfies Conditions 2(c)-(e).

Now we show that this semantics also coincides with the progression semantics. Suppose that $\mathcal{M}^\infty(\Pi) = \mathcal{M}$. Then, Condition 1 holds. Otherwise, there exists an assignment η and a rule r such that, for all i $(1 \leq i \leq m)$, $\beta_i\eta \in \mathcal{M}$ and for all j $(1 \leq j \leq l)$, $\gamma_j\eta \notin \mathcal{M}$ but $\alpha\eta \notin \mathcal{M}$. Since $\beta_i\eta \in \mathcal{M}^\infty(\Pi)$, there exists a bound k such that for all i $(1 \leq i \leq m)$, $\beta_i\eta \in \mathcal{M}^k(\Pi)$. Then, $\alpha\eta \in \mathcal{M}^{k+1}(\Pi)$ by the definition. This means that $\alpha\eta \in \mathcal{M}^\infty(\Pi)$. Therefore, $\alpha\eta \in \mathcal{M}$, a contradiction. In addition, Condition 2 must hold as well. Otherwise, let us assume that there exists such an $\mathcal{M}'$. By induction on the evaluation stage t, it can be shown that for all t, $\mathcal{M}^t(\Pi) \subseteq \mathcal{M}'$. Therefore, $\mathcal{M}^\infty(\Pi) \subseteq \mathcal{M}'$. Hence, $\mathcal{M}^\infty(\Pi) \subseteq \mathcal{M}' \subset \mathcal{M}$, a contradiction. On the other hand, suppose that a structure $\mathcal{M}$ satisfies both Conditions 1 and 2. Then, it can be shown that $\mathcal{M}^t(\Pi) \subseteq \mathcal{M}$ by induction on the evaluation stage t by Condition 1. Hence, $\mathcal{M}^\infty(\Pi) \subseteq \mathcal{M}$. Now we prove $\mathcal{M}^\infty(\Pi) \not\subset \mathcal{M}$. Otherwise, we construct a structure $\mathcal{M}'$ of $\tau(\Pi)$ in the following way: $\mathsf{Dom}(\mathcal{M}') = \mathsf{Dom}(\mathcal{M})$, for each constant $c \in \tau(\Pi)$, $c^{\mathcal{M}'} = c^{\mathcal{M}}$, for each extensional predicate $P \in \tau_{ext}(\Pi)$, $P^{\mathcal{M}'} = P^{\mathcal{M}}$, and for each intensional predicate $Q \in \Omega_{\Pi}$, $Q^{\mathcal{M}'} = Q^{\mathcal{M}^{\infty}(\Pi)}$. So $\mathcal{M}'$ satisfies Conditions 2(a)-(e) as well, a contradiction. Hence, $\mathcal{M}^{\infty}(\Pi) = \mathcal{M}$. $\square$

4. Answer Set Programming vs Datalog

Datalog is one of the roots of Answer Set Programming. From a syntactic point of view, ASP is an extension of Datalog, in which the negation operator (in fact negation-as-failure) might be applied on intensional predicates as well in the body of rules. If we only consider Hebrand structures, ASP is also an extension of Datalog from a semantic point of view in the sense that the Hebrand stable model of a datalog program coincides with the intended values of this program. Further studies showed that ASP (i.e. normal logic programs under the stable model semantics) in general is strictly more expressive than Datalog as a query language on databases [14].

However, since then, the relationships between ASP and Datalog have been long stagnant. For instance, many of the important topics in Datalog, e.g. boundedness [19, 29], first-order definability [2] and the monadic subclass [24], have not been thoroughly studied in the literature. Perhaps one major reason is that their semantics are presented in a very different way. While the semantics for Datalog is defined directly on the first-order level, for ASP, it is essentially propositional by grounding on the Hebrand universe [21, 22]. As a result, in the last two decades, research in ASP has been mainly focused on the propositional level.

Our progression semantics (see Definitions 6 and 7) bridges the gap between ASP and Datalog since it is a natural generalization of the progression semantics for Datalog. It enables us to compare ASP and Datalog in a more detailed level, and also the possibility to apply many of the important techniques in Datalog to ASP.

4.1. From Datalog to ASP

According to Definitions 6 and 7, it can be observed that ASP is an extension of Datalog. The key point is that, in the progression semantics for normal logic programs, the negative parts of rules are fixed by a candidate structure, and then the progression behaves basically the same as that of datalog programs.

The major differences between the progression semantics for ASP and that for Datalog are discussed in the previous section. Now consider the similarities. First of all, both definitions are progression in the sense that the $t+1$-th stages are expanded from the t-th stages by adding some intensional ground atoms, which are the heads of some rules applicable at the t-th stage. Secondly, during the whole process of evaluation, the extensional parts are fixed, and the intensional parts are increasing till a fixed point is obtained. Thirdly, at the initial stage, the intensional parts are both set to be empty.

Syntactically, a datalog program is a normal logic program as well. So the progression semantics for normal logic programs (i.e. Definitions 6 and 7) is also applicable to datalog programs. By induction, it can be shown that progression in this manner is actually irrelevant to the intensional part of the candidate structure. Suppose that Π is a datalog program (to be treated as a normal program) and $\mathcal{M}$ a structure of $\tau(\Pi)$. Applying Definition 6, at the initial stage, we just take the extensional part of $\mathcal{M}$ into account and set the intensional part to be empty. Hence, the 0-th evaluation stage – $\mathcal{M}^0(\Pi)$ – is irrelevant to $\mathcal{M}|\tau_{int}(\Pi)$. Now suppose that $\mathcal{M}^t(\Pi)$ is irrelevant to $\mathcal{M}|\tau_{int}(\Pi)$. Consider the $t+1$-th evaluation stage. The negative parts of rules in Π are fixed by $\mathcal{M}$. Since Π is a datalog program (i.e., the negative parts of rules in Π mention no intensional predicates), the negative parts are irrelevant to $\mathcal{M}|\tau_{int}(\Pi)$. Also, the positive parts are justified by $\mathcal{M}^t(\Pi)$, which is irrelevant to $\mathcal{M}|\tau_{int}(\Pi)$ as well according to the induction hypothesis. Hence, $\mathcal{M}^{t+1}(\Pi)$ – the $t+1$-th simultaneous evaluation stage is irrelevant to $\mathcal{M}|\tau_{int}(\Pi)$. This shows that $\mathcal{M}^\infty(\Pi)$ is irrelevant to $\mathcal{M}|\tau_{int}(\Pi)$. In other words, the progression of a datalog program as a normal program only relies on the extensional part of the candidate structure. That is, for any two structures with the same extensional part (extensional database), their progressions on a datalog program under Definition 6 are exactly the same.

Given an extensional database $\mathcal{D}$ and a structure $\mathcal{M}$ whose extensional part is $\mathcal{D}$ (i.e. $\mathcal{M}|\tau_{ext}(\Pi) = \mathcal{D}$), the progression of a datalog program Π with respect to $\mathcal{M}$ under Definition 6 is only depending on $\mathcal{D}$. Let $\mathcal{A} = \mathcal{M}^\infty(\Pi)$. Clearly, $\mathcal{A} = \mathcal{A}^\infty(\Pi)$ as $\mathcal{A}|\tau_{ext}(\Pi) = \mathcal{D}$. This shows that $\mathcal{A}$ is a stable model of Π. Moreover, it is the only stable model of Π whose extensional part is $\mathcal{D}$ because for any other structure $\mathcal{M}$ where $\mathcal{M}|\tau_{ext}(\Pi) = \mathcal{D}$, $\mathcal{M}^\infty(\Pi) = \mathcal{A} \neq \mathcal{M}$. Hence, fixing an extensional database $\mathcal{D}$, there exists and only exists a stable model of Π according to Definition 7. In addition, this stable model is corresponding to the progression of Π as a datalog program. Comparing Definition 6 and Definition 4, it is clear that for any intensional predicate Q, the interpretation of Q in $\mathcal{A}$, i.e. $Q^\infty(\Pi, \mathcal{A})$, is exactly the same as the intended value of Q on $\mathcal{D}$ for Π, i.e. $Q^\infty(\Pi, \mathcal{D})$.

In this sense, we can conclude that ASP is an extension of Datalog as well from a semantic point of view.

Proposition 2. *Let Π be a datalog program and $\mathcal{D}$ a structure of $\tau_{ext}(\Pi)$. Then, Π has a unique stable model with respect to $\mathcal{D}$, where the interpretation of any intensional predicate Q is the intended value of Q on $\mathcal{D}$ for Π. More precisely, $\mathcal{D} \cup \bigcup_{Q_i \in \tau_{int}(\Pi)} Q_i^{\infty}(\Pi, \mathcal{D})$ is a stable model of Π and it is the only one whose extensional part is $\mathcal{D}$.*

Proof: Let $\mathcal{A} = \mathcal{D} \cup \bigcup_{Q_i \in \tau_{int}(\Pi)} Q_i^{\infty}(\Pi, \mathcal{D})$. By induction on t, it can be shown that for any $Q \in \tau_{int}(\Pi)$, $Q^t(\Pi, \mathcal{A}) = Q^t(\Pi, \mathcal{D})$. Therefore, $Q^{\infty}(\Pi, \mathcal{A}) = Q^{\infty}(\Pi, \mathcal{D})$. This shows that $\mathcal{A}^{\infty}(\Pi) = \mathcal{A}$. Hence, $\mathcal{A}$ is a stable model of Π.

On the other hand, suppose that $\mathcal{M}$ is a $\tau(\Pi)$-structure such that $\mathcal{M}|\tau_{ext}(\Pi) = \mathcal{D}$ and $\mathcal{M} \neq \mathcal{A}$. By induction on t, it can be shown that, for any $Q \in \tau_{int}(\Pi)$, $Q^t(\Pi, \mathcal{M}) = Q^t(\Pi, \mathcal{A})$ as the progression of Π only depends on the extensional part of the candidate structure. Therefore, $\mathcal{M}^{\infty}(\Pi) = \mathcal{A}^{\infty}(\Pi) = \mathcal{A} \neq \mathcal{M}$. This shows that $\mathcal{M}$ is not a stable model of Π. $\Box$

Notice that for a normal program, the progression semantics is not only depending on the extensional part of the candidate structures. In general, given an extensional database, a normal program may have 0, 1 or more stable models.

4.2. From ASP to Datalog

Based on our progression semantics, Proposition 2 simply observes that Datalog is a special subclass of ASP. A question arises: conversely, can ASP be translated back to Datalog in some way?

The progression semantics suggests a simple translation from ASP to Datalog. According to Definition 6, the progression of a normal program is similar to that of a datalog program, except that the negative parts of rules are fixed by the candidate structure. A natural idea is to use extra predicates to simulate the intensional atoms occurring in the negative parts. Then, all rules will turn into datalog rules.

Formally, let Π be a program and Ω_{Π} the set of intensional predicates in Π. We introduce $\Omega'_{\Pi} = \{Q'_1, \dots, Q'_n\}$ to 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}. Ideally, Q'_i is used to match the complement of Q_i.

Let r be a rule. We can separate $Neg(r)$ into two parts: the one mentions only extensional predicates and the other one mentions only intensional predicates. Then, a rule r is of the

$$\alpha \leftarrow \beta_1, \dots, \beta_m, \mathsf{not}\, \gamma_1, \dots, \mathsf{not}\, \gamma_l, \mathsf{not}\, \delta_1, \dots, \mathsf{not}\, \delta_k,$$

where the predicate used in γ_i $(1 \leq i \leq l)$ is intensional and the one used in δ_j $(1 \leq j \leq k)$ is extensional. By r^D ("D" for Datalog), we denote the following rule

$$\alpha \leftarrow \beta_1, \ldots, \beta_m, \gamma'_1, \ldots, \gamma'_l, \mathsf{not}\, \delta_1, \ldots, \mathsf{not}\, \delta_k,$$

where for any i $(1 \leq \imath \leq l)$, γ'_i is $Q'(\overrightarrow{t})$ if γ_i is $Q(\overrightarrow{t})$.

Let Π be a program. By Π^D, we denote the program $\{r^D \mid r \in \Pi\}$. Clearly, Π^D is a datalog program, where $\tau_{int}(\Pi^D) = \tau_{int}(\Pi)$ and $\tau_{ext}(\Pi^D) = \tau_{ext}(\Pi) \cup \Omega'_\Pi$. We use $\Omega_\Pi = \neg\Omega'_\Pi$ to denote the abbreviation of

$$\bigwedge_{Q \in \Pi} \forall \overrightarrow{x} Q(\overrightarrow{x}) \leftrightarrow \neg Q'(\overrightarrow{x}).$$

Intuitively, this formula is to claim that Q' in Ω'_Π simulates the complement of Q in Ω_Π.

Proposition 3. *Let* Π *be a normal logic program and* $\mathcal{A}$ *a structure of* $\tau(\Pi)$. *Let* $\mathcal{A}'$ *be the conservative extension of* $\mathcal{A}$ *under* $\Omega_\Pi = \neg\Omega'_\Pi$. *Then,* $\mathcal{A}$ *is a stable model of* Π *if and only if* $\mathcal{A}'$ *is a stable model of* Π^D.

Proof: By induction on t, it can be shown that for any $Q \in \Omega_\Pi$, $Q^t(\Pi, \mathcal{A}) = Q^t(\Pi^D, \mathcal{A}')$. It follows that $Q^\infty(\Pi, \mathcal{A}) = Q^\infty(\Pi^D, \mathcal{A}')$. Then, $\mathcal{A}$ is a stable model of Π iff $\mathcal{A}^\infty(\Pi) = \mathcal{A}$ iff for any $Q \in \Omega_\Pi$, $Q^\infty(\Pi, \mathcal{A}) = Q^\mathcal{A}$ iff for any $Q \in \Omega_\Pi$, $Q^\infty(\Pi^D, \mathcal{A}') = Q^{\mathcal{A}'}$ (since $Q^\infty(\Pi, \mathcal{A}) = Q^\infty(\Pi^D, \mathcal{A}')$ and $Q^\mathcal{A} = Q^{\mathcal{A}'}$) iff $\mathcal{A}'^\infty(\Pi^D) = \mathcal{A}'$ iff $\mathcal{A}'$ is a stable model of Π^D. □

Nevertheless, Proposition 3 does not mean that ASP can be converted back to Datalog. This is because the program Π^D may have other stable models where the formula $\Omega_\Pi = \neg\Omega'_\Pi$ does not hold. For instance, it is possible that Π has no stable model but Π^D has. Consider the following program Π with a single rule

$$P(x) \leftarrow \mathsf{not}\, P(x).$$

Clearly, Π has no stable model. However, Π^D

$$P(x) \leftarrow P'(x),$$

has many stable models, but none of them satisfies $\Omega_\Pi = \neg\Omega'_\Pi$.

Hence, the translation Π^D is complete but not sound in the sense that for any stable model of the original program Π, there is a corresponding stable model of

Π^D but not vice versa. Here, the difficulty to preserve soundness is to capture $\Omega_\Pi = \neg\Omega'_\Pi$ in Datalog. For this purpose, we need constraint rules, and we shall discuss this in Section 5.

In fact, there is no translation from ASP to Datalog in general that preserves both soundness and completeness. Firstly, it can be observed that fixing the signature, ASP is more expressive than Datalog in terms of models. That is, there exists a normal logic program Π which is not equivalent to any datalog program on the same signature, i.e., there is no datalog program Π' such that $\tau_{ext}(\Pi) = \tau_{ext}(\Pi')$, $\tau_{int}(\Pi) = \tau_{int}(\Pi')$ and $AS(\Pi) = AS(\Pi')$. This is simply because for any datalog program, there must be a unique stable model with respect to a given extensional database. However, this is not the case for answer set programs.

The problem becomes more challenging if extra predicates are allowed. That is, for a normal program Π, does there exist a datalog program Π' with a larger signature and the reducts of the stable models of Π' exactly correspond to the stable models of Π, i.e. $AS(\Pi) = \{\mathcal{M}|\tau(\Pi) \mid \mathcal{M} \in AS(\Pi')\}$? The answer is "no" either providing some general assumptions in the complexity theory. This follows from the studies of expressive power of different query languages. In terms of query language, ASP is strictly more expressive than Datalog [14]. For instance, checking whether a graph is 3-colorable can be captured by a normal logic program but this cannot be done by a Datalog program.

To sum up, the progression semantics further validates that Datalog is exactly the monotonic part of ASP, both from a syntactic and a semantic point of view. Syntactically, Datalog is the monotonic part of ASP. Although the negation-as-failure operator can be used in Datalog, it does not cause nonmonotonicity because the operator can only be applied on extensional predicates, whose interpretations are pre-given and fixed. ASP releases this restriction by allowing it to be applied on intensional predicates as well. Then, the ASP becomes nonmonotonic because one can use those rules to derive new facts from "unknown" facts. Semantically, Datalog is the monotonic part of ASP as well. Datalog is monotonic in the sense that, given any two datalog programs Π and Π' on the same signature such that $\Pi \subseteq \Pi'$, for any extensional database $\mathcal{D}$ and intensional predicate Q, the intended value of Q on $\mathcal{D}$ for Π is always a subset of that for Π', i.e., $Q^\infty(\Pi, \mathcal{D}) \subseteq Q^\infty(\Pi', \mathcal{D})$. In other words, Π' can always derive more information than Π. However, this is not the case for ASP, which is actually nonmonotonic. For instance, the program $\{a \leftarrow \mathsf{not}\, b\}$ has a unique answer set $\{a\}$. However, after adding a new rule $\{b\}$, $\{a\}$ is not an answer set any more. The only answer set for $\{b \leftarrow, a \leftarrow \mathsf{not}\, b\}$ is $\{b\}$.

5. Boundedness, Recursion-free and Loop-free

The progression semantics for normal logic programs is a natural extension of that for datalog programs. As discussed in the previous section, it is important for understanding the deep and long neglected connections between ASP and Datalog. More interestingly, with this semantics, we are able to consider some important notions and techniques originated from Datalog for first-order answer set programming.

Among them, one fundamental notion is boundedness. Roughly speaking, a datalog program is bounded if there exists a natural number k such that every evaluation stage of this program must be ended within k steps. Boundedness is one of the key notions in Datalog, e.g., to study the expressive power of Datalog and classical first-order logic [2].

With our progression semantics, we are able to define the boundedness notion for first-order ASP. Certainly, the basic idea shares the same, i.e., we may require every evaluation of a normal program is bounded by some fixed number as well. However, as we shall see in this section, the definition is not that straightforward as the progression of a normal program is relative to a candidate structure.

Boundedness also plays an important role in first-order ASP. In this section, we shall show that it actually coincides with the syntactic notions of recursion-free and loop-free under program equivalence. Roughly speaking, recursion-free programs are those programs without recursions, that is, the positive bodies of any rules in the program contain no intensional predicates, while loop-free programs, also called tight program in the literature [17], are those programs without loops [11, 27]. In the next section, we will use boundedness as a key tool to study the expressive power of first-order ASP, in particular, its relationships to classical first-order logic.

5.1. Boundedness for normal logic programs

We first review the notion of boundedness in Datalog, which had attracted many attentions in the area of deductive database [1, 29].

Definition 8 (Datalog boundedness). *A datalog program* Π *is* bounded *if there exists a natural number k, such that for every extensional database $\mathcal{D}$, the evaluation stage of* Π *on $\mathcal{D}$ is bounded within k steps, i.e., $Q^\infty = Q^k$ for all intensional predicates Q in* Π.

The boundedness notion can be extended for first-order answer set programming based on our progression semantics for normal logic programs (i.e. Definition 7).

Definition 9 (Boundedness). *A program* Π *is* bounded *if there exists a natural number* k*, such that for all intensional predicates* Q *of* Π *and all stable models* $\mathcal{M}$ *of* Π*,* $Q^\infty(\Pi, \mathcal{M}) = Q^k(\Pi, \mathcal{M})$*; or equivalently,* $\mathcal{M}^\infty(\Pi) = \mathcal{M}^k(\Pi)$*. In this case,* k *is called a* bound *of* Π*, and* Π *is called a* k-bounded program.

Definition 9 is not the same as saying that for all stable models $\mathcal{M}$, there exists a natural number k such that $\mathcal{M}^\infty(\Pi) = \mathcal{M}^k(\Pi)$. It is important to note that, similar to the boundedness notion for Datalog (see Definition 8), the fixed constant k applies on all stable models, i.e., such k is independent from specific structures (stable models). However, the major difference between boundedness for ASP and that for Datalog is that the former only takes the stable models but not all $\tau(\Pi)$-structures into account. Hence, for a k-bounded program Π, there may exist a $\tau(\Pi)$-structure $\mathcal{M}$ such that $\mathcal{M}^\infty(\Pi) \neq \mathcal{M}^k(\Pi)$, where $\mathcal{M}$ is not a stable modal of Π.

Boundedness is a semantic notion in the sense that its definition is only depending on the progression semantics. It intends to capture a certain subclass of all programs, for which their progressions are very restricted.

Example 3. Consider the following program Π_V:

$$Visits(x, y) \leftarrow Interested(x, y), \texttt{not}\ Busy(x),$$
$$Visits(x, y) \leftarrow Visits(z, y), Attraction(y), \texttt{not}\ Busy(x).$$

In program Π_V, $Visits$ is the only intensional predicate. According to Definition 6, it is easy to verify that for any stable model $\mathcal{M}$ of Π_V, the evaluation time for all intended values of $Visits$ is not more than 2. In other words, program Π_V is a 2-bounded program. □

Clearly, the boundedness notion for normal programs is an extension of that for datalog programs.

Proposition 4. *Let* Π *be a datalog program. Then,* Π *is bounded under Definition 9 iff it is bounded under Definition 8.*

As a consequence, some results in the Datalog literature can be directly applied under the context of ASP.

Corollary 5. *Checking boundedness for normal logic programs is undecidable.*

Proof: This assertion follows directly from Proposition 4 and the result that checking boundedness for datalog programs is undecidable (see Theorem 2.5 in [19]). □

5.2. Recursion-free and loop-free

Now we introduce two syntactic notions for first-order normal programs, namely recursion-free and loop-free, which are used to characterize the expressiveness of first-order answer set programs from a syntactic point of view.

Definition 10 (Recursion-free). *A program is said to be* recursion-free *if no intensional predicates occurs in the positive bodies of any rules in the program.*

Note that it is possible that the intensional predicates may occur negatively in a recursion-free program.

Example 4. Consider the following program Π_{VP}:

$$\begin{aligned}&Visits(x,y) \leftarrow Interested(x,y),\\&PossVisit(x,y) \leftarrow Attraction(y), \mathsf{not}\, Visits(x,y).\end{aligned}$$

There are two intensional predicates $Visits$ and $PossVisit$ in program Π_{VP}. Since none of them positively occurs in the bodies of two rules, Π_{VP} is a recursion-free program. $\square$

It is generally considered that recursion is one of the most important feature for datalog and normal logic programs. Hence, recursion-free programs can be considered as "trivial" programs to some extent.

According to the definitions, it is easy to see that the following result holds.

Proposition 6. *If* Π *is a recursion-free program, then* $\mathcal{M}^{\infty}(\Pi) = \mathcal{M}^{1}(\Pi)$ *for any structure* $\mathcal{M}$ *of* $\tau(\Pi)$.

Proposition 6 states that for recursion-free programs, the stable models of the program can be verified within one step. It immediately follows that all recursion-free programs are bounded.

Corollary 7. *A recursion-free program must be bounded.*

A closely related notion is loop-free.[2] For this purpose, we first review the concepts of loops for first-order normal programs [11]. Let Π be a program.

[2]Loop-free is also called tight in the literature [17], particularly in the propositional case. We call it loop-free here in order to compare it with the notion of recursion-free.

The positive dependency graph of Π, denoted by G_Π, is a graph (maybe infinite) (V, E), where V is the set of atoms of $\tau_{int}(\Pi)$, and (α, β) is an edge in E if (a) there exists a rule $r \in \Pi$, and α' and β' in r such that α' is the head of r and β' is one of the positive atoms of intensional predicate in the body of r, and (b) there exists a substitution θ such that $\alpha'\theta = \alpha$ and $\beta'\theta = \beta$. A finite non-empty subset L of V is said to be a *loop* of Π if there exists a cycle in G_Π that goes through only and all the nodes in L.

Loops and their corresponding loop formulas are critical concepts in answer set programming. As shown in [11], under answer set semantics, a logic program can be captured by its completion together with all its loop formulas on finite structures. Also, it initiates an alternative way to compute the answer sets of a program by transforming it to propositional formulas [27].

Definition 11 (Loop-free). *A program is said to be loop-free if it has no loops.*

The answer sets of a loop-free program can be exactly captured by its Clark's completion [11, 17].

Proposition 8 ([11]). *Let Π be a loop-free program. Then, a $\tau(\Pi)$ structure $\mathcal{M}$ is an answer set of Π iff it is a model of $Comp(\Pi)$.*

Example 5. Consider programs Π_V and Π_{VP} once again in Examples 3 and 4 respectively. It is easy to see that Π_V has a loop $L = \{Visits(x, y), Visits(z, y)\}$. So Π_V is not loop-free. On the other hand, program Π_{VP} in Example 4 is loop-free obviously. $\Box$

Clearly, recursion-free programs are loop-free as their dependency graphs have no edges at all.

Proposition 9. *A recursion-free program must be loop-free.*

However, the converse of Proposition 9 does not hold in general. For example, the following program

$$\begin{aligned} Visits(x, y) &\leftarrow Friends(x, y), \\ Friends(x, y) &\leftarrow Likes(x, y), \text{not}\, Hate(x, y). \end{aligned}$$

is loop-free but not recursion-free.

5.3. On the relationships among boundedness, recursion-free and loop-free

In this subsection, we shall show that the syntactic notion of recursion-free and loop-free are closely related with the semantic notion of boundedness. More precisely, these three notions coincide under program equivalence, that is, a program is bounded if and only if it is equivalent to a recursion-free program if and only if it is equivalent to a loop-free program.

Some straightforward observations are presented earlier, e.g., Corollary 7 and Proposition 9. Corollary 7 states that all recursion-free programs must be bounded. We can extend this into the following result.

Proposition 10. *A loop-free program must be bounded.*

Proof: We prove this assertion by contradiction. Assume that Π is not bounded. Then for an arbitrary k, there exists some stable model $\mathcal{M}$ of Π, such that for some intensional predicate Q in Ω_Π, $Q(\overrightarrow{a}) \in \mathcal{M}^{k+1}(\Pi)$ but $Q(\overrightarrow{a}) \notin \mathcal{M}^{k}(\Pi)$. Then from Definition 6, there must exist a rule r in Π:

$$Q(\overrightarrow{x}) \leftarrow \beta_1, \ldots, \beta_m, \mathsf{not}\,\gamma_1, \ldots, \mathsf{not}\,\gamma_l, \tag{11}$$

and an assignment η such that (1) $Q(\overrightarrow{a}) = Q(\overrightarrow{x})\eta$, and (2) for all i $(1 \leq i \leq m)$, $\beta_i\eta \in \mathcal{M}^k(\Pi)$, and for all j $(1 \leq j \leq l)$, $\gamma_j\eta \notin \mathcal{M}$.

Based on this observation, for the given stable model $\mathcal{M}$ of Π, we define the *intensional dependency tree* $\mathcal{T}(Q(\overrightarrow{a}), \mathcal{M})$ for $Q(\overrightarrow{a})$ as follows:

(a) the root of $\mathcal{T}(Q(\overrightarrow{a}, \mathcal{M})$ is $Q(\overrightarrow{a})$,

(b) in (11), for each β_i $(1 \leq i \leq m)$, if β_i is an intensional atom, then $\beta_i\eta$ is a child of $Q(\overrightarrow{a})$,

(c) for each child $\beta_i\eta$ of $Q(\overrightarrow{a})$, we build the subtree $\mathcal{T}(\beta_i\eta, \mathcal{M})$ as in (a) and (b), and repeat the process until no more subtree can be built.

It is clear that $\mathcal{T}(Q(\overrightarrow{a}), \mathcal{M})$ has depth $k+1$. Now from $\mathcal{T}(Q(\overrightarrow{a}), \mathcal{M})$, we construct an *atom based* intensional dependency tree $\mathcal{T}(Q(\overrightarrow{x}))$ for atom $Q(\overrightarrow{x})$ as follows.

(i) let $\theta_0 = \overrightarrow{a}/\overrightarrow{x}$ be a substitution, replace $Q(\overrightarrow{a})$ in $\mathcal{T}(Q(\overrightarrow{a}, \mathcal{M})$ by $Q(\overrightarrow{a})\theta = Q(\overrightarrow{x})$ as the root of $\mathcal{T}(Q(\overrightarrow{x}))$;

(ii) let $\theta_1 = \overrightarrow{b}/\overrightarrow{y}$, where $\overrightarrow{b}$ be the tuple of elements occurring in $\beta_1\eta, \cdots, \beta_l\eta$ but not occurring in $Q(\overrightarrow{a})$, and $\overrightarrow{y}$ be the tuple of variables not occurring in θ_0, then for each child $\beta_i\eta$ in $\mathcal{T}(Q(\overrightarrow{a}, \mathcal{M})$, replace $\beta_i\eta$ by $((\beta_i\eta)\theta_0)\theta_1$ accordingly;

(iii) this process continues until all ground atoms in $\mathcal{T}(Q(\overrightarrow{a}), \mathcal{M})$ have been replaced by the corresponding atoms.

Then $\mathcal{T}(Q(\overrightarrow{x}))$ is a tree with depth $k+1$ where only variables occur in each atom node[3].

From the construction of $\mathcal{T}(Q(\overrightarrow{x}))$, we observe that for each parent-child pair $(Q_i(\overrightarrow{x}), Q_j(\overrightarrow{y}))$ in tree $\mathcal{T}(Q(\overrightarrow{x}))$, there is a corresponding edge $(Q_i(\overrightarrow{x}), Q_j(\overrightarrow{y}))$ in Π's positive dependency graph G_Π.

On the other hand, since Π is not bounded, for any arbitrary k, there exists some stable model $\mathcal{M}$ and $Q(\overrightarrow{a}) \in \mathcal{M}$, we can construct the tree $\mathcal{T}(Q(\overrightarrow{x}))$ with depth $k+1$. Let N be the number of intensional predicates in Π, and we choose some $M > N$. Then it is clear that for some intensional predicate Q, we can construct a tree $\mathcal{T}(Q(\overrightarrow{x}))$ which has a depth $(M+1) > N$. Consequently, there must exist a path from the root to some leaf such that an intensional predicate Q' occurs two or more than two times, i.e. atoms $Q'(\overrightarrow{x})$ and $Q'(\overrightarrow{y})$ are in the path. Therefore, a loop must exist in the corresponding positive dependency graph G_Π. This concludes that Π is not loop-free. $\Box$

Proposition 10 is an extension of Corollary 7 since all recursion-free programs are loop-free by Proposition 9.

Now we consider the other way around, that is, whether or not a bounded program can be converted to a recursion-free/loop-free program. First of all, the following example shows that there exists a bounded program that is neither recursion-free nor loop-free.

Example 6. Let Π_{flag} be the following program:

$$\begin{array}{rcl} Reach(a) & & \\ Reach(x) & \leftarrow & Reach(y), Edge(x,y), flag \\ Reach(x) & \leftarrow & \mathsf{not}\, Reach(x) \\ flag & \leftarrow & flag \end{array} \qquad (12)$$

Clearly, Π_{flag} is not a recursive-free program as the positive body of rule (12) mentions the intensional predicates $Reach$. It is not a loop-free program either since rule (12) forms some loops. However, Π_{flag} is a bounded program. The

[3]Recall that we assume Π is in a normalized form where each intensional predicate is associated with a tuple of distinguished variables.

reason is that the only recursion rule, i.e., rule (12), is guarded by the 0-ary intensional predicate $flag$. As $flag$ will never be generated in the progression, this rule will never be triggered. Thus, the syntactic recursion in rule (12) is actually blocked semantically.

It is easy to see that the above program Π_{flag} can be equivalently transformed to a recursion-free one by simply deleting rule (12). In this sense, Π_{flag} is "semantically" recursion-free to some extent. The following proposition confirms that this kind of semantically recursion-free indeed can be implied by boundedness.

Proposition 11. *If a program is bounded, then it is equivalent to a recursion-free program.*

The proof of Proposition 11 is a little tedious, although a similar result for datalog programs holds straightforwardly. We plan to prove it by constructions and we decompose the constructions into several steps. First, we show that every k-bounded program is equivalent to a 1-bounded program. Then, we show that a 1-bounded program can be equivalently transformed to a recursion-free program.

Let Π be a program. We construct a program Π_t to simulate the t-th evaluation stage of Π. We define Π_t inductively and show that Π_t is a normalized program as well. Firstly, set $\Pi_1 = \Pi$. Since Π is in a normalized form, Π_1 is normalized too. We now specify Π_{t+1} by giving Π_t, which is expanded from Π_t by adding some new rules. Suppose that there exists a rule r in Π of the form

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

and for all i ($1 \leq i \leq m$), if $\beta_i = Q_i(\overrightarrow{t})$ is an intensional atomic formula, then there exists a rule r_i in Π_t such that $Head(r_i)\theta_i = \beta_i$, where θ_i is the substitution $\overrightarrow{x_{Q_i}}/\overrightarrow{t}$. We add a new rule r^* into Π_{t+1} such that:

$$\begin{aligned}
&Head(r^*) = Head(r),\\
&Pos(r^*) = Pos(r)\backslash\{\beta_{i_1}, \ldots, \beta_{i_n}\} \cup Pos(r_1)\theta_1 \cup \ldots\\
&\qquad\qquad \cup Pos(r_n)\theta_n,\\
&Neg(r^*) = Neg(r) \cup Neg(r_1)\theta_1 \cup \cdots \cup Neg(r_n)\theta_n,
\end{aligned}$$

where $\{\beta_{i_1}, \ldots, \beta_{i_n}\}$ is the set of *all* intensional atomic formulas in $\{\beta_1, \ldots, \beta_m\}$, $r_1, \ldots, r_n$ are the corresponding rules in Π_t as discussed above, and θ_i are defined accordingly. In addition, we apply necessary substitutions such that the sets of local variables in rules in Π_{t+1} are pairwise disjoint. It is easy to see that Π_{t+1} is a normalized program as well. Such process is similar to the *unfolding* in propositional logic programs.

Lemma 1. *Let* Π *be a program and* k *an integer. Then,* $\mathcal{M}^k(\Pi) = \mathcal{M}^1(\Pi_k)$ *for any structure* $\mathcal{M}$ *of* $\tau(\Pi)$.

Proof: We prove this assertion by induction on k. Clearly, this assertion holds when $k = 1$. Suppose that for all $k < t$, this assertion holds. Now we prove that it holds when $k = t$ as well.

We first prove that $\mathcal{M}^t(\Pi) \subseteq \mathcal{M}^1(\Pi_t)$. Let $(a_1, \dots, a_n) \in Q^t(\mathcal{M})$, where Q is an intensional predicate of Π. If the evaluation time of $Q(a_1, \dots, a_n)$ is less than t, then $Q(a_1, \dots, a_n) \in \mathcal{M}^1(\Pi_t)$ by induction assumption. If the evaluation time of $Q(a_1, \dots, a_n)$ is exactly t, then according to the definition, there exists a rule $r \in \Pi$ of form (1) and an assignment η such that (a) $\overrightarrow{x_Q}\eta = (a_1, \dots, a_n)$, (b) for all i $(1 \leq i \leq m)$, $\beta_i\eta \in \mathcal{M}^{t-1}(\Pi)$, and (c) for all j $(1 \leq j \leq l)$, $\gamma_j\eta \notin \mathcal{M}$. By induction assumption, for all i $(1 \leq i \leq m)$, $\beta_i\eta \in \mathcal{M}^1(\Pi_{t-1})$. If β_i is of the form $Q(\overrightarrow{t})$, where Q is an intensional predicate, then according to Definition 6, there exists a rule $r_i \in \Pi_{t-1}$ such that $\beta_i\eta$ can be computed by r_i within one step by assuming $\mathcal{M}$. Therefore, $\alpha\eta$ can be computed by the following rule r^* within one step (note that Π_k is normalized for all k).

$$\begin{aligned}
&Head(r^*) = Head(r),\\
&Pos(r^*) = Pos(r)\backslash\{\beta_{i_1}, \dots, \beta_{i_n}\} \cup Pos(r_1)\theta_1 \cup \dots\\
&\qquad\qquad \cup Pos(r_n)\theta_n,\\
&Neg(r^*) = Neg(r) \cup Neg(r_1)\theta_1 \cup \dots \cup Neg(r_n)\theta_n,
\end{aligned}$$

where $\beta_{i_1}, \dots, \beta_{i_n}$ are the atoms discussed above, and r_i and θ_i are defined accordingly. This shows that $Q(a_1, \dots, a_n) \in \mathcal{M}^1_t(\Pi_t)$.

We now prove $\mathcal{M}^1(\Pi_t) \subseteq \mathcal{M}^t(\Pi)$. Suppose that $Q(a_1, \dots, a_n)$ can be computed from Π_t within one step by assuming $\mathcal{M}$, where Q is an intensional predicate of Π. Then there exists a rule $r^* \in \Pi_t$, and an assignment η such that $Head(r^*)\eta = Q(a_1, \dots, a_n)$. Suppose that r^* has the form

$$\begin{aligned}
&Head(r^*) = Head(r),\\
&Pos(r^*) = Pos(r)\backslash\{\beta_{i_1}, \dots, \beta_{i_n}\} \cup Pos(r_1)\theta_1 \cup \dots\\
&\qquad\qquad \cup Pos(r_n)\theta_n,\\
&Neg(r^*) = Neg(r) \cup Neg(r_1)\theta_1 \cup \dots \cup Neg(r_n)\theta_n,
\end{aligned}$$

where $r \in \Pi$, $r_i \in \Pi_{t-1}$, and the others are defined accordingly. Then, $\beta_{i_j}\eta$ can be computed from r_i within one step by assuming $\mathcal{M}$. So $\beta_{i_j}\eta \in \mathcal{M}^{t-1}(\Pi)$ by induction assumption. Consequently, $\alpha\eta \in \mathcal{M}^t(\Pi)$ since it can be computed through rule r. $\Box$

Now we show that every k-bounded program Π is equivalent to a 1-bounded program, more precisely, Π_k.

Proposition 12. *If Π is a k-bounded program, then Π is equivalent to Π_k, which is a 1-bounded program.*

Proof: We first show that Π is equivalent to Π_k by proving that for any structure $\mathcal{M}$, $\mathcal{M}^\infty(\Pi) = \mathcal{M}^\infty(\Pi_k)$. Clearly, $\mathcal{M}^\infty(\Pi) \subseteq \mathcal{M}^\infty(\Pi_k)$ since $\Pi \subseteq \Pi_k$. It suffices to show that $\mathcal{M}^\infty(\Pi_k) \subseteq \mathcal{M}^\infty(\Pi)$. We prove this by induction that for any natural number t, $\mathcal{M}^t(\Pi_k) \subseteq \mathcal{M}^\infty(\Pi)$. The induction basis follows from Lemma 1. Suppose that it holds for all natural numbers less than t, now we prove the case for t. Let $Q(a_1, \dots, a_n)$ be a ground atom in $\mathcal{M}^t(\Pi_k)$ but not in $\mathcal{M}^{t-1}(\Pi_k)$. If it is obtained from a rule in Π itself together with an assignment, then the inductive step holds obviously. Otherwise, there exists a rule $r^* \in \Pi_k$, and an assignment η such that $Head(r^*)\eta = Q(a_1, \dots, a_n)$ and its body can be applied at the current evaluation stage. Suppose that r^* has the form

$$\begin{aligned}
&Head(r^*) = Head(r),\\
&Pos(r^*) = Pos(r)\backslash\{\beta_{i_1}, \dots, \beta_{i_n}\} \cup Pos(r_1)\theta_1 \cup \dots\\
&\qquad\qquad \cup Pos(r_n)\theta_n,\\
&Neg(r^*) = Neg(r) \cup Neg(r_1)\theta_1 \cup \dots \cup Neg(r_n)\theta_n,
\end{aligned}$$

where $r \in \Pi$, $r_i \in \Pi_{t-1}$, and the others are defined accordingly. Notice that the negative parts are irrelevant here as they are fixed by $\mathcal{M}$. Considering the positive parts, for all $i, 1 \le i \le n$, $Pos(r_i)\theta_i \subseteq \mathcal{M}^{t-1}(\Pi_k)$. By the induction hypothesis, $Pos(r_i)\theta_i \subseteq \mathcal{M}^\infty(\Pi)$. This shows that, for all $i, 1 \le i \le n$, $\beta_i\eta \in \mathcal{M}^\infty(\Pi)$. It follows that for the rule $r\eta$, $Pos(r)\eta \subseteq \mathcal{M}^\infty(\Pi)$. Therefore, $Head(r)\eta \in \mathcal{M}^\infty(\Pi)$. Hence, $Q(a_1, \dots, a_n) \in \mathcal{M}^\infty(\Pi)$.

We now show that Π_k is a 1-bounded program. If $\mathcal{M}$ is an answer set of Π_k, then $\mathcal{M}$ is an answer set of Π as well. In addition, $\mathcal{M}^\infty(\Pi_k) = \mathcal{M} = \mathcal{M}^\infty(\Pi) = \mathcal{M}^k(\Pi) = \mathcal{M}^1(\Pi_k)$(by Lemma 1). This shows that Π_k is 1-bounded. $\Box$

We now show that every 1-bounded program is equivalent to a recursion-free program. For this purpose, we decompose this task into two steps. We first show that a 1-bounded program is equivalent to a program with only non-recursive rules and constraints, and then show that constraints can be eliminated into non-recursive rules as well.

Constraints are of the same as normal rules of the form (1) except that the head is empty instead of an atom. More precisely, a *constraint* is of the form

$$\leftarrow \beta_1, \ldots, \beta_m, \mathsf{not}\, \gamma_1, \ldots, \mathsf{not}\, \gamma_l. \tag{13}$$

Let r be a constraint of the form (13). By $\widehat{r}$, we denote the first-order formula

$$\neg(\beta_1 \wedge \cdots \wedge \beta_m \wedge \neg\gamma_1 \wedge \cdots \wedge \neg\gamma_l).$$

Let Π be a program and C a set of constraints. A first-order structure $\mathcal{M}$ is a *stable model* of $\Pi \cup C$ if it is a stable model of Π and for all $c \in C$, $\mathcal{M} \models \widehat{c}$.

Constraints can be simulated by normal rules. More precisely, a constraint of the form (13) can be simulated by the following normal rule

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

where p is a new 0-ary predicate. Also, datalog programs with constraints are able to capture normal logic programs. Following Proposition 3, every normal program Π is equivalent to

$$\Pi^D \cup \{\leftarrow Q(\overrightarrow{x}), Q'(\overrightarrow{x}) \mid Q \in \Omega_\Pi\} \cup \{\leftarrow \mathsf{not}\, Q(\overrightarrow{x}), \mathsf{not}\, Q'(\overrightarrow{x}) \mid Q \in \Omega_\Pi\}.$$

Here, we use constraints to help us to prove that any 1-bounded program can be equivalently transformed into a recursion-free program. First, we show that any 1-bounded program can be equivalently transformed into a recursion-free program with constraints. Let r be a rule of the form (1), by r^C, we denote the constraint

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

Let Π be a program. By Π^C, we denote the program obtained from Π by replacing every recursive rule r with r^C.

Proposition 13. *If* Π *is a 1-bounded program, then* Π *is equivalent to* Π^C.

Proof: First of all, we split the program Π^C into two parts, namely Π^{NR} that contains all non-recursive rules in Π and Π^{RC} that contains all constraints obtained from recursive rules in Π. Then, a stable model of Π^C is a stable model of Π^{NR} that satisfies all constraints in Π^{RC}, which is a stable model of Π^{NR} that satisfies $\widehat{\Pi}$. In addition, a structure $\mathcal{M}$ is a stable model of Π^{NR} iff

- for all ground atoms $Q(\overrightarrow{a}) \in \mathcal{M}$, there exist a non-recursive rule $r \in \Pi$ and an assignment η such that $Head(r)\eta = Q(\overrightarrow{a})$ and $\mathcal{M} \models Body(r)\eta$;
- for all ground atoms $Q(\overrightarrow{a}) \notin \mathcal{M}$, there do not exist a non-recursive rule $r \in \Pi$ and an assignment η such that $Head(r)\eta = Q(\overrightarrow{a})$ and $\mathcal{M} \models Body(r)\eta$.

On one side, suppose that $\mathcal{M}$ is a stable model of Π. Since Π is 1-bounded, we have $\mathcal{M} = \mathcal{M}^1(\Pi)$. Hence, $\mathcal{M}$ is a stable model of Π^{NR} as $\mathcal{M}^1(\Pi)$ satisfies the two conditions mentioned above (according to the definition of the evaluation stage). In addition, $\mathcal{M} \models \widehat{\Pi}$ since $\mathcal{M}$ is a stable model of Π. Hence, $\mathcal{M}$ is a stable model of Π^C.

On the other side, suppose that $\mathcal{M}$ is a stable model of Π^C. Since $\mathcal{M}$ is a stable model of Π^C thus Π^{NR}, $\mathcal{M}^1(\Pi) = \mathcal{M}$. Now assume that $\mathcal{M}$ is not a stable model of Π. Then, $\mathcal{M} \subset \mathcal{M}^\infty(\Pi)$. There exists a ground atom $Q(\overrightarrow{a})$ in $\mathcal{M}^\infty(\Pi)$ but not in $\mathcal{M}$. In fact, there exists such a $Q(\overrightarrow{a})$ in $\mathcal{M}^2(\Pi)$ but not in $\mathcal{M}^1(\Pi)$ (which is the same as $\mathcal{M}$). Otherwise, if $\mathcal{M}^2(\Pi) = \mathcal{M}^1(\Pi)$, then $\mathcal{M} = \mathcal{M}^1(\Pi) = \mathcal{M}^2(\Pi) = \mathcal{M}^3(\Pi) = \cdots = \mathcal{M}^\infty(\Pi)$, a contradiction. Now suppose that $Q(\overrightarrow{a})$ is derived by a rule r together with an assignment η in the second step of the evaluation stage and $Q(\overrightarrow{a}) = Head(r)\eta$. Then, $\mathcal{M} \models Neg(r)\eta$, and $\mathcal{M}^1(\Pi) \models Pos(r)\eta$. Therefore, $\mathcal{M} \models Body(r)\eta$ since $\mathcal{M} = \mathcal{M}^1(\Pi)$. It follows that $\mathcal{M} \models Head(r)\eta$. This shows that $Q(\overrightarrow{a}) \in \mathcal{M}$, a contradiction. $\Box$

Next, we show that recursive-free programs with constraints can always be equivalently transformed into recursive-free programs. Let Π be a recursive-free program and c a constraint of the form (13). Suppose that all the rules in Π whose head is $\beta_i, 1 \leq i \leq m$ are

$$\begin{aligned}&\beta_i \leftarrow \mathsf{not}\, Body_{i1},\\ &\ldots,\\ &\beta_i \leftarrow \mathsf{not}\, Body_{ib_i},\end{aligned}$$

and all the rules in Π whose head is $\gamma_j, 1 \leq j \leq l$ are

$$\begin{aligned}&\gamma_j \leftarrow \mathsf{not}\, Body_{j1},\\ &\ldots,\\ &\gamma_j \leftarrow \mathsf{not}\, Body_{jc_j}.\end{aligned}$$

By $\Pi \oplus c$, we denote the program obtained from Π and c by replacing each $\beta_i \leftarrow$

$\mathsf{not}\, Body_k, i1 \leq k \leq ib_i$ with the following set (*) of rules

$$\begin{aligned}
&\beta_i \leftarrow \mathsf{not}\,\beta_1, \mathsf{not}\, Body_k,\\
&\ldots,\\
&\beta_i \leftarrow \mathsf{not}\,\beta_m, \mathsf{not}\, Body_k,\\
&\beta_i \leftarrow \mathsf{not}\, Body_{11}, \mathsf{not}\, Body_k,\\
&\ldots,\\
&\beta_i \leftarrow \mathsf{not}\, Body_{1c_1}, \mathsf{not}\, Body_k,\\
&\beta_i \leftarrow \mathsf{not}\, Body_{21}, \mathsf{not}\, Body_k,\\
&\ldots,\\
&\beta_i \leftarrow \mathsf{not}\, Body_{2c_2}, \mathsf{not}\, Body_k,\\
&\ldots,\\
&\beta_i \leftarrow \mathsf{not}\, Body_{l1}, \mathsf{not}\, Body_k,\\
&\ldots,\\
&\beta_i \leftarrow \mathsf{not}\, Body_{lc_l}, \mathsf{not}\, Body_k.
\end{aligned}$$

Proposition 14. *If* Π *is a recursive-free program and* c *a constraint with at least one positive atom, then* $\Pi \cup \{c\}$ *is equivalent to* $\Pi \oplus c$.

Proof: Suppose that $\mathcal{M}$ is a stable model of $\Pi \cup \{c\}$. Then, $\mathcal{M}$ is stable model of Π and $\mathcal{M} \models \widehat{c}$. Then, for any assignment η, there are two cases.

Case 1: There exists $\beta_i, 1 \leq i \leq m$ such that $\mathcal{M} \not\models \beta_i\eta$. In this case, consider any $\beta_j\eta, 1 \leq j \neq i \leq m$. Clearly, if $\beta_j\eta \notin \mathcal{M}$, then $\beta_j\eta \notin \mathcal{M}^1(\Pi \oplus c)$ according to the construction of the rule set (*). On the other side, if $\beta_j\eta \in \mathcal{M}$, then there exists a rule in Π of the form $\beta_j \leftarrow \mathsf{not}\, Body_{jk}, 1 \leq jk \leq jb_j$ such that $\mathcal{M} \not\models Body_{jk}\eta$. Then, $\mathcal{M} \not\models [Body_{jk} \cup \{\beta_i\}]\eta$. Then, $\beta_j\eta$ is in $\mathcal{M}^1(\Pi \oplus c)$ since it is justified by the rule $\beta_j \leftarrow \mathsf{not}\,\beta_i, \mathsf{not}\, Body_{jk}$ in the rule set (*).

Case 2: There exists $\gamma_j, 1 \leq j \leq l$ such that $\mathcal{M} \models \gamma_j\eta$. In this case, there exists a rule of the form $\gamma_j \leftarrow \mathsf{not}\, Body_{jk}, 1 \leq k \leq c_j$ such that $\mathcal{M} \not\models Body_{jk}\eta$. Similarly, consider any $\beta_i\eta, 1 \leq i \leq m$. Again, if $\beta_i\eta \notin \mathcal{M}$, then $\beta_i\eta \notin \mathcal{M}^1(\Pi \oplus c)$. If $\beta_i\eta \in \mathcal{M}$, then there exists a rule in Π of the form $\beta_i \leftarrow \mathsf{not}\, Body_{is}, 1 \leq is \leq ib_i$ such that $\mathcal{M} \not\models Body_{is}\eta$. Then, $\beta_i\eta$ is in $\mathcal{M}^1(\Pi \oplus c)$ since it is justified by the rule $\beta_i \leftarrow \mathsf{not}\, Body_{jk}, \mathsf{not}\, Body_{is}$ in the rule set (*).

In addition, for all other atoms α not in the positive body of c, Π and $\Pi \oplus c$ have the same set of rules whose head is α. This shows that for all ground atoms, it is in $\mathcal{M}$ iff it is in $\mathcal{M}^1(\Pi \oplus c)$. It follows that $\mathcal{M}$ is a stable model of $\Pi \oplus c$ as $\Pi \oplus c$ is a recursive-free program.

Suppose that $\mathcal{M}$ is a stable model of Π but does not satisfy c. Then, there exists an assignment η such that for all $\beta_i, 1 \leq i \leq m$, $\beta_i\eta \in \mathcal{M}$ and for all $\gamma_j, 1 \leq j \leq l$, $\gamma_j\eta \notin \mathcal{M}$. We use contradiction to prove that $\mathcal{M}$ is not a stable model of $\Pi \oplus c$. Otherwise, $\beta_1\eta \in \mathcal{M}^1(\Pi \oplus c)$, there exists a rule in (*) that justifies $\beta_1\eta$ when $i = 1$. It cannot be of the form $\beta_1 \leftarrow \mathsf{not}\, \beta_j, \mathsf{not}\, Body_k$ since $\beta_j\eta \in \mathcal{M}$. Suppose that it is of the form $\beta_1 \leftarrow \mathsf{not}\, Body_{ij}, \mathsf{not}\, Body_k$. Then, $\mathcal{M} \not\models Body_{ij}\eta$. It follows that $\mathcal{M} \models \gamma_i\eta$ because of the rule $\gamma_i \leftarrow \mathsf{not}\, Body_{ij}$ is in Π, a contradiction. This shows that $\mathcal{M}$ is not a stable model of $\Pi \oplus c$.

Finally suppose that $\mathcal{M}$ is not a stable model of Π. Then there are two cases.

Case 1: There exists a ground atom that is in $\mathcal{M}$ but not in $\mathcal{M}^1(\Pi)$. In this case, this ground atom is not in $\mathcal{M}^1(\Pi \oplus c)$ either according to the construction of the rule set (*).

Case 2: There exists a ground atom that is in $\mathcal{M}^1(\Pi)$ but not in $\mathcal{M}$. If this atom is not in the positive body of c, then it is in $\mathcal{M}^1(\Pi \oplus c)$ as well. Hence, $\mathcal{M}$ is not a stable model of $\Pi \oplus c$. Suppose that it is of the form $\beta_i\eta, 1 \leq i \leq m$. Since it is in $\mathcal{M}^1(\Pi)$, there exists a rule of the form $\beta_i \leftarrow \mathsf{not}\, Body_k$ such that $\mathcal{M} \not\models Body_k\eta$. Hence, $\mathcal{M} \not\models [Body_k \cup \{\beta_i\}]\eta$ as $\beta_i\eta \notin \mathcal{M}$. Therefore, $\beta_i\eta \in \mathcal{M}^1(\Pi \oplus c)$ as it is justified by the rule $\beta_i \leftarrow \mathsf{not}\, \beta_i, \mathsf{not}\, Body_k$. It follows that $\mathcal{M}$ is not a stable model of $\Pi \oplus c$.

No matter which case is, $\mathcal{M}$ is not a stable model of $\Pi \oplus c$.

This shows that $\Pi \cup \{c\}$ is equivalent to $\Pi \oplus c$. $\Box$

As a consequence,

Corollary 15. *If* Π *is a recursive-free program and* C *a set of constraints, then* Π *is equivalent to a recursion-free program.*

Proof: Note that for eliminating constraints $\leftarrow \mathsf{not}\, \gamma_1, \ldots, \mathsf{not}\, \gamma_l$ without positive body, one only needs to convert it to a non-recursive rule $\gamma_1 \leftarrow \mathsf{not}\, \gamma_1, \ldots, \mathsf{not}\, \gamma_l$. The assertion follows from Proposition 14 and this fact since constraints can be eliminated one-by-one. That is, for a constraint c and a set of constraints C, $\mathcal{M}$ is a stable model of $\Pi \cup C \cup \{c\}$ iff $\mathcal{M}$ is a stable model of $\Pi \cup \{c\}$ and $\mathcal{M}$ satisfies

C iff $\mathcal{M}$ is a stable model of $\Pi \oplus c$ and $\mathcal{M}$ satisfies C. $\square$

Finally, we are able to prove Proposition 11.

Proof of Proposition 11: Proposition 11 follows from Propositions 12 and 13 and Corollary 15. $\square$

It immediately follows from Proposition 11 and Proposition 9 that any bounded program can be equivalently transformed to a loop-free program.

Corollary 16. *If a program is bounded, then it is equivalent to a loop-free program.*

From Corollary 7, Proposition 9, Proposition 10, Proposition 11 and Corollary 16, we can see that the notions of boundedness, recursion-free and loop-free are highly connected. However, these results are not enough to justify the claim made in the beginning of this subsection that boundedness, recursion-free and loop-free coincide under program equivalence. The missing assertion is: if a program is equivalent to a recursion-free or loop-free program (but not necessarily is recursion-free or loop-free itself), must it be bounded? The answer is again yes, and we shall prove it in Section 6. Nevertheless, for this purpose, more tools and techniques are needed.

Notice that the proofs provided in this section are independent on the cardinality of a particular structure. Hence, the main results proved in this section hold both on arbitrary structures and on finite structures.

6. From Answer Set Programming to First-Order Logic

Classical First-Order Logic (FOL) plays a central role in the logic family. As a first-order rule based formalism, Answer Set Programming shares some similarities with FOL but essentially differs from it in several aspects. From a syntactic point of view, although both ASP and FOL use a first-order language, the former uses rule connectives while the latter uses classical connectives, for instance, classical negation $\neg$ is used in FOL but negation-as-failure not is used instead in ASP. From a semantic point of view, both stable models of ASP and classical models of FOL are defined as first-order structures. However, the semantics of ASP embraces recursion and nonmonotonic reasoning but the semantics of FOL does not.

Certainly, the relationship between first-order ASP and FOL is one of most important topics in this area, and it has been well-studied in the literature [3, 4, 12].

Researches in this direction are mainly focused on translating (some subclasses of) first-order ASP into classical FOL. This is because FOL is a well-established formalism so that translations from ASP to FOL would be helpful to understand some essential properties of the former. Also, normal logic programming is only a fragment of first-order logic programming. For instance, it lacks the support of disjunctive heads nor existential quantifiers. Hence, it makes no much sense to translate the full language classical logic to just a fragment of logic programming. Interestingly, some recent work are proposed to translate fragments of FOL (e.g., various description logics) into fragments of ASP (e.g., normal logic programs enhanced with existential quantifiers in the heads), largely driven by the need of rule-based reasoning and defeasible reasoning in ontology engineering [23].

For the possibility of translating first-order normal logic programs under the stable model semantics into classical first-order logic, a rather complete answer has been provided by Asuncion et al. [3] based on previous results in the literature (see Table 2 in [3]). Interestingly and surprisingly, the answer is depending on three factors, considering arbitrary structures or only finite structures, introducing auxiliary predicates or not, allowing the results to be infinite or not. To conclude, there is no translation from normal ASP to FOL when considering arbitrary structures. For finite structures, if no new predicates are introduced and the results are restricted to be finite, again, such translation does not exist. However, there exists translations from normal ASP to FOL when releasing any of the above two conditions. Loop formula is a translation from normal ASP to FOL on finite structures without introducing any new predicates but the translated results could be infinite [11]. Ordered completion is an alternative translation that guarantees the result to be finite but a polynomial number of extra predicates are needed [3].

Although normal ASP cannot be translated into FOL on arbitrary structures in general, this can be done for some subclasses. A well known subclass is the loop-free programs (also called tight programs) [17]. It is shown that the stable models of a loop-free program can be captured by its Clark's completion, which is a first-order sentence. This result is extended to so-called loop-separable programs [12]. In fact, work in this direction is not only theoretically important but also practically relevant. For instance, some modern ASP solvers are built based on the loop-formula approaches, e.g. ASSAT and CMODELS.

However, it still remains an open problem that whether there is an exact characterization of the first-order definability of first-order normal answer set programs, that is, whether we can exactly capture what kind of normal programs are first-order definable. In this paper, we bridge this gap and show that the concept of boundedness exactly capture first-order definability for first-order normal programs on arbitrary structures. That is, a program is first-order definable if and only if it is bounded. Moreover, we show that these two notions coincide with the syntactic notions of recursion-free and loop-free (i.e. tight) under program equivalence.

6.1. First-order definability of answer set programs

We start our discussions with a formal definition of first-order definability of normal logic programs.

Definition 12 (First-order definability). *Let* Π *be a program and* ϕ *a first-order sentence of the signature* $\tau(\Pi)$. *Let* $\mathcal{C}$ *be a class of first-order structures. We say that* ϕ defines Π *on* $\mathcal{C}$ *if the models of* ϕ *in* $\mathcal{C}$ *are exactly the stable models of* Π *in* $\mathcal{C}$.

A program Π *is said to be* first-order definable *on* $\mathcal{C}$ *if there exists such a first-order sentence that defines* Π.

In this paper, we normally consider $\mathcal{C}$ to be the class of all structures or the class of finite structures.

Example 7. Let us consider Π_V again in Example 3. It can be verified that Π_V is defined by the following sentence:
$\forall xy(Visits(x, y) \leftrightarrow (Interested(x, y) \wedge \neg Busy(x) \vee$
$\exists z(z \neq x \wedge Visits(z, y) \wedge Attraction(y) \wedge \neg Busy(x))))$. $\Box$

It is shown in the literature that the stable models of a loop-free program can be exactly captured by its Clark's completion.

Proposition 17. *[[12]] If* Π *is a loop-free program, then* $Comp(\Pi)$ *defines* Π *on both arbitrary structures and finite structures.*

Consequently, by Proposition 9, a recursion-free program is defined by its Clark's completion as well.

6.2. Boundedness = first-order definability

Now we prove that the semantic notion of first-order definability can be exactly captured by the semantic notion of boundedness presented in Section 5 on arbitrary structures, which are further corresponding to the syntactic notions of recursion-free and loop-free under program equivalence.

Theorem 2. *Let* Π *be a program. The following four statements are equivalent on arbitrary structures.*

1. Π *is bounded.*
2. Π *is equivalent to a recursion-free program.*
3. Π *is equivalent to a loop-free program.*
4. Π *is first-order definable.*

Notice that $1 \Rightarrow 2$ is Proposition 11; $2 \Rightarrow 3$ follows straightforwardly from Proposition 9 and $3 \Rightarrow 4$ follows straightforwardly from Proposition 17. We only need to prove $4 \Rightarrow 1$ for Theorem 2.

For this purpose, we need to introduce some background knowledge and results on least fixed-point logic. Let τ be a vocabulary and P a new predicate not in τ with the arity n. Let $\phi(\overrightarrow{x}, P)$ be a first-order formula, where $\overrightarrow{x}$ is the tuple of all free variables in ϕ with length n, and P only occurs positively in ϕ (i.e. every occurrence of P in ϕ is in the scope of even numbers of negations[4]). Given a structure $\mathcal{A}$ of τ, the formula $\phi(\overrightarrow{x}, P)$ defines an operator $\Phi(T)$ from n-ary relation T on $\mathsf{Dom}(\mathcal{A})$ to n-ary relation on $\mathsf{Dom}(\mathcal{A})$:

$$\Phi(T) = \{\overrightarrow{a} \in \mathsf{Dom}(\mathcal{A})^n : \mathcal{A} \models \phi(\overrightarrow{x}/\overrightarrow{a}, T)\}.$$

The least-fixed point formula $\phi^\infty(\overrightarrow{x}, P)$ (ϕ^∞ for short) on $\mathcal{A}$ is constructed inductively as follows:

$$\begin{aligned}&\phi^0(\overrightarrow{x}, P) = \emptyset;\\ &\phi^t(\overrightarrow{x}, P) = \Phi(\textstyle\bigcup_{r<t} \phi^r(\overrightarrow{x}, P)).\end{aligned}$$

Given $\overrightarrow{a} \in \mathsf{Dom}(\mathcal{A})^n$, $\mathcal{A} \models \phi^t(\overrightarrow{x}/\overrightarrow{a}, P)$ iff $\overrightarrow{a} \in \phi^t(\overrightarrow{x}, P)$; $\mathcal{A} \models \phi^\infty(\overrightarrow{x}/\overrightarrow{a}, P)$ iff $\overrightarrow{a} \in \phi^\infty(\overrightarrow{x}, P)$. Since P only positively occurs in ϕ, the sequence $\phi^1, \ldots, \phi^t, \ldots$ always increases. Thus, there exists a least ordinal k such that $\phi^k = \phi^t = \phi^\infty$, where $t > k$.

Since P occurs positively in ϕ, the operator Φ has a *least fixed-point* T_0 in the sense that $\Phi(T_0) = T_0$ and for every T such that $\Phi(T) = T$, $T_0 \subseteq T$. We write $\phi^\infty(\overrightarrow{x}, P)$ (ϕ^∞ for short), which is called *the least fixed-point formula*, to denote the least fixed-point of Φ.

[4]Here we assume that ϕ is constructed only from connectives of $\neg$, $\wedge$ and $\vee$, while $\rightarrow$ and $\leftrightarrow$ are defined in terms of $\neg$, $\wedge$ and $\vee$.

A *fixed-point query* is a formula in fixed-point logic that defines a global relation. More precisely, let $\phi(x_1, \dots, x_n)$ be a formula in fixed-point logic of vocabulary τ, where $x_1, \dots, x_n$ are all the free variables in ϕ. We say that $\phi(x_1, \dots, x_n)$ *expresses* an n-ary global relation of τ if for every structure $\mathcal{A}$ of τ, $\phi(x_1, \dots, x_n)$ yields the following n-ary relation on $\mathsf{Dom}(\mathcal{A})$:

$$\{(a_1, \dots, a_n) \mid \mathcal{A} \models \phi(a_1, \dots, a_n)\}.$$

The notion of definability and boundedness can be defined for least fixed-point logic as well. Let $\mathcal{K}$ be a class of τ-structures. We say that a formula $\psi(\overrightarrow{y})$, where $\overrightarrow{y}$ is the tuple of all free variables in ψ with length n, of τ *defines* the fixed-point $\phi^\infty(\overrightarrow{x}, P)$ on $\mathcal{K}$ iff for every $\mathcal{A} \in \mathcal{K}$ and every $\overrightarrow{a} \in \mathsf{Dom}(\mathcal{A})^n$,

$$\mathcal{A} \models \phi^\infty(\overrightarrow{x}/\overrightarrow{a}, P) \text{ iff } \mathcal{A} \models \psi(\overrightarrow{y}/\overrightarrow{a}).$$

We say that the least-fixed point formula $\phi^\infty(\overrightarrow{x}, P)$ is *bounded* on $\mathcal{K}$ if there exists a fixed natural number k such that for all $\mathcal{A} \in \mathcal{K}$ and every $\overrightarrow{a} \in \mathsf{Dom}(\mathcal{A})^n$, $\mathcal{A} \models \phi^\infty(\overrightarrow{x}/\overrightarrow{a}, P)$ iff $\mathcal{A} \models \phi^k(\overrightarrow{x}/\overrightarrow{a}, P)$.

Barwise and Moschovakis [9] revealed the important correspondence between definability and boundedness on arbitrary structures in least fixed-point logic.

Proposition 18. *[9] Let $\mathcal{K}$ be a class of τ-structures which is first-order finitely axiomatizable*[5]*. A least fixed-point formula is bounded on $\mathcal{K}$ iff it is defined by a first-order formula on $\mathcal{K}$.*

We shall prove $4 \Rightarrow 1$ based on Proposition 18. The basic ideas are divided into two steps. First, we show that for each program, we can construct a program with a single intensional predicate to simulate the original program. Then we show that each program with a single intensional predicate can be translated to an equivalent fixed-point formula.

Let Π be a program. Let $\{P_1, \dots, P_n\}$ be the set of intensional predicates of Π. Suppose that k is the maximal arity among all $P_i, (1 \le i \le n)$. Let $0, 1, \dots, n$ be $n + 1$ distinguishable new constants. Construct a new predicate P whose arity is $k + 1$. Let Π^S be the program obtained from Π by simultaneously replacing each atom $P_i(\overrightarrow{t_i})$ in Π with $P(\overrightarrow{t_i}, 0, \dots, 0, i)$, where the number of occurrences of 0 is equal to $k - |\overrightarrow{t_i}|$. We show that Π^S simulates Π.

[5]That is, there exists a first-order sentence ϕ on τ whose models are exactly captured by $\mathcal{K}$.

Lemma 2. *Let Π be a program and Π^S be the program constructed above. Let $\mathcal{M}$ be a structure of $\tau(\Pi)$. We construct a structure $\mathcal{M}^S$ on $\tau_{ext}(\Pi) \cup \{P\}$ such that*

- *the domain of is $\mathcal{M}^S$ is $M \cup \{0, 1, \dots, n\}$;*
- *for all extensional predicates Q of Π, $Q^{\mathcal{M}^S} = Q^{\mathcal{M}}$;*
- *for all constants c in Π, $c^{\mathcal{M}^S} = c^{\mathcal{M}}$;*
- *for all intensional predicates P_i, $P_i(\overrightarrow{a}) \in \mathcal{M}$ iff $P(\overrightarrow{a}, 0, \dots, 0, i) \in \mathcal{M}^S$.*

Then, for any integer k, P_i, and $\overrightarrow{a}$ that matches the arity of P_i, $P_i(\overrightarrow{a}) \in \mathcal{M}^k(\Pi)$ iff $P(\overrightarrow{a}, 0, \dots, 0, i) \in (\mathcal{M}^S)^k(\Pi^S)$.

Proof: This assertion follows from the constructions and definitions by induction on k. $\Box$

Lemma 2 shows that Π^S can simulate Π in the sense that every intensional atom $P_i(\overrightarrow{t_i})$ in Π is associated with the intensional atom $P(\overrightarrow{t_i}, 0, \dots, 0, i)$ in Π^S.

Now we show that each program with a single intensional predicate can be equivalently transferred into a fixed-point formula on a class of axiomatizable structures. Let Π be a program that only contains a single intensional predicate, say P. Then, all the heads of rules in Π are of the form $P(\overrightarrow{x})$ since Π is normalized. Let P^* be a new predicate that has the same arity as P. Let $\psi(\Pi, P^*)$ be the first-order formula obtained from Π and P^* by two steps: (1) construct a program Π^* by replacing every occurrence of $P(\overrightarrow{t})$ in the negative bodies of any rules in Π with $P^*(\overrightarrow{t})$, (2) let $\psi(\Pi, P^*)$ be the formula $\bigvee_{r \in \Pi^*} \exists \overrightarrow{y} \widehat{Body(r)}$, where $\overrightarrow{y}$ is the set of local variables in rule r. Clearly, $\psi(\Pi, P^*)$ is a first-order formula of the vocabulary $\tau(\Pi) \cup \{P^*\}$, where P only occurs positively and $\overrightarrow{x}$ are all the free variables.

Let $\mathcal{M}$ be a $\tau(\Pi)$-structure. By $\mathcal{M}^*$, we denote the structure of the vocabulary $\tau(\Pi) \cup \{P^*\}$ such that

- $\mathsf{Dom}(\mathcal{M}^*) = \mathsf{Dom}(\mathcal{M})$;
- for all $\overrightarrow{a}$, $P^*(\overrightarrow{a}) \in \mathcal{M}^*$ iff $P(\overrightarrow{a}) \in \mathcal{M}$;
- the interpretations of all constants and other predicates are the same as those in $\mathcal{M}$.

Proof: If Π is defined by the first-order sentence ϕ, then $\mathcal{K}$ is axiomatized by the first-order sentence $\phi \wedge \forall \overrightarrow{x}(P(\overrightarrow{x}) \leftrightarrow P^*(\overrightarrow{x}))$. $\Box$

The fixed-point formula $\psi(\Pi, P^*)^\infty(\overrightarrow{x}, P)$ simulates the program Π on all answer sets of Π. By induction on k, the following lemma holds.

Lemma 3. *Let Π be a program that has a single intensional predicate P, and $\mathcal{M}$ an answer set of $\tau(\Pi)$. Suppose that $\psi(\Pi, P^*)$ and $\mathcal{M}^*$ are constructed as above. Then, for any integer k and any $\overrightarrow{a}$, $P(\overrightarrow{a}) \in \mathcal{M}^k(\Pi)$ iff $\overrightarrow{a} \in \psi(\Pi, P^*)^k(\overrightarrow{x}, P)$.*

Proposition 3 shows that Π^* can simulate Π on the class of structures $\mathcal{K}$. Consequently, the answer set program Π can be simulated by the fixed-point formula $\widehat{\Pi^*}$ on $\mathcal{K}$. Together with Proposition 18, we can finally prove Theorem 2 in this paper.

We finish the proof of Theorem 2 as follows.

Proof of Theorem 2: We only need to prove $4 \Rightarrow 1$. From Lemma 2, it suffices to prove the case in which the program only contains a single intensional predicate. Let Π be such a program, which has a single intensional predicate P and is defined by a first-order sentence ϕ. Let $\mathcal{K} = \{\mathcal{M}^* \mid \mathcal{M} \in AS(\Pi)\}$. Then, $\mathcal{K}$ is first-order axiomatized by $\phi \wedge \forall \overrightarrow{x}(P(\overrightarrow{x}) \leftrightarrow P^*(\overrightarrow{x}))$. By Lemma 3, the fixed-point formula $\psi(\Pi, P^*)^\infty(\overrightarrow{x}, P)$ on $\mathcal{K}$ is defined by the formula $\phi^* \wedge P^*(x)$, where ϕ^* is obtained from ϕ by simultaneously replacing each occurrence of $P(\overrightarrow{t})$ with $P^*(\overrightarrow{t})$. Then, by Proposition 18, $\psi(\Pi, P^*)^\infty(\overrightarrow{x}, P)$ is bounded on $\mathcal{K}$. Again, by Lemma 3, Π is bounded. $\Box$

Corollary 19. *Boundedness is closed under program equivalence. That is, if two programs Π_1 and Π_2 are equivalent, then Π_1 is bounded iff Π_2 is bounded.*

Proof: Since Π_1 is bounded, then it is first-order definable. Therefore, Π_2 is first-order definable by the same sentence as Π_2 is equivalent to Π_1. It follows that Π_2 is bounded as well. $\Box$

7. Yet Another Translation from ASP to SMT

The progression semantics sheds new insights on first-order ASP from a theoretical point of view. For instance, Theorem 2 states that first-order definability of normal programs can be exactly captured by the notion of boundedness, which is defined based on the progression semantics. In this section, we show that the progression semantics on sheds new insights into first-order ASP from a practical point of view. More precisely, the progression semantics suggests a natural way to encode first-order normal ASP into Satisfiability Modulo Theories (SMT) [32], which are classical first-order theories enhanced with some modular theories to represent some components that cannot be easily handled in a logical setting, for instance, arithmetical formulas such as $2x - y \leq 10$.

This work follows the same research line of ordered completion, which translates a normal program to a first-order (SMT) sentence. The ordered completion approach initiates a new way of computing answer sets by grounding on the ordered completion (a first-order sentence) of programs instead of the first-order program itself, as most of the modern ASP solvers do. Some preliminary experimental results show that this new direction is promising as a first prototype performs reasonably good on the Hamiltonian program, particulary on big problem instances.

Inspired from the progression semantics (see Definition 7), we can define an alternative translation from normal ASP to first-order logic/first-order SMT. In fact, the progression semantics directly specifies a derivation order. Let us take a closer look at Definition 7 again. At the k stage of the progression, the accumulating structure $\mathcal{M}^k(\Pi)$ will be extended by some ground atoms, which are heads of some rules applicable at the k stage. Notice that the initial structure $\mathcal{M}^0(\Pi)$ is empty and the final structure $\mathcal{M}^\infty(\Pi)$ coincides with $\mathcal{M}$ itself if $\mathcal{M}$ is a stable model of the program. This means that for any ground atom α to be true in the stable model $\mathcal{M}$, it must be generated at a particular stage t in the progression, that is, there exists a rule r in the program that generates the atom α at the stage t in the progression. Furthermore, we need to precisely capture what it means by a rule r is exactly applicable/applied at a particular stage t. In fact, this is equivalent to

- the negative body of r is satisfied by the intended structure $\mathcal{M}$;
- the positive body of r is satisfied by the t-th evaluation stage $\mathcal{M}^t(\Pi)$;
- and the positive body of r is not satisfied by the $t-1$-th evaluation stage $\mathcal{M}^{t-1}(\Pi)$ (otherwise the rule r must be applied before),

which is further equivalent to (since $\mathcal{M}^k(\Pi)$ is monotonic)

- the negative body of r is satisfied by the intended structure $\mathcal{M}$;

- the positive body of r is satisfied by the intended structure $\mathcal{M}$;
- there exists at least one ground atom in the positive body of r, which is generated at the $t-1$-th stage, and all other ground atoms in the positive body of r must be generated even earlier.

Having explained our intuitions, we are now able to define the new translation from normal ASP to SMT. Again, for every intensional predicate P, we introduce an integer predicate n_P with the same arity.

Definition 13 (Progression based completion). *Let* Π *be a program. The* progression based completion *of* Π*, written* $PC(\Pi)$*, is the following sentence*

$$\widehat{\Pi} \wedge \bigwedge_{P \in \tau_{int}(\Pi)} \forall \overrightarrow{x}[P(\overrightarrow{x}) \rightarrow \bigvee_{1 \leq i \leq k} \exists \overrightarrow{y_i} \widehat{Body_i} \wedge n_P(\overrightarrow{x}) = succ(max(\{n_Q(\overrightarrow{z})\}))], \tag{14}$$

where

- *some notations are borrowed from Definition 3. Once again,* $Q(\overrightarrow{z})$ *ranges over all intensional atoms in the positive part of* $Body_i$*;*
- $succ$ *and* max *stand for the successor function and the maximum function in arithmetic respectively.*

Similar to the Clark's completion and ordered completion, progression based completion has to satisfy the program itself, namely $\widehat{(\Pi)}$. The differences are about the other way around, which is the justification part, stating that if a ground atom is in the answer set, then it has to be justified. Nevertheless, there are different understandings of justification. In the Clark's completion, it simply states that there is a rule in the program to support this ground atom, i.e., whose head is the ground atom and whose body is also satisfied by the structure. It turns out this kind of justification is not powerful enough to capture the answer set semantics. In ordered completion, justification is a bit stronger in the sense that not only there exists a rule to support the ground atom but also all ground atoms of that rule have to be justified earlier. As shown by Asunction et al. [3], this is enough to capture the answer set semantics. In progression based completion, justification is even stronger as it enforces a particular derivation order, which actually coincides with the derivation order obtained in the progression semantics. Intuitively, for a ground atom $P(\overrightarrow{a})$, $n_P(\overrightarrow{a})$ exactly represents its level in the progression of Π with respect to $\mathcal{M}$. Here, the arithmetical formula

$n_P(\overrightarrow{x}) = succ(max(\{n_Q(\overrightarrow{z})\}))$ means that the stage of the head atom $P(\overrightarrow{x})$ is exactly the maximal stage of the positive body atoms plus 1. That is, the rule is exactly triggered at this stage in the $max(\{n_Q(\overrightarrow{z})\})$-th evaluation stage.

Example 8. Let Π_R be the following program to check the reachability of a graph, whose edges are represented by the extensional predicate $Edge$.

$$\begin{aligned} &Reach(a) \\ &Reach(x) \leftarrow Reach(y), Edge(x, y), && (15) \\ &Reach(x) \leftarrow \text{not}\, Reach(x). && (16) \\ & && (17) \end{aligned}$$

Then, $OC'(\Pi)$ is

$$\widehat{\Pi_R} \wedge \forall x(R(x) \rightarrow x = a \vee \exists y[R(y) \wedge E(y, x) \wedge n_R(x) < n_R(y)],$$

while $PC(\Pi)$ is

$$\widehat{\Pi_R} \wedge \forall x(R(x) \rightarrow x = a \vee \exists y[R(y) \wedge E(y, x) \wedge n_R(x) = succ(n_R(y))].$$

Progression based completion and ordered completion share something in common. Both of them modify the justification part of the Clark's completion into a logically stronger formula by adding some extra statements about the derivation order of ground atoms. Nevertheless, ordered completion only requires the ground atoms are justified in some order, i.e., bodies should be justified earlier than heads, while progression based completion strictly enforces one particular derivation order on ground atoms, which coincides with the one obtained in the progression semantics. Thus, progression based completion yields a stronger version.

Proposition 20. *Let Π be a program. Then, $PC(\Pi) \models OC'(\Pi)$.*

Proof: This follows from the definitions since if $n_P(\overrightarrow{x}) = succ(max(\{n_Q(\overrightarrow{z})\}))$, then for all $n_Q(\overrightarrow{z})$, $n_Q(\overrightarrow{z}) < n_P(\overrightarrow{x})$. $\square$

Another difference between these two translations is the host SMT language. Ordered completion needs to use the built-in comparison operators $<$, while progression based completion needs to use two built-in functions, namely the maximum function and the successor function. However, many benchmark programs are linear in the sense that all rules in the program contain at most one intensional predicate. For linear programs, the maximum function is not needed in progression based completion. In this case, we believe that progression based has some computational advantages in implementations as the arity of the successor function is much less than the comparison operators $<$.

We end up this section by showing that, on finite structures, progression based completion, namely $PC(\Pi)$, exactly captures the stable model semantics as well.

Theorem 3. *Let* Π *be an extended program. Then, a finite* $\tau(\Pi)$ *structure is a stable model of* Π *if and only if it can be expanded to a model of* $PC(\Pi)$.

Proof: The "if" part follows from Proposition 1 and Proposition 20. We show the "only if" part. Let $\mathcal{M}$ be a stable model of Π. For a ground atom $P(\overrightarrow{a})$, we define $n_P(\overrightarrow{a})$ as its evaluation time in the progression. Now we show that $\mathcal{M}^+$, the structure obtained from $\mathcal{M}$ by expanding the interpretations on the integer predicates n_P as mentioned above, is a model of $PC(\Pi)$. First, $\mathcal{M}^+$ is a model of $\widehat{\Pi}$ since $\mathcal{M}$ is a model of $\widehat{\Pi}$. For any ground atom $P(\overrightarrow{a}) \in \mathcal{M}$, it must be justified by a rule r together with an assignment η at step $n_P(\overrightarrow{a})$ in the progression. Then, for any intensional ground atom $Q(\overrightarrow{b})$ in the positive body of $r\eta$, $Q(\overrightarrow{b})$ has to be justified before in the progression since $Q(\overrightarrow{b}) \in \mathcal{M}^{n_P(\overrightarrow{a})}(\Pi)$. In addition, there exists some $Q(\overrightarrow{b})$ in the positive body of $r\eta$ whose evaluation time is exactly $n_P(\overrightarrow{a}) - 1$. Otherwise, $P(\overrightarrow{a})$ should be justified earlier in the progression. Hence, $n_P(\overrightarrow{a}) = succ(max(\{Q(\overrightarrow{b}) \mid Q(\overrightarrow{b}) \in Pos(r\eta), Q \in \Omega_\Pi\}))$. This shows that $\mathcal{M}^+$ is a model of $PC(\Pi)$. $\Box$

8. Ongoing and Related Work

In this paper, we have restricted our discussions to first-order normal logic programs with rules only of the form (1) — the most important and fundamental fragment of first-order answer set programming. Driven by needs, normal logic programs are extended with some useful building blocks, including disjunctive heads, constraints and choice rules, existentially quantified heads, functions, nested expressions and so on. A problem arises how to extend the progression semantics for programs with those building blocks. Unfortunately, this seems to be a challenging task as the underlying principles of some building blocks are essentially different from the nature of the progression semantics. In the progression semantics, all intensional ground atoms in a stable model of a program must be justified at some step in the evaluation stage. Starting from the empty intensional database, each step justifies a set of ground atoms, which are the heads of all rules applicable at the current stage. Here, a ground rule is applicable if its positive body is satisfied by the current progression stage and its negative body is satisfied by the candidate structure itself.

Disjunctive logic programming is a natural extension of normal logic programming [22]. The head of a disjunctive rule is a disjunction of atoms, which represents a non-deterministic choice if the body is satisfied. The key point for extending the progression semantics for disjunctive programs is how to add the ground atoms when a ground rule is satisfied at a progression stage. There are two existing solutions. The first is to select a minimal hitting set of all heads of applicable rules (a collection of sets of ground atoms) as the justified ground atoms at this stage [38]. In this sense, there could be many different progression sequences with respect to a given disjunctive program and a candidate structure. The second approach is to collect the disjunctions of atoms (i.e., clauses) derivable at the current stage, and finally computes the minimal model of all collected clauses [36]. Both extensions are equivalent to the translation stable models semantics. Again, some interesting consequences follow from the progression semantics for disjunctive programs, e.g., a translation to SMT [38] and a characterization of first-order definability via boundedness [36].

It remains an open problem to further extend the progression semantics for incorporating other building blocks, for instance, aggregates, functions, existentially quantified heads and nested expressions. Work in this direction is worth pursuing as the progression semantics has some important theoretical and practical consequences. Incorporating extensional functions in the progression semantics is straightforward as their interpretations are fixed in the extensional database. However, this task seems not easy for intensional functions [7, 26]. Existentially quantified heads are of special interests as Datalog (ASP) enhanced with existentially quantified heads is able to capture some interesting fragments in description logics [23]. For incorporating existentially quantified heads in the progression semantics, again, the key point is how to add the ground atoms when a ground rule is satisfied at a progression stage. We leave these to our future investigations.

The notion of boundedness (see Definition 9) presents an exact characterization of the first-order definability for normal logic programs on arbitrary structures (see Theorem 2). Hence, it covers the notion of loop separability [12], a sufficient condition for first-order definability based on loop formulas. Roughly speaking, a first-order program is loop-separable iff all its loop patterns can be separated in some sense so that all its loop formulas can be finitely characterized. As a consequence, the stable models of a loop separable program can be defined by the classical models of its Clark's completion together with a finite set of loop formulas. Since boundedness is an if and only if condition for first-order definability, all loop-separable programs are bounded. In fact, this can also be observed from the proof (see Section 5 in [12]), which essentially shows that if a program is loop-separable, then we only need to take some loops with a bounded size into account. However, the converse does not hold. That is, there exists a bounded program that is not loop separable, e.g., the program Π_{flag} in Example 6. Nevertheless, loop separability is a syntactic condition, while boundedness is semantic. In addition, it is decidable to check whether a program is loop separable (see Theorem 3 in [12]), but checking boundedness is undecidable.

9. Conclusions

The main contributions of this paper are summarized as follows.

- We proposed a progression semantics (Definition 7) for first-order normal logic programs, and showed that it coincides with the translation stable model semantics (Theorem 1).
- As the progression semantics for ASP is a direct extension of the one for Datalog, it can be served as a foundation for comparing these two formalisms. Further evident from the progression semantics, Datalog is exactly the monotonic counterpart of first-order ASP.
- Consequently, many important concepts and techniques in Datalog can be applied to first-order ASP as well. As an example, we extended the notion of boundedness for ASP (Definition 9), and showed that it provides deeper understandings of the expressiveness of first-order ASP. In particular, we showed that boundedness coincides with both the notions of loop-free and recursion-free under program equivalence. More interestingly, it can be used to exactly capture first-order definability of first-order answer set programs on arbitrary structures (Theorem 2).
- We showed that the progression semantics directly suggests an alternative translation from first-order ASP to SMT, namely the progression base completion (Theorem 3), which yields a new way of implementing first-order ASP by utilizing SMT solvers.

Acknowledgement

We specially thank Fangzhen Lin for his initial inspiration on this work. Our results presented in this paper provides a complete answer to his conjecture on the equivalence between loop-free and first-order definability under answer set semantics. We also thank Heng Zhang for pointing out that the original proof of Proposition 11 in our conference paper is incomplete.

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Miklós Ajtai and Yuri Gurevich. Datalog vs first-order logic. *J. Comput. Syst. Sci.*, 49(3):562–588, 1994.

[3] Vernon Asuncion, Fangzhen Lin, Yan Zhang, and Yi Zhou. Ordered completion for first-order logic programs on finite structures. *Artif. Intell.*, 177-179:1–24, 2012.

[4] Vernon Asuncion, Yan Zhang, and Yi Zhou. Ordered completion for logic programs with aggregates. In *AAAI-2012*, pages 691–697, 2012.

[5] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[6] Michael Bartholomew and Joohyung Lee. Functional stable model semantics and answer set programming modulo theories. In *IJCAI 2013*, 2013.

[7] Michael Bartholomew and Joohyung Lee. On the stable model semantics for intensional functions. *TPLP*, 13(4-5):863–876, 2013.

[8] Michael Bartholomew, Joohyung Lee, and Yunsong Meng. First-order extension of the FLP stable model semantics via modified circumscription. In *IJCAI-2011*, pages 724–730, 2011.

[9] J. Barwise and Y. Moschovakis. Global inductive definability. *Journal of Symbolic Logic*, 43:521–534, 1978.

[10] Yin Chen, Fangzhen Lin, Yisong Wang, and Mingyi Zhang. First-order loop formulas for normal logic programs. In *KR'06*, pages 298–307, 2006.

[11] Yin Chen, Fangzhen Lin, Yisong Wang, and Mingyi Zhang. First-order loop formulas for normal logic programs. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 298–307, 2006.

[12] Yin Chen, Fangzhen Lin, Yan Zhang, and Yi Zhou. Loop-separable programs and their first-order definability. *Artif. Intell.*, 175(3-4):890–913, 2011.

[13] Keith L. Clark. Negation as failure. In *Logics and Databases*, pages 293–322, 1978.

[14] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[15] Marc Denecker, Yuliya Lierler, Miroslaw Truszczynski, and Joost Vennekens. A tarskian informal semantics for answer set programming. In *ICLP 2012*, pages 277–289, 2012.

[16] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

[17] François Fages. Consistency of clark's completion and existence of stable models. *Meth. of Logic in CS*, 1(1):51–60, 1994.

[18] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription. *Artif. Intell.*, 175(1):236–263, 2011.

[19] Haim Gaifman, Harry G. Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *J. ACM*, 40(3):683–713, 1993.

[20] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[21] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[22] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[23] Georg Gottlob, André Hernich, Clemens Kupke, and Thomas Lukasiewicz. Stable model semantics for guarded existential rules and description logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*, 2014.

[24] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.

[25] Joohyung Lee and Yunsong Meng. First-order stable model semantics and first-order loop formulas. *J. Artif. Intell. Res. (JAIR)*, 42:125–180, 2011.

[26] Vladimir Lifschitz. Logic programs with intensional functions. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*, 2012.

[27] Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.

[28] Fangzhen Lin and Yi Zhou. From answer set logic programming to circumscription via logic of GK. *Artif. Intell.*, 175(1):264–277, 2011.

[29] David Maier, Jeffrey D. Ullman, and Moshe Y. Vardi. On the foundations of the universal relation model. *ACM Trans. Database Syst.*, 9(2):283–308, 1984.

[30] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[31] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. and AI*, 25(3-4):241–273, 1999.

[32] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: from an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(t). *Journal of the ACM (JACM)*, 53(6):937–977, 1999.

[33] David Pearce and Agustín Valverde. Towards a first order equilibrium logic for nonmonotonic reasoning. In *JELIA'2004*, pages 147–160, 2004.

[34] Raymond Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[35] Yi-Dong Shen, Kewen Wang, Thomas Eiter, Michael Fink, Christoph Redl, Thomas Krennwallner, and Jun Deng. FLP answer set semantics without circular justifications for general logic programs. *Artif. Intell.*, 213:1–41, 2014.

[36] Heng Zhang and Yan Zhang. First-order expressibility and boundedness of disjunctive logic programs. In *IJCAI 2013*, 2013.

[37] Yan Zhang and Yi Zhou. On the progression semantics and boundedness of answer set programs. In *KR 2010*, 2010.

[38] Yi Zhou and Yan Zhang. Progression semantics for disjunctive logic programs. In *AAAI 2011*, 2011.