Artificial Intelligence 250 (2017) 58–79

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

A progression semantics for first-order logic programs

Yi Zhou[a,b,*], Yan Zhang[a,c]

[a] School of Computing, Engineering and Mathematics, Western Sydney University, Sydney, Australia
[b] School of Computer Science and Technology, TianJin University, TianJin, China
[c] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

ARTICLE INFO

Article history:
Received 21 August 2015
Received in revised form 19 May 2017
Accepted 6 June 2017
Available online 9 June 2017

Keywords:
Logic programming
Stable model
Progression
First-order

ABSTRACT

In this paper, we propose a progression semantics for first-order normal logic programs, and show that it is equivalent to the well-known stable model (answer set) semantics. The progressional definition sheds new insights into Answer Set Programming (ASP), for instance, its relationships to Datalog, First-Order Logic (FOL) and Satisfiability Modulo Theories (SMT). As an example, we extend the notion of boundedness in Datalog for ASP, and show that it coincides with the notions of recursion-freeness and loop-freeness 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 progressional definition 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 [6,20,33,34]. 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 [4,5,7,10,15,18,24,26,28,31,36,38,40,42].

A typical approach on this research line is to use a translation to another host language, e.g. second-order language [18] or circumscription [31]. 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 definition 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 progressional definition for first-order normal logic programs. Intuitively, this definition may be viewed as a generalization of the Gelfond–Lifschitz transformation [6] to the first-order case as well as a generalization of the progression semantics for Datalog [1,32]. Also, it shares some fundamental ideas with Reiter's semantics for default

* Corresponding author.
E-mail address: yzhou@scm.uws.edu.au (Y. Zhou).

http://dx.doi.org/10.1016/j.artint.2017.06.001
0004-3702/© 2017 Elsevier B.V. All rights reserved.

logic [37]. Simply enough, in the progressional definition, 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 progressional definition is equivalent to the general stable model semantics defined by $SM(\Pi)$.

The progressional definition 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 progressional definition 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 progressional definition, we are able to define the notion of boundedness for first-order answer set programs, which is critical for understanding the relationship between first-order ASP and classical first-order logic.

With the features of iterative and nonmonotonic reasoning, ASP is a representative rule-based formalism that is significantly different from classical logics. Nevertheless, ASP and classical logics are very closely related. Hence, the relationships between them have attracted a lot of attention in the literature [4,5,12–14,17,25,26,39]. Among them, a central topic is first-order definability, that is, what kind of answer set programs can be captured in classical first-order logic in the sense that their answer sets/stable models are exactly the classical models of a first-order sentence. Our notion of boundedness provides a complete answer for this. We prove that an answer set program is first-order definable if and only if it is bounded. Moreover, the notion of boundedness/first-order definability is also equivalent to two important syntactic notions of recursion-freeness and loop-freeness (tightness) 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 progressional definition is not only of theoretical interest 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 [4,5], it is logically stronger as it has less models.

This paper is organized as follows. Section 2 introduces necessary backgrounds. Section 3 proposes the progressional definition and shows that it is equivalent to the translational definition. Then, Section 4 extends the notion of boundedness in Datalog for ASP and shows that it is equivalent to the notions of recursion-freeness and loop-freeness under program equivalence. Section 5 further shows that boundedness exactly captures first-order definability of ASP. Section 6 reports a natural translation from first-order ASP to SMT based on the progressional definition. Finally, Section 7 discusses some related and ongoing works and Section 8 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 $\mathcal{A}$, $c_i^{\mathcal{A}}$ $(1 \le i \le m)$ is an element in A for every constant c_i in τ, and $P_j^{\mathcal{A}}$ $(1 \le j \le n)$ is a k-ary *relation* over A for every k-ary predicate P_j in τ. $P_j^{\mathcal{A}}$ is also called the *interpretation* 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 $\mathcal{A}$. For convenience, we also write $P(\overrightarrow{x})\eta \in \mathcal{A}$ for the fact that $\eta(\overrightarrow{x}) \in P^{\mathcal{A}}$. 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 is independent from the assignment of variables not in $\overrightarrow{x}$. In this case, we also write $\mathcal{A} \models \phi(\overrightarrow{x}/\overrightarrow{a})$ for convenience, where $\overrightarrow{a}$ is a tuple of elements in A. In particular, if ϕ is a sentence, then the satisfaction relation is independent of the assignment. In this case, we simply write $\mathcal{A} \models \phi$ for short. A *ground atom* in 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 $P(\overrightarrow{a}) \in \mathcal{A}$, or $\mathcal{A} \models P(\overrightarrow{a})$, to denote $\overrightarrow{a} \in P^{\mathcal{A}}$.

Given a structure $\mathcal{A}$ of τ, Q a predicate in τ and some ground atoms $Q(\overrightarrow{a_1}), \ldots, Q(\overrightarrow{a}_n)$, we use $\mathcal{A} \cup \{Q(\overrightarrow{a_1}), \ldots, Q(\overrightarrow{a_n})\}$ 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 $Q^{\mathcal{A}} \cup \{\overrightarrow{a_1}, \ldots, \overrightarrow{a_n}\}$.

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:

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

where α is a proper atom, β_i $(0 \le i \le m)$, and γ_j $(0 \le j \le l)$ 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 $\{\mathrm{not}\,\gamma_1, \ldots, \mathrm{not}\,\gamma_l\}$ 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 heads of some rules in Π are said to be *intensional*; all other predicates are said to be *extensional*.[1] 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 constant. 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.

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 $\mathrm{not}\,P(\overrightarrow{t}) \in Neg(r)$, $\mathcal{M} \not\models P(\overrightarrow{t})\eta$; $\mathcal{M}$ 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 translational stable model definition

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 as predicate Q_i in Ω_Π. Let r be a rule in Π of the form

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

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

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

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

$$\beta^*_1 \wedge \cdots \wedge \beta^*_m \wedge \neg\gamma_1 \wedge \cdots \wedge \neg\gamma_l \to \alpha^*,$$

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

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

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

$$\bigwedge_{1\le i\le n} \forall\overrightarrow{x}\,(Q^*_i(\overrightarrow{x}) \to Q_i(\overrightarrow{x})) \wedge \neg \bigwedge_{1\le i\le n} \forall\overrightarrow{x}\,(Q_i(\overrightarrow{x}) \to Q^*_i(\overrightarrow{x})).$$

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

[1] Here, we follow the notions used in Datalog to distinguish between intensional and extensional predicates. According to the definition, predicates defined by sets of facts in the program are also considered to be intensional.

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

This definition is originated from Lin and Shoham's work to translate normal logic programs under the answer set/stable model semantics into circumscription – a second order sentence [29]. Later on, a number of approaches have been proposed to generalize this work for translating richer forms of logic programs into fragments of second-order logic [8,18,28,31,36,42]. Restricted to normal logic programs, these translations are essentially equivalent.

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 significantly different from the classical first-order logic. However, these two types of formalisms are closely related. The relationships between answer set programming and classical logics have been one of the central topics in this area since its origin, and have attracted much attention in the literature [4,5,12–14,17,25,26,39].

Among them, one influential work is the completion approaches [14], 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. *Clark's Completion* (*completion* for short if clear from the context) of Π, denoted by $Comp(\Pi)$, is the following first-order sentence:

$$\bigwedge_{P\in\tau_{int}(\Pi)} \forall\overrightarrow{x}\,(P(\overrightarrow{x}) \leftrightarrow \bigvee_{1\le i\le k} \exists\overrightarrow{y_i}\,\widehat{Body_i}), \tag{2}$$

where

- $P(\overrightarrow{x}) \leftarrow Body_1, \ldots, P(\overrightarrow{x}) \leftarrow Body_k$ 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 was 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. The loop formula approach [12,26,30] showed that, together with so-called loop formulas, Clark's completion can exactly capture the stable model semantics. That is, a finite structure is a stable model of a program if and only if it is a classical model of the program's Clark's completion and all its loop formulas. Nevertheless, in the first-order case, there could be infinite number of loop formulas. In contrast, the ordered completion approach [4,5] introduces some extra comparison predicates to keep track of the derivation order so that the stable models can exactly be captured by ordered completion – a modified version of Clark's completion.

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

- For each intensional predicate P, the following sentences:

$$\forall\overrightarrow{x}\,(\bigvee_{1\le i\le k} \exists\overrightarrow{y_i}\,\widehat{Body_i} \rightarrow P(\overrightarrow{x})), \tag{3}$$

$$\begin{aligned}&\forall\overrightarrow{x}\,(P(\overrightarrow{x}) \rightarrow \bigvee_{1\le i\le k} \exists\overrightarrow{y_i}\,(\widehat{Body_i} \wedge \\ &\qquad \bigwedge_{Q(\overrightarrow{z})\in Pos_i,\, Q\in\Omega_\Pi} \le_{QP}(\overrightarrow{z},\overrightarrow{x}) \wedge \neg \le_{PQ}(\overrightarrow{x},\overrightarrow{z}))),\end{aligned} \tag{4}$$

  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 $Body_i$;
  - $\le_{QP}$ ($\le_{PQ}$) is a new predicate, and $\le_{QP}(\overrightarrow{z},\overrightarrow{x})$ intuitively means that the evaluation time of $Q(\overrightarrow{z})$ is less or equal than the one 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:

$$\bigwedge_{P,Q,R\in\Omega_\Pi} \forall\overrightarrow{x}\,\overrightarrow{y}\,\overrightarrow{z}\,(\le_{PQ}(\overrightarrow{x},\overrightarrow{y}) \wedge \le_{QR}(\overrightarrow{y},\overrightarrow{z}) \rightarrow \le_{PR}(\overrightarrow{x},\overrightarrow{z})). \tag{5}$$

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

Proposition 1 *(Theorem 1, [4]). 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 (5)) 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 (3) and formula (4), where the second the line of formula (4), i.e.,

$$\bigwedge_{Q(\overrightarrow{z})\in Pos_i, Q\in\Omega_\Pi} \leq_{QP}(\overrightarrow{z},\overrightarrow{x}) \wedge \neg \leq_{PQ}(\overrightarrow{x},\overrightarrow{z})$$

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 (5) 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 predicate, 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 heads 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 $Q_i^0, Q_i^1,\ldots,Q_i^k,\ldots$ is monotonic in the sense that $Q_i^k\subseteq Q_i^{k+1}$ for any k. Hence, a convergence always exists on finite structures.

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 5 can be extended for structures $\mathcal{D}$ with arbitrary cardinality by using transfinite iteration. 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 $\mathcal{D}\cup Q_1^\xi\cup\cdots\cup Q_n^\xi\models\widehat{Body}[\eta]\}$.

Again, a least fixed point always exists, which is called the intended value. Nevertheless, for simplicity and clarity, we mainly use the notion of evaluation stage proposed in Definition 5 unless stated otherwise. This should not affect the major conclusions drawn in this paper.

3. A progression definition for normal logic programs

In this section, we propose a progressional definition for first-order normal logic programs and show that it is equivalent to the translational stable model definition.

3.1. The progressional definition

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 $\Omega_\Pi = \{Q_1, \ldots, Q_n\}$ 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 restriction[2] 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;
- $\mathcal{M}^{k+1}(\Pi) = \mathcal{M}^k(\Pi) \cup \{Head(r)\eta \mid$ there exists $r = Q(\overrightarrow{x}) \leftarrow \beta_1, \ldots, \beta_m, \text{not}\,\gamma_1, \ldots, \text{not}\,\gamma_l \in \Pi$ 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 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:

$$\mathcal{M}^\infty(\Pi) = \bigcup_{0 \le j} \mathcal{M}^j(\Pi).$$

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

Similarly to Datalog, Definition 6 can be extended for structures with arbitrary cardinality by using transfinite iteration. 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 characterize the stable model semantics for first-order normal logic programs by using a progressional definition, similar to the one for Datalog.

Definition 7 *(Progressional stable model).* Let Π be a normal program and $\mathcal{M}$ a structure of $\tau(\Pi)$. $\mathcal{M}$ is called a *progressional stable model* of Π iff $\mathcal{M}^\infty(\Pi) = \mathcal{M}$.

We call this definition the *progressional definition*. Intuitively, a structure $\mathcal{M}$ is a progressional 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.

Example 1. Consider the following program Π_G:

$$GoShopping(x, y) \leftarrow Friends(x, y),$$

$$GoShopping(x, y) \leftarrow GoShopping(x, z), Likes(z, y), \text{not}\, Hate(x, y).$$

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

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 progressional stable model of Π_G. □

The progressional definition 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 progressional stable model (answer set) of the underlying program.

On the other hand, the progressional definition for normal programs can be viewed as an extension of the progressional definition for Datalog [1]. From a syntactic point of view, a Datalog program is a special case of normal program, where the negative bodies mention no intensional predicate. To address this difference semantically, one needs to handle the occurrences of intensional predicates in the negative bodies. For this purpose, we use several techniques. First, we guess a candidate structure on the signature of the program instead of just using a structure on 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 progressional stable model if it coincides with the structure obtained from the progression.

Our progressional definition also shares some fundamental ideas with Reiter's semantics for default logic [37]. 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 differences. First of all, in Reiter's default logic, what we guess is a theory, but in our progress definition, 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. Progressional stable models = translational stable models

We show that the progressional definition (i.e. Definition 7) is indeed equivalent to the translational definition (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 equivalent definition, and show that it is equivalent to both the progressional definition and the translational definition described above.

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

1. for every assignment η and every rule r of form (1) in Π, if for all i $(1 \le i \le m)$, $\beta_i\eta \in \mathcal{M}$ and for all j $(1 \le j \le l)$, $\gamma_j\eta \notin \mathcal{M}$, 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 $(1 \le i \le m)$, $\beta_i\eta \in \mathcal{M}'$ and for all j $(1 \le j \le l)$, $\gamma_j\eta \notin \mathcal{M}$, then $\alpha\eta \in \mathcal{M}'$.

We first show that this definition is equivalent to the translational definition. 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 $\mathcal{M} \models \exists\Omega^*_\Pi((\Omega^*_\Pi < \Omega_\Pi) \wedge \Pi^*)$. Suppose that there exists such an $\mathcal{M}'$, we construct n new relations in $\mathcal{M}$ on predicates $\Omega^*_\Pi = \{Q^*_1, \dots, Q^*_n\}$ corresponding to $\Omega_\Pi = \{Q_1, \dots, Q_n\}$ 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 definition is also equivalent to the progressional definition. 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 $(1 \le i \le m)$, $\beta_i\eta \in \mathcal{M}$ and for all j $(1 \le j \le l)$, $\gamma_j\eta \notin \mathcal{M}$ 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 $(1 \le i \le m)$, $\beta_i\eta \in \mathcal{M}^k(\Pi)$. 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 show that $\mathcal{M} \subseteq \mathcal{M}^\infty(\Pi)$. Otherwise, $\mathcal{M}^\infty(\Pi) \subset \mathcal{M}$. 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, $\mathcal{M}^\infty(\Pi) = \mathcal{M}$. □

4. Boundedness, recursion-freeness and loop-freeness

The progressional definition 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 definition, 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 progressional definition, we are able to define the boundedness notion for first-order ASP. Certainly, the basic idea is similar, i.e., we may require that 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 is actually equivalent to the syntactic notions of recursion-freeness and loop-freeness 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 predicate, while loop-free programs, also called tight program in the literature [17], are those programs without loops [12,30]. 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.

4.1. Boundedness for normal logic programs

We first review the notion of boundedness in Datalog, which had attracted much attention in the area of deductive databases [1,32].

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 progressional definition 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 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 model of Π.

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

Example 2. Consider the following program Π_V:

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

$$Visits(x, y) \leftarrow Visits(z, y), Attraction(y), \text{not}\ Busy(x). \tag{6}$$

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. □

Nevertheless, let $\Pi_{V'}$ be the program obtained from Π_V by replacing the rule (6) with the following one:

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

Then, $\Pi_{V'}$ is unbounded. One can construct a structure $\mathcal{M}$ with an infinite domain $a_0, a_1, \ldots, a_n, \ldots$, $Busy^{\mathcal{M}} = \emptyset$, $Interested^{\mathcal{M}} = \{(a_i, a_{i+1}) \mid i \geq 0\}$ and $Visits^{\mathcal{M}} = \{(a_i, a_j) \mid i < j\}$. It can be verified that $\mathcal{M}$ is a stable model of $\Pi_{V'}$ but there does not exist a number k such that $\mathcal{M}^\infty(\Pi_{V'}) = \mathcal{M}^k(\Pi_{V'})$.

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

Proposition 2. *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 3. *Checking boundedness for normal logic programs is undecidable.*

Proof. This assertion follows directly from Proposition 2 and the result that checking boundedness for Datalog programs is undecidable (see Theorem 2.5 in [19]). □

4.2. Recursion-freeness and loop-freeness

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

Recursion-freeness is an important notion in Datalog and it is well-studied in the Datalog community [1,2,32]. It can be lifted for first-order normal programs as follows.

Definition 10 *(Recursion-freeness).* A program is said to be *recursion-free* if no intensional predicate occurs in the positive body of any rule in the program.

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

Example 3. Consider the following program Π_{VP}:

$$Visits(x, y) \leftarrow Interested(x, y),$$

$$PossVisit(x, y) \leftarrow Attraction(y), \text{not}\ Visits(x, y).$$

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

It is generally considered that recursion is one of the most important features 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 4. *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 4 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 5. *Recursion-free programs are bounded.*

A closely related notion is loop-freeness.[3] For this purpose, we first review the concepts of loops for first-order normal programs [12]. Let Π be a program. 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 [12], under the stable model 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 stable models of a program by transforming it to propositional formulas [30].

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

The stable models of a loop-free program can be exactly captured by its Clark's completion [12,17].

Proposition 6 *([12]). Let Π be a loop-free program. Then, a $\tau(\Pi)$-structure $\mathcal{M}$ is a stable model of Π iff it is a model of $Comp(\Pi)$.*

Example 4. Consider programs Π_V and Π_{VP} once again in Examples 2 and 3 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 3 is loop-free obviously. $\square$

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

Proposition 7. *A recursion-free program must be loop-free.*

However, the converse of Proposition 7 does not hold in general. For example, the following program

$$Visits(x, y) \leftarrow Friends(x, y),$$

$$Friends(x, y) \leftarrow Likes(x, y), \text{not}\, Hate(x, y).$$

is loop-free but not recursion-free.

4.3. On the relationships among boundedness, recursion-freeness and loop-freeness

In this subsection, we shall show that the syntactic notions of recursion-freeness and loop-freeness 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 5 and Proposition 7. Corollary 5 states that all recursion-free programs must be bounded. We can extend this into the following result.

Proposition 8. *A loop-free program must be bounded.*

We leave the proof to the Appendix. Proposition 8 is an extension of Corollary 5 since all recursion-free programs are loop-free by Proposition 7.

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.

[3] 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.

Example 5. Let Π_{flag} be the following program:

$$\begin{aligned}&Reach(a)\\&Reach(x) \leftarrow Reach(y), Edge(x,y), flag \qquad (7)\\&Reach(x) \leftarrow \text{not}\, Reach(x)\\&\quad flag \leftarrow flag\end{aligned}$$

Clearly, Π_{flag} is not a recursion-free program as the positive body of rule (7) mentions the intensional predicate *Reach*. It is not a loop-free program either since rule (7) forms some loops. However, Π_{flag} is a bounded program. The reason is that the only recursion rule, i.e., rule (7), 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 (7) 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 (7) and the rule $flag \leftarrow flag$. In this sense, Π_{flag} is "semantically" recursion-free to some extent. The following proposition confirms that this kind of semantical recursion-freeness indeed can be implied by boundedness.

Proposition 9. *If a program is bounded, then it is equivalent to a recursion-free program.*

As the proof of Proposition 9 is a little tedious, although a similar result for Datalog programs holds straightforwardly. We leave it to the appendix.

It immediately follows from Proposition 9 and Proposition 7 that any bounded program can be equivalently transformed to a loop-free program.

Corollary 10. *If a program is bounded, then it is equivalent to a loop-free program.*

From Corollary 5, Proposition 7, Proposition 8, Proposition 9 and Corollary 10, we can see that the notions of boundedness, recursion-freeness and loop-freeness are highly connected. However, these results are not enough to justify the claim made in the beginning of this subsection that boundedness, recursion-freeness and loop-freeness 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 5. Nevertheless, for this purpose, more tools and techniques are needed.

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

5. First-order definability of answer set programs and boundedness

The relationship between first-order ASP and FOL is one of the most important topics in this area, and it has been well-studied in the literature [4,5,12–14,17,25,26,39]. 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 and existential quantifiers. Hence, it makes little sense to translate the full version of classical logic into a fragment of logic programming. Interestingly, some recent works 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 problem 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. [4] based on previous results in the literature (see Table 2 in [4]). Interestingly and surprisingly, the answer is depending on three factors, considering arbitrary structures or only finite structures, introducing auxiliary predicates or not, and 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 exist translations from normal ASP to FOL when relaxing any of the above two conditions. Loop formulas provide a translation from normal ASP to FOL on finite structures without introducing any new predicates but the translated results could be infinite [12]. Ordered completion is an alternative translation that guarantees the result to be finite but a polynomial number of extra predicates are needed [4].

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 class of loop-free programs (also called tight programs) [17]. It was 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 the so-called loop-separable programs [13]. 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 [30] and CMODELS [27].

However, it still remains an open problem 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 captures 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-freeness and loop-freeness under program equivalence.

5.1. *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 6. Let us consider Π_V again in Example 2. It can be verified that Π_V is defined by the following sentence:
$\forall xy(Visits(x,y) \leftrightarrow (Interested(x,y) \wedge \neg Busy(x) \vee \exists z(z \neq x \wedge Visits(z,y) \wedge Attraction(y) \wedge \neg Busy(x))))$. □

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

Proposition 11. *[13] If Π is a loop-free program, then $Comp(\Pi)$ defines Π on both arbitrary structures and finite structures.*

Consequently, by Proposition 7, a recursion-free program is defined by its Clark's completion as well.

5.2. *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 4 on arbitrary structures, which further corresponds to the syntactic notions of recursion-freeness and loop-freeness 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 9; $2 \Rightarrow 3$ follows straightforwardly from Proposition 7 and $3 \Rightarrow 4$ follows straightforwardly from Proposition 11. We only need to prove $4 \Rightarrow 1$ for Theorem 2. Nevertheless, the proof of this is rather technical and tedious. Hence, we leave it to the appendix.

Corollary 12. *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. □

6. Yet another translation from ASP to SMT

The progressional definition 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 progressional definition. In this section, we show that the progressional definition sheds new insights into first-order ASP from a practical point of view. More precisely, the progressional definition suggests a natural way to encode first-order normal ASP into Satisfiability Modulo Theories (SMT) [35], 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 ordered completion approach, which translates a normal program to a first-order (SMT) sentence. The ordered completion approach initiates a new way of computing stable models by grounding on ordered completion (a first-order sentence) of programs instead of the first-order program itself, as most of the modern ASP solvers do.

Inspired from the progressional definition (see Definition 7), we can define an alternative translation from normal ASP to first-order logic/first-order SMT. In fact, the progressional definition directly specifies a derivation order. Let us take a closer look at Definition 7 again. At the k-th 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-th stage. Notice that the intensional part of 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. 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 function 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

$$\widehat{\Pi} \wedge \bigwedge_{P \in \tau_{int}(\Pi)} \forall \overrightarrow{x} [P(\overrightarrow{x}) \rightarrow \bigvee_{1 \leq i \leq k} \exists \overrightarrow{y_i} \widehat{Body_i} \wedge n_P(\overrightarrow{x}) = succ(max(\{n_Q(\overrightarrow{z})\}))], \tag{8}$$

where

- some notations, including y_i, $Body_i$ and $\widehat{Body_i}$, are borrowed from Definitions 2 and 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 Clark's completion and ordered completion, progression based completion has to satisfy the program itself, namely $\widehat{\Pi}$. The main difference is the justification part, which states that if a ground atom is in the stable model, then it has to be justified. There are different understandings of justification. In 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 that this kind of justification is not powerful enough to capture the stable model semantics. In ordered completion, justification is a bit stronger in the sense that not only that there exists a rule to support the ground atom but also that all ground atoms of that rule have to be justified earlier. As shown by Asunction et al. [4], this is enough to capture the stable model 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. Intuitively, for a ground atom $P(\overrightarrow{a})$, $n_P(\overrightarrow{a})$ exactly represents its evaluation time 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 7. Let Π_R be the following program to check the reachability of a graph, whose edges are represented by the extensional predicate $Edge$.

$$Reach(a)$$

$$Reach(x) \leftarrow Reach(y), Edge(x, y),$$

$$Reach(x) \leftarrow \text{not}\, Reach(x).$$

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 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 that 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 progressional definition. Thus, progression based completion yields a stronger version.

Proposition 13. *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 $n_Q(\overrightarrow{z})$, $n_Q(\overrightarrow{z}) < n_P(\overrightarrow{x})$. □

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. Note that, for linear programs (in which all bodies of rules contain at most one intensional predicate) such as reachability, the maximum function is not needed in progression based completion.

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 13. 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 $PC(\Pi)$. □

7. 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 when extending the progressional definition 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 progressional definition. In the progressional definition, 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 progressional definition 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 [42]. 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 compute the minimal model of all collected clauses [39]. Both extensions are equivalent to the translational stable models definition. Again, some interesting consequences follow from the progressional definition for disjunctive programs, e.g., a translation to SMT [42] and a characterization of first-order definability via boundedness [39].

Constraints, choice rules and aggregates are essential building blocks for answer set programming, which are extensively used in most benchmark programs. Again, extending the progressional definition for them seems not easy as the underlying principles of the progressional definition and these building blocks are incompatible. For instance, while the progressional definition justifies the stable models step-by-step, the aggregate atoms are interpreted globally.

It remains an open problem to further extend the progressional definition for incorporating other building blocks, for instance, functions, existentially quantified heads and nested expressions. We expect that such a progression definition, if defined, should be equivalent to the stale model semantics on these richer formalisms [5,18,24,38]. Nevertheless, this seems to be a challenging task as the progression definition needs to be defined step-by-step. Work in this direction is worth pursuing as the progressional definition has some important theoretical and practical consequences. Incorporating extensional functions in the progressional definition is straightforward as their interpretations are fixed in the extensional database. However, this task seems not easy for intensional functions [8,9,28]. 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 progressional definition, 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.

Naive extensions to richer syntactic classes do not work. New notions and techniques have to be developed. For instance, only boundedness itself cannot make a difference between first-order disjunctive logic programs and classical first-order logic. Recently, we coined a new term called "choice-boundedness" for this purpose [41]. Also, we found that the progression definition may work for certain aggregates such as convex aggregates [3]. We consider this to be one of the most important future directions as the progression definition can help us understanding first-order answer set programming much more deeply, from not only a theoretical but also a practical point of view.

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 [13], 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 equivalent to the condition of first-order definability, all loop-separable programs are bounded. In fact, this can also be observed from the proof (see Section 5 in [13]), 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 5. 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 [13]), but checking boundedness is undecidable. Nevertheless, given a fixed number k, checking k-boundedness should be decidable. This might help us to rewrite some logic programs into loop-free ones so that they can be solved more easily.

Another important future direction is to apply our theoretical results into practices, for instance, to develop a new ASP solver based on the translation into SMT proposed in Section 6. Alternatively, we may utilize some notions and techniques developed in this paper, e.g., boundedness and k-boundedness, for solving certain subclasses of answer set programs more easily.

8. Conclusions

The main contributions of this paper are summarized as follows:

- We extended the progression semantics for Datalog into a progression definition for first-order normal logic programming and showed that it is equivalent to the well-known stable model semantics. As a consequence, many important and useful notions and techniques in Datalog can be lifted for first-order ASP.
- We introduced a notion of boundedness for first-order ASP and showed that it coincides with the notions of recursion-freeness and loop-freeness under program equivalence. More interestingly, we showed that these notions exactly capture first-order definability of ASP for normal logic programs. This clearly clarifies the expressive power of the intersection between first-order ASP and classical First-Order Logic (FOL), both from a syntactic and a semantic point of view. Syntactically, it is well known that recursion-free and loop-free logic programs are first-order definable [12,17,30]. Our result proved a long standing conjecture that this assertion holds the other way around. That is, a first-order definable logic program is essentially equivalent to a recursion-free (loop-free) one. Semantically, our result showed that boundedness draws a clear boundary between first-order definable and indefinable normal logic programs.
- The progression semantics naturally suggests a new translation from first-order ASP to Satisfiability Modulo Theories (SMT) by introducing new predicates. This translation is of practical relevance since it is has less models than so-called ordered completion [4].

To conclude, the progression definition sheds new insights into first-order Answer Set Programming (ASP), including its deep connections and relationships to Datalog, FOL and SMT.

Acknowledgements

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-freeness and first-order definability under stable model semantics. We thank Heng Zhang for pointing out that the original proof of Proposition 9 in our conference paper is incomplete. We are also grateful to the anonymous reviewers for their valuable comments.

Appendix. Proofs of Proposition 9 and Theorem 2

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.

Proposition 8. *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:

$$Q(\overrightarrow{x}) \leftarrow \beta_1, \ldots, \beta_m, \text{not}\, \gamma_1, \ldots, \text{not}\, \gamma_l, \tag{9}$$

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

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 (9), for each β_i $(1 \le i \le m)$, 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 $\overrightarrow{b}$ is 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.

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. □

Proposition 9. *If a program is bounded, then it is equivalent to a recursion-free program.*

We 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 normalized program and t a number. We define a program Π_t inductively as follows. Firstly, set $\Pi_1 = \Pi$. 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

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

and for all i $(1 \le i \le m)$, 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:

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

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. Clearly, Π_t is normalized if Π is normalized.

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 $(a_1, \ldots, a_n) \in Q^t(\mathcal{M})$, where Q is an intensional predicate of Π. If the evaluation time of $Q(a_1, \ldots, a_n)$ is less than t, then $Q(a_1, \ldots, a_n) \in \mathcal{M}^1(\Pi_t)$ by induction assumption. If the evaluation time of $Q(a_1, \ldots, a_n)$ 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) $\overrightarrow{x_Q}\eta = (a_1, \ldots, a_n)$, (b) for all i $(1 \le i \le m)$, $\beta_i\eta \in \mathcal{M}^{t-1}(\Pi)$, and (c) for all j $(1 \le j \le l)$, $\gamma_j\eta \notin \mathcal{M}$. By induction assumption, for all i $(1 \le i \le m)$, $\beta_i\eta \in \mathcal{M}^1(\Pi_{t-1})$. 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).

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

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

We now prove $\mathcal{M}^1(\Pi_t) \subseteq \mathcal{M}^t(\Pi)$. Suppose that $Q(a_1, \ldots, a_n)$ 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 $Head(r^*)\eta = Q(a_1, \ldots, a_n)$. Suppose that r^* has the form

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

where $r \in \Pi$, $r_i \in \Pi_{t-1}$, 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. $\square$

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

Lemma 2. *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 $Q(a_1, \ldots, a_n)$ 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 $Head(r^*)\eta = Q(a_1, \ldots, a_n)$ and its body can be applied at the current evaluation stage. Suppose that r^* has the form

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

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 a stable model of Π_k, then $\mathcal{M}$ is a stable model 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. □

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

$$\leftarrow \beta_1, \dots, \beta_m, \text{not}\,\gamma_1, \dots, \text{not}\,\gamma_l. \tag{10}$$

Let r be a constraint of the form (10). By $\widehat{r}$, we denote the first-order formula

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

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 $c \in C$, $\mathcal{M} \models \widehat{c}$.

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

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

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

Lemma 3. *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 $\mathcal{M} = \mathcal{M}^1(\Pi) = \mathcal{M}^2(\Pi) = \mathcal{M}^3(\Pi) = \dots = \mathcal{M}^{\infty}(\Pi)$, 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. □

Next, we show that recursion-free programs with constraints can always be equivalently transformed into recursion-free programs. Let Π be a recursion-free program and c a constraint of the form (10). Suppose that all the rules in Π whose head is $\beta_i, 1 \le i \le m$ are

$$\beta_i \leftarrow Body_{i1},$$

$$\dots,$$

$$\beta_i \leftarrow Body_{ib_i},$$

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

$$\gamma_j \leftarrow Body_{j1},$$
$$\ldots,$$
$$\gamma_j \leftarrow Body_{jc_j}.$$

Here, $Body_{ik}, 1 \le k \le b_i$ ($Body_{jl}, 1 \le l \le c_j$) is a body without positive intensional atoms since Π is a recursion-free program.

By $\Pi \oplus c$, we denote the program obtained from Π by replacing each $\beta_i \leftarrow Body_k, i1 \le k \le ib_i$ with the following set (*) of rules

$$\beta_i \leftarrow \text{not}\,\beta_1, Body_k,$$
$$\ldots,$$
$$\beta_i \leftarrow \text{not}\,\beta_m, Body_k,$$
$$\beta_i \leftarrow Body_{11}, Body_k,$$
$$\ldots,$$
$$\beta_i \leftarrow Body_{1c_1}, Body_k,$$
$$\beta_i \leftarrow Body_{21}, Body_k,$$
$$\ldots,$$
$$\beta_i \leftarrow Body_{2c_2}, Body_k,$$
$$\ldots,$$
$$\beta_i \leftarrow Body_{l1}, Body_k,$$
$$\ldots,$$
$$\beta_i \leftarrow Body_{lc_l}, Body_k.$$

Lemma 4. *If Π is a recursion-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 $\beta_i, 1 \le i \le m$ such that $\mathcal{M} \not\models \beta_i\eta$. In this case, consider any $\beta_j\eta, 1 \le j \ne i \le m$. 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 $\beta_j \leftarrow Body_{jk}, 1 \le jk \le jb_j$ such that $\mathcal{M} \models Body_{jk}\eta$. Then, $\mathcal{M} \models [Body_{jk} \cup \{\neg\beta_i\}]\eta$. Then, $\beta_j\eta$ is in $\mathcal{M}^1(\Pi \oplus c)$ since it is justified by the rule $\beta_j \leftarrow \text{not}\beta_i, Body_{jk}$ in the rule set (*).

Case 2: There exists $\gamma_j, 1 \le j \le l$ such that $\mathcal{M} \models \gamma_j\eta$. In this case, there exists a rule of the form $\gamma_j \leftarrow Body_{jk}, 1 \le k \le c_j$ such that $\mathcal{M} \models Body_{jk}\eta$. Similarly, consider any $\beta_i\eta, 1 \le i \le m$. 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 $\beta_i \leftarrow Body_{is}, 1 \le is \le ib_i$ such that $\mathcal{M} \models Body_{is}\eta$. Then, $\beta_i\eta$ is in $\mathcal{M}^1(\Pi \oplus c)$ since it is justified by the rule $\beta_i \leftarrow Body_{jk}, Body_{is}$ 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 recursion-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 β_i, $1 \le i \le m$, $\beta_i\eta \in \mathcal{M}$ and for all γ_j, $1 \le j \le l$, $\gamma_j\eta \notin \mathcal{M}$. 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 $\beta_1 \leftarrow \text{not}\beta_j, Body_k$ since $\beta_j\eta \in \mathcal{M}$. Suppose that it is of the form $\beta_1 \leftarrow Body_{ij}, Body_k$. Then, $\mathcal{M} \models Body_{ij}\eta$. It follows that $\mathcal{M} \models \gamma_i\eta$ because of the rule $\gamma_i \leftarrow Body_{ij}$ 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 $\beta_i\eta, 1 \le i \le m$. Since it is in $\mathcal{M}^1(\Pi)$, there exists a rule of the form $\beta_i \leftarrow Body_k$ such that $\mathcal{M} \models Body_k\eta$. Hence, $\mathcal{M} \models [Body_k \cup \{\neg\beta_i\}]\eta$

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 $\beta_i \leftarrow \text{not}\,\beta_i, Body_k$. 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 $\Pi \oplus c$. □

Corollary 10. *If Π is a recursion-free program and C a set of constraints, then $\Pi \cup C$ is equivalent to a recursion-free program.*

Proof. Note that for eliminating constraints $\leftarrow \text{not}\,\gamma_1, \dots, \text{not}\,\gamma_l$ without positive body, one only needs to convert it to a non-recursive rule $\gamma_1 \leftarrow \text{not}\,\gamma_1, \dots, \text{not}\,\gamma_l$. The assertion follows from Lemma 4 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. □

Finally, we are able to prove Proposition 9.

Proof of Proposition 9. Proposition 9 follows from Lemmas 2 and 3 and Corollary 10. □

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.*

Here, we 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 an n-ary relation to an n-ary relation on $\mathsf{Dom}(\mathcal{A})$:

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

Starting from the empty set, Φ gives rise to a sequence of n-ary relations as follows:

$$\Phi^0(\overrightarrow{x}, P) = \emptyset;$$
$$\Phi^t(\overrightarrow{x}, P) = \Phi(\bigcup_{r<t} \Phi^r(\overrightarrow{x}, P)).$$

Since P only occurs positively in ϕ, the sequence $\Phi^1(\overrightarrow{x}, P), \dots, \Phi^t(\overrightarrow{x}, P), \dots$ always increases. Thus, there exists a least ordinal k such that $\Phi^k(\overrightarrow{x}, P) = \Phi^t(\overrightarrow{x}, P) = \Phi^\infty(\overrightarrow{x}, P)$, 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 use $\Phi^\infty(\overrightarrow{x}, P)$ to denote the least fixed point of Φ.

This defines a corresponding iterative formula $\phi^t(\overrightarrow{x}, P)$ in the sense that for any $\overrightarrow{a} \in \mathsf{Dom}(\mathcal{A})^n$, $\mathcal{A} \models \phi^t(\overrightarrow{x}/\overrightarrow{a}, P)$ iff $\overrightarrow{a} \in \Phi^t(\overrightarrow{x}, P)$; $\mathcal{A} \models \phi^\infty(\overrightarrow{x}/\overrightarrow{a}, P)$ iff $\overrightarrow{a} \in \phi^\infty(\overrightarrow{x}, P)$. We write $\phi^\infty(\overrightarrow{x}, P)$ (ϕ^∞ for short) to denote the *the least fixed-point formula* obtained from $\phi(\overrightarrow{x}, P)$.

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 $\overrightarrow{a} \in \mathsf{Dom}(\mathcal{A})^n$,

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

[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 $\vee$.

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 $\overrightarrow{a} \in \mathrm{Dom}(\mathcal{A})^n$, $\mathcal{A} \models \phi^{\infty}(\overrightarrow{x}/\overrightarrow{a}, P)$ iff $\mathcal{A} \models \phi^{k}(\overrightarrow{x}/\overrightarrow{a}, P)$.

Barwise and Moschovakis [11] revealed the important correspondence between definability and boundedness on arbitrary structures in least fixed-point logic.

Lemma 5. *[11] 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 Lemma 5. 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.

Lemma 6. *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 $\mathcal{M}^S$ 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. $\square$

Lemma 6 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

- $\mathrm{Dom}(\mathcal{M}^*) = \mathrm{Dom}(\mathcal{M})$;
- 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 $\phi \wedge \forall \overrightarrow{x} (P(\overrightarrow{x}) \leftrightarrow P^*(\overrightarrow{x}))$. $\square$

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

Lemma 7. *Let Π be a program that has a single intensional predicate P, and $\mathcal{M}$ a stable model 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 $\overrightarrow{a} \in \psi(\Pi, P^*)^k(\overrightarrow{x}, P)$.*

Lemma 7 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 Lemma 5, we can finally prove Theorem 2 in this paper.

We finish the proof of Theorem 2 as follows.

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

Proof of Theorem 2. We only need to prove $4 \Rightarrow 1$. From Lemma 6, 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 7, 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 Lemma 5, $\psi(\Pi, P^*)^\infty(\overrightarrow{x}, P)$ is bounded on $\mathcal{K}$. Again, by Lemma 7, Π is bounded. □

References

[1] Serge Abiteboul, Richard Hull, Victor Vianu, Foundations of Databases, Addison-Wesley, 1995.
[2] Miklós Ajtai, Yuri Gurevich, Datalog vs first-order logic, J. Comput. Syst. Sci. 49 (3) (1994) 562–588.
[3] Vernon Asuncion, Yin Chen, Yan Zhang, Yi Zhou, Ordered completion for logic programs with aggregates, Artif. Intell. 224 (2015) 72–102.
[4] Vernon Asuncion, Fangzhen Lin, Yan Zhang, Yi Zhou, Ordered completion for first-order logic programs on finite structures, Artif. Intell. 177–179 (2012) 1–24.
[5] Vernon Asuncion, Yan Zhang, Yi Zhou, Ordered completion for logic programs with aggregates, in: AAAI-2012, 2012, pp. 691–697.
[6] Chitta Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003.
[7] Michael Bartholomew, Joohyung Lee, Functional stable model semantics and answer set programming modulo theories, in: IJCAI 2013, 2013.
[8] Michael Bartholomew, Joohyung Lee, On the stable model semantics for intensional functions, Theory Pract. Log. Program. 13 (4–5) (2013) 863–876.
[9] Michael Bartholomew, Joohyung Lee, Stable models of multi-valued formulas: partial versus total functions, in: Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20–24, 2014, 2014.
[10] Michael Bartholomew, Joohyung Lee, Yunsong Meng, First-order extension of the FLP stable model semantics via modified circumscription, in: IJCAI-2011, 2011, pp. 724–730.
[11] J. Barwise, Y. Moschovakis, Global inductive definability, J. Symb. Log. 43 (1978) 521–534.
[12] Yin Chen, Fangzhen Lin, Yisong Wang, 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, 2006, pp. 298–307.
[13] Yin Chen, Fangzhen Lin, Yan Zhang, Yi Zhou, Loop-separable programs and their first-order definability, Artif. Intell. 175 (3–4) (2011) 890–913.
[14] Keith L. Clark, Negation as failure, in: Logics and Databases, 1978, pp. 293–322.
[15] Marc Denecker, Yuliya Lierler, Miroslaw Truszczynski, Joost Vennekens, A Tarskian informal semantics for answer set programming, in: ICLP 2012, 2012, pp. 277–289.
[16] Heinz-Dieter Ebbinghaus, 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) (1994) 51–60.
[18] Paolo Ferraris, Joohyung Lee, Vladimir Lifschitz, Stable models and circumscription, Artif. Intell. 175 (1) (2011) 236–263.
[19] Haim Gaifman, Harry G. Mairson, Yehoshua Sagiv, Moshe Y. Vardi, Undecidable optimization problems for database logic programs, J. ACM 40 (3) (1993) 683–713.
[20] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012.
[21] Michael Gelfond, Vladimir Lifschitz, The stable model semantics for logic programming, in: Proceedings of International Logic Programming Conference and Symposium, MIT Press, 1988, pp. 1070–1080.
[22] Michael Gelfond, Vladimir Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (3/4) (1991) 365–386.
[23] Georg Gottlob, André Hernich, Clemens Kupke, 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] Amelia Harrison, Vladimir Lifschitz, David Pearce, Agustín Valverde, Infinitary equilibrium logic and strongly equivalent logic programs, Artif. Intell. 246 (2017) 22–33.
[25] Tomi Janhunen, Ilkka Niemela, Compact translations of non-disjunctive answer set programs to propositional clauses, in: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning – Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, 2011, pp. 111–130.
[26] Joohyung Lee, Yunsong Meng, First-order stable model semantics and first-order loop formulas, J. Artif. Intell. Res. 42 (2011) 125–180.
[27] Yuliya Lierler, cmodels – SAT-based disjunctive answer set solver, in: Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5–8, 2005, Proceedings, 2005, pp. 447–451.
[28] 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, Springer, 2012.
[29] Fangzhen Lin, Yoav Shoham, A logic of knowledge and justified assumptions, Artif. Intell. 57 (2–3) (1992) 271–289.
[30] Fangzhen Lin, Yuting Zhao, ASSAT: computing answer sets of a logic program by SAT solvers, Artif. Intell. 157 (1–2) (2004) 115–137.
[31] Fangzhen Lin, Yi Zhou, From answer set logic programming to circumscription via logic of GK, Artif. Intell. 175 (1) (2011) 264–277.
[32] David Maier, Jeffrey D. Ullman, Moshe Y. Vardi, On the foundations of the universal relation model, ACM Trans. Database Syst. 9 (2) (1984) 283–308.
[33] Victor W. Marek, Miroslaw Truszczynski, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: A 25-Year Perspective, Springer-Verlag, 1999, pp. 375–398.
[34] Ilkka Niemelä, Logic programs with stable model semantics as a constraint programming paradigm, Ann. Math. Artif. Intell. 25 (3–4) (1999) 241–273.
[35] Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli, Solving SAT and SAT modulo theories: from an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(t), J. ACM 53 (6) (1999) 937–977.
[36] David Pearce, Agustín Valverde, Towards a first order equilibrium logic for nonmonotonic reasoning, in: JELIA'2004, 2004, pp. 147–160.
[37] Raymond Reiter, A logic for default reasoning, Artif. Intell. 13 (1–2) (1980) 81–132.
[38] Yi-Dong Shen, Kewen Wang, Thomas Eiter, Michael Fink, Christoph Redl, Thomas Krennwallner, Jun Deng, FLP answer set semantics without circular justifications for general logic programs, Artif. Intell. 213 (2014) 1–41.
[39] Heng Zhang, Yan Zhang, First-order expressibility and boundedness of disjunctive logic programs, in: IJCAI 2013, 2013.
[40] Yan Zhang, Yi Zhou, On the progression semantics and boundedness of answer set programs, in: KR 2010, 2010.
[41] Yi Zhou, First-order disjunctive logic programming vs normal logic programming, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015, 2015, pp. 3292–3298.
[42] Yi Zhou, Yan Zhang, Progression semantics for disjunctive logic programs, in: AAAI 2011, 2011.