A Decidable Class of Groundable Formulas in the General Theory of Stable Models

Michael Bartholomew and Joohyung Lee
School of Computing, Informatics and Decision System Engineering
Arizona State University, Tempe, AZ, USA
{mjbartho, joolee}@asu.edu

Abstract

We present a decidable class of first-order formulas in the general theory of stable models that can be instantiated even in the presence of function constants. The notion of an *argument-restricted formula* presented here is a natural generalization of both the notion of an argument-restricted program and the notion of a semi-safe sentence that have been studied in different contexts. Based on this new notion, we extend the notion of safety defined by Cabalar, Pearce and Valverde to arbitrary formulas that allow function constants, and apply the result to RASPL-1 programs and programs with arbitrary aggregates, ensuring finite groundability of those programs in the presence of function constants. We also show that under a certain syntactic condition, argument-restricted formulas can be turned into argument-restricted programs.

Introduction

Grounding is a widely used approach that allows us to identify the Herbrand models[1] of a first-order signature with the models of the corresponding propositional signature. For instance, assuming the signature $\{p/1, q/1, a, b\}$, grounding turns

$$p(a) \wedge q(b) \wedge \forall x(p(x) \rightarrow q(x)) \tag{1}$$

into

$$p(a) \wedge q(b) \wedge (p(a) \rightarrow q(a)) \wedge (p(b) \rightarrow q(b)) \tag{2}$$

by replacing the universal quantification with multiple conjunctions in which x ranges over the Herbrand universe $\{a, b\}$. The Herbrand models of (1) are the same as the Herbrand models of (2). Containing no variables, formula (2) can also be viewed as a propositional formula of the signature $\{p(a), p(b), q(a), q(b)\}$ (i.e., "$p(a)$" is an atom in propositional logic). The (propositional) models of the propositional signature can be identified with the Herbrand models of the first-order signature.

[1] Recall that an *Herbrand interpretation* of a signature σ (containing at least one object constant) is an interpretation of σ such that its universe is the set of all ground terms of σ, and every ground term represents itself. An Herbrand interpretation can be identified with the set of ground atoms (not containing equality) to which it assigns the value *true*.

However, this approach does not work in the presence of arbitrary function constants. For instance, one cannot turn

$$p(a) \wedge q(b) \wedge \forall x(p(x) \rightarrow q(f(x))) \tag{3}$$

into multiple conjunctions over the Herbrand universe because the universe is infinite: $\{a, b, f(a), f(b), f(f(a)), f(f(b)), \dots\}$.

On the other hand, in answer set programming, where rules are used instead of formulas, function constants are allowed to some degree. In order to ensure finite grounding, answer set solvers impose syntactic conditions on the input languages, such as ω-restricted (Syrjänen 2001; 2004), λ-restricted (Gebser *et al.* 2007b), or finite domain programs (Calimeri *et al.* 2008). Recently Lierler and Lifschitz [2009] defined the most general decidable class of finitely groundable programs, called argument-restricted programs, that covers the syntactic conditions above.

In this paper, we extend the notion of an argument-restricted program to the syntax of an arbitrary formula, whose stable models are defined in (Ferraris *et al.* 2007; 2010). Unlike in first-order logic, argument-restricted formulas under the stable model semantics can be grounded even in the presence of function constants. For instance, formula (3) is argument-restricted, and the results of this paper tell us that the Herbrand *stable* models of formula (3) can be identified with the Herbrand *stable* models of

$$p(a) \wedge q(b) \wedge (p(a) \rightarrow q(f(a))) \wedge (p(b) \rightarrow q(f(b))). \tag{4}$$

It turns out that the notion of an argument-restricted formula can also be viewed as an extension of semi-safety, which is defined in (Lee *et al.* 2009; Cabalar *et al.* 2009) for formulas containing no function constants other than object constants. Based on this new notion, we show that the notion of safety that was generalized to arbitrary formulas (Lee *et al.* 2008b; Cabalar *et al.* 2009) can be further generalized to allow function constants.

We apply the results to justify finite groundability of programs that contain aggregates, whose semantics are as given in (Lee *et al.* 2008a) and (Faber *et al.* 2004). Lee *et al.* [2008a] showed that the meaning of choice rules and counting aggregates in answer set programming can be understood by identifying these constructs as shorthand for first-order formulas under the stable model semantics. For instance, the "FOL-representation" of the RASPL-1 rule

$$s \leftarrow \{x : p(x)\}\, 1 \tag{5}$$

is the formula

$$\neg\exists xy(p(x) \wedge p(y) \wedge x \neq y) \rightarrow s. \quad (6)$$

The counterpart of (5) in the language defined in (Faber *et al.* 2004) is

$$s \leftarrow \#count\{x : p(x)\} \leq 1. \quad (7)$$

In that language, the meaning of $\#count\{x : p(x)\} \leq 1$ is understood by instantiating the expression $\{x : p(x)\}$ over the Herbrand universe, and then applying an interpretation on the ground set. Obviously this process results in an infinite ground set once the program contains function constants of positive arity as in the following:

$$\begin{array}{l} p(f(x)) \leftarrow q(x) \\ q(x) \leftarrow p(x), r(x) \\ p(a) \qquad r(a) \qquad r(f(a)) \\ s \leftarrow \#count\{x : p(x)\} \leq 1 \end{array} \quad (8)$$

This program cannot be handled by DLV, which implements a large subset of the language defined in (Faber *et al.* 2004). On the other hand, the recent system DLV-COMPLEX, an extension of DLV with function constants, returns one answer set $\{p(a), p(f(a)), p(f(f(a))), q(a), q(f(a)), r(a), r(f(a))\}$ for the program above. While the system is based on the theory from (Calimeri *et al.* 2008) that ensures finite groundability of an answer set program, that paper does not consider aggregates. To the best of our knowledge, no justification has been provided regarding finite groundability of programs such as (8).

We also show that under a certain syntactic condition, argument-restricted formulas can be turned into argument-restricted programs. This may be of interest in applications of system F2LP (Lee and Palla 2009), which computes answer sets of arbitrary formulas under certain conditions, by reducing formulas to logic programs.

The rest of the paper is organized as follows. In the next section, we review the definitions of argument-restricted programs, answer sets of first-order formulas, and semi-safety and safety in the absence of function constants. In Section "Argument-Restricted Formulas," we present the definition of argument-restricted formulas, and show that such formulas have the *small predicate property*. We extend the notion of safety to allow function constants by taking into account argument-restrictedness; we show that safe sentences can be grounded w.r.t. a finite domain, and their answer sets are not affected by enlarging the domain. In Section "Deciding Argument-Restricted Formulas," we show that the class of argument-restricted formulas is decidable. In the next three sections, we apply the concept of argument-restricted formulas and safety to RASPL-1 programs and FLP programs, and extend the notion of argument-restrictedness and safety to allow extensional predicates. In the last two sections, we show that under a certain condition, argument-restricted formulas can be turned into argument-restricted programs and how the similar results hold for λ-restricted formulas, an extension of λ-restricted programs.

Preliminaries

A *signature* consists of *function constants* and *predicate constants*. Function constants of arity 0 are called *object constants*.

We distinguish between atoms and atomic formulas as follows: an *atom* of a signature σ is an n-ary predicate constant followed by a list of n terms; *atomic formulas* of σ are atoms of σ, equalities between terms of σ, and the 0-place connective $\bot$.

Review: Argument-Restricted Programs

A *program* considered in (Lierler and Lifschitz 2009) is a finite list of *rules* of the form

$$A_1; \ldots; A_k \leftarrow A_{k+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n, \quad (9)$$

($0 \leq k \leq m \leq n$), where each A_i is an atom, possibly containing function constants of positive arity. A program is called *safe* if every variable occurring in a rule of Π occurs also in the positive body of that rule.

We follow the terminology and the notations used in (Lierler and Lifschitz 2009). Given an atom $p(t_1, \ldots, t_n)$, expression $p(t_1, \ldots, t_n)^0$ represents its predicate constant p, and $p(t_1, \ldots, t_n)^i$, where $1 \leq i \leq n$, represents its argument term t_i. As in (Calimeri *et al.* 2008), an *argument* is an expression of the form $p[i]$, where i is one of the argument positions $1, \ldots, n$. The *depth* of a variable x in a term t that contains x, denoted by $d(x, t)$, is defined recursively as follows:

$$d(x,t) = \begin{cases} 0, & \text{if } t \text{ is } x; \\ 1 + \max(\{d(x,t_i) \mid t_i \text{ contains } x\}), & \text{if } t \text{ is } f(t_1, \ldots, t_n). \end{cases}$$

An *argument ranking* for Π is a function α from arguments to integers such that, for every rule R of Π, every atom A occurring in the head of R, and every variable x occurring in an argument term A^i, the positive body of R contains an atom B such that x occurs in an argument term B^j satisfying

$$\alpha(A^0[i]) - d(x, A^i) \geq \alpha(B^0[j]) - d(x, B^j). \quad (10)$$

A program is called *argument-restricted* if it is safe and has an argument ranking.

Review: Answer Sets of First-Order Formulas

This review follows (Ferraris *et al.* 2010), a journal version of (Ferraris *et al.* 2007), which is based on the *stable model operator with the intensional predicates* $\mathbf{p}$, denoted by $\mathrm{SM}_{\mathbf{p}}$. Let $\mathbf{p}$ be a list of distinct predicate constants $p_1, \ldots, p_n$, and let $\mathbf{u}$ be a list of distinct predicate variables $u_1, \ldots, u_n$ of the same length as $\mathbf{p}$. By $\mathbf{u} = \mathbf{p}$ we denote the conjunction of the formulas $\forall \mathbf{x}(u_i(\mathbf{x}) \leftrightarrow p_i(\mathbf{x}))$, where $\mathbf{x}$ is a list of distinct object variables of the same arity as the length of p_i, for all $i = 1, \ldots n$. By $\mathbf{u} \leq \mathbf{p}$ we denote the conjunction of the formulas $\forall \mathbf{x}(u_i(\mathbf{x}) \rightarrow p_i(\mathbf{x}))$ for all $i = 1, \ldots n$, and $\mathbf{u} < \mathbf{p}$ stands for $(\mathbf{u} \leq \mathbf{p}) \wedge \neg(\mathbf{u} = \mathbf{p})$.

For any first-order formula F, $\mathrm{SM}_{\mathbf{p}}[F]$ stands for the second-order formula

$$F \wedge \neg\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})), \quad (11)$$

where $F^*(\mathbf{u})$ is defined recursively:[2]

- $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ for any tuple $\mathbf{t}$ of terms;
- $F^* = F$ for any atomic formula F that does not contain members of $\mathbf{p}$;[3]
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \to G)^* = (F^* \to G^*) \wedge (F \to G)$;
- $(\forall x F)^* = \forall x F^*$;
- $(\exists x F)^* = \exists x F^*$.

For any sentence F, a $\mathbf{p}$-*stable model* of F is an interpretation of the underlying signature that satisfies $\mathrm{SM}_{\mathbf{p}}[F]$. In this paper we will often omit the subscript $\mathbf{p}$ when it is the list of all predicate constants occurring in F.

The answer sets are defined as follows. By $\sigma(F)$ we denote the signature consisting of object, function and predicate constants occurring in F. For any sentence F that contains at least one object constant, we define the answer sets of F as the Herbrand interpretations of $\sigma(F)$ that satisfy $\mathrm{SM}[F]$. Ferraris *et al.* [2010] shows that this definition of an answer set coincides with the usual definition of an answer set, which is given in terms of grounding and reduct when F can be identified with a logic program.

Review: Semi-Safety and Safety in the Absence of Function Symbols

The concept of safety was extended to first-order formulas in (Lee *et al.* 2008a) and was further generalized in (Cabalar *et al.* 2009), but the work there does not consider function constants of positive arity. This section reviews some definitions from (Cabalar *et al.* 2009) in a slightly modified form.[4]

We assume that the signature contains no function constants of positive arity. To every quantifier-free formula F, we assign a set $\mathrm{RV}(F)$ of its *restricted variables* as follows:

- For an atomic formula F,
  - if F is an equality between two variables, then $\mathrm{RV}(F) = \emptyset$;
  - otherwise, $\mathrm{RV}(F)$ is the set of all variables occurring in F;
- $\mathrm{RV}(\bot) = \emptyset$;
- $\mathrm{RV}(F \wedge G) = \mathrm{RV}(F) \cup \mathrm{RV}(G)$;
- $\mathrm{RV}(F \vee G) = \mathrm{RV}(F) \cap \mathrm{RV}(G)$;
- $\mathrm{RV}(F \to G) = \emptyset$.

Recall that the occurrence of one formula in another is called *positive* if the number of implications containing that occurrence in the antecedent is even, and *negative* otherwise. We say that an occurrence of a subformula or a variable in a formula F is *strictly positive* if that occurrence is not in the antecedent of any implication. For example, in (6), the occurrences of $p(x)$ and $p(y)$ are positive, but not strictly positive; the occurrence of s is strictly positive.

Consider a sentence F in prenex form:

$$Q_1 x_1 \cdots Q_n x_n M \tag{12}$$

(each Q_i is $\forall$ or $\exists$; $x_1, \ldots, x_n$ are distinct variables; the matrix M is quantifier-free). It is shown that every first-order formula under the stable model semantics can be turned into prenex form (Lee and Palla 2007, Theorem 2). We say that F is *semi-safe* if every strictly positive occurrence of every variable x_i belongs to a subformula $G \to H$ where x_i belongs to $\mathrm{RV}(G)$. If a sentence has no strictly positive occurrence of a variable, it is clearly semi-safe. The FOL-representation of (9) is semi-safe if every variable occurring in the head of a rule occurs also in the positive body of that rule. Thus the notion of semi-safety, applied to a logic program, is weaker than the notion of safety.

Proposition 1 below shows that all stable models of a semi-safe sentence have the small predicate property: the relation represented by any of its predicate constants p_i can hold for a tuple of arguments only if each member of the tuple is represented by an object constant occurring in F. To make this idea precise, we will use the following notation: For any finite set $\mathbf{c}$ of object constants, $in_{\mathbf{c}}(x)$ stands for the formula

$$\bigvee_{c \in \mathbf{c}} x = c.$$

The small predicate property can be expressed by the conjunction of the sentences

$$\forall v_1, \ldots, v_n \Big(p(v_1, \ldots, v_n) \to \bigwedge_{i=1,\ldots,n} in_{\mathbf{c}}(v_i) \Big)$$

for all predicate constants p occurring in F, where $v_1, \ldots, v_n$ are distinct variables. We denote this sentence by $SPP_{\mathbf{c}}$. By $c(F)$ we denote the set of all object constants occurring in F.

Proposition 1 *(Lee* et al. *2009) For any semi-safe sentence F, formula* $\mathrm{SM}[F]$ *entails* $SPP_{\mathbf{c}(F)}$.

For instance, in application to the prenex form of (1), which is semi-safe, this proposition asserts that $\mathrm{SM}[(1)]$ entails

$$\forall x(p(x) \to x = a \vee x = b) \wedge \forall x(q(x) \to x = a \vee x = b).$$

Cabalar *et al.* [2009] generalized the definition of safety from (Lee *et al.* 2008b) as follows. They consider the following transformations:

- $\neg\bot \mapsto \top, \quad \neg\top \mapsto \bot,$
- $\bot \wedge F \mapsto \bot, \quad F \wedge \bot \mapsto \bot, \quad \top \wedge F \mapsto F, \quad F \wedge \top \mapsto F,$
- $\bot \vee F \mapsto F, \quad F \vee \bot \mapsto F, \quad \top \vee F \mapsto \top, \quad F \vee \top \mapsto \top,$
- $\bot \to F \mapsto \top, \quad F \to \top \mapsto \top, \quad \top \to F \mapsto F.$

We say that a variable x is *positively weakly restricted* in a formula G if the formula obtained from G by

- first replacing every atomic formula A in it such that x belongs to $RV(A)$ by $\bot$,

[2] We treat $\neg F$ as shorthand for $F \to \bot$.

[3] This includes the case when F is $\bot$.

[4] This reformulation is described in the unpublished draft (Lee *et al.* 2009).

- and then applying the transformations above

is $\top$. Similarly, we say that x is *negatively weakly restricted* in G if the formula obtained from G by the same procedure is $\bot$.

A semi-safe sentence (12) is called *safe* if, for every occurrence of a variable x_i in (12),

(a) if Q_i is $\forall$, then the occurrence belongs to
   - a positive subformula of (12) in which x_i is positively weakly restricted, or
   - a negative subformula of (12) in which x_i is negatively weakly restricted;

(b) if Q_i is $\exists$, then the occurrence belongs to
   - a negative subformula of (12) in which x_i is positively weakly restricted, or
   - a positive subformula of (12) in which x_i is negatively weakly restricted.

This definition generalizes the definition of safety for logic programs. The prenex form of (6)

$$\exists xy(\neg(p(x) \wedge p(y) \wedge x \neq y) \rightarrow s) \tag{13}$$

is safe because the antecedent of the implication is a negative subformula in which both x and y are positively weakly restricted in it. (Or $p(x) \wedge p(y) \wedge x \neq y$ can be taken as a positive subformula in which x and y are negatively weakly restricted.) Formula $\exists x \forall y((p(x) \rightarrow q(y)) \rightarrow r)$ is safe because, for x, $p(x)$ can be taken as a positive subformula, and, for y, $q(y)$ can be taken as a negative subformula. Formula $\exists x(\neg p(x) \rightarrow q)$ is safe, while $\forall x(\neg p(x) \rightarrow q)$ is semi-safe, but not safe.

Argument-Restricted Formulas

We extend the notion of argument-restricted programs to arbitrary formulas as follows. We visualize any term t as a rooted tree, with object constants and variables as leaves, and function constants of positive arity as internal nodes. The *height* $h(t)$ of a term t is the height of the corresponding tree. Let F be a quantifier-free formula (possibly containing function constants of positive arity) and let α be a function from arguments to integers. Function $lb_\alpha(x, F)$ maps a variable and a formula to an integer or ω, defined as follows:

- $lb_\alpha(x, p(t_1, \dots, t_n)) =$
  $\min(\{\alpha(p[i]) - d(x, t_i) \mid t_i \text{ contains } x\} \cup \{\omega\})$;
- When F is $t_1 = t_2$, if t_1 is x and t_2 is a ground term, then $lb_\alpha(x, F)$ is $h(t_2)$; Similar if t_2 is x and t_1 is a ground term; otherwise $lb_\alpha(x, F)$ is ω;
- $lb_\alpha(x, \bot) = \omega$;
- $lb_\alpha(x, G \wedge H) = \min(\{lb_\alpha(x, G), lb_\alpha(x, H)\})$;
- $lb_\alpha(x, G \vee H) = \max(\{lb_\alpha(x, G), lb_\alpha(x, H)\})$;
- $lb_\alpha(x, G \rightarrow H) = \omega$.

An *argument ranking* for F is a function α from arguments to integers such that, for every atom A occurring strictly positively in F and every variable x occurring in an argument term A^i, the occurrence of A is in a subformula $G \rightarrow H$ such that

$$\alpha(A^0[i]) - d(x, A^i) \geq lb_\alpha(x, G). \tag{14}$$

An *argument ranking* for a sentence in prenex form is defined as an argument ranking for its matrix. We say that the sentence (12) is *argument-restricted* if it has an argument ranking.

For example, the prenex form of formula (3) is argument-restricted. We can take $\alpha(p[1]) = 0$, $\alpha(q[1]) = 1$. Formula

$$\forall x(p(a) \wedge (p(f(x)) \vee p(x) \rightarrow p(f(x))))$$

is not argument-restricted because for any function α,

$$\begin{aligned}\alpha(p[1]) - d(x, f(x)) &= \alpha(p[1]) - 1 \\ &< lb_\alpha(x, p(f(x)) \vee p(x)) = \alpha(p[1]).\end{aligned}$$

On the other hand,

$$\forall x(p(a, f(a)) \wedge (p(x, f(x)) \vee p(f(x), x) \rightarrow p(x, f(x)))) \tag{15}$$

is argument-restricted. Take $\alpha(p[1]) = \alpha(p[2]) = 1$. Formula

$$\forall xy(p(x) \wedge y = f(x) \rightarrow p(y)) \tag{16}$$

is not argument-restricted, while

$$\forall xy(p(x) \wedge q(y, f(x)) \rightarrow p(y)) \tag{17}$$

is argument-restricted. Take $\alpha(p[1]) = \alpha(q[1]) = 0$, $\alpha(q[2]) = 1$.

The small predicate property can be extended to argument-restricted sentences as follows. For any finite set $\mathbf{f}$ of function constants (including object constants) and any integer m, by $gt^m_{\mathbf{f}}$ we denote the set of ground terms of height up to m which can be constructed from function constants in $\mathbf{f}$. By $in^m_{\mathbf{f}}(x)$ we denote the formula

$$\bigvee_{c \in gt^m_{\mathbf{f}}} x = c.$$

For any formula F and any mapping α from arguments to nonnegative integers, by $SPP^\alpha_{\mathbf{f}}$ we denote the conjunction of the sentences

$$\forall v_1 \dots v_n \Big(p(v_1, \dots, v_n) \rightarrow \bigwedge_{i=1,\dots,n} in^{\alpha(p[i])}_{\mathbf{f}}(v_i) \Big)$$

for all predicate constants p occurring in F. Clearly this is a proper generalization of $SPP_{\mathbf{c}}$ in the previous section.

Intuitively, the small predicate property tells that the extent of a predicate is named by a finite set of ground terms that can be constructed from function constants in $\mathbf{f}$ whose maximum height is given by α. We will show that argument-restrictedness is a syntactic condition that ensures the small predicate property.

We consider the following axiom set *the relaxed unique name assumption (*RUNA*)* for signature σ.

$$\begin{aligned}\forall x_1 \dots x_n y_1 \dots y_n (f(x_1, \dots, x_n) = f(y_1, \dots, y_n) \\ \rightarrow (x_1 = y_1 \wedge \dots \wedge x_n = y_n))\end{aligned} \tag{18}$$

for all function constants f of arity > 0;

$$\forall x_1 \dots x_m y_1 \dots y_n (f(x_1, \dots, x_m) \neq g(y_1, \dots, y_n)) \tag{19}$$

for all pairs of distinct function constants f, g except for the pair in which both f and g have arity 0. In other words, the assumption is slightly weaker than the standard unique name assumption in that object constants are not assumed to be distinct from each other.

We say that an argument ranking α is *strict* if $\alpha(A^0[i]) \geq h(A^i)$ for every strictly positive occurrence of an atom A and any of its argument position i. The following theorem tells that any argument-restricted sentence has the small predicate property. By $\mathbf{f}(F)$ we denote the set of all function constants occurring in F.

Theorem 1 *For any argument-restricted sentence F and any strict argument ranking for F, under the relaxed unique name assumption, formula* SM$[F]$ *entails* $SPP^{\alpha}_{\mathbf{f}(F)}$.

For instance, under the relaxed unique name assumption SM$[(3)]$ entails

$$\forall x(p(x) \rightarrow (x = a \vee x = b)),$$

and

$$\forall x(q(x) \rightarrow (x = a \vee x = b \vee x = f(a) \vee x = f(b)).$$

with $\alpha(p[1]) = 0$, $\alpha(q[1]) = 1$.

Theorem 1 becomes incorrect if we drop the relaxed unique name assumption. For example, take

$$F = p(0) \wedge \forall x(p(f(x)) \rightarrow p(x)),$$

which is argument-restricted (take $\alpha(p[1]) = 1$). However, consider a model I of SM$[F]$ such that the universe is the set of integers, $f^I(m) = m + 1$, $p^I = \{0, -1, -2, \dots\}$. The model I satisfies SM$[F]$ but not $SPP^{\alpha}_{\mathbf{f}}$.

Note that the notion of an argument-restricted program in (Lierler and Lifschitz 2009) requires that the program be safe, while the notion of an argument-restricted formula above does not.

Proposition 2 *Let Π be a program and F a prenex form of the FOL representation of Π. Π is argument-restricted in the sense of (Lierler and Lifschitz 2009) iff F is argument-restricted (in our sense) and safe.*

The notion of an argument-restricted formula is also an extension of semi-safety with function constants.

Proposition 3 *For any sentence F in prenex form that contains no function constants of positive arity, F is semi-safe according to (Lee* et al. *2009) iff it is argument-restricted.*

The proof of the proposition uses the following lemma that describes the relationship between the notions *RV* and lb_α.

Lemma 1 *For any quantifier-free formula F without function constant of positive arity and any mapping α from arguments to integers,*

$$x \in \mathit{RV}(F) \ \textit{ iff } \ lb_\alpha(x, F) \neq \omega.$$

However, the small predicate property alone is not enough to ensure the domain independence of answer sets.

$$\forall x(\neg q(x) \rightarrow p) \wedge q(a)$$

is semi-safe, and thus argument-restricted. The answer set is $\{q(a)\}$ if the Herbrand universe is $\{a\}$; the answer set is $\{p, q(a)\}$ if the Herbrand universe is any proper superset of $\{a\}$ (e.g., $\{a, b\}$). It is not difficult to see that, due to the small predicate property, any semi-safe sentence has at most two kinds of answer sets depending on the underlying signature, be it $\sigma(F)$ or any of its strict superset. In the absence of function constants of positive arity, it is known that, if we further require the sentence to be safe (Lee *et al.* 2008a; Cabalar *et al.* 2009), then such distinction between the underlying signature disappears: answer sets are not affected by the underlying signature.

The definition of safety in the previous section can be applied in the presence of function constants. The only changes required are

- in the definitions of positively (negatively, resp.) weakly restricted variables, to replace "x belongs to *RV*(A)" by "$lb_\alpha(x, A) \neq \omega$".
- replace "semi-safe sentence" with "argument-restricted sentence."

For any safe sentence F (in prenex form; possibly containing function constants of positive arity), and any strict argument ranking α for F, the result of its grounding is the ground formula $\text{Ground}^{\alpha}_{\mathbf{f}}[F]$ defined as follows. If F is quantifier-free, then $\text{Ground}^{\alpha}_{\mathbf{f}}[F] = F$. Otherwise

$$\text{Ground}^{\alpha}_{\mathbf{f}}[\forall x F(x)] = \bigwedge_{c \in gt^m_{\mathbf{f}}} \text{Ground}^{\alpha}_{\mathbf{f}}[F(c)],$$

$$\text{Ground}^{\alpha}_{\mathbf{f}}[\exists x F(x)] = \bigvee_{c \in gt^m_{\mathbf{f}}} \text{Ground}^{\alpha}_{\mathbf{f}}[F(c)],$$

where m is

$$\max(\{lb_\alpha(x, A) \mid A \text{ is an atomic formula occurring in } F(x) \text{ such that } lb_\alpha(x, A) \neq \omega\} \cup \{0\}).$$

For example, for formula (3) with $\alpha(p[1]) = 0$, $\alpha(q[1]) = 1$, $\text{Ground}^{\alpha}_{\{a,b,f\}}[(3)]$ is (4). For formula (15) with $\alpha(p[1]) = \alpha(p[2]) = 1$, formula $\text{Ground}^{\alpha}_{\{a,f\}}[(15)]$ is

$$p(a, f(a)) \wedge (p(a, f(a)) \vee p(f(a), a) \rightarrow p(a, f(a))).$$

In the rest of the section, we present generalizations of the theorems from (Lee *et al.* 2009). We assume that F is a safe sentence, α is a strict argument for the matrix of F and $\mathbf{f}$ is a finite set of function constants that contains at least one object constant and $\mathbf{f}(F)$.

As in (Lifschitz *et al.* 2007), by $\mathbf{INT}^{=}$ we denote intuitionistic predicate logic with equality, and DE stands for the decidable equality axiom.

$$x = y \vee x \neq y \tag{20}$$

The provability of a sentence $F \leftrightarrow G$ in this system implies that SM$[F]$ is equivalent to SM$[G]$ (Lifschitz *et al.* 2007).

Proposition 4

$$\text{Ground}^{\alpha}_{\mathbf{f}}[F] \leftrightarrow F$$

is derivable from SPP$^{\alpha}_{\mathbf{f}}$ in $\mathbf{INT}^{=} + \text{DE} + \text{RUNA}$.

Using Proposition 4 we can prove that the variable-free formula obtained by grounding a safe sentence F has the same stable models as F.

Theorem 2 *Under the relaxed unique name assumption,* $\text{SM}[\text{Ground}^{\alpha}_{\mathbf{f}}[F]]$ *is equivalent to* $\text{SM}[F]$.

In general, the second-order definition of a stable model cannot be expressed in first-order logic. The following theorem shows, however, that in the case of a safe sentence, stable models can be characterized by a very simple first-order formula, almost variable-free:

Theorem 3 *There exists a variable-free formula G such that* $\text{SM}[F]$ *is equivalent to $G \wedge SPP^{\alpha}_{\mathbf{f}(F)}$ under the relaxed unique name assumption.*

The safety of a sentence does indeed imply that its meaning does not depend on the presence of irrelevant object constants in the signature:

Theorem 4 *If F contains at least one object constant, and σ is a signature obtained from $\sigma(F)$ by adding function constants, then an Herbrand interpretation of σ satisfies* $\text{SM}[F]$ *iff it is an answer set of* $\text{Ground}^{\alpha}_{\mathbf{f}(F)}[F]$.

Deciding Argument-Restricted Formulas

The algorithm described in (Lierler and Lifschitz 2009) can be extended to argument-restricted formulas in a straightforward way. Let F be a formula (12). Similar to (Lierler and Lifschitz 2009), we define the operator Ω on the set U of functions from arguments to nonnegative integers by the formula

$$\Omega(\alpha)(p[i]) = \max(\{\min(\{d(x, A^i) + lb_{\alpha}(x, G) \mid G \to H\} \cup \{\omega\}) \mid A, x\} \cup \{0\})$$

where

- A and x range over atoms and variables such that A occurs strictly positively in F, $A^0 = p$, and x occurs in A^i;
- given such A and x, formula $G \to H$ ranges over subformulas of F such that the occurrence of A is in $G \to H$.

It is easy to check that $\alpha \in U$ is an argument ranking for the matrix of F iff $\alpha \geq \Omega(\alpha)$.

The operator Ω is monotone, so that the same argument in (Lierler and Lifschitz 2009) applies to this extension. If F is argument-restricted, then the set of nonnegative argument rankings has the least element $\alpha_{\min} = \Omega^i(\mathbf{0})$ for the smallest i such that $\Omega^{i+1}(\mathbf{0}) = \Omega^i(\mathbf{0})$.[5] On the other hand, for any argument-restricted formula F, all values of $\alpha_{\min}$ do not exceed the product M of two numbers: the total number of arguments and the largest of the numbers $d(x, t)$ for the terms t occurring strictly positively in F and for the variables x occurring in t.

We can use the polynomial time algorithm given in (Lierler and Lifschitz 2009) with the extended definition of Ω to determine whether F is argument-restricted by iterating on $\mathbf{0}$ until

- $\Omega^{i+1}(\mathbf{0}) = \Omega^i(\mathbf{0})$ ($\alpha_{\min}$ is found), or
- one of the values of $\Omega^i(\mathbf{0})$ exceeds M (F is not argument-restricted).

[5] $\mathbf{0}$ denotes a function that maps every argument positions to 0.

Applying to RASPL-1 programs

(Lee *et al.* 2008a) defines the semantics of a program with choice and counting in terms of reduction to the general theory of stable models. That paper assumes that the signature does not contain function constants of positive arity, but here we remove that restriction.

A *cardinality expression* is an expression of the form

$$b\,\{\mathbf{x} : F(\mathbf{x})\} \tag{21}$$

where b is a positive integer ("the bound"), $\mathbf{x}$ is a list of variables (possibly empty), and $F(\mathbf{x})$ is a conjunction of literals (possibly containing variables other than $\mathbf{x}$). This expression reads: there are at least b values of $\mathbf{x}$ such that $F(\mathbf{x})$.

A *RASPL-1 rule* is an expression of the form

$$A_1\ ;\ldots;\ A_l \leftarrow\ E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n \tag{22}$$

($l \geq 0$; $n \geq m \geq 0$), where each A_i is an atom, and each E_i is a cardinality expression. A *RASPL-1 program* is a finite list of rules.

The semantics of a RASPL-1 program is defined by turning it into a first-order sentence (*FOL-representation*) and applying the definition of an answer set for a first-order sentence. The FOL-representation of cardinality expression (21) is the formula

$$\exists \mathbf{x}^1 \cdots \mathbf{x}^b \left[\bigwedge_{1 \leq i \leq b} F(\mathbf{x}^i) \wedge \bigwedge_{1 \leq i < j \leq b} \neg(\mathbf{x}^i = \mathbf{x}^j) \right] \tag{23}$$

where $\mathbf{x}^1, \ldots \mathbf{x}^b$ are lists of new variables of the same length as $\mathbf{x}$. An expression $1\{:\ A\}$, where A is an atom, can be identified with A.

The FOL-representation of a RASPL-1 rule *Head* $\leftarrow$ *Body* is the universal closure of the implication *Body* $\rightarrow$ *Head* with each cardinality expression in *Body* replaced by its FOL-representation. The FOL-representation of a RASPL-1 program is the conjunction of the FOL-representations of its rules.

For any RASPL-1 program Π containing at least one object constant, the *answer sets* of Π are defined as the answer sets of the FOL-representation of Π, as defined earlier.

A variable v is *free* in a rule (22) if

- v occurs in the head $A_1\ ;\ldots;\ A_l$ of the rule, or
- the body $E_1, \ldots, \mathit{not}\ E_n$ of the rule contains a cardinality expression $b\{\mathbf{x} : F(\mathbf{x})\}$ such that v occurs in F and does not belong to $\mathbf{x}$.

In the following we assume that every free variable is distinct from variables $\mathbf{x}$ occurring in (21) by possibly renaming the variables.

An *argument ranking* for Π is a function α from arguments to integers such that, for every rule R of Π, every atom A occurring in the head of R, and every variable v

occurring in an argument term A^i, the positive body of R contains a cardinality expression $b\{\mathbf{x} : F(\mathbf{x})\}$ such that

$$\alpha(A^0[i]) - d(v, A^i) \geq lb_\alpha(v, F(\mathbf{x})).$$

A program is called *argument-restricted* if it has an argument ranking.

The safety condition for RASPL-1 programs defined in (Lee *et al.* 2008a) can be extended to allow function constants as follows. Given an argument ranking α, we say that a cardinality expression $b\{\mathbf{x} : F(\mathbf{x})\}$ is *allowed* if, for every member x of $\mathbf{x}$, $lb_\alpha(x, F) \neq \omega$. For instance, $2\{x : p(f(x), y)\}$ is allowed; $2\{x : \mathit{not}\ p(f(x), y)\}$ is not allowed.

A RASPL-1 program is *safe* if it has an argument ranking α and, for each rule (22) in it,

- each cardinality expression in its body is allowed, and
- for each of its free variables v, there is a cardinality expression $b\{\mathbf{x} : F(\mathbf{x})\}$ among $E_1, \ldots, E_m$ such that $lb_\alpha(v, F) \neq \omega$.

The following proposition follows from Theorem 4 since RASPL-1 programs are shorthand for first-order formulas.

Proposition 5 *Let Π be a safe RASPL-1 program (allowing function constants of positive arity) containing at least one object constant, let F be its FOL-representation, and let α be a strict argument ranking for F. For any signature σ obtained by adding any function constants to $\sigma(F)$, an Herbrand interpretation of σ satisfies* SM$[F]$ *iff it is an answer set of* Ground$^\alpha_{\mathbf{f}(F)}[F]$.

For example, program (8) (assuming the last rule is written in the syntax of RASPL-1) is safe, and can be equivalently rewritten as a ground formula under the stable model semantics.

Applying to FLP-Aggregates

An *FLP-aggregate expression* is of the form

$$\text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b \tag{24}$$

where

- OP is a symbol for an *aggregate function*, such as SUM, COUNT;
- $\mathbf{x}$ is a nonempty list of distinct object variables;
- $F(\mathbf{x})$ is a quantifier-free formula;
- $\succeq$ is a symbol for a binary relation over integers, such as $\leq, \geq, <, >, =, \neq$;
- b is an integer constant.

An *FLP-rule* is an expression of the form

$$A_1; \ldots; A_l \leftarrow E_1, \ldots, E_m, \mathit{not}\ E_{m+1}, \ldots, \mathit{not}\ E_n \tag{25}$$

($l \geq 0$; $n \geq m \geq 0$), where each A_i is an atom and each E_i is an atom or an FLP-aggregate expression. An *FLP-program* is a finite list of rules.

The definition of an argument ranking for an FLP program is the same as that for a program whose rules have the form (9). The definitions of "allowed expressions" and "free variables" for FLP programs are defined similar to those for RASPL-1 programs.

According to (Faber *et al.* 2004), a rule (25) is *safe* if

- each FLP-aggregate expression in its body is allowed, and
- for each of its free variables v, there is an atom E_i ($i = 1, \ldots, m$) that contains v.

Again we assume that every free variable is distinct from variables $\mathbf{x}$ in (24) by possibly renaming the variables. For any argument-restricted FLP-program Π, any strict argument ranking α for Π and any finite set $\mathbf{f}$ of function constants, ground FLP program Ground$^\alpha_{\mathbf{f}}[\Pi]$ is defined as follows:

- replace every free occurrence of a variable x with a ground term from $gt^m_{\mathbf{f}}$ where m is

$$\max(\{lb_\alpha(x, A) \mid A \text{ is an atomic formula occurring in } \Pi \text{ such that } lb_\alpha(x, A) \neq \omega\} \cup \{0\});$$

- replace every occurrence of $\text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b$ in the resulting program with $\text{OP}\langle S\rangle \succeq b$ where S is the set of all ground instantiations of $\mathbf{x} : F(\mathbf{x})$ w.r.t. $gt^m_{\mathbf{f}}$ where m is

$$\max(\{lb_\alpha(x, A) \mid x \text{ is in } \mathbf{x} \text{ and } A \text{ is an atomic formula occurring in } F(\mathbf{x}) \text{ such that } lb_\alpha(x, A) \neq \omega\} \cup \{0\}).$$

In the following proposition, the definition of an answer set refers to the one in (Faber *et al.* 2004). By $\sigma(\Pi)$ we denote the signature consisting of object, function and predicate constants occurring in Π. By $\mathbf{f}(\Pi)$ we denote the set of all function constants occurring in Π.

Proposition 6 *Let Π be a safe FLP program containing at least one object constant, let α be a strict argument ranking for Π. For any signature σ obtained by adding function constants to $\sigma(\Pi)$, a ground set of atoms of σ is an answer set of Π iff it is an answer set of ground FLP program* Ground$^\alpha_{\mathbf{f}(\Pi)}[\Pi]$.

For example, program (8) is safe, and grounding the last rule of (8) yields

$$s \leftarrow \#count\{a{:}p(a), f(a){:}p(f(a)), f(f(a)){:}p(f(f(a))), a{:}q(a), f(a){:}q(f(a)), a{:}r(a), f(a){:}r(f(a))\} \leq 1$$

with $\alpha(p[1]) = 2, \alpha(q[1]) = 1, \alpha(r[1]) = 1$.

The following is another safe program in the syntax of the DLV-COMPLEX language.

```
string(cons(a, cons(b, cons(a, cons(c, nil))))).
letter(a).  letter(b).  letter(c).  letter(d).
tail(X) :- string(X).
tail(Y) :- tail(cons(X,Y)), letter(X).
containsTwo(X):- #count{Y: tail(cons(X,Y))}>=2,
                 letter(X).
```

The program has only one answer set. For the `string` "[a,b,a,c]", `tail(X)` belongs to the answer set if `X` is the tail of the string. `containsTwo(X)` belongs to the answer set if letter `X` occurs at least twice in the string. The answer set returned by DLV-COMPLEX contains `containsTwo(a)`.

Allowing Extensional Predicates

We defined the notion of an argument-restricted formula only in the case when the intensional predicates $\mathbf{p}$ in $\mathrm{SM}_{\mathbf{p}}[F]$ are all the predicates occurring in F. The notion can be extended to the general case, in which intensional predicates $\mathbf{p}$ are any subset of the predicates in the signature, by only modifying the first clause of the definition of $lb_\alpha(x, F)$ as

$$lb_\alpha(x, p(t_1, \dots, t_n)) = \begin{cases} \min(\{\alpha(p[i]) - d(x, t_i) \mid t_i \text{ contains } x\} \cup \{\omega\}) \\ \qquad\qquad \text{if } p \text{ is intensional;} \\ \omega \qquad\qquad \text{otherwise.} \end{cases}$$

For example, similar to (16), formula (17) becomes non-argument-restricted if q is regarded as extensional.

The definition of safety in the general case remains the same as in the case when all predicate constants are regarded intensional.

Theorems 1–4 can be extended to allow extensional predicates by allowing $\mathbf{p}$ to be any subset of the predicate constants in the signature, and by replacing $\mathrm{SM}[F]$ in the statements with $\mathrm{SM}_{\mathbf{p}}[F]$.

Reducing Argument-Restricted Formulas to Argument-Restricted Programs

System F2LP (Lee and Palla 2009) turns a first-order formula into a program whose rules have the form

$$A_1 \,;\dots;\, A_k \leftarrow A_{k+1}, \dots, A_m, \mathit{not}\ A_{m+1}, \dots, \mathit{not}\ A_n, \mathit{not\ not}\ A_{n+1}, \dots, \mathit{not\ not}\ A_p \tag{26}$$

($0 \le k \le m \le n \le p$), where each A_i is an atomic formula. The definition of an argument-restricted program from (Lierler and Lifschitz 2009) can be extended to this syntax as follows: A program whose rules have the form (26) is called *argument-restricted* if its FOL-representation is argument-restricted.

Is it possible to turn argument-restricted formulas into such argument-restricted programs? This is an important question in applications of system F2LP. A part of the transformation that is implemented in F2LP follows the one from (Cabalar *et al.* 2005) in order to turn a quantifier-free formula into a set of rules. However, this transformation does not preserve argument-restrictedness: among the transformation rules defined in (Cabalar *et al.* 2005), only one rule (L5) does not preserve it. For instance, given

$$\begin{array}{l} \forall x((p(x) \to r(f(x))) \\ \wedge((p(x) \to q(x)) \wedge r(x) \to s(x))), \end{array} \tag{27}$$

F2LP applies the rule (L5) to turn it into the following program

$$\begin{array}{rcl} r(f(x)) & \leftarrow & p(x) \\ s(x) & \leftarrow & \mathit{not}\ p(x), r(x) \\ s(x) & \leftarrow & q(x), r(x) \\ p(x)\,;s(x) & \leftarrow & r(x), \mathit{not\ not}\ q(x), \end{array}$$

which is finitely groundable, but not argument-restricted.

A reduction that preserves argument-restrictedness is possible under a certain syntactic condition. We call a formula *singular* if every subformula $G \to H$ that occurs in the antecedent of another implication is contained in a formula that has no strictly positive occurrence of predicate constants. Formula (27) is not singular because of $p(x) \to q(x)$.

Proposition 7 *Every singular argument-restricted quantifier-free formula is strongly equivalent to an argument-restricted program.*

It remains an open question if the restriction that formula be singular can be removed from the statement.

Extending λ-restricted Programs to Formulas

Currently, there is no implementation of argument-restricted programs. On the other hand, the input language of GRINGO is assumed to be λ-restricted, which is a stronger notion than argument-restrictedness. Similar to the generalization of argument-restricted programs to argument-restricted formulas, we generalize the notion of λ-restricted programs to λ-restricted formulas, and show that the latter can be reduced to the former under some condition. This justifies the use of F2LP for computing answer sets of λ-restricted formulas.

According to (Gebser *et al.* 2007b), a λ-ranking for a program Π is a function α from predicate constants of $\sigma(\Pi)$ to integers such that for every rule r of Π and every variable x occurring in r, there is an atom B in the positive body of r such that B contains x and

$$\alpha(A^0) > \alpha(B^0).$$

We say that a program is λ-*restricted* if the program has a λ-ranking.

This concept can be extended to formulas similar to the definition of argument-restricted formulas as follows. Let α be a function from predicate constants of $\sigma(\Pi)$ to integers. For a quantifier-free formula F, function $lb'_\alpha(x, F)$ is defined recursively:

- $lb'_\alpha(x, p(t_1, \dots, t_n)) = \alpha(p)$;
- $lb'_\alpha(x, t_1 = t_2) = \omega$;
- $lb'_\alpha(x, \bot) = \omega$;
- $lb'_\alpha(x, G \wedge H) = \min(\{lb'_\alpha(x, G), lb'_\alpha(x, H)\})$;
- $lb'_\alpha(x, G \vee H) = \max(\{lb'_\alpha(x, G), lb'_\alpha(x, H)\})$;
- $lb'_\alpha(x, G \to H) = \omega$.

α is called a λ *ranking* for F if every strictly positive occurrence of every variable x in an atom A is contained in a subformula $G \to H$ such that

$$\alpha(A^0) > lb'_\alpha(x, G).$$

We say that prenex formula (12) is λ-*restricted* if G has a λ-ranking. This definition is similar to argument-restricted formulas, but does not take into account the depth of a variable in a predicate. It is easy to check that every λ-restricted formula is argument-restricted. Also this definition reduces to the definition from (Gebser *et al.* 2007b) when it is applied to logic programs.

Proposition 8 *Let* Π *be a program and* F *a prenex form of the FOL representation of* Π. Π *is* λ*-restricted in the sense of (Gebser* et al. *2007b) iff* F *is* λ*-restricted (in our sense).*

It is clear from the definitions that every λ-restricted formula is argument-restricted. Similar to Proposition 7, the following holds.

Proposition 9 *Every singular* λ*-restricted quantifier-free formula is strongly equivalent to a* λ*-restricted program.*

Conclusion

We have presented a decidable class of the general language of stable models that can be instantiated even in the presence of function constants. The notion naturally extends the notion of an argument-restricted program and the notion of a semi-safe sentence that have been studied in different contexts. We view argument-restrictedness, as well as λ-restrictedness and other similar definitions as syntactic conditions to ensure the small predicate property, which says that the extent of a predicate can be represented by a finite set of ground terms. Under the small predicate property, safety condition can be imposed to ensure that grounding relative to any domain does not affect the answer sets. This definition of safety is a generalization of the definition of safety from (Cabalar *et al.* 2009).

Based on this, we presented a syntactic condition under which RASPL-1 programs and FLP programs can be finitely instantiated. We expect that the notions introduced in the paper will be useful for clarifying other important constructs in ASP.

Acknowledgements

We are grateful to Yuliya Lierler, Vladimir Lifschitz, Yunsong Meng, Ravi Palla and anonymous referees for their useful comments and discussions related to this paper. Some theorems in this paper are natural generalizations of the theorems reported in (Lee *et al.* 2009), the joint work by the second author, Vladimir Lifschiz and Ravi Palla. Ravi also contributed to Proposition 7. This work was partially supported by the National Science Foundation under Grant IIS-0916116 and by the IARPA SCIL program.

References

Pedro Cabalar, David Pearce, and Agustin Valverde. Reducing propositional theories in equilibrium logic to logic programs. In *Proceedings of Portuguese Conference on Artificial Intelligence (EPIA)*, pages 4–17, 2005.

Pedro Cabalar, David Pearce, and Agustin Valverde. A revised concept of safety for general answer set programs. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 58–70, 2009.

Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in ASP: theory and implementation. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 407–424, 2008.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 2004.

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 372–379, 2007.

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription. *Artificial Intelligence*, 2010. To appear.

Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo: A new grounder for answer set programming. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.

Joohyung Lee and Ravi Palla. Yet another proof of the strong equivalence between propositional theories and logic programs. In *Working Notes of the Workshop on Correspondence and Equivalence for Nonmonotonic Theories*, 2007.

Joohyung Lee and Ravi Palla. System F2LP – computing answer sets of first-order formulas. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 515–521, 2009.

Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 472–479, 2008.

Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. Safe formulas in the general theory of stable models (preliminary report). In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 672–676, 2008.

Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. Safe formulas in the general theory of stable models.[6] Unpublished Draft, 2009.

Yuliya Lierler and Vladimir Lifschitz. One more decidable class of finitely ground programs. In Patricia M. Hill and David Scott Warren, editors, *ICLP*, volume 5649 of *Lecture Notes in Computer Science*, pages 489–493. Springer, 2009.

Vladimir Lifschitz, David Pearce, and Agustin Valverde. A characterization of strong equivalence for logic programs with variables. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 2007.

Tommi Syrjänen. Omega-restricted logic programs. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 267–279, 2001.

Tommi Syrjänen. Cardinality constraint programs. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, pages 187–199, 2004.

[6] *http://peace.eas.asu.edu/joolee/papers/safety.pdf* .