Intelligent Instantiation and Supersafe Rules*

Vladimir Lifschitz[1]

1 **Department of Computer Science, University of Texas at Austin, 2317 Speedway, Stop D9500, Austin, TX 78712, USA** vl@cs.utexas.edu

Abstract

In the input languages of most answer set solvers, a rule with variables has, conceptually, infinitely many instances. The primary role of the process of intelligent instillation is to identify a finite set of ground instances of rules of the given program that are "essential" for generating its stable models. This process can be launched only when all rules of the program are safe. If a program contains arithmetic operations or comparisons then its rules are expected to satisfy conditions that are even stronger than safety. This paper is an attempt to make the idea of an essential instance and the need for "supersafety" in the process of intelligent instantiation mathematically precise.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory

Keywords and phrases answer set programming

Digital Object Identifier 10.4230/OASIcs.ICLP.2016.first-page-number

1 Introduction

The input languages of most answer set solvers are not typed. When a program in such a language is grounded, the variables occurring in it can be replaced by arbitrary ground terms not containing arithmetic operations, and that includes arbitrary integers.[1] Thus the set of ground instances of any non-ground rule is infinite. The primary role of the process of intelligent instantiation is to identify a finite set of ground instances of the rules of the program that are "essential" for generating its stable models.

The possibility of intelligent instantiation is predicated on the assumption that every rule of the given program is safe—that each variable occurring in the rule appears also nonnegated in its body. If an unsafe rule is found in the program then the solver produces an error message and stops. For example, generating the stable models of a program will not be attempted if it contains the rule

$$p(X,Y) \leftarrow q(X),$$

because the variable Y occurs in its head but not in the body.

The safety assumption does not guarantee that the set of essential instances is finite. For example, all rules of the program

$$\begin{array}{l} p(a), \\ p(b), \\ p(f(f(X))) \leftarrow p(X) \end{array} \tag{1}$$

* This work was supported in part by the National Science Foundation under Grant IIS-1422455.

[1] The language SPARC [1] is a notable exception. In a SPARC program, finite sorts are assigned to arguments of all predicates, and the range of integers allowed in the process of grounding is finite.

Technical Communications of the 32nd Int'l Conference on Logic Programming (ICLP'16).
Editors: Manuel Carro, Andy King, Marina De Vos, and Neda Saeedloei; pp. 1–14

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are safe, but its third rule has infinitely many instances essential for constructing the stable model:

$$\begin{array}{ll} p(f(f(a))) \leftarrow p(a), & p(f(f(b))) \leftarrow p(b), \\ p(f(f(f(f(a))))) \leftarrow p(f(f(a))), & p(f(f(f(f(b))))) \leftarrow p(f(f(b))), \end{array}$$
$$\ldots$$

The rules in the first line are essential because their bodies $p(a)$ and $p(b)$ are facts from (1). The rules in the second line are essential because their bodies are identical to the heads of the rules in the first line, and so on. An attempt to form all essential instances will add a finite set of rules at each step, but it will not terminate. In the terminology of Calimeri et al. [4], program (1) is not finitely ground. The safety of all rules does guarantee, however—for programs containing neither arithmetic operations nor comparisons—that all essential instances can be found in a stepwise manner, as in the example above, with finitely many instances added at every step.

In the presence of arithmetic operations and comparisons, on the other hand, the possibility of launching the process of accumulating essential instances is not ensured by the safety of all rules. For example, each of the rules

$$p(X, Y) \leftarrow q(X + Y), \tag{2}$$
$$p(X, Y) \leftarrow X < Y, \tag{3}$$
$$p(X, Y) \leftarrow X = Y \tag{4}$$

is safe in the sense that both variables occurring in it appear nonnegated in the body. But the presence of any of these rules in a program causes the grounder GRINGO to stop execution with the same error message as in the presence of an unsafe rule. On the other hand, GRINGO does not object against the rules

$$p(X, Y) \leftarrow X = Y,\ q(X) \tag{5}$$

and

$$p(X) \leftarrow X + 3 = 4. \tag{6}$$

The discussion of safety in Version 2.0 of the Potassco User Guide (`http://sourceforge.net/projects/potassco/files/guide/`) shows that the conditions under which GRINGO treats a rule as safe are quite complicated.[2] Such conditions have to be imposed because safety in the traditional sense does not guarantee the possibility of calculating essential instances in the step-by-step manner; the rules of the program must be "supersafe."

This paper shows how our informal discussion of essential instances, of the role of intelligent instantiation, and of the need for supersafety can be made mathematically precise. In the next section we define which elements of a set Γ of ground rules are essential and

[2] According to that document, occurrences of variables in the scope of arithmetic functions can only justify safety for "simple arithmetic terms"—terms containing a single occurrence of a variable and no arithmetic operations other than addition, subtraction, and multiplication. This explains why GRINGO does not accept rule (2) as safe: the term $X + Y$ is not simple. Moreover, if multiplication is used, then the constant factor must not evaluate to 0 for the variable occurrence to justify safety. Furthermore, according to the User Guide, safety is not justified by occurrences of variables in inequalities; hence (3) is not accepted. This restriction does not apply to equalities. "However, this only works when unification can be made directionally, i.e., it must be possible to instantiate one side without knowing the values of variables on the other side." This explains why GRINGO considers rule (4) unsafe but accepts (5) and (6).

prove that the set $E(\Gamma)$ of essential rules has the same stable models as the whole Γ. The set $E(\Gamma)$ is defined as the union of a monotone sequence of subsets $E_k(\Gamma)$, representing the stepwise process of accumulating essential rules. After describing a class of logic programs with variables and arithmetic in Section 3, we define and study the concept of a supersafe rule (Section 4). The main result of this paper, proved in Section 5, shows that if Γ is the propositional image of a program consisting of supersafe rules then each of the sets $E_k(\Gamma)$ is finite. This theorem clarifies the role of the additional conditions that GRINGO imposes on safe rules.

2 Essential Rules

2.1 Propositional Programs

We start by describing the class of programs without variables for which the concept of an essential rule will be defined.[3] Consider a fixed propositional signature—a set of symbols called *atoms*. (In applications to the study of logic programs with variables and arithmetic, the signature will consist of the ground atoms not containing arithmetic operations.) A *(propositional) rule* is an expression of the form $H \leftarrow B$, where the *head* H and the *body* B are propositional formulas formed from atoms and the symbols $\top$ (true) and $\bot$ (false) using the connectives $\wedge$, $\vee$, $\neg$.

A *(propositional) program* is a set of rules.

A set M of atoms will be identified with the truth assignment that maps all elements of M to *true* and all other atoms to *false*. The *reduct* R^M of a rule R relative to M is the rule obtained by replacing, in the head and in the body of R, each subformula F that begins with negation and is not in the scope of negation with $\top$ if M satisfies F, and with $\bot$ otherwise. The reduct Γ^M of a program Γ is defined as the set of the reducts R^M of all rules R of Γ. We say that M is a *stable model* of a program Γ if M is minimal among the sets satisfying Γ^M.

2.2 Essential Part of a Propositional Program

Consider a propositional program Γ such that the body of every rule of Γ is a conjunction of formulas of three types: (a) symbols $\top$ and $\bot$; (b) atoms; (c) formulas beginning with negation. In the definition of the essential part of Γ below, the following terminology is used. A *nonnegated atom* of a propositional formula F is an atom A such that at least one occurrence of A in F is not in the scope of negation. A rule from Γ is *trivial* if at least one of the conjunctive terms of its body is $\bot$.

The subsets $E_0(\Gamma)$, $E_1(\Gamma), \dots$ of Γ are defined as follows:

- $E_0(\Gamma) = \emptyset$,
- $E_{k+1}(\Gamma)$ is the set of all nontrivial rules R of Γ such that every nonnegated atom of the body of R is also a nonnegated atom of the head of some rule from $E_k(\Gamma)$.

It is clear that every member of the sequence $E_0(\Gamma), E_1(\Gamma), \dots$ is a subset of the one that follows (by induction). It is clear also that if $E_{k+1}(\Gamma) = E_k(\Gamma)$ then $E_l(\Gamma) = E_k(\Gamma)$ for all l that are greater than k.

The set of *essential rules* of Γ, denoted by $E(\Gamma)$, is defined as the union $\bigcup_{k\geq 0} E_k(\Gamma)$. The *degree* of an essential rule R is the smallest k such that $R \in E_k(\Gamma)$.

[3] Programs considered here are programs with nested expressions [7] without classical negation, with the usual symbols for propositional connectives used instead of the comma, the semicolon, and "*not*" in the original publication.

▶ **Theorem 1.** *Programs* Γ *and* $E(\Gamma)$ *have the same stable models.*

▶ **Example 2.** If the rules of Γ are

$$\begin{aligned} &a_1 \vee a_2 \leftarrow \neg a_0, \\ &\quad b_n \leftarrow a_n \wedge a_{n+1} \qquad (n \geq 0) \end{aligned}$$

then

$$\begin{aligned} &E_0(\Gamma) = \emptyset, \\ &E_1(\Gamma) = \{a_1 \vee a_2 \leftarrow \neg a_0\}, \\ &E_2(\Gamma) = \{a_1 \vee a_2 \leftarrow \neg a_0,\ b_1 \leftarrow a_1 \wedge a_2\}, \end{aligned}$$

and $E_3(\Gamma) = E_2(\Gamma)$. It follows that Γ has two essential rules: rule $a_1 \vee a_2 \leftarrow \neg a_0$ of degree 1 and rule $b_1 \leftarrow a_1 \wedge a_2$ of degree 2. The program consisting of these two rules has the same stable models as Γ: $\{a_1\}$ and $\{a_2\}$.

▶ **Example 3.** If the rules of Γ are

$$\begin{aligned} &a_1 \leftarrow \top, \\ &a_{2n} \leftarrow a_n \qquad (n \geq 0) \end{aligned}$$

then

$$\begin{aligned} &E_0(\Gamma) = \emptyset, \\ &E_1(\Gamma) = \{a_1 \leftarrow \top\}, \\ &E_2(\Gamma) = \{a_1 \leftarrow \top,\ a_2 \leftarrow a_1\}, \\ &E_3(\Gamma) = \{a_1 \leftarrow \top,\ a_2 \leftarrow a_1,\ a_4 \leftarrow a_2\}, \\ &\dots, \end{aligned}$$

so that

$$E(\Gamma) = \{a_1 \leftarrow \top\} \cup \{a_{2^{k+1}} \leftarrow a_{2^k} \ :\ k \geq 0\}.$$

The set of essential rules in this case is infinite, but for every positive k the program has only one essential rule of degree k. The set $E(\Gamma)$ has the same stable model as Γ: $\{a_1, a_2, a_4, \dots\}$.

▶ **Example 4.** If the rules of Γ are

$$\begin{aligned} &a_0 \leftarrow \top, \\ &b_{m,n} \leftarrow a_{n-m} \qquad (n \geq m \geq 0) \end{aligned}$$

then

$$\begin{aligned} &E_0(\Gamma) = \emptyset, \\ &E_1(\Gamma) = \{a_0 \leftarrow \top\}, \\ &E_2(\Gamma) = \{a_0 \leftarrow \top\} \cup \{b_{n,n} \leftarrow a_0 \ :\ n \geq 0\}, \end{aligned}$$

and $E_3(\Gamma) = E_2(\Gamma)$. It follows that Γ has infinitely many essential rules: rule $a_0 \leftarrow \top$ of degree 1 and rules $b_{n,n} \leftarrow a_0$, for all n, of degree 2. The program consisting of these rules has the same stable model as Γ: $\{a_0, b_{0,0}, b_{1,1}, \dots\}$.

In Section 5 we will apply the concept of an essential rule to "propositional images" of programs with variables and arithmetic operations, and we will see that the behaviors observed in the examples above correspond to programs of three types: (1) programs for which the process of intelligent instantiation terminates; (2) programs for which this process can be launched but does not terminate; and (3) programs for which this process cannot be even launched. We will see also that if every rule of a program is supersafe then the program cannot belong to the third group.

2.3 Proof of Theorem 1

▶ **Lemma 5.** *Let Δ be a subset of a propositional program Γ, and let H be the set of all nonnegated atoms of the heads of the rules of Δ. If the body of every nontrivial rule of $\Gamma \setminus \Delta$ contains a nonnegated atom that does not belong to H then Γ and Δ have the same stable models.*

Proof. Consider first the case when the rules of Γ do not contain negation. We need to show that Γ and Δ have the same minimal models. Assume that M is a minimal model of Δ. Then $M \subseteq H$, so that the body of every nontrivial rule of $\Gamma \setminus \Delta$ contains a nonnegated atom that does not belong to M. It follows that M satisfies all rules of $\Gamma \setminus \Delta$, so that M is a model of Γ, and consequently a minimal model of Γ. In the other direction, assume than M is a minimal model of Γ. To show that M is minimal even among the models of Δ, consider a subset M' of M that satisfies all rules of Δ. Then $M' \cap H$ satisfies all rules of Δ as well, so that every nontrivial rule of $\Gamma \setminus \Delta$ contains a nonnegated atom that does not belong to $M' \cap H$. It follows that this set satisfies all rules of $\Gamma \setminus \Delta$, so that it is a model of Γ. Since it is a subset of a minimal model M of Γ, we can conclude that $M' \cap H = M$. Since M' is a subset of M, it follows that $M' = M$.

If some rules of Γ contain negation then consider the reducts Γ^M of Γ and Δ^M of Δ with respect to the same set M of atoms. It is clear that Δ^M is a subset of Γ^M, that H is the set of all nonnegated atoms of the heads of the rules of Δ^M, and that the body of every nontrivial rule of $\Gamma^M \setminus \Delta^M$ contains a nonnegated atom that does not belong to H. Furthermore, the rules of Γ^M do not contain negation. It follows, by the special case of the lemma proved earlier, that Γ^M and Δ^M have the same minimal models. In particular, M is a minimal model of Γ^M iff M is a minimal model of Δ^M. In other words, M is a stable model of Γ iff M is a stable model of Δ. ◀

To prove the theorem, consider the set H of all nonnegated atoms of the heads of the rules of $E(\Gamma)$. We will show that the body of every nontrivial rule of $\Gamma \setminus E(\Gamma)$ contains a nonnegated atom that does not belong to H; then the assertion of the theorem will follow from the lemma. Assume that R is a nontrivial rule of Γ such that all nonnegated atoms in the body of R belong to H. Then each of these atoms A is a nonnegated atom of the head of a rule that belongs to $E_k(\Gamma)$ for some k. This k can be chosen uniformly for all these atoms A: take the largest of the values of k corresponding to all nonnegated atoms in the head of R. Then R belongs to $E_{k+1}(\Gamma)$, and consequently to $E(\Gamma)$.

3 Programs with Variables and Arithmetic

The programming language defined in this section is a subset of the "Abstract Gringo" language AG [5]. The meaning of a program in this language is characterized by means of a transformation, denoted by τ, that turns rules and programs into their propositional images—propositional programs in the sense of Section 2.1. The stable models of a program are defined as the stable models of its propositional image.[4]

[4] Gebser et al. [5] write rules $H \leftarrow B$ of $\tau\Pi$ as implications $B \rightarrow H$. More importantly, H and B are allowed in that paper to contain implications and infinitely long conjunctions and disjunctions. This additional generality is not needed here because the programs that we study contain neither conditional literals nor aggregates.

3.1 Syntax

We assume that three disjoint sets of symbols are selected—*numerals*, *symbolic constants*, and *variables*—which do not contain the symbols

$$+ \quad - \quad \times \quad / \quad .. \tag{7}$$
$$= \quad \neq \quad < \quad > \quad \leq \quad \geq \tag{8}$$
$$not \quad \wedge \quad \vee \quad , \quad (\quad) \tag{9}$$

(The symbol .. is used to represent intervals.) We assume that a 1–1 correspondence between the set of numerals and the set $\mathbf{Z}$ of integers is chosen. The numeral corresponding to an integer n will be denoted by $\overline{n}$.

Terms are defined recursively, as follows:

- all numerals, symbolic constants, and variables are terms;
- if f is a symbolic constant and $\mathbf{t}$ is a tuple of terms separated by commas then $f(\mathbf{t})$ is a term;
- if t_1 and t_2 are terms and $\star$ is one of the symbols (7) then $(t_1 \star t_2)$ is a term.

An *atom* is an expression of the form $p(\mathbf{t})$, where p is a symbolic constant and $\mathbf{t}$ is a tuple of terms separated by commas.

A term or an atom is *precomputed* if it contains neither variables nor symbols (7). We assume a total order on precomputed terms such that for any integers m and n, $\overline{m} \leq \overline{n}$ iff $m \leq n$.

For any atom A, the expressions A, *not* A, *not not* A are *literals*. A *comparison* is an expression of the form $t_1 \prec t_2$ where t_1, t_2 are terms and $\prec$ is one of the symbols (8).

A *rule* is an expression of the form

$$H_1 \vee \cdots \vee H_k \leftarrow B_1 \wedge \cdots \wedge B_m \tag{10}$$

or a "choice rule" of the form

$$\{A\} \leftarrow B_1 \wedge \cdots \wedge B_m \tag{11}$$

$(k, m \geq 0)$, where each H_i and each B_j is a literal or a comparison, and A is an atom. A *program* is a set of rules.

A rule or another syntactic expression is *ground* if it does not contain variables. A rule (10) or (11) is *safe* if each variable occurring in it appears also in one of the expressions B_j which is an atom or a comparison.

3.2 Propositional Image of a Program

The signature of the propositional program $\tau\Pi$, defined below, is the set of precomputed atoms.

3.2.1 Semantics of Ground Terms

Every ground term t represents a finite set $[t]$ of precomputed terms, which is defined recursively:

- if t is a numeral or a symbolic constant then $[t]$ is $\{t\}$;
- if t is $f(t_1, \ldots, t_n)$ then $[t]$ is the set of terms $f(r_1, \ldots, r_n)$ for all $r_1 \in [t_1], \ldots, r_n \in [t_n]$;
- if t is $t_1 + t_2$ then $[t]$ is the set of numerals $\overline{n_1 + n_2}$ for all integers n_1, n_2 such that $\overline{n_1} \in [t_1]$ and $\overline{n_2} \in [t_2]$; similarly when t is $t_1 - t_2$ or $t_1 \times t_2$;

- if t is t_1/t_2 then $[t]$ is the set of numerals $\overline{\lfloor n_1/n_2 \rfloor}$ for all integers n_1, n_2 such that $\overline{n_1} \in [t_1]$, $\overline{n_2} \in [t_2]$, and $n_2 \neq 0$;
- if t is $t_1 \,..\, t_2$ then $[t]$ is the set of numerals $\overline{m}$ for all integers m such that, for some integers n_1, n_2,

$$\overline{n_1} \in [t_1], \qquad \overline{n_2} \in [t_2], \qquad n_1 \leq m \leq n_2.$$

For example, $[\overline{1} \,..\, (\overline{7} - \overline{5})] = \{\overline{1}, \overline{2}\}$; if t contains a symbolic constant in the scope of an arithmetic operation then $[t] = \emptyset$.

3.2.2 Propositional Images of Ground Literals and Comparisons

If A is a ground atom $p(t_1, \dots, t_n)$ then

- $\tau_\wedge A$ stands for the conjunction of the atoms $p(r_1, \dots, r_n)$ for all $r_1 \in [t_1], \dots, r_n \in [t_n]$, and $\tau_\vee A$ is the disjunction of these atoms;
- $\tau_\wedge(not\ A)$ is $\neg\tau_\vee A$, and $\tau_\vee(not\ A)$ is $\neg\tau_\wedge A$;
- $\tau_\wedge(not\ not\ A)$ is $\neg\neg\tau_\wedge A$, and $\tau_\vee(not\ not\ A)$ is $\neg\neg\tau_\vee A$.

For any ground terms t_1, t_2,

- $\tau_\wedge(t_1 \prec t_2)$ is $\top$ if the relation $\prec$ holds between the terms r_1 and r_2 for all $r_1 \in [t_1]$ and $r_2 \in [t_2]$, and $\bot$ otherwise;
- $\tau_\vee(t_1 \prec t_2)$ is $\top$ if the relation $\prec$ holds between the terms r_1 and r_2 for some r_1, r_2 such that $r_1 \in [t_1]$ and $r_2 \in [t_2]$, and $\bot$ otherwise.

For example, $\tau_\vee(\overline{3} = \overline{1}..\overline{3})$ is $\top$.

3.2.3 Propositional Images of Rules and Programs

For any ground rule R of form (10), τR stands for the propositional rule

$$\tau_\wedge H_1 \vee \dots \vee \tau_\wedge H_k \leftarrow \tau_\vee B_1 \wedge \dots \wedge \tau_\vee B_m.$$

For any ground rule R of form (11), where A is $p(t_1, \dots, t_n)$, τR stands for the propositional rule

$$\bigwedge_{r_1 \in [t_1], \dots, r_n \in [t_n]} (p(r_1, \dots, r_n) \vee \neg p(r_1, \dots, r_n)) \leftarrow \tau_\vee B_1 \wedge \dots \wedge \tau_\vee B_m.$$

A *ground instance* of a rule R is a ground rule obtained from R by substituting precomputed terms for variables. The propositional image τR of a rule R with variables is the set of the propositional images of the instances of R. For any program Π, $\tau\Pi$ is the union of the sets τR for all rules R of Π.

3.2.4 Examples

▶ **Example 6.** The propositional image of the ground rule

$$a(\overline{1} \,..\, \overline{3}) \leftarrow b(\overline{4} \,..\, \overline{6})$$

is the propositional rule

$$a(\overline{1}) \wedge a(\overline{2}) \wedge a(\overline{3}) \leftarrow b(\overline{4}) \vee b(\overline{5}) \vee b(\overline{6}).$$

▶ **Example 7.** The propositional image of the rule

$$\{a(X)\} \leftarrow X = \overline{1}\,..\,\overline{3}$$

consists of the propositional rules

$$\begin{aligned} a(\overline{n}) \vee \neg a(\overline{n}) &\leftarrow \top && \text{for } n \in \{1,2,3\}, \\ a(r) \vee \neg a(r) &\leftarrow \bot && \text{for all precomputed terms } r \text{ other than } \overline{1},\ \overline{2},\ \overline{3}. \end{aligned}$$

▶ **Example 8.** The propositional image of the program

$$\begin{aligned} a(\overline{1}) \vee a(\overline{2}) &\leftarrow \mathit{not}\ a(\overline{0}), \\ b(X) &\leftarrow a(X) \wedge a(X + \overline{1}) \end{aligned}$$

is

$$\begin{aligned} a(\overline{1}) \vee a(\overline{2}) &\leftarrow \neg a(\overline{0}), \\ b(\overline{n}) &\leftarrow a(\overline{n}) \wedge a(\overline{n+1}) && \text{for all } n \in \mathbf{Z}, \\ b(r) &\leftarrow a(r) \wedge \bot && \text{for all precomputed terms } r \text{ other than numerals.} \end{aligned}$$

The first two lines are similar to the propositional program from Example 2 (Section 2.2); the rules in the last line are trivial, as defined in Section 2.2.

▶ **Example 9.** The propositional image of the program

$$\begin{aligned} a(\overline{1}) &\leftarrow, \\ a(\overline{2} \times X) &\leftarrow a(X) \end{aligned}$$

is

$$\begin{aligned} a(\overline{1}) &\leftarrow \top, \\ a(\overline{2n}) &\leftarrow a(\overline{n}) && \text{for all } n \in \mathbf{Z}, \\ \top &\leftarrow a(r) && \text{for all precomputed terms } r \text{ other than numerals.} \end{aligned}$$

The first two lines are similar to the propositional program from Example 3.

▶ **Example 10.** The propositional image of the program

$$\begin{aligned} a(\overline{0}) &\leftarrow, \\ b(X, Y) &\leftarrow a(Y - X) \end{aligned}$$

is

$$\begin{aligned} a(\overline{0}) &\leftarrow \top, \\ b(\overline{m}, \overline{n}) &\leftarrow a(\overline{n-m}) && \text{for all } m, n \in \mathbf{Z}, \\ b(r, s) &\leftarrow \bot && \text{for all precomputed terms } r,\ s \text{ such that at least one of them} \\ & && \text{is not a numeral.} \end{aligned}$$

The first two lines are similar to the propositional program from Example 4.

4 Supersafe Rules

The idea of the definition below can be explained in terms of a "guessing game." Imagine that you and I are looking at a safe rule R, and I form a ground instance of R by substituting precomputed terms for its variables in such a way that all comparisons in the body of the rule become true. I do not show you that instance, but for every term that occurs as an argument of a nonnegated atom in its body I tell you what the value of that term is. If R is supersafe then on the basis of this information you will be able to find out which terms I substituted for the variables of R; or, at the very least, you will be able to restrict the possible choices to a finite set. If, on the other hand, R is not supersafe then the information that I give you will be compatible with infinitely many substitutions.

Consider, for example, the rule

$$p(X,Y,Z) \leftarrow X = \overline{5}\,..\,\overline{7} \wedge q(\overline{2} \times Y) \wedge \mathit{not}\ q(\overline{3} \times Y) \wedge Y = Z + \overline{1}. \tag{12}$$

Imagine that I chose an instance of this rule such that both comparisons in its body are true, and told you that the value of $\overline{2} \times Y$ in that instance is, for example, 10. You will be able to conclude that the value chosen for Y is 5, and that consequently the value of Z is 4. About the value that I chose for X you will be able to say that it is one of the numbers 5, 6, and 7. We see that rule (12) has only three ground instances compatible with the information about the value of $\overline{2} \times Y$ that I gave you; the rule is supersafe.

On the other hand, if we replace $\overline{2} \times Y$ in rule (12) by $\overline{0} \times Y$ then the situation will be different: I will tell you that the value of $\overline{0} \times Y$ is $\overline{0}$, and this information will not allow you to restrict the possible substitutions to a finite set. The modified rule is not supersafe.

4.1 Definition of Supersafety

A term or an atom is *interval-free* if it does not contain the interval symbol (..). We will define when a safe rule R is supersafe assuming that R satisfies the following additional condition:

all nonnegated atoms in the body of R are interval-free. (IF)

This simplifying assumption eliminates rules like the one in Example 6. It is useful because, for a term t containing intervals, the set $[t]$ may have more than one element; in the description of the guessing game we glossed over this complication when we talked above about *the* value of a term as if it were a uniquely defined object. On the other hand, if a rule R satisfies condition (IF) then for every term t occurring in a nonnegated atom in the body of a ground instance of R the set $[t]$ has at most one element. (An atom violating condition (IF) can be eliminated using a new variable; for instance, we can replace $b(\overline{4}..\overline{6})$ in Example 4 by $b(X) \wedge X = \overline{4}..\overline{6}$.)

The *positive body arguments* of R are the members of the tuples $\mathbf{t}$ for all nonnegated atoms $p(\mathbf{t})$ in the body of R. For example, the only positive body argument of (12) is $\overline{2} \times Y$. The values of positive body arguments constitute the information about an instance of the rule that is available to you in the guessing game.

The instances of a rule that are allowed in the guessing game can be characterized by "acceptable tuples of terms," defined as follows. Let $\mathbf{x}$ be the list of all variables occurring in R, and let $\mathbf{r}$ be a tuple of precomputed terms of the same length as $\mathbf{x}$. We say that $\mathbf{r}$ is *acceptable (for R)* if

$$\begin{array}{cccc} \cdots & \overline{5}, \overline{-1}, \overline{-2} & \overline{5}, \overline{0}, \overline{-1} & \overline{5}, \overline{1}, \overline{0} & \cdots \\ \cdots & \overline{6}, \overline{-1}, \overline{-2} & \overline{6}, \overline{0}, \overline{-1} & \overline{6}, \overline{1}, \overline{0} & \cdots \\ \cdots & \overline{7}, \overline{-1}, \overline{-2} & \overline{7}, \overline{0}, \overline{-1} & \overline{7}, \overline{1}, \overline{0} & \cdots \end{array}$$

Figure 1 Acceptable tuples for rules (12) and (13).

(i) for each comparison C in the body of R, $\tau_\vee(C^{\mathbf{x}}_{\mathbf{r}}) = \top$;[5]
(ii) for each positive body argument t of R, the set $[t^{\mathbf{x}}_{\mathbf{r}}]$ is non-empty (and consequently is a singleton).

For instance, a tuple r_1, r_2, r_3 is acceptable for rule (12) if

- r_1 is one of the numerals $\overline{5}$, $\overline{6}$, $\overline{7}$, so that $\tau_\vee(r_1 = \overline{5}\,..\,\overline{7}) = \top$;
- r_2 is a numeral $\overline{l}$ (rather than symbolic constant), so that the set $[\overline{2} \times r_2]$ is non-empty;
- r_3 is the numeral $\overline{l-1}$, so that $\tau_\vee(r_2 = r_3 + \overline{1}) = \top$.

(See Figure 1.)

The information about the values of positive arguments that I give you in the guessing game can be described in terms of equivalence classes of acceptable tuples. About acceptable tuples $\mathbf{r}$, $\mathbf{s}$ we say that they are *equivalent* if for each positive body argument t of rule R, $[t^{\mathbf{x}}_{\mathbf{r}}] = [t^{\mathbf{x}}_{\mathbf{s}}]$. In the case of rule (12), for example, acceptable tuples r_1, r_2, r_3 and s_1, s_2, s_3 are equivalent iff r_2 equals s_2 (so that $[\overline{2} \times r_2] = [\overline{2} \times s_2]$). In Figure 1, each column is an equivalence class of this relation.

We say that R is *supersafe* if all equivalence classes of acceptable tuples for it are finite.

For example, rule (12) is supersafe because each equivalence class of acceptable tuples for it has 3 elements. Consider now the rule obtained from (12) by replacing $\overline{2} \times Y$ with $\overline{0} \times Y$:

$$p(X, Y, Z) \leftarrow X = \overline{5}\,..\,\overline{7} \wedge q(\overline{0} \times Y) \wedge \neg q(\overline{3} \times Y) \wedge Y = Z + \overline{1}. \tag{13}$$

The set of acceptable tuples does not change, but now all of them are equivalent: for any $\mathbf{r}$ and $\mathbf{s}$,

$$[\overline{0} \times r_2] = [\overline{0} \times s_2] = \{\overline{0}\}.$$

The only equivalence class is the set of all acceptable tuples, so that the rule is not supersafe.

It is easy to check that rules (1), (5), (6) and all rules in Examples 7–9 are supersafe,[6] and that rules (2)–(4) and the second rule in Example 10 are not.

The concept of supersafety can be applied also to individual variables occurring in a rule. As before, let R be a safe rule satisfying condition (IF). Let $\mathbf{x}$ be the list of all variables $X_1, \ldots, X_n$ occurring in R, and let $\mathbf{r}$ be an acceptable tuple of precomputed terms $r_1, \ldots, r_n$. We say that a variable X_i is *supersafe* in R if, for every equivalence class E of acceptable tuples, the i-th projection of E (that is, the set of the terms r_i over all tuples $r_1, \ldots, r_n$ from E) is finite. It is easy to see that R is supersafe iff all variables occurring in R are supersafe. Indeed, a subset E of the Cartesian product of finitely many sets is finite iff all projections of E are finite.

As an example, consider the only equivalence class of acceptable tuples for rule (13), shown in Figure 1. The first projection of that set is $\{\overline{5}, \overline{6}, \overline{7}\}$; the second projection is the set of all numerals, and the third projection is the set of all numerals as well. Consequently the variable X is supersafe, and the variables Y and Z are not.

---

[5] By $C^{\mathbf{x}}_{\mathbf{r}}$ we denote the result of substituting the terms $\mathbf{r}$ for the variables $\mathbf{x}$ in C.

[6] In the syntax of Section 3.1, rules (5) and (6) would be written as $p(X, Y) \leftarrow X = Y \wedge q(X)$ and $p(X) \leftarrow X + \overline{3} = \overline{4}$.

4.2 Supersafety in the Absence of Arithmetic Operations

As could be expected, the difference between safety and supersafety disappears in the absence of arithmetic operations. In the following theorem, R is a safe rule satisfying condition (IF).

▶ **Theorem 11.** *If a positive body argument t of R does not contain the arithmetic operations*

$$+ \quad - \quad \times \quad /$$

then all variables occurring in t are supersafe.

Proof. We will show that if X_i occurs in a positive body argument t of R that does not contain arithmetic operations then the i-th projection of any equivalence class of acceptable tuples is a singleton. We will prove, in other words, that for any pair of equivalent acceptable tuples $\mathbf{r}$ and $\mathbf{s}$, $r_i = s_i$. Assume that $\mathbf{r}$ is equivalent to $\mathbf{s}$. Then $[t^{\mathbf{x}}_{\mathbf{r}}] = [t^{\mathbf{x}}_{\mathbf{s}}]$. Since t is interval-free and does not contain arithmetic operations, and $\mathbf{r}$, $\mathbf{s}$ are precomputed, both $t^{\mathbf{x}}_{\mathbf{r}}$ and $t^{\mathbf{x}}_{\mathbf{s}}$ are precomputed also, so that $[t^{\mathbf{x}}_{\mathbf{r}}]$ is the singleton $\{t^{\mathbf{x}}_{\mathbf{r}}\}$, and $[t^{\mathbf{x}}_{\mathbf{s}}]$ is the singleton $\{t^{\mathbf{x}}_{\mathbf{s}}\}$. It follows that the term $t^{\mathbf{x}}_{\mathbf{r}}$ is the same as $t^{\mathbf{x}}_{\mathbf{s}}$. Since X_i is a member of the tuple $\mathbf{x}$ and occurs in t, we can conclude that $r_i = s_i$. ◀

▶ **Corollary 12.** *If the body of R contains neither arithmetic operations nor comparisons then R is supersafe.*

Indeed, since R is safe and its body does not contain comparisons, every variable occurring in R occurs also in one of its positive body arguments.

4.3 Supersafety is Undecidable

The conditions imposed on variables by GRINGO (see Footnote 2) ensure their supersafety, but they can be relaxed without losing this property. There is no need, for example, to reject the rule

$$p(X) \leftarrow q(X/\overline{2})$$

—it is supersafe. The use of unnecessarily strong restrictions on variables in the design of GRINGO can be explained by the desire to make the grounding algorithm less complicated.

There is, however, a more fundamental reason why the class of rules accepted by GRINGO for grounding does not exactly match the class of supersafe rules:

▶ **Theorem 13.** *Membership in the class of supersafe rules is undecidable.*

Proof. The undecidable problem of determining whether a Diophantine equation has a solution [8] can be reduced to deciding whether a rule is supersafe as follows. The safe rule

$$p(Y) \leftarrow f(\mathbf{x}) = \overline{0} \times Y,$$

where $f(\mathbf{x})$ is a polynomial with integer coefficients and Y is a variable different from the members of $\mathbf{x}$, is supersafe iff the equation $f(\mathbf{x}) = 0$ has no solutions. Indeed, if the equation has no solutions then the set of acceptable tuples is empty and the rule is trivially supersafe. If it has a solution $\mathbf{r}$ then the set of acceptable tuples is infinite, because an acceptable tuple can be formed from $\mathbf{r}$ by appending any numeral $\overline{n}$. All acceptable tuples form one equivalence class, because the rule in question has no positive body arguments. ◀

5 Intelligent Instantiation

5.1 Intelligent Instantiation as Selecting Essential Instances

Consider a program Π such that its rules satisfy condition (IF). As observed in Section 4.1, for every term t occurring in a nonnegated atom in the body of a ground instance of a rule of Π, the set $[t]$ has at most one element. It follows that for every nonnegated atom A in the body of a ground instance of a rule of Π, the formula $\tau_\vee A$ is either an atom or the symbol $\bot$. Consequently the body of every rule of the propositional image $\tau\Pi$ of Π is a conjunction of formulas of three types: symbols $\top$ and $\bot$, atoms, and formulas beginning with negation. In other words, the definition of an essential rule in Section 2.2 is applicable to the propositional program $\tau\Pi$, and we can talk about its essential rules.

For instance, if Π is the program from Example 8 then the propositional program $\tau\Pi$ has two essential rules:

$$a(\overline{1}) \vee a(\overline{2}) \leftarrow \neg a(\overline{0}) \tag{14}$$

of degree 1, and

$$b(\overline{1}) \leftarrow a(\overline{1}) \wedge a(\overline{2}) \tag{15}$$

of degree 2.

If, for every k, $\tau\Pi$ has only finitely many essential rules of degree k, as in this example, then the stepwise process of generating the essential rules of $\tau\Pi$ of higher and higher degrees can be thought of as a primitive, but useful, theoretical model of the process of intelligent instantiation. It is primitive in the sense that this process involves not only identifying essential instances but also simplifying them. In application to the program from Example 8, GRINGO will not only find the essential instances (14) and (15); it will also simplify rule (14) by dropping its body.

The supersafety of all rules of a program guarantees the possibility of launching the process of intelligent instantiation, although without guarantee of termination:

▶ **Theorem 14.** *If Π is a finite program, and all rules of Π are supersafe, then each of the sets $E_k(\tau\Pi)$ is finite.*

The program from Example 10 shows that the assertion of the theorem would be incorrect without the supersafety assumption. The propositional image of that program has infinitely many essential rules of degree 2—rules $b(\overline{n}, \overline{n}) \leftarrow a(\overline{0})$ for all integers n.

5.2 Proof of Theorem 14

5.2.1 Plan of the Proof

The assertion of the theorem will be derived from the two lemmas stated below.

Consider a finite program Π such that all rules of Π are supersafe. For any rule R of Π and any set S of atoms from $\tau\Pi$, by $\rho(R, S)$ we denote the set of all tuples $\mathbf{r}$ of precomputed terms that are acceptable for R such that all nonnegated atoms of the body of $\tau(R^{\mathbf{x}}_{\mathbf{r}})$ (where $\mathbf{x}$ is the list of variables of R) belong to S.

▶ **Lemma 15.** *If R is safe and S is finite then $\rho(R, S)$ is finite.*

By S_k we denote the set of the nonnegated atoms of the heads of the rules of $E_k(\tau\Pi)$.

▶ **Lemma 16.** *Every rule of* $E_{k+1}(\tau\Pi)$ *has the form* $\tau(R^{\mathbf{x}}_{\mathbf{r}})$*, where* R *is a rule of* Π*,* $\mathbf{x}$ *is the list of its variables, and* $\mathbf{r}$ *belongs to* $\rho(R, S_k)$*.*

Given these lemmas, Theorem 14 can be proved by induction on k as follows. If $E_k(\tau\Pi)$ is finite then S_k is finite also. By Lemma 15, we can further conclude that for every rule R of Π, $\rho(R, S_k)$ is finite. Hence, by Lemma 16, $E_{k+1}(\tau\Pi)$ is finite as well.

5.2.2 Proof of Lemma 15

Let B be the set of positive body arguments of R, and let T be the set of the members of the tuples $\mathbf{t}$ for all atoms $p(\mathbf{t})$ in S. For every function ϕ from B to T, by $\rho_\phi(R, S)$ we denote the subset of $\rho(R, S)$ consisting of the tuples $\mathbf{r}$ such that $\phi(t) \in [t^{\mathbf{x}}_{\mathbf{r}}]$. We will prove the following two assertions:

Claim 1: The subsets $\rho_\phi(R, S)$ cover the whole set $\rho(R, S)$.

Claim 2: Each subset $\rho_\phi(R, S)$ is finite.

It will follow then that $\rho(R, S)$ is finite, because there are only finitely many functions from B to T.

To prove Claim 1, consider an arbitrary tuple $\mathbf{r}$ from $\rho(R, S)$. We want to find a function ϕ from B to T such that $\mathbf{r}$ belongs to $\rho_\phi(R, S)$. For every term t from B, the set $[t^{\mathbf{x}}_{\mathbf{r}}]$, where $\mathbf{x}$ is the list of variables of R, is non-empty, in view of the fact that $\mathbf{r}$, like all tuples in $\rho(R, S)$, is acceptable for R. Since t is interval-free, we can further conclude that $[t^{\mathbf{x}}_{\mathbf{r}}]$ is a singleton. Choose the only element of this set as $\phi(t)$. Let us check that $\phi(t)$ belongs to T; it will be clear then that $\mathbf{r}$ belongs to $\rho_\phi(R, S)$. Since t is a positive body argument of R, it is a member of the tuple $\mathbf{u}$ for some atom $p(\mathbf{u})$ of the body of R. Then $\tau(p(\mathbf{u}^{\mathbf{x}}_{\mathbf{r}}))$ is a nonnegated atom in the body of $\tau(R^{\mathbf{x}}_{\mathbf{r}})$. It has the form $p(\mathbf{t})$, where $\mathbf{t}$ is a tuple of terms containing $\phi(t)$. Since $\mathbf{r}$ belongs to $\rho(R, S)$, the atom $p(\mathbf{t})$ belongs to S, so that $\phi(t)$ belongs to T.

To prove Claim 2, note that all tuples from $\rho_\phi(R, S)$ are equivalent to each other. Indeed, if $\mathbf{r_1}$ and $\mathbf{r_2}$ belong to $\rho_\phi(R, S)$ then, for every t from B, $\phi(t)$ belongs both to $[t^{\mathbf{x}}_{\mathbf{r_1}}]$ and to $[t^{\mathbf{x}}_{\mathbf{r_2}}]$; since both sets are singletons, it follows that they are equal to each other. We showed, in other words, that $\rho_\phi(R, S)$ is a subset of a class of equivalent tuples. Since R is supersafe, this equivalence class is finite.

5.2.3 Proof of Lemma 16

Every rule of $\tau\Pi$ is obtained by applying τ to an instance $R^{\mathbf{x}}_{\mathbf{r}}$ of some rule R of Π. Assuming that a rule $\tau(R^{\mathbf{x}}_{\mathbf{r}})$ belongs to $E_{k+1}(\tau\Pi)$, we need to show that $\mathbf{r}$ belongs to $\rho(R, S_k)$. In other words, we need to check, first, that r is acceptable for R, and second, that all nonnegated atoms of the body of $\tau(R^{\mathbf{x}}_{\mathbf{r}})$ belong to S_k. The first property follows from the fact that all rules of $E_{k+1}(\tau\Pi)$ are nontrivial, because if $\mathbf{r}$ is not acceptable for R then the body of $\tau(R^{\mathbf{x}}_{\mathbf{r}})$ includes the conjunctive term $\bot$. According to the definition of S_k, the second property can be expressed as follows: every nonnegated atom of the body of $\tau(R^{\mathbf{x}}_{\mathbf{r}})$ is a nonnegated atom of the head of some rule of $E_k(\tau\Pi)$. This is immediate from the assumption that rule $\tau(R^{\mathbf{x}}_{\mathbf{r}})$ belongs to $E_{k+1}(\tau\Pi)$.

6 Conclusion

Supersafety is a property of rules with variables and arithmetic operations. If all rules of a program are supersafe then the process of accumulating the ground instances of its rules that are essential for finding its stable models will produce only a finite set of rules at every step.

This paper extends earlier work on the mathematics of the input language of GRINGO [5]. Unlike other publications on the theory of safe rules and intelligent instantiation in answer set programming [2, 3, 4, 6, 9], it concentrates on the difficulties related to the use of arithmetic operations. It is limited, however, to programs without GRINGO constructs that involve local variables—conditional literals and aggregates. Extending the theory of supersafety to local variables is a topic for future work.

Acknowledgements Thanks to Amelia Harrison, Roland Kaminski, Dhananjay Raju, and the anonymous referees for useful comments.

References

1 Evgenii Balai, Michael Gelfond, and Yuanlin Zhang. Towards answer set programming with sorts. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 135–147, 2013.

2 Annamaria Bria, Wolfgang Faber, and Nicola Leone. Normal form nested programs. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 2008.

3 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)*, 2009.

4 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.

5 Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz, and Torsten Schaub. Abstract Gringo. *Theory and Practice of Logic Programming*, 15:449–463, 2015.

6 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.

7 Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

8 Yuri Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

9 Norman McCain and Hudson Turner. Language independence and language tolerance in logic programs. In Pascal Van Hentenryck, editor, *Proceedings Eleventh International Conference on Logic Programming*, pages 38–57, 1994.