

Safe Rules and Intelligent Instantiation

Vladimir Lifschitz

DRAFT

Abstract

The set of rules that are accepted by the ASP grounder GRINGO as safe is rather complicated, and we propose to view it as an approximation to a more general concept of a safe rule. We introduce a simple model of the process of intelligent instantiation and show that this process can be launched, without guarantee of termination, as long as all rules of the program are safe according to the new definition. In this sense, the proposed definition is not overly general.

1 Introduction

The semantics of the input language of the ASP grounder GRINGO defined by Gebser *et al.* [2015] matches the behavior of the system CLINGO in the sense that CLINGO correctly computes the stable models as defined in that paper whenever the process of grounding terminates without error messages.¹ There are many programs, however, that GRINGO refuses to ground because some of their rules are “unsafe.” But what do we mean by an unsafe rule?

The informal discussion of safety in the Potassco User Guide² shows that this concept is quite complicated. Traditionally, a safe rule is defined as a rule such that every variable occurring in it appears also in a positive literal in its body. The User Guide explains that this description is not completely adequate, as far as the input language of GRINGO is concerned. First of all, an occurrence of a variable in the scope of an arithmetic function only counts as positive in the sense of this description for “simple arithmetic

¹The only exception, as far as we know, is related to the possibility of overflow when arithmetic operations are performed. Overflow is not reported by Version 4.5 of GRINGO, the latest available at the time of this writing.

²<http://sourceforge.net/projects/potassco/files/guide/>

terms”—terms containing a single occurrence of a variable and no arithmetic operations other than addition, subtraction, and multiplication. Moreover, if multiplication is used, then the constant factor must not evaluate to 0 for the variable occurrence to be considered positive. For instance, the rule

$\text{:- } p(X*2).$

is safe, but the rules

$\text{:- } p(X+Y).$
 $\text{:- } p(X+X).$
 $\text{:- } p(X/2).$
 $\text{:- } p(X*0).$

are not. Furthermore, occurrences of variables in inequalities are not counted as positive in the sense of the definition of safety: the rule

$p(X) \text{ :- } X > 5.$

is unsafe. This restriction does not apply to equalities: the rule

$p(X) \text{ :- } X = 5.$

is safe, according to GRINGO. “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.” Thus the rule

$p(X, Y) \text{ :- } X = Y.$

is unsafe, because the condition $X=Y$ does not allow us to find the value of any of the variables X , Y without knowing the value of the other. But the rule

$p(X, Y, Z) \text{ :- } 2*X=Y+3, Y+1=Z, q(5*Z).$

is safe: the equality $2*X=Y+3$ allows us to express X in terms of Y , the equality $Y+1=Z$ allows us to express Y in terms of Z , and Z occurs in the positive literal $q(5*Z)$.

In view of these complications, we propose to think of the safety condition implemented in the current version of GRINGO as an approximation to a more general concept of a safe rule, and to study that more general concept. It is possible that future, more sophisticated versions will recognize larger subsets of the class of rules that are safe in the sense of the definition given

below. But we will see that that there is no general algorithm for checking our safety condition, so that no safety-checking program can provide a perfect match.

If a grounder, such as GRINGO, accepts every rule of a program as safe then it starts executing an intelligent instantiation algorithm, albeit without guarantee of termination. We propose here a simplified description of the process of intelligent instantiation and show that this process can be launched—without guarantee of termination—as long as all rules of the program are safe according to the new definition. In this sense, the new definition is reasonable, not overly general.

2 A Guessing Game

The idea of the new definition of safety can be explained as follows. Imagine that you and I are looking at a rule R , and I chose an instance of R that is nontrivial, in the sense that the arithmetic literals in the body of that instance are true. I didn't show you that instance. But for every term that occurs as an argument of a positive, non-arithmetic literal in the body of that instance, I told you what the value of that term is. If R is safe then on the basis of this information you will be able to find out which values are substituted for the variables in the instance that I chose or, at the very least, you'll be able to restrict the possible choices to a finite set. If, on the other hand, R is unsafe then the information that I gave you will be compatible with infinitely many substitutions.

Consider, for instance, the rule

$$p(X,Y,Z) \text{ :- } X=5..7, \text{ } q(2*Y), \text{ } Y=Z+1.$$

This rule is similar to the last example from the introduction, except that its body contains an interval term, $5..7$. (The arithmetic literal $X = 5..7$ expresses that X is an integer between 5 and 7.) Imagine that I chose an instance of this rule such that both arithmetic literals in its body are true, and told you that the value of $2*Y$ in that instance is, for example, 10. You will be able to conclude that the value of Y in the instance that I chose is 5, and that consequently the value of Z is 4. About the value of X you will be able to say that it is one of the numbers 5, 6, 7. We see that there are only three instances compatible with the information about the value of $2*Y$ that I gave you; the rule is safe.

On the other hand, if we replace $2*Y$ in this rule by $0*Y$ then the rule will become unsafe: I'll tell you that the value of $0*Y$ is 0, and this

information will not allow you to restrict the possible substitutions to a finite set.

In the next section we turn this idea into a precise definition in the context of the language AG [Gebser *et al.*, 2015]—a subset of the input language of GRINGO that uses “abstract” notation, which is convenient for theoretical analysis. For instance, the rule used above as an example, rewritten in the syntax of AG, will look like this:

$$p(X, Y, Z) \leftarrow X = \bar{5}..\bar{7} \wedge q(\bar{2} \times Y) \wedge Y = Z + \bar{1}. \quad (1)$$

(In AG, $\bar{n}$ is the symbol, “numeral,” representing the integer n).

3 Safe Rules

In the definition of safety below we assume that the body of the rule satisfies two conditions.

Condition 1. The body is a conjunction of symbolic and arithmetic literals. In other words, we allow neither conditional literals (other than symbolic and arithmetic) nor aggregate literals.

This condition implies that every conjunctive term of the body is

- (a) an arithmetic literal, or
- (b) an atom, or
- (c) an expression beginning with negation.

It implies also that all variables occurring in the rule are global.

The second condition is a restriction on the atoms of the body, that is, on its conjunctive members of type (b).

Condition 2. The interval symbol ($..$) and the pooling symbol ($;$) don’t occur in the atoms of the body. In other words, each of these atoms has the form $p(\mathbf{t})$, where $\mathbf{t}$ is an interval-free tuple of terms.

This assumption is useful in view of the fact that the object $[t]$, which represents the meaning of a ground term t [Gebser *et al.*, 2015, Section 4.2], is a set that may include several terms. For instance, $[\bar{2} + \bar{2}]$ is the singleton $\{4\}$, but

$$\begin{aligned} [\bar{5}..\bar{7}] &= \{\bar{5}, \bar{6}, \bar{7}\}, \\ [\bar{5}; \bar{7}; abc] &= \{\bar{5}, \bar{7}, abc\}. \end{aligned}$$

We glossed over this complication when we talked in Section 2 about *the* value of a term as if it were a uniquely defined object. But if t is interval-free then the cardinality of $[t]$ is at most 1. It is still possible that this set is empty; for instance,

$$[\overline{5}..\overline{2}] = [\overline{1}/\overline{0}] = [\overline{7} + abc] = \emptyset.$$

We address this difficulty by including clause (ii) in the definition of an acceptable tuple below.

Consider a rule R satisfying Conditions 1 and 2. The *positive body arguments* of R are the members of the tuples $\mathbf{t}$ for all atoms $p(\mathbf{t})$ of the body of r . For instance, the only positive body argument of rule (1) 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 described above.

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 terms³ of the same length as $\mathbf{x}$. We say that $\mathbf{r}$ is *acceptable (for R)* if

- (i) for each arithmetic literal L in the body of R , the ground literal $L_{\mathbf{r}}^{\mathbf{x}}$ is true;⁴
- (ii) for each positive body argument t of R , the set $[t_{\mathbf{r}}^{\mathbf{x}}]$ is non-empty (and consequently is a singleton).

For instance, a tuple r_1, r_2, r_3 is acceptable for rule (1) if

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

(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

³A ground term is said to be precomputed if it contains neither arithmetic operations nor intervals [Gebser *et al.*, 2015, Section 2.1].

⁴Here $L_{\mathbf{r}}^{\mathbf{x}}$ stands for the result of substituting the terms $\mathbf{r}$ for the variables $\mathbf{x}$ in L . In the notation of [Gebser *et al.*, 2015, Section 4.3], this condition can be written as $\tau_{\vee}(L_{\mathbf{r}}^{\mathbf{x}}) = \top$.

...	$\bar{5}, \bar{-1}, \bar{-2}$	$\bar{5}, \bar{0}, \bar{-1}$	$\bar{5}, \bar{1}, \bar{0}$	...
...	$\bar{6}, \bar{-1}, \bar{-2}$	$\bar{6}, \bar{0}, \bar{-1}$	$\bar{6}, \bar{1}, \bar{0}$	...
...	$\bar{7}, \bar{-1}, \bar{-2}$	$\bar{7}, \bar{0}, \bar{-1}$	$\bar{7}, \bar{1}, \bar{0}$	...

Figure 1: Acceptable tuples for rules (1) and (2).

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 , $[t_{\mathbf{r}}^{\mathbf{x}}] = [t_{\mathbf{s}}^{\mathbf{x}}]$. For instance, in the case of rule (1) 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 $[\bar{2} \times r_2] = [\bar{2} \times s_2]$). In Figure 1, each column is an equivalence class of this relation.

Definition. A rule is *safe* if all equivalence classes of acceptable tuples for it are finite.

For instance, rule (1) is safe because each equivalence class of acceptable tuples for it has 3 elements. Consider, on the other hand, the rule obtained from (1) by replacing $\bar{2} \times Y$ with $\bar{0} \times Y$:

$$p(X, Y, Z) \leftarrow X = \bar{5}.. \bar{7} \wedge q(\bar{0} \times Y) \wedge Y = Z + \bar{1}. \quad (2)$$

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

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

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

Theorem 1. *Membership in the class of safe rules is undecidable.*

Proof. The undecidable problem of determining whether a Diophantine equation has a solution [Matiyasevich, 1993] can be reduced to deciding whether a rule is safe as follows. The rule

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

where $f(\mathbf{x})$ is a polynomial with integer coefficients written in the syntax of AG and Y is a variable different from the members of $\mathbf{x}$, is safe iff the equation in the body has no solutions. Indeed, if the equation has no solutions then the set of acceptable tuples is empty and the rule is trivially safe. 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 $\bar{n}$ (or any other precomputed term). All acceptable tuples form one equivalence class, because the rule in question has no body arguments.

4 Safe Variables

We can talk not only about the safety of a rule as a whole, but also about the safety of individual variables occurring in it. As before, let R be a rule satisfying Conditions 1 and 2, let $\mathbf{x}$ be the list of all variables $X_1, \dots, X_n$ occurring in R , and let $\mathbf{r}$ be an acceptable tuple of precomputed terms $r_1, \dots, r_n$.

Definition. A variable X_i is *safe (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 $r_1, \dots, r_n$ from E) is finite.

A rule is safe iff all its variables are safe. Indeed, a subset of the cross product of finitely many sets is finite iff all its projections are finite.

As an example, consider the only equivalence class of acceptable tuples for rule (2), that is, the set of all tuples in Figure 1. The first projection of that set is the 3-element set $\{\bar{5}, \bar{6}, \bar{7}\}$; the second projection is the set of all numerals; the third projection is the set of all numerals as well. Consequently the variable X is safe in this rule, and the variables Y and Z are not safe.

We pointed out in the introduction that the traditional understanding of a safe variable as a variable that appears in a positive literal in the body is unsatisfactory when that literal is an arithmetic literal, and also when the occurrence of the variable is in the scope of an arithmetic operation. The following proposition shows that these are the only two cases when the traditional safety condition doesn't imply safety in the sense of this paper.

Theorem 2. *If a term t is a positive body argument of a rule R and it doesn't contain the arithmetic operations*

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

then all variables occurring in t are safe in R .

Proof. We will show that if X_i occurs in a positive body argument t of R that doesn't 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 $[t_{\mathbf{r}}^{\mathbf{x}}] = [t_{\mathbf{s}}^{\mathbf{x}}]$. Since t is interval-free and doesn't contain arithmetic operations, and $\mathbf{r}, \mathbf{s}$ are precomputed, both $t_{\mathbf{r}}^{\mathbf{x}}$ and $t_{\mathbf{s}}^{\mathbf{x}}$ are precomputed also, so that $[t_{\mathbf{r}}^{\mathbf{x}}]$ is the singleton $\{t_{\mathbf{r}}^{\mathbf{x}}\}$, and $[t_{\mathbf{s}}^{\mathbf{x}}]$ is the singleton $\{t_{\mathbf{s}}^{\mathbf{x}}\}$. It follows that the term $t_{\mathbf{r}}^{\mathbf{x}}$ is the same as $t_{\mathbf{s}}^{\mathbf{x}}$. Since X_i is a member of the tuple $\mathbf{x}$ and occurs in t , we can conclude that $r_i = s_i$.

5 Intelligent Instantiation as Selection

As a step towards the study of the relationship between safety and the process of intelligent instantiation, we present in this section a simplified description of that process.

Intelligent instantiation allows us to find the instances of rules that are “essential” for generating the stable models of the given program. To illustrate this idea, consider the AG program consisting of two rules

$$\begin{aligned} p(\bar{5}), \\ p(\bar{2} \times X) \leftarrow p(X). \end{aligned} \tag{3}$$

The set of instances of the second rule includes the rules

$$p(\bar{2} \times \bar{n}) \leftarrow p(\bar{n})$$

for all integers n . But this instance is “essential” only if n has the form $5 \cdot 2^k$. Disregarding the other instances of that rule will have no effect on the stable model of program (3). As another example, take the program

$$\begin{aligned} a(\bar{1}.. \bar{3}), \\ b(\bar{3}) \vee b(\bar{4}), \\ c(X) \leftarrow a(X) \wedge b(X). \end{aligned} \tag{4}$$

The only instance of the last rule that is essential for finding the stable models of this program is

$$c(\bar{3}) \leftarrow a(\bar{3}) \wedge b(\bar{3}). \tag{5}$$

We will view intelligent instantiation as the process of generating the essential instances of the given rules. Intelligent instantiation algorithms do actually more than weeding out inessential instances; they also simplify. Given program (4), for example, GRINGO simplifies the instance (5) of the last rule and replaces it by

$$c(\bar{3}) \leftarrow b(\bar{3}).$$

But this aspect of the process of intelligent instantiation is not relevant to the issue at hand—the role of safety.

The semantics of AG programs is defined by Gebser *et al.* [2015] using a translation τ that turns AG rules and programs into sets of infinitary propositional formulas. The stable models of an AG program Π are characterized in terms of the stable models of $\tau\Pi$ in the sense of Truszczyński [2012]. If all rules of an AG program satisfy Condition 1 from Section 3 then each

formula in $\tau\Pi$ is an implication such that its antecedent and consequent are finite formulas formed from atoms and the 0-place connectives $\top$, $\perp$ using conjunction, disjunction, and negation. Such formulas can be viewed as programs with nested expressions (without classical negation)⁵ in the sense of Lifschitz *et al.* [1999]. For instance, the result of applying τ to program (3), written as a program with nested expressions, consists of the rules

$$\begin{aligned} p(\bar{5}) &\leftarrow \top, \\ p(\overline{2n}) &\leftarrow p(\bar{n}) \quad \text{for all integers } n, \\ \top &\leftarrow p(r) \quad \text{for all precomputed terms } r \text{ other than numerals.} \end{aligned} \tag{6}$$

The result of applying τ to (4) consists of the rules

$$\begin{aligned} a(\bar{1}) \wedge a(\bar{2}) \wedge a(\bar{3}) &\leftarrow \top, \\ b(\bar{3}) \vee b(\bar{4}) &\leftarrow \top, \\ c(r) &\leftarrow a(r) \wedge b(r) \end{aligned} \tag{7}$$

for all precomputed terms r .⁶

If an AG program Π satisfies not only Condition 1, but also Condition 2, then we can assert that the body of each rule of $\tau\Pi$ is a conjunction of formulas of three forms, corresponding to the forms (a)–(c) of the conjunctive terms of the bodies of the rules of Π (see Section 3):⁷

- (a') 0-place connectives $\top$ and $\perp$,
- (b') atoms,
- (c') formulas beginning with negation.

In the rest of this section and in the section that follows, Γ is a program with nested expressions such that the body of each of its rules is a conjunction of formulas of the forms (a')–(c'). We will define which rules of Γ are considered essential. The following terminology will be useful. 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 any negation. A rule from Γ is *trivial* if at least one of the conjunctive terms of its body is $\perp$.

The subsets $E_k(\Gamma)$ of Γ ($k \geq 0$) are defined as follows:

⁵Technically, there is no classical negation in AG: it is simulated by "negated constants" [Gebser *et al.*, 2015, Section 2.1].

⁶In this paper, when we write a program with nested expressions, the standard symbols for conjunction and disjunction are used, rather than commas and semicolons as in the original publication.

⁷Condition 2 is needed here because the result of applying τ to a ground atom in the body is, generally, a disjunction of several atoms, rather than a single atom. For instance, τ turns $p(\bar{1}.\bar{3})$ in the body of a rule into $p(\bar{1}) \vee p(\bar{2}) \vee p(\bar{3})$.

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

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 that if $E_{k+1}(\Gamma) = E_k(\Gamma)$ then $E_l(\Gamma) = E_k(\Gamma)$ for all l that are greater than k .

Definition. The *essential part* $E(\Gamma)$ of Γ is the set $\bigcup_{k \geq 0} E_k(\Gamma)$.

For example, if Γ is (6) then

$$\begin{aligned}
E_0(\Gamma) &= \emptyset, \\
E_1(\Gamma) &= \{p(\bar{5}) \leftarrow \top\}, \\
E_2(\Gamma) &= \{p(\bar{5}) \leftarrow \top, p(\bar{10}) \leftarrow p(\bar{5})\}, \\
E_3(\Gamma) &= \{p(\bar{5}) \leftarrow \top, p(\bar{10}) \leftarrow p(\bar{5}), p(\bar{20}) \leftarrow p(\bar{10})\}, \\
&\dots,
\end{aligned}$$

so that

$$E(\Gamma) = \{p(\bar{5}) \leftarrow \top\} \cup \{p(\overline{5 \cdot 2^{k+1}}) \leftarrow p(\overline{5 \cdot 2^k}) : k \geq 0\}.$$

If Γ is (7) then

$$\begin{aligned}
E_0(\Gamma) &= \emptyset, \\
E_1(\Gamma) &= \{a(\bar{1}) \wedge a(\bar{2}) \wedge a(\bar{3}) \leftarrow \top, b(\bar{3}) \vee b(\bar{4}) \leftarrow \top\}, \\
E_2(\Gamma) &= \{a(\bar{1}) \wedge a(\bar{2}) \wedge a(\bar{3}) \leftarrow \top, b(\bar{3}) \vee b(\bar{4}) \leftarrow \top, c(\bar{3}) \leftarrow a(\bar{3}) \wedge b(\bar{3})\}, \\
E_3(\Gamma) &= E_2(\Gamma),
\end{aligned}$$

so that

$$E(\Gamma) = \{a(\bar{1}) \wedge a(\bar{2}) \wedge a(\bar{3}) \leftarrow \top, b(\bar{3}) \vee b(\bar{4}) \leftarrow \top, c(\bar{3}) \leftarrow a(\bar{3}) \wedge b(\bar{3})\}.$$

The following proposition shows that the essential part of a program does indeed contain all rules needed for finding its stable models.

Theorem 3. Γ and $E(\Gamma)$ have the same stable models.

6 Proof of Theorem 3

Lemma 1. *Let Δ be a subset of 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 every nontrivial rule of $\Gamma \setminus \Delta$ contains a nonnegated atom that doesn't 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 that 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 doesn't 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.⁸ 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 .

Proof of Theorem 3. Let H be the set 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

⁸We refer here to reducts in the sense of Lifschitz *et al.* [1999]. The process of constructing the reduct defined in that paper replaces some formulas beginning with negation by $\top$, and some by $\perp$, so that all occurrences of negation are eliminated.

all nonnegated atoms in the head of R . Then R belongs to $E_{k+1}(\Gamma)$, and consequently to $E(\Gamma)$.

7 Relationship between Safety and Instantiation

Theorem 3 can be used for generating the stable models of an AG program Π that satisfies our two simplifying assumptions (Section 3) if the essential part of $\tau\Pi$ is finite. There can be infinitely many rules in $\tau\Pi$ even if all rules of Π are safe, as in the case of program (3). But safety does guarantee the finiteness of the approximations used in the definition of $E(\Gamma)$:

Theorem 4. *If all rules of Π are safe then each of the sets $E_k(\tau\Pi)$ is finite.*

In other words, if all rules of Π are safe then we can at least launch the (possibly nonterminating) process of generating the essential rules of $\tau\Pi$ by calculating the sets $E_k(\tau\Pi)$ for larger and larger values of k .

To illustrate the need for the safety assumption in Theorem 4, consider the one-rule program

$$p(X) \leftarrow X \neq \bar{0}. \quad (8)$$

After applying τ , we get the rules

$$\begin{aligned} p(\bar{0}) &\leftarrow \perp, \\ p(r) &\leftarrow \top \quad \text{for all precomputed terms } r \text{ other than } \bar{0}. \end{aligned} \quad (9)$$

If Γ is (9) then

$$\begin{aligned} E_0(\Gamma) &= \emptyset, \\ E_1(\Gamma) &= \{p(r) \leftarrow \top : r \neq \bar{0}\}, \\ E_2(\Gamma) &= E_1(\Gamma), \end{aligned}$$

so that $E(\Gamma)$ is the set of rules in the second line of (9):

$$E(\Gamma) = \{p(r) \leftarrow \top : r \neq \bar{0}\}.$$

The whole infinite set of essential rules is generated here at one step.

8 Proof of Theorem 4

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

Consider an AG program Π satisfying Conditions 1 and 2 from Section 3. 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 $\tau(R_{\mathbf{r}}^{\mathbf{x}})$ (where $\mathbf{x}$ is the list of variables of R) belong to S .

Lemma 2. *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 3. *Every rule of $E_{k+1}(\tau\Pi)$ has the form $\tau(R_{\mathbf{r}}^{\mathbf{x}})$, 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, the theorem can be proved by induction on k as follows. Assume that all rules of Π are safe. If $E_k(\tau\Pi)$ is finite then S_k is finite also. By Lemma 2, we can further conclude that for every rule R of Π , $\rho(R, S_k)$ is finite. Hence, by Lemma 3, $E_{k+1}(\tau\Pi)$ is finite as well.

Proof of Lemma 2. 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 $\phi(t) \in [t_{\mathbf{r}}^{\mathbf{x}}]$. 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 $[t_{\mathbf{r}}^{\mathbf{x}}]$, 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 $[t_{\mathbf{r}}^{\mathbf{x}}]$ 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 $\tau(p(\mathbf{u}_{\mathbf{r}}^{\mathbf{x}}))$ is a nonnegated body element of $\tau(R_{\mathbf{r}}^{\mathbf{x}})$. 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 all tuples from $\rho_{\phi}(R, S)$ are equivalent to each other in the sense of Section 3. 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 $[t_{\mathbf{r}_1}^{\mathbf{x}}]$ and to $[t_{\mathbf{r}_2}^{\mathbf{x}}]$; 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 safe, this equivalence class is finite.

Proof of Lemma 3. Every rule of $\tau\Pi$ is obtained by applying τ to an instance $R_{\mathbf{r}}^{\mathbf{x}}$ of some rule R of Π [Gebser *et al.*, 2015, Section 4.8]. Assuming that a rule $\tau(R_{\mathbf{r}}^{\mathbf{x}})$ 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 $\tau(R_{\mathbf{r}}^{\mathbf{x}})$ 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 $\tau(R_{\mathbf{r}}^{\mathbf{x}})$ includes the conjunctive term $\perp$. According to the definition of S_k , the second property can be expressed as follows: every nonnegated atom of the body of $\tau(R_{\mathbf{r}}^{\mathbf{x}})$ is a nonnegated atom of the head of some rule of $E_k(\tau\Pi)$. This is immediate from the assumption that the rule $\tau(R_{\mathbf{r}}^{\mathbf{x}})$ belongs to $E_{k+1}(\tau\Pi)$.

9 Conclusion

We proposed a definition of a safe rule for a subset of the input language of GRINGO. Membership in the class of safe rules in the sense of this note is undecidable. If all rules of a program are safe then it is possible to launch the process of generating the instances of its rules that are essential for finding its stable models.

Available publications on the theory of safe rules and intelligent instantiation in answer set programming [McCain and Turner, 1994, Lee *et al.*, 2008, Bria *et al.*, 2008, Calimeri *et al.*, 2008, Cabalar *et al.*, 2009] didn't cover rules with arithmetic constructs, which play an important role in this paper.

The work presented here is restricted by two simplifying assumptions, which guarantee that all variables are global.

Acknowledgements

Thanks to Amelia Harrison, Roland Kaminski, and Dhananan Raju for comments on earlier drafts. This research was supported in part by the National Science Foundation under Grant IIS-1422455.

References

[Bria *et al.*, 2008] Annamaria Bria, Wolfgang Faber, and Nicola Leone. Normal form nested programs. In *Proceedings of European Conference on*

Logics in Artificial Intelligence (JELIA), 2008.

- [Cabalar *et al.*, 2009] 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.
- [Calimeri *et al.*, 2008] 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.
- [Gebser *et al.*, 2015] Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz, and Torsten Schaub. Abstract Gringo. In *Proceedings of International Conference on Logic Programming (ICLP)*, 2015. <http://www.cs.utexas.edu/users/vl/papers/AG.pdf>; to appear.
- [Lee *et al.*, 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.
- [Lifschitz *et al.*, 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.
- [Matiyasevich, 1993] Yuri Matiyasevich. *Hilbert’s Tenth Problem*. MIT Press, 1993.
- [McCain and Turner, 1994] Norman McCain and Hudson Turner. Language independence and language tolerance in logic programs. In Pascal Van Hentenryck, editor, *Proceedings Eleventh Int’l Conf. on Logic Programming*, pages 38–57, 1994.
- [Truszczyński, 2012] Mirosław Truszczyński. Connecting first-order ASP and the logic FO(ID) through reducts. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning: Essays on Logic-Based AI in Honor of Vladimir Lifschitz*, pages 543–559. Springer, 2012.