On the Semantics of Gringo

Amelia Harrison, Vladimir Lifschitz, and Fangkai Yang

University of Texas, Austin, Texas, USA
{ameliaj, vl, fkyang}@cs.utexas.edu

Abstract. Input languages of answer set solvers are based on the mathematically simple concept of a stable model. But many useful constructs available in these languages, including local variables, conditional literals, and aggregates, cannot be easily explained in terms of stable models in the sense of the original definition of this concept and its straightforward generalizations. Manuals written by designers of answer set solvers usually explain such constructs using examples and informal comments that appeal to the user's intuition, without references to any precise semantics. We propose to approach the problem of defining the semantics of GRINGO programs by translating them into the language of infinitary propositional formulas. This semantics allows us to study equivalent transformations of GRINGO programs using natural deduction in infinitary propositional logic.

1 Introduction

In this note, Gringo is the name of the input language of the grounder GRINGO,[1] which is used as the front end in many answer set programming (ASP) systems. Several releases of GRINGO have been made public, and more may be coming in the future; accordingly, we can distinguish between several "dialects" of the language Gringo. We concentrate here on Version 4, released in March of 2013. (It differs from Version 3, described in the *User's Guide* dated October 4, 2010,[2] in several ways, including the approach to aggregates—it is modified as proposed by the ASP Standardization Working Group.[3])

The basis of Gringo is the language of logic programs with negation as failure, with the syntax and semantics defined in [6]. Our goal here is to extend that semantics to a larger subset of Gringo. Specifically, we would like to cover arithmetical functions and comparisons, conditions, and aggregates.[4]

[1] `http://potassco.sourceforge.net/.`

[2] The *User's Guide* can be downloaded from the Potassco website (Footnote 1). It is posted also at `http://www.cs.utexas.edu/users/vl/teaching/lbai/clingo_guide.pdf`.

[3] `https://www.mat.unical.it/aspcomp2013/ASPStandardization.`

[4] The subset of Gringo discussed in this note includes also constraints, disjunctive rules, and choice rules, treated along the lines of [7] and [3]. The first of these papers introduces also "classical" (or "strong") negation—a useful feature that we do not include. (Extending our semantics of Gringo to programs with classical negation

Our proposal is based on the informal and sometimes incomplete description of the language in the *User's Guide*, on the discussion of ASP programming constructs in [4], on experiments with GRINGO, and on the clarifications provided in response to our questions by its designers.

The proposed semantics uses a translation from Gringo into the language of infinitary propositional formulas—propositional formulas with infinitely long conjunctions and disjunctions. Including infinitary formulas is essential, as we will see, when conditions or aggregates use variables ranging over infinite sets (for instance, over integers).

Alternatively, the semantics of Gringo can be approached using quantified equilibrium logic [12] or its syntactic counterpart defined in [2]. This method involves translating rules into the language of first-order logic. For instance, the rule

$$p(Y) \leftarrow \mathit{count}\{X, Y : q(X, Y)\} \geq 1 \tag{1}$$

can be represented by the sentence

$$\forall y(\exists x Q(x, y) \rightarrow P(y)).$$

However, this translation is not sufficiently general. For instance, it is not clear how to represent the rule

$$\mathit{total\_hours}(N) \leftarrow \mathit{sum}\{H, C : \mathit{enroll}(C), \mathit{hours}(H, C)\} = N \tag{2}$$

from Section 3.1.10 of the Gringo 3 *User's Guide* with a first-order formula. One reason is that the aggregate *sum* is used here instead of *count*. The second difficulty is that the variable N is used rather than a constant.

General aggregate expressions, as used in Gringo, can be represented by first-order formulas with generalized quantifiers.[5] The advantage of infinitary propositional formulas as the target language is that properties of these formulas, and of their stable models, are better understood. We may be able to prove, for instance, that two Gringo programs have the same stable models by observing that the corresponding infinitary formulas are equivalent in one of the natural deduction systems discussed in [8]. We give here several examples of reasoning about Gringo programs based on this idea.

The process of converting Gringo programs into infinitary propositional formulas defined in this note uses substitutions to eliminate variables. This form of grounding is quite different, of course, from the process of intelligent instantiation implemented in GRINGO and other grounders. Mathematically, it is much simpler than intelligent instantiation; as a computational procedure, it is much less efficient, not to mention the fact that sometimes it produces infinite objects. Like grounding in the original definition of a stable model [6], it is modular, in the sense that it applies to the program rule by rule, and it is applicable even if

is straightforward, using the process of eliminating classical negation in favor of additional atoms described in [7, Section 4].)

[5] Stable models of formulas with generalized quantifiers are defined by Lee and Meng [9][10][11].

the program is not safe. From this perspective, GRINGO's safety requirement is an implementation restriction.

Our description of the syntax of Gringo disregards some of the features related to representing programs as strings of ASCII characters, such as using `:-` to separate the head from the body, using semicolons, rather than parentheses, to indicate the boundaries of a conditional literal, and representing falsity (which we denote here by $\bot$) as `#false`. Since the subset of Gringo discussed in this note does not include assignments, we can disregard also the requirement that equality be represented by two characters `==`.

2 Syntax

We begin with a signature σ in the sense of first-order logic that includes, among others,

(i) numerals—object constants representing all integers,
(ii) arithmetical functions—binary function constants $+$, $-$, $\times$,
(iii) comparisons—binary predicate constants $<$, $>$, $\leq$, $\geq$.

We will identify numerals with the corresponding elements of the set $\mathbf{Z}$ of integers. Object, function, and predicate symbols not listed under (i)–(iii) will be called *symbolic*. A term over σ is *arithmetical* if it does not contain symbolic object or function constants. A ground term is *precomputed* if it does not contain arithmetical functions.

We assume that in addition to the signature, a set of symbols called *aggregate names* is specified, and that for each aggregate name α, *the function denoted by* α, $\widehat{\alpha}$, maps every tuple of precomputed terms to an element of $\mathbf{Z} \cup \{\infty, -\infty\}$.

Examples. The functions denoted by the aggregate names *count*, *max*, and *sum* are defined as follows. For any set T of tuples of precomputed terms,

- $\widehat{count}(T)$ is the cardinality of T if T is finite, and ∞ otherwise;
- $\widehat{max}(T)$ is the least upper bound of the set of the integers t_1 over all tuples $(t_1, \dots, t_m)$ from T in which t_1 is an integer;
- $\widehat{sum}(T)$ is the sum of the integers t_1 over all tuples $(t_1, \dots, t_m)$ from T in which t_1 is a positive integer; it is ∞ if there are infinitely many such tuples.[6]

A *literal* is an expression of one of the forms

$$p(t_1, \dots, t_k),\ t_1 = t_2,\ \mathit{not}\ p(t_1, \dots, t_k),\ \mathit{not}\ (t_1 = t_2)$$

where p is a symbolic predicate constant of arity k, and each t_i is a term over σ, or

$$t_1 \prec t_2,\ \mathit{not}\ (t_1 \prec t_2)$$

[6] To allow negative numbers in this example, we would have to define summation for a set that contains both infinitely many positive numbers and infinitely many negative numbers. It is unclear how to do this in a natural way.

where $\prec$ is a comparison, and t_1, t_2 are arithmetical terms. A *conditional literal* is an expression of the form $H : \mathbf{L}$, where H is a literal or the symbol $\bot$, and $\mathbf{L}$ is a list of literals, possibly empty. The members of $\mathbf{L}$ will be called *conditions*. If $\mathbf{L}$ is empty then we will drop the colon after H, so that every literal can be viewed as a conditional literal.

Example. If *available* and *person* are unary predicate symbols then

$$available(X) : person(X)$$

and

$$\bot : (person(X), not\ available(X))$$

are conditional literals.

An *aggregate expression* is an expression of the form

$$\alpha\{\mathbf{t} : \mathbf{L}\} \prec s$$

where α is an aggregate name, $\mathbf{t}$ is a list of terms, $\mathbf{L}$ is a list of literals, $\prec$ is a comparison or the symbol $=$, and s is an arithmetical term.

Example. If *enroll* is a unary predicate symbol and *hours* is a binary predicate symbol then

$$sum\{H, C : enroll(C), hours(H, C)\} = N$$

is an aggregate expression.

A *rule* is an expression of the form

$$H_1 \mid \cdots \mid H_m \leftarrow B_1, \ldots, B_n \tag{3}$$

$(m, n \geq 0)$, where each H_i is a conditional literal, and each B_i is a conditional literal or an aggregate expression. A *program* is a set of rules.

If p is a symbolic predicate constant of arity k, and $\mathbf{t}$ is a k-tuple of terms, then

$$\{p(\mathbf{t})\} \leftarrow B_1, \ldots, B_n$$

is shorthand for

$$p(\mathbf{t}) \mid not\ p(\mathbf{t}) \leftarrow B_1, \ldots, B_n.$$

Example. For any positive integer n,

$$\begin{array}{ll} \{p(i)\}\leftarrow & (i = 1, \ldots, n), \\ \quad \leftarrow p(X), p(Y), p(X+Y) & \end{array} \tag{4}$$

is a program.

3 Semantics

We will define the semantics of Gringo using a syntactic transformation τ. It converts Gringo rules into infinitary propositional combinations of atoms of the form $p(\mathbf{t})$, where p is a symbolic predicate constant, and $\mathbf{t}$ is a tuple of precomputed terms. Then the stable models of a program will be defined as stable models, in the sense of [13], of the set consisting of the translations of all rules of the program. Truszczynski's definition of stable models for infinitary propositional formulas is reviewed below.

Prior to defining the translation τ for rules, we will define it for ground literals, conditional literals, and aggregate expressions.

3.1 Review: Stable Models of Infinitary Formulas

Let σ be a propositional signature, that is, a set of propositional atoms. The sets $\mathcal{F}_0^\sigma$, $\mathcal{F}_1^\sigma$, ... are defined as follows:

- $\mathcal{F}_0^\sigma = \sigma \cup \{\bot\}$,
- $\mathcal{F}_{i+1}^\sigma$ is obtained from $\mathcal{F}_i^\sigma$ by adding expressions $\mathcal{H}^\wedge$ and $\mathcal{H}^\vee$ for all subsets $\mathcal{H}$ of $\mathcal{F}_i^\sigma$, and expressions $F \to G$ for all $F, G \in \mathcal{F}_i^\sigma$.

The elements of $\bigcup_{i=0}^\infty \mathcal{F}_i^\sigma$ are called *(infinitary) formulas* over σ. Negation and equivalence are abbreviations.

Subsets of a signature σ will be also called its *interpretations*. The satisfaction relation between an interpretation and a formula is defined in a natural way.

The *reduct* F^I of a formula F w.r.t. an interpretation I is defined as follows:

- $\bot^I = \bot$.
- For $p \in \sigma$, $p^I = \bot$ if $I \not\models p$; otherwise $p^I = p$.
- $(\mathcal{H}^\wedge)^I = \{G^I \mid G \in \mathcal{H}\}^\wedge$.
- $(\mathcal{H}^\vee)^I = \{G^I \mid G \in \mathcal{H}\}^\vee$.
- $(G \to H)^I = \bot$ if $I \not\models G \to H$; otherwise $(G \to H)^I = G^I \to H^I$.

An interpretation I is a *stable model* of a set $\mathcal{H}$ of formulas if it is minimal w.r.t. set inclusion among the interpretations satisfying the reducts of all formulas from $\mathcal{H}$.

3.2 Semantics of Well-Formed Ground Literals

A term $\mathbf{t}$ is *well-formed* if it contains neither symbolic object constants nor symbolic function constants in the scope of arithmetical functions. For instance, all arithmetical terms and all precomputed terms are well-formed; $c+2$ is not well-formed. The definition of "well-formed" for literals, aggregate expressions, and so forth is the same.

For every well-formed ground term t, by $[t]$ we denote the precomputed term obtained from t by evaluating all arithmetical functions, and similarly for tuples of terms. For instance, $[f(2+2)]$ is $f(4)$.

The translation τL of a well-formed ground literal L is defined as follows:

– $\tau(p(\mathbf{t}))$ is $p([\mathbf{t}])$;
– $\tau(t_1 \prec t_2)$, where $\prec$ is the symbol $=$ or a comparison, is $\top$ if the relation $\prec$ holds between $[t_1]$ and $[t_2]$, and $\bot$ otherwise;
– $\tau(not\ A)$ is $\neg\tau A$.

For instance, $\tau(not\ p(f(2{+}2)))$ is $\neg p(f(4))$, and $\tau(2{+}2{=}\,4)$ is $\top$.

Furthermore, $\tau\bot$ stands for $\bot$, and, for any list $\mathbf{L}$ of ground literals, $\tau\mathbf{L}$ is the conjunction of the formulas τL for all members L of $\mathbf{L}$.

3.3 Global Variables

About a variable we say that it is *global*

– in a conditional literal $H : \mathbf{L}$, if it occurs in H but does not occur in $\mathbf{L}$;
– in an aggregate expression $\alpha\{\mathbf{t} : \mathbf{L}\} \prec s$, if it occurs in the term s;
– in a rule (3), if it is global in at least one of the expressions H_i, B_i.

For instance, the head of the rule (2) is a literal with the global variable N, and its body is an aggregate expression with the global variable N. Consequently N is global in the rule as well.

A conditional literal, an aggregate expression, or a rule is *closed* if it has no global variables. An *instance* of a rule R is any well-formed closed rule that can be obtained from R by substituting precomputed terms for global variables. For instance,

$$total\_hours(6) \leftarrow sum\{H, C : enroll(C), hours(H, C)\} = 6$$

is an instance of rule (2). It is clear that if a rule is not well-formed then it has no instances.

3.4 Semantics of Closed Conditional Literals

If t is a term, $\mathbf{x}$ is a tuple of distinct variables, and $\mathbf{r}$ is a tuple of terms of the same length as $\mathbf{x}$, then the term obtained from t by substituting $\mathbf{r}$ for $\mathbf{x}$ will be denoted by $t^{\mathbf{x}}_{\mathbf{r}}$. Similar notation will be used for the result of substituting $\mathbf{r}$ for $\mathbf{x}$ in expressions of other kinds, such as literals and lists of literals.

The result of applying τ to a closed conditional literal $H : \mathbf{L}$ is the conjunction of the formulas

$$\tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}) \rightarrow \tau(H^{\mathbf{x}}_{\mathbf{r}})$$

where $\mathbf{x}$ is the list of variables occurring in $H : \mathbf{L}$, over all tuples $\mathbf{r}$ of precomputed terms of the same length as $\mathbf{x}$ such that both $\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}$ and $H^{\mathbf{x}}_{\mathbf{r}}$ are well-formed. For instance,

$$\tau(available(X) : person(X))$$

is the conjunction of the formulas $person(r) \rightarrow available(r)$ over all precomputed terms r;

$$\tau(\bot : p(2 \times X))$$

is the conjunction of the formulas $\neg p(2 \times i)$ over all numerals i. When a conditional literal occurs in the head of a rule, we will translate it in a different way. By $\tau_h(H : \mathbf{L})$ we denote the disjunction of the formulas

$$\tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}) \wedge \tau(H^{\mathbf{x}}_{\mathbf{r}})$$

where $\mathbf{x}$ and $\mathbf{r}$ are as above. For instance,

$$\tau_h(\mathit{available}(X) : \mathit{person}(X))$$

is the disjunction of the formulas $\mathit{person}(r) \wedge \mathit{available}(r)$ over all precomputed terms r.

3.5 Semantics of Closed Aggregate Expressions

In this section, the semantics of ground aggregates proposed in [1, Section 4.1] is adapted to closed aggregate expressions. Let E be a closed aggregate expression $\alpha\{\mathbf{t} : \mathbf{L}\} \prec s$, and let $\mathbf{x}$ be the list of variables occurring in E. A tuple $\mathbf{r}$ of precomputed terms of the same length as $\mathbf{x}$ is *admissible* (w.r.t. E) if both $\mathbf{t}^{\mathbf{x}}_{\mathbf{r}}$ and $\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}$ are well-formed. About a set Δ of admissible tuples we say that it *justifies* E if the relation $\prec$ holds between $\widehat{\alpha}(\{[\mathbf{t}^{\mathbf{x}}_{\mathbf{r}}] : \mathbf{r} \in \Delta\})$ and $[s]$. For instance, consider the aggregate expression

$$\mathit{sum}\{H, C : \mathit{enroll}(C), \mathit{hours}(H, C)\} = 6. \tag{5}$$

In this case, admissible tuples are arbitrary pairs of precomputed terms. The set $\{(3, \mathit{cs101}), (3, \mathit{cs102})\}$ justifies (5), because

$$\widehat{\mathit{sum}}(\{(H, C)^{H,C}_{3,\mathit{cs101}}, (H, C)^{H,C}_{3,\mathit{cs102}}\}) = \widehat{\mathit{sum}}(\{(3, \mathit{cs101}), (3, \mathit{cs102})\}) = 3+3 = 6.$$

More generally, a set Δ of pairs of precomputed terms justifies (5) whenever Δ contains finitely many pairs (h, c) in which h is a positive integer, and the sum of the integers h over all these pairs is 6.

We define τE as the conjunction of the implications

$$\bigwedge_{\mathbf{r} \in \Delta} \tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}) \rightarrow \bigvee_{\mathbf{r} \in A \setminus \Delta} \tau(\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}) \tag{6}$$

over all sets Δ of admissible tuples that do not justify E, where A is the set of all admissible tuples. For instance, if E is (5) then the conjunctive terms of τE are the formulas

$$\bigwedge_{(h,c) \in \Delta} (\mathit{enroll}(c) \wedge \mathit{hours}(h, c)) \rightarrow \bigvee_{(h,c) \notin \Delta} (\mathit{enroll}(c) \wedge \mathit{hours}(h, c)).$$

The conjunctive term corresponding to $\{(3, \mathit{cs101})\}$ as Δ says: if I am enrolled in CS101 for 3 hours then I am enrolled in at least one other course.

3.6 Semantics of Rules and Programs

For any rule R, τR stands for the conjunction of the formulas

$$\tau B_1 \wedge \cdots \wedge \tau B_n \to \tau_h H_1 \vee \cdots \vee \tau_h H_m$$

for all instances (3) of R. A *stable model* of a program Π is a stable model, in the sense of [13], of the set consisting of the formulas τR for all rules R of Π.

Consider, for instance, the rules of program (4). If R is the rule $\{p(i)\}$ then τR is

$$p(i) \vee \neg p(i) \tag{7}$$

$(i = 1, \ldots, n)$. If R is the rule

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

then the instances of R are rules of the form

$$\leftarrow p(i), p(j), p(i+j)$$

for all numerals i, j. (Substituting precomputed ground terms other than numerals would produce a rule that is not well-formed.) Consequently τR is in this case the infinite conjunction

$$\bigwedge_{\substack{i,j,k \in \mathbf{Z} \\ i+j=k}} \neg(p(i) \wedge p(j) \wedge p(k)). \tag{8}$$

The stable models of program (4) are the stable models of formulas (7), (8), that is, sets of the form $\{p(i) : i \in S\}$ for all sum-free subsets S of $\{1, \ldots, n\}$.

4 Reasoning about Gringo Programs

In this section we give examples of reasoning about Gringo programs on the basis of the semantics defined above. These examples use the results of [8], and we assume here that the reader is familiar with that paper.

4.1 Simplifying a Rule from Example 3.7 of User's Guide

Consider the rule[7]

$$\mathit{weekdays} \leftarrow \mathit{day}(X) : (\mathit{day}(X), \mathit{not}\ \mathit{weekend}(X)). \tag{9}$$

Replacing this rule with the fact *weekdays* within any program will not affect the set of stable models. Indeed, the result of applying translation τ to (9) is the formula

$$\bigwedge_r (\mathit{day}(r) \wedge \neg \mathit{weekend}(r) \to \mathit{day}(r)) \ \to\ \mathit{weekdays}, \tag{10}$$

[7] This rule is similar to a rule from Example 3.7 of the Gringo 3 *User's Guide* (see Footnote 2).

where the conjunction extends over all precomputed terms r. The formula

$$day(r) \wedge \neg weekend(r) \rightarrow day(r)$$

is intuitionistically provable. By the replacement property of the basic system of natural deduction from [8], it follows that (10) is equivalent to *weekdays* in the basic system. By the main theorem of [8], it follows that replacing (10) with the atom *weekdays* within any set of formulas does not affect the set of stable models.

4.2 Simplifying the Sorting Rule

The rule

$$order(X,Y) \leftarrow p(X),\ p(Y),\ X<Y,\ not\ p(Z) : (p(Z), X<Z, Z<Y) \tag{11}$$

can be used for sorting.[8] It can be replaced by either of the following two shorter rules within any program without changing that program's stable models.

$$order(X,Y) \leftarrow p(X),\ p(Y),\ X<Y,\ \bot : (p(Z), X<Z, Z<Y) \tag{12}$$

$$order(X,Y) \leftarrow p(X),\ p(Y),\ X<Y,\ not\ p(Z) : (X<Z, Z<Y) \tag{13}$$

Let's prove this claim for rule (12). By the main theorem of [8] it is sufficient to show that the result of applying τ to (11) is equivalent in the basic system to the result of applying τ to (12). The instances of (11) are the rules

$$order(i,j) \leftarrow p(i),\ p(j),\ i<j,\ not\ p(Z) : (p(Z), i<Z, Z<j),$$

and the instances of (12) are the rules

$$order(i,j) \leftarrow p(i),\ p(j),\ i<j,\ \bot : (p(Z), i<Z, Z<j)$$

where i and j are arbitrary numerals. The result of applying τ to (11) is the conjunction of the formulas

$$p(i) \wedge p(j) \wedge i<j \wedge \bigwedge_k (\neg p(k) \wedge i<k \wedge k<j \rightarrow p(k)) \rightarrow order(i,j) \tag{14}$$

for all numerals $i,\ j$. The result of applying τ to (12) is the conjunction of the formulas

$$p(i) \wedge p(j) \wedge i<j \wedge \bigwedge_k (\neg p(k) \wedge i<k \wedge k<j \rightarrow \bot) \rightarrow order(i,j). \tag{15}$$

By the replacement property of the basic system, it is sufficient to observe that

$$p(k) \wedge i<k \wedge k<j \rightarrow \neg p(k)$$

is intuitionistically equivalent to

$$p(k) \wedge i<k \wedge k<j \rightarrow \bot.$$

The proof for rule (13) is similar. Rule (12), like rule (11), is safe; rule (13) is not.

[8] This rule was communicated to us by Roland Kaminski on October 21, 2012.

4.3 Eliminating Choice in Favor of a Conditional Literal

Replacing the rule

$$\{p(X)\} \leftarrow q(X) \tag{16}$$

with

$$p(X) \leftarrow q(X), \bot : not\ p(X) \tag{17}$$

within any program will not affect the set of stable models. Indeed, the result of applying translation τ to (16) is

$$\bigwedge_r (q(r) \rightarrow p(r) \vee \neg p(r)) \tag{18}$$

where the conjunction extends over all precomputed terms r, and the result of applying τ to (17) is

$$\bigwedge_r (q(r) \wedge \neg\neg p(r) \rightarrow p(r)). \tag{19}$$

The implication from (18) is equivalent to the implication from (19) in the extension of intuitionistic logic obtained by adding the axiom schema

$$\neg F \vee \neg\neg F,$$

and consequently in the extended system presented in [8, Section 7]. By the replacement property of the extended system, it follows that (18) is equivalent to (19) in the extended system as well.

4.4 Eliminating a Trivial Aggregate Expression

The rule (1) says, informally speaking, that we can conclude $p(Y)$ once we established that there exists at least one X such that $q(X, Y)$. Replacing this rule with

$$p(Y) \leftarrow q(X, Y) \tag{20}$$

within any program will not affect the set of stable models.

To prove this claim, we need to calculate the result of applying τ to rule (1). The instances of (1) are the rules

$$p(t) \leftarrow count\{X, t : q(X, t)\} \geq 1 \tag{21}$$

for all precomputed terms t. Consider the aggregate expression E in the body of (21). Any precomputed term r is admissible w.r.t. E. A set Δ of precomputed terms justifies E if

$$\widehat{count}(\{(r, t) : r \in \Delta\}) \geq 1,$$

that is to say, if Δ is non-empty. Consequently τE consists of only one implication (6), with the empty Δ. The antecedent of this implication is the empty

conjunction $\top$, and its consequent is the disjunction $\bigvee_u q(u,t)$ over all precomputed terms u. Then the result of applying τ to (1) is

$$\bigwedge_t \left(\bigvee_u q(u,t) \ \rightarrow \ p(t) \right). \tag{22}$$

On the other hand, the result of applying τ to (20) is

$$\bigwedge_{t,u} (q(u,t) \rightarrow p(t)).$$

This formula is equivalent to (22) in the basic system [8, Example 2].

4.5 Replacing an Aggregate Expression with a Conditional Literal

Informally speaking, the rule

$$q \leftarrow count\{X : p(X)\} = 0 \tag{23}$$

says that we can conclude q once we have established that the cardinality of the set $\{X : p(X)\}$ is 0; the rule

$$q \leftarrow \bot : p(X) \tag{24}$$

says that we can conclude q once we have established that $p(X)$ does not hold for any X. We'll prove that replacing (23) with (24) within any program will not affect the set of stable models. To this end, we'll show that the results of applying τ to (23) and (24) are equivalent to each other in the extended system from [8, Section 7].

First, we'll need to calculate the result of applying τ to rule (23). Consider the aggregate expression E in the body of (23). Any precomputed term r is admissible w.r.t. E. A set Δ of precomputed terms justifies E if

$$\widehat{count}(\{r : r \in \Delta\}) = 0,$$

that is to say, if Δ is empty. Consequently τE is the conjunction of the implications

$$\bigwedge_{r \in \Delta} p(r) \rightarrow \bigvee_{r \in A \setminus \Delta} p(r) \tag{25}$$

for all non-empty subsets Δ of the set A of precomputed terms. The result of applying τ to (23) is

$$\left(\bigwedge_{\substack{\Delta \subseteq A \\ \Delta \neq \emptyset}} \left(\bigwedge_{r \in \Delta} p(r) \rightarrow \bigvee_{r \in A \setminus \Delta} p(r) \right) \right) \rightarrow q. \tag{26}$$

The result of applying τ to (24), on the other hand, is

$$\left(\bigwedge_{r \in A} \neg p(r) \right) \rightarrow q. \tag{27}$$

The fact that the antecedents of (26) and (27) are equivalent to each other in the extended system can be established by essentially the same argument as in [8, Example 7]. By the replacement property of the extended system, it follows that (26) is equivalent to (27) in the extended system as well.

4.6 Eliminating Summation over the Empty Set

Informally speaking, the rule

$$q \leftarrow sum\{X : p(X)\} = 0 \tag{28}$$

says that we can conclude q once we have established that the sum of the elements of the set $\{X : p(X)\}$ is 0. In the presence of the constraint

$$\leftarrow p(X), \tag{29}$$

replacing (28) with the fact q will not affect the stable models.

To see this, first we calculate the result of applying τ to rule (28). Consider the aggregate expression E in the body of (28). Any precomputed term r is admissible w.r.t. E. A set Δ of precomputed terms justifies E if

$$\widehat{sum}(\{r : r \in \Delta\}) = 0,$$

that is to say, if Δ contains no positive integers. Consequently τE is the conjunction of the implications

$$\bigwedge_{r \in \Delta} p(r) \rightarrow \bigvee_{r \in A \setminus \Delta} p(r) \tag{30}$$

for subsets Δ of the set A of precomputed terms that contain at least one positive integer. The result of applying τ to (28) is

$$\left(\bigwedge_{\substack{\Delta \subseteq A \\ \Delta \cap \mathbf{Z} \neq \emptyset}} \left(\bigwedge_{r \in \Delta} p(r) \rightarrow \bigvee_{r \in A \setminus \Delta} p(r) \right) \right) \rightarrow q. \tag{31}$$

The result of applying τ to (29), on the other hand, is

$$\bigwedge_{r \in A} \neg p(r). \tag{32}$$

For every nonempty Δ, the antecedent of (30) contradicts (32). Consequently, the antecedent of (31) can be derived from (32) in the basic system. It follows that the equivalence between (31) and the atom q can be derived in the basic system under assumption (32).

5 Conclusion

In this note we approached the problem of defining the semantics of Gringo by reducing Gringo programs to infinitary propositional formulas. We argued that this approach to semantics may allow us to study equivalent transformations of programs using natural deduction in infinitary propositional logic.

In the absence of a precise semantics, it is impossible to put the study of some important issues on a firm foundation. This includes the correctness of ASP programs, grounders, solvers, and optimization methods, and also the relationship between input languages of different solvers (for instance, the equivalence of the semantics of aggregate expressions in Gringo to their semantics in the ASP Core language and in the language proposed in [5] under the assumption that aggregates are used nonrecursively). As future work, we are interested in addressing some of these tasks on the basis of the semantics proposed in this note. Proving the correctness of the intelligent instantiation algorithms implemented in GRINGO will provide justification for our informal claim that for a safe program, the semantics proposed here correctly describes the output produced by GRINGO.

Acknowledgements

Many thanks to Roland Kaminski and Torsten Schaub for helping us understand the input language of GRINGO. Roland, Michael Gelfond, Yuliya Lierler, Joohyung Lee, and anonymous referees provided valuable comments on drafts of this note.

References

1. Ferraris, P.: Answer sets for propositional theories. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 119–131 (2005)
2. Ferraris, P., Lee, J., Lifschitz, V.: Stable models and circumscription. Artificial Intelligence 175, 236–263 (2011)
3. Ferraris, P., Lifschitz, V.: Weight constraints as nested expressions. Theory and Practice of Logic Programming 5, 45–74 (2005)
4. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers (2012)
5. Gelfond, M.: Representing knowledge in A-Prolog. Lecture Notes in Computer Science 2408, 413–451 (2002)
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K. (eds.) Proceedings of International Logic Programming Conference and Symposium. pp. 1070–1080. MIT Press (1988)
7. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991)

8. Harrison, A., Lifschitz, V., Truszczynski, M.: On equivalent transformations of infinitary formulas under the stable model semantics (preliminary report)[9]. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR) (2013), to appear
9. Lee, J., Meng, Y.: Stable models of formulas with generalized quantifiers. In: Working Notes of the 14th International Workshop on Non-Monotonic Reasoning (NMR) (2012)
10. Lee, J., Meng, Y.: Stable models of formulas with generalized quantifiers (preliminary report). In: Technical Communications of the 28th International Conference on Logic Programming (ICLP). pp. 61–71 (2012)
11. Lee, J., Meng, Y.: Two new definitions of stable models of logic programs with generalized quantifiers. In: Working Notes of the 5th Workshop of Answer Set Programming and Other Computing Paradigms (ASPOCP) (2012)
12. Pearce, D., Valverde, A.: Towards a first order equilibrium logic for nonmonotonic reasoning. In: Proceedings of European Conference on Logics in Artificial Intelligence (JELIA). pp. 147–160 (2004)
13. Truszczynski, M.: Connecting first-order ASP and the logic FO(ID) through reducts. In: Correct Reasoning: Essays on Logic-Based AI in Honor of Vladimir Lifschitz. Springer (2012)

[9] `http://www.cs.utexas.edu/users/vl/papers/etinf.pdf`