The Semantics of Gringo and Infinitary Propositional Formulas

Amelia Harrison and **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, so that the properties of these programs can be more precisely characterized. In this way, we aim to create a foundation on which important issues such as the correctness of GRINGO programs and optimization methods may be more formally studied.

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 (Gelfond and Lifschitz 1988). 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]

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 (Gebser et al. 2012), 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 programs can be approached using quantified equilibrium logic (Pearce and Valverde 2004) or its syntactic counterpart defined in (Ferraris, Lee, and Lifschitz 2011). 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 approach is not very general. For instance, it is not clear how to represent the rule

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

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 in it rather than a constant.

[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 (Gelfond and Lifschitz 1991) and (Ferraris and Lifschitz 2005). 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 is straightforward, using the process of eliminating classical negation in favor of additional atoms described in (Gelfond and Lifschitz 1991, Section 4).

General aggregate expressions allowed in Gringo can be represented by first-order formulas with generalized quantifiers. Stable models of formulas with generalized quantifiers are defined by Lee and Meng (2012a, 2012b, 2012c). The advantage of infinitary propositional formulas as the target language is that we have more mathematical results regarding the properties of these formulas. 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 (Harrison, Lifschitz, and Truszczynski 2014). 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 (Gelfond and Lifschitz 1988), it is modular, in the sense that it applies to the program rule by rule, and it is applicable even if 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 `==`.

The syntax and semantics of the fragment of Gringo studied in this note are defined in Sections 2 and 3. In Sections 4 and 5 we give examples of reasoning about stable models of Gringo programs on the basis of the proposed semantics. The semantics of aggregate expressions, based on (Ferraris 2005), is rather complicated, and in Section 6 we show how it can be simplified in the case of monotone and anti-monotone aggregate expressions. Definitions and theorems regarding stable models of infinitary formulas used in this paper are reproduced in Section 3.1 and in the appendix.

A preliminary report on this project was presented at the 2013 Workshop on Answer Set Programming and Other Computing Paradigms.

2 Syntax

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

- numerals—object constants representing all integers,
- arithmetical functions—binary function constants $+$, $-$, $\times$,
- 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 which do not fall into any of the categories listed above 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 symbol $\widehat{\alpha}$ represents *the function denoted by* α. This function 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.[5]

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

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

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

$$t_1 \prec t_2$$

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

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

and

$$\bot : (\mathit{person}(X), \mathit{not}\ \mathit{available}(X))$$

are conditional literals.

An *aggregate expression* is an expression of the form

$$\alpha\{\mathbf{t}_1 : \mathbf{L}_1; \dots; \mathbf{t}_n : \mathbf{L}_n\} \prec s \tag{3}$$

($n > 0$), where

- α is an aggregate name,
- each $\mathbf{t}_i$ is a non-empty list of terms,
- each $\mathbf{L}_i$ is a non-empty list of literals,

[5] 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.

- $\prec$ is a comparison or one of the symbols $=, \neq$,
- 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

$$H_1 \mid \cdots \mid H_m \leftarrow B_1, \ldots, B_n \tag{4}$$

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

$$\begin{array}{ll} \{p(i)\} & \leftarrow \\ & \leftarrow p(X), p(Y), p(X+Y) \end{array} \qquad (i = 1, \ldots, n), \tag{5}$$

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 (Truszczynski 2012), 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 $\mathcal{F}_0^\sigma, \mathcal{F}_1^\sigma, \ldots$ are defined as follows:

- $\mathcal{F}_0^\sigma = \sigma$,
- $\mathcal{F}_{i+1}^\sigma$ 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.

A set $\mathcal{H}$ of formulas is *bounded* if it is contained in one of the sets $\mathcal{F}_i^\sigma$. For a bounded set $\mathcal{H}$ of formulas, $\mathcal{H}^\wedge$ and $\mathcal{H}^\vee$ are infinitary formulas.

The symbols $\top$ and $\bot$ will be understood as abbreviations for $\emptyset^\wedge$ and $\emptyset^\vee$ respectively; $\neg F$ stands for $F \to \bot$, and $F \leftrightarrow G$ stands for $(F \to G) \wedge (G \to F)$.

We will write $\{F, G\}^\wedge$ as $F \wedge G$, and $\{F, G\}^\vee$ as $F \vee G$. This convention allows us to view finite propositional formulas over σ as a special case of infinitary formulas. For any bounded family $\{F_\alpha\}_{\alpha \in A}$ of formulas, we denote the formula $\{F_\alpha : \alpha \in A\}^\wedge$ by $\bigwedge_{\alpha \in A} F_\alpha$, and similarly for disjunctions.

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:

- 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 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 $=$, $\neq$, 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 (3), if it occurs in the term s;
- in a rule (4), if it is global in at least one of the expressions H_i, B_j.

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,

$$\begin{array}{l} \mathit{total\_hours}(6) \leftarrow \\ \quad \mathit{sum}\{H, C : \mathit{enroll}(C), \mathit{hours}(H, C)\} = 6 \end{array}$$

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,

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

is the conjunction of the formulas

$$\mathit{person}(r) \rightarrow \mathit{available}(r)$$

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, it is translated in a different way.[6] By $\tau_h(H : \mathbf{L})$ we denote the disjunction of the formulas

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

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

$$(\mathit{person}(r) \rightarrow \mathit{available}(r)) \wedge \neg\neg \mathit{person}(r)$$

over all precomputed terms r.

3.5 Semantics of Closed Aggregate Expressions

In this section, the semantics of ground aggregates proposed in (Ferraris 2005, Section 4.1) is adapted to closed aggregate expressions. Let E be a closed aggregate expression (3), and let $\mathbf{x}_i$ be the list of variables occurring in $\mathbf{t}_i : \mathbf{L}_i$ $(1 \leq i \leq n)$. By A_i we denote the set of tuples $\mathbf{r}$ of precomputed terms of the same length as $\mathbf{x}_i$ such that both $(\mathbf{t}_i)^{\mathbf{x}_i}_{\mathbf{r}}$ and $(\mathbf{L}_i)^{\mathbf{x}_i}_{\mathbf{r}}$ are well-formed. By A we denote the set $\{(i, \mathbf{r}) : i \in \{1, \ldots, n\}, \mathbf{r} \in A_i\}$.

About a subset Δ of A we say that it *justifies* E if the relation $\prec$ holds between $\widehat{\alpha}(\{[(\mathbf{t}_i)^{\mathbf{x}_i}_{\mathbf{r}}] : (i, \mathbf{r}) \in \Delta\})$ and $[s]$. We define τE as the conjunction of the implications

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

over all sets Δ that do not justify E.

Example Consider the aggregate expression

$$\mathit{count}\{X : p(X);\ N : q(2 \times N)\} > 0. \qquad (7)$$

In this case, A_1 is the set of all precomputed terms, and A_2 is the set of all numerals. Therefore, A is the set of all pairs whose first member is either 1 or 2 and whose second member is a precomputed term when the first member is 1 and is a numeral when the first member is 2. A subset Δ of A justifies (7) iff

$$\widehat{\mathit{count}}\left(\left\{X^X_{\mathbf{r}} : (1, \mathbf{r}) \in \Delta\right\} \cup \left\{N^N_{\mathbf{r}} : (2, \mathbf{r}) \in \Delta\right\}\right) > 0.$$

Since

$$\{X^X_{\mathbf{r}} : (1, \mathbf{r}) \in \Delta\} = \{\mathbf{r} : (1, \mathbf{r}) \in \Delta\}$$

and

$$\{N^N_{\mathbf{r}} : (2, \mathbf{r}) \in \Delta\} = \{\mathbf{r} : (2, \mathbf{r}) \in \Delta\}$$

we conclude that the set Δ justifies (7) iff

$$\widehat{\mathit{count}}(\Delta) > 0,$$

that is, iff the set Δ is non-empty. Since the only $\Delta \subseteq A$ that does not justify (7) is $\emptyset$, the result of applying τ to (7) has only one conjunctive term of the form (6). Its antecedent is the trivial empty conjunction; its consequent is

$$\bigvee_{x \in A_1} p(x) \vee \bigvee_{n \in A_2} q(2n).$$

If E is an aggregate expression of the form

$$\alpha\{\mathbf{t}_1 : \mathbf{L}_1; \ldots; \mathbf{t}_n : \mathbf{L}_n\} = s,$$

$E_{\leq}$ is

$$\alpha\{\mathbf{t}_1 : \mathbf{L}_1; \ldots; \mathbf{t}_n : \mathbf{L}_n\} \leq s,$$

and $E_{\geq}$ is

$$\alpha\{\mathbf{t}_1 : \mathbf{L}_1; \ldots; \mathbf{t}_n : \mathbf{L}_n\} \geq s,$$

then $\tau(E)$ has the same conjunctive terms as

$$\tau(E_{\leq}) \wedge \tau(E_{\geq}).$$

In this sense, E has the same meaning of the conjunction of $E_{\leq}$ and $E_{\geq}$.

In the special case when E has the form $\alpha\{\mathbf{t} : \mathbf{L}\} \prec s$, the definition of τE can be stated as follows. Let $\mathbf{x}$ be the list of variables occurring in $\mathbf{t} : \mathbf{L}$, and let A be the set of tuples $\mathbf{r}$ of precomputed terms of the same length as $\mathbf{x}$ such that both $\mathbf{t}^{\mathbf{x}}_{\mathbf{r}}$ and $\mathbf{L}^{\mathbf{x}}_{\mathbf{r}}$ are well-formed. A subset Δ of A *justifies* E if the relation $\prec$ holds between $\widehat{\alpha}(\{[\mathbf{t}^{\mathbf{x}}_{\mathbf{r}}] : \mathbf{r} \in \Delta\})$ and $[s]$. We define τE as the conjunction of the implications

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

[6] Torsten Schaub, Martin Gebser, and Roland Kaminski, personal communication, October 2013.

over all subsets Δ of A that do not justify E.

Examples Consider the aggregate expression

$$\mathit{sum}\{H, C : \mathit{enroll}(C), \mathit{hours}(H, C)\} = 6. \tag{8}$$

In this case, A is the set of pairs (h, c) of precomputed terms. The subset $\{(3, \mathit{cs101}), (3, \mathit{cs102})\}$ of A justifies (8), because

$$\begin{aligned}\widehat{\mathit{sum}}\left(\left\{(H,C)^{H,C}_{3,\mathit{cs101}}, (H,C)^{H,C}_{3,\mathit{cs102}}\right\}\right) &= \\ \widehat{\mathit{sum}}(\{(3, \mathit{cs101}), (3, \mathit{cs102})\}) &= 3 + 3 = 6.\end{aligned}$$

More generally, a set of pairs of precomputed terms justifies (8) whenever it 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. The conjunctive terms of τE are the formulas

$$\begin{aligned}\bigwedge_{(h,c)\in\Delta} (\mathit{enroll}(c) \wedge \mathit{hours}(h, c)) \rightarrow \\ \bigvee_{(h,c)\notin\Delta} (\mathit{enroll}(c) \wedge \mathit{hours}(h, c)).\end{aligned} \tag{9}$$

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. In Section 6.1 we will see how the result of applying τ to (8) can be simplified.

As a final example illustrating the semantics of aggregate expressions, consider a pair of closed aggregate expressions of the form

$$\mathit{sum}\{1, X : \mathbf{L}\} \prec s \quad \text{and} \quad \mathit{count}\{X : \mathbf{L}\} \prec s, \tag{10}$$

where the tuple $\mathbf{L}$ of literals contains no variables other than X. Intuitively, these expressions have the same meaning; accordingly, the translation τ transforms them to identical formulas. Note first that the result of applying τ to each of them is the conjunction of implications of the form

$$\bigwedge_{r\in\Delta} \tau(\mathbf{L}^X_r) \rightarrow \bigvee_{r\in A\setminus\Delta} \tau(\mathbf{L}^X_r).$$

To check that both conjunctions extend over the same set Δ of precomputed terms, observe that

$$\widehat{\mathit{sum}}(\{(1, X)^X_r : r \in \Delta\}) = \widehat{\mathit{sum}}(\{(1, r) : r \in \Delta\})$$

and

$$\begin{aligned}\widehat{\mathit{count}}(\{X^X_r : r \in \Delta\}) &= \widehat{\mathit{count}}(\{r : r \in \Delta\}) \\ &= \widehat{\mathit{count}}(\Delta),\end{aligned}$$

so that both expressions evaluate to the cardinality of Δ.

3.6 Semantics of Rules and Programs

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

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

for all instances (4) of R. A *stable model* of a program Π is a stable model, in the sense of (Truszczynski 2012), of the set consisting of the formulas τR for all rules R of Π.

Example Consider the rules of program (5). If R is the rule $\{p(i)\}$ then τR is

$$p(i) \vee \neg p(i) \tag{11}$$

$(i = 1, \ldots, n)$. 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

$$\bigwedge_{\substack{i,j,k\in\mathbf{Z}\\ i+j=k}} \neg(p(i) \wedge p(j) \wedge p(k)). \tag{12}$$

The stable models of program (5) are the stable models of formulas (11), (12), 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

As an example, we will show how we can reason about the stable models of the program Π consisting of rule (2) and an arbitrary finite set S of atoms of the forms $\mathit{hours}(h, c)$ and $\mathit{enroll}(c)$, where h is a positive integer and c is an object constant.

Let h^* be the sum of the numbers h over all pairs (h, c) such that both $\mathit{hours}(h, c)$ and $\mathit{enroll}(c)$ are in S. As could be expected, the set

$$S \cup \{\mathit{total\_hours}(h^*)\} \tag{13}$$

is the only stable model of Π.

To justify this claim, consider the result of applying τ to Π. This set of formulas consists of the atoms S and the implications

$$\begin{aligned}\tau(\mathit{sum}\{H, C : \mathit{enroll}(C), hours(H, C)\} = i) \\ \rightarrow \mathit{total\_hours}(i)\end{aligned} \tag{14}$$

for all numerals i. The conjunction of the formulas in $\tau\Pi$ is a supertight infinitary program (see Appendix A.1). Consequently, it is sufficient to show that (13) is the only supported model of $\tau\Pi$, which can be derived from the following fact:

Lemma *Let I be an interpretation such that S is the set of all atoms of the forms hours(h,c) and enroll(c) that are satisfied by I. For any numeral i, I satisfies the antecedent of (14) if and only if $i = h^*$.*

5 Equivalent Transformations of 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 theorem from (Harrison, Lifschitz, and Truszczynski 2014) reproduced in the appendix.

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

Consider the rule[7]

$$\mathit{weekdays} \leftarrow \mathit{day}(X) : (\mathit{day}(X), \mathit{not\ weekend}(X)). \quad (15)$$

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 (15) is the formula

$$\bigwedge_r (\mathit{day}(r) \wedge \neg \mathit{weekend}(r) \rightarrow \mathit{day}(r)) \rightarrow \mathit{weekdays}, \quad (16)$$

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

$$\mathit{day}(r) \wedge \neg \mathit{weekend}(r) \rightarrow \mathit{day}(r)$$

is intuitionistically provable. By the theorem from Section A.2, it follows that replacing (16) with the atom *weekdays* within any set of formulas does not affect the set of stable models.

5.2 Simplifying the Sorting Rule

The rule

$$\begin{aligned} \mathit{order}(X,Y) \leftarrow\ & p(X),\ p(Y),\ X<Y, \\ & \mathit{not}\ p(Z) : (p(Z), X<Z, Z<Y) \end{aligned} \quad (17)$$

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.

$$\begin{aligned} \mathit{order}(X,Y) \leftarrow\ & p(X),\ p(Y),\ X<Y, \\ & \bot : (p(Z), X<Z, Z<Y) \end{aligned} \quad (18)$$

$$\begin{aligned} \mathit{order}(X,Y) \leftarrow\ & p(X),\ p(Y),\ X<Y, \\ & \mathit{not}\ p(Z) : (X<Z, Z<Y) \end{aligned} \quad (19)$$

Let's prove this claim for rule (18). By the theorem from Section A.2, it is sufficient to show that the result of applying τ to (17) is equivalent in the extended system to the result of applying τ to (18). The instances of (17) are the rules

$$\begin{aligned} \mathit{order}(i,j) \leftarrow\ & p(i),\ p(j),\ i<j, \\ & \mathit{not}\ p(Z) : (p(Z), i<Z, Z<j), \end{aligned}$$

and the instances of (18) are the rules

$$\begin{aligned} \mathit{order}(i,j) \leftarrow\ & p(i),\ p(j),\ i<j, \\ & \bot : (p(Z), i<Z, Z<j) \end{aligned}$$

where i and j are arbitrary numerals. The result of applying τ to (17) is the conjunction of the formulas

$$\begin{aligned} & p(i) \wedge p(j) \wedge \tau(i<j) \wedge \\ & \quad \textstyle\bigwedge_k (\neg p(k) \wedge \tau(i<k) \wedge \tau(k<j) \rightarrow p(k)) \rightarrow \\ & \qquad\qquad \mathit{order}(i,j) \end{aligned} \quad (20)$$

for all numerals $i,\ j$. The result of applying τ to (18) is the conjunction of the formulas

$$\begin{aligned} & p(i) \wedge p(j) \wedge \tau(i<j) \wedge \\ & \quad \textstyle\bigwedge_k (\neg p(k) \wedge \tau(i<k) \wedge \tau(k<j) \rightarrow \bot) \rightarrow \\ & \qquad\qquad \mathit{order}(i,j). \end{aligned} \quad (21)$$

It remains to observe that for any numerals i, j, k,

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

is intuitionistically equivalent to

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

The proof for rule (19) is similar. Rule (18), like rule (17), is safe; rule (19) is not.

5.3 Eliminating Choice in Favor of a Conditional Literal

Replacing the rule

$$\{p(X)\} \leftarrow q(X) \quad (22)$$

with

$$p(X) \leftarrow q(X), \bot : \mathit{not}\ p(X) \quad (23)$$

within any program will not affect the set of stable models. Indeed, the result of applying translation τ to (22) is

$$\bigwedge_r (q(r) \rightarrow p(r) \vee \neg p(r)) \quad (24)$$

where the conjunction extends over all precomputed terms r, and the result of applying τ to (23) is

$$\bigwedge_r (q(r) \wedge \neg\neg p(r) \rightarrow p(r)). \quad (25)$$

Each conjunctive term of (24) is equivalent to the corresponding conjunctive term of (25) 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 the appendix. It follows that (24) is equivalent to (25) in the extended system as well.

5.4 Eliminating a Trivial Aggregate Expression

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

$$p(Y) \leftarrow q(X,Y) \quad (26)$$

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

$$p(t) \leftarrow \mathit{count}\{X, t : q(X,t)\} \geq 1 \quad (27)$$

for all precomputed terms t. Consider the aggregate expression E in the body of (27). Any precomputed term r is admissible w.r.t. E. A set Δ of precomputed terms justifies E if

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

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 $\top$,

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

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

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

$$\bigwedge_t \left(\bigvee_u q(u,t) \ \rightarrow \ p(t) \right). \tag{28}$$

On the other hand, the result of applying τ to (26) is

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

This formula is equivalent to (28) in the extended system defined in the appendix (see Example 2 from (Harrison, Lifschitz, and Truszczynski 2014)).

6 Monotone and Anti-Monotone Aggregate Expressions

A closed aggregate expression E is *monotone* if for any set Δ that justifies E, all supersets of Δ also justify E. Similarly, E is *anti-monotone* if for any set Δ that justifies E, all subsets of Δ also justify E.

For example, if α is one of the symbols *count*, *max*, or *sum*, then (3) is monotone when $\prec$ is $>$ or $\geq$, and anti-monotone when $\prec$ is $<$ or $\leq$. (According to the definition for $\widehat{sum}$ given in this paper negative summands are disregarded.) As observed in Section 3.5, if $\prec$ is $=$ then E can be equivalently replaced by the pair of aggregate expressions $E_{\leq}, E_{\geq}$, which are anti-monotone and monotone respectively when α is one of the symbols listed above.

6.1 Simplifying Monotone and Anti-Monotone Aggregate Expressions

Recall that τE for a closed aggregate expression E is defined as the conjunction of the implications (6) over all sets Δ that do not justify E. The two theorems stated below show that the antecedent in (6) can be dropped if E is monotone, and the consequent can be dropped if E is anti-monotone. These equivalences can be proved in the extended system described in the appendix. The theorems are similar to the properties of monotone and anti-monotone ground aggregates stated in (Ferraris 2005).

Theorem 1 *If a closed aggregate expression E is monotone, then τE is equivalent in the extended system to*

$$\bigwedge_{\Delta} \bigvee_{(i,\mathbf{r}) \in A \setminus \Delta} \tau((\mathbf{L}_i)^{\mathbf{x}_i}_{\mathbf{r}}), \tag{29}$$

where the conjunction extends over the subsets Δ of A that do not justify E.

Theorem 2 *If a closed aggregate expression E is anti-monotone, then τE is equivalent in the extended system to*

$$\bigwedge_{\Delta} \neg \bigwedge_{(i,\mathbf{r}) \in \Delta} \tau((\mathbf{L}_i)^{\mathbf{x}_i}_{\mathbf{r}}), \tag{30}$$

where the conjunction extends over all subsets Δ of A that do not justify E.

Example We saw in Section 3.5 that the result of applying τ to (8) is the conjunction of formulas (9) over all sets Δ of pairs (h,c) of precomputed terms that do not justify (8). Theorems 1 and 2 allow us to simplify this conjunction. Note first that the result of applying τ to (8) can be rewritten as the conjunction of

$$\tau(sum\{H, C : enroll(C), hours(H,C)\} \leq 6) \tag{31}$$

and

$$\tau(sum\{H, C : enroll(C), hours(H,C)\} \geq 6). \tag{32}$$

By Theorem 1, (32) is equivalent to

$$\bigwedge_{\widehat{sum}(\Delta) < 6} \ \bigvee_{(h,c) \notin \Delta} (enroll(c) \wedge hours(h,c)),$$

and by Theorem 2, (31) is equivalent to

$$\bigwedge_{\widehat{sum}(\Delta) > 6} \ \neg \bigwedge_{(h,c) \in \Delta} (enroll(c) \wedge hours(h,c)),$$

where Δ ranges over pairs (h,c) of precomputed terms.

In Section 6.2 we show how Theorem 2 can be used for simplifying and aggregate expression. Proofs of Theorems 1 and 2 are given in Section 6.3.

6.2 Eliminating an Aggregate Expression in Favor of a Conditional Literal

We will show that the rule

$$q \leftarrow count\{X : p(X)\} < 1 \tag{33}$$

can be replaced by the rule

$$q \leftarrow \bot : p(X) \tag{34}$$

within any program without changing its stable models. The bodies of both rules express that the set p is empty, but the first uses an aggregate expression and the second a conditional literal.

Since the aggregate expression in the body of (33) is anti-monotone, by Theorem 2, the result of applying τ to it is equivalent (in the extended system) to

$$\bigwedge_{\Delta \neq \emptyset} \neg \bigwedge_{a \in \Delta} p(a) \tag{35}$$

where Δ ranges over arbitrary sets of precomputed terms.

On the other hand, the result of applying τ to the body of (34) is

$$\bigwedge_{a} \neg p(a) \tag{36}$$

where a ranges over all precomputed terms.

It remains to observe that all conjunctive terms of (35) can be derived in the extended system from the conjunctive terms in which Δ is a singleton, and that the conjunctive terms in which Δ is a singleton are identical to the conjunctive terms of (36).

6.3 Proofs of Theorems 1 and 2

Proof of Theorem 1 The implication from (29) to τE is obvious. Now, assume τE, and consider the conjunctive term of (29) corresponding to a set Δ_0 which does not justify E. Since E is monotone, from τE we can derive the conjunction of all terms (6) where $\Delta \subseteq \Delta_0$. Any formula of the form

$$\left(F \to \bigvee_{i \in I} G_i\right) \to \left(\bigvee_{i \in I} (F \to G_i)\right)$$

is provable in the extended system defined in the appendix (see Section 7 of (Harrison, Lifschitz, and Truszczynski 2014)). Consequently, we can derive the conjunction of the disjunctions

$$\bigvee_{(i,\mathbf{r}) \in A \setminus \Delta} \left(\left(\bigwedge_{(j,\mathbf{s}) \in \Delta} \tau((\mathbf{L}_j)_{\mathbf{s}}^{\mathbf{x}_j})\right) \to \tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i})\right)$$

over all $\Delta \subseteq \Delta_0$. By distributivity, this is equivalent in the extended system to the disjunction of the conjunctions

$$\bigwedge_{\Delta \subseteq \Delta_0} \left(\bigwedge_{(j,\mathbf{s}) \in \Delta} \tau((\mathbf{L}_j)_{\mathbf{s}}^{\mathbf{x}_j}) \to \tau\left((\mathbf{L}_{f(\Delta)})_{g(\Delta)}^{\mathbf{x}_{f(\Delta)}}\right)\right) \quad (37)$$

over all pairs of functions (f, g) defined on all subsets Δ of Δ_0 such that the pair $(f(\Delta), g(\Delta))$ is an element of $A \setminus \Delta$. Now we reason by cases, with one case corresponding to each disjunctive term (37). For a particular disjunctive term (37), we wish to show that we can derive from it $\tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i})$ for some pair $(i, \mathbf{r})$ in $A \setminus \Delta_0$. Consider the set Δ^* of all pairs $(i, \mathbf{r})$ such that $\tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i})$ is derivable from (37) in the extended system. We will show that Δ^* is not a subset of Δ_0. Assume $\Delta^* \subseteq \Delta_0$, so that

$$(f(\Delta^*), g(\Delta^*)) \in A \setminus \Delta^*, \quad (38)$$

and one of the conjunctive terms of (37) is

$$\bigwedge_{(j,\mathbf{s}) \in \Delta^*} \tau((\mathbf{L}_j)_{\mathbf{s}}^{\mathbf{x}_j}) \to \tau\left((\mathbf{L}_{f_{\Delta^*}})_{g(\Delta^*)}^{\mathbf{x}_{f(\Delta^*)}}\right).$$

By the definition of Δ^*, every conjunctive term in the antecedent of this implication is derivable from (37). It follows that the consequent is derivable as well, so that $(f(\Delta^*), g(\Delta^*))$ belongs to Δ^*, which contradicts (38). Therefore, Δ^* is not a subset of Δ_0, so that at least one disjunctive term of (29) is derivable from (37).

Proof of Theorem 2 The implication from (30) to τE is obvious. We will derive (30) from τE using the disjunction

$$\bigvee_{\Delta \subseteq A} \left(\neg \bigvee_{(i,\mathbf{r}) \in A \setminus \Delta} \tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i}) \wedge \neg\neg \bigwedge_{(i,\mathbf{r}) \in \Delta} \tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i})\right). \quad (39)$$

This is a special case of the generalized law of weak excluded middle given in Section 7 of (Harrison, Lifschitz, and Truszczynski 2014). Each disjunctive term (39) corresponds to a subset Δ of A. We will reason by cases with one case corresponding to each disjunctive term D_Δ of (39). Each D_Δ is equivalent to

$$\neg\left(\bigwedge_{(i,\mathbf{r}) \in \Delta} \tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i}) \to \bigvee_{(i,\mathbf{r}) \in A \setminus \Delta} \tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i})\right) \quad (40)$$

in the extended system described in the appendix. If Δ does not justify E then (40) contradicts one of the conjunctive terms from (6) and (30) follows. Consider now the case when Δ justifies E. Assume

$$\bigwedge_{(i,\mathbf{r}) \in \Delta_0} \tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i}) \quad (41)$$

for some Δ_0 that does not justify E. Since E is anti-monotone Δ_0 is not a subset of Δ. We conclude that Δ_0 and $A \setminus \Delta$ overlap on at least one element. It follows that

$$\bigvee_{(i,\mathbf{r}) \in A \setminus \Delta} \tau((\mathbf{L}_i)_{\mathbf{r}}^{\mathbf{x}_i})$$

can be derived from (41) since at least one of its disjunctive terms can be derived. This disjunction contradicts (40), and so we may conclude the negation of (41).

7 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 (Gelfond 2002) 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.

Appendix

We review here conditions for characterizing the stable models of a set of infinitary formulas in terms of supportedness

(Lifschitz and Yang 2013), and transformations of infinitary formulas which preserve their stable models (Harrison, Lifschitz, and Truszczynski 2014).

A.1 Tight Infinitary Programs

An *infinitary program* is a conjunction of (possibly infinitely many) infinitary formulas of the form $G \to A$, where A is an atom. We say that an infinitary program Π is *supertight* if it is impossible to find an infinite sequence of atoms $A_1, A_2, \dots$ such that for all i, Π contains a conjunctive term $G \to A_i$, where A_{i+1} occurs in G. We say that an interpretation I is *supported* by Π if each atom $A \in I$ is the consequent of a conjunctive term $G \to A$ of Π such that $I \models G$.

Theorem *For any model I of a supertight infinitary program Π, I is stable iff I is suported by Π.*

This theorem is a special case of Lemma 2 from (Lifschitz and Yang 2013). In the statement of that lemma Π is required to be "tight on I." That condition is significantly more general than supertightness as defined above.

A.2 Equivalent Transformations of Infinitary Formulas

The extended system of natural deduction presented in (Harrison, Lifschitz, and Truszczynski 2014) includes the axiom schema

$$F \Rightarrow F,$$

the axiom schema

$$\Rightarrow F \vee (F \to G) \vee \neg G$$

that characterizes (in the finite case) the logic of here-and-there (Hosoi 1966), and the following additional axiom schemas:

$$\Rightarrow \neg \bigwedge_{F \in \mathcal{H}} F \to \bigvee_{F \in \mathcal{H}} \neg F, \tag{42}$$

$$\Rightarrow \left(\bigwedge_{i \in I} \bigvee_{F \in \mathcal{H}_i} F \right) \to \left(\bigvee_{\{F_i\}_{i \in I}} \bigwedge_{i \in I} F_i \right), \tag{43}$$

and

$$\Rightarrow \left(\bigwedge_{\{F_i\}_{i \in I}} \bigvee_{i \in I} F_i \right) \to \left(\bigvee_{i \in I} \bigwedge_{F \in \mathcal{H}_i} F \right). \tag{44}$$

In formula (42) $\mathcal{H}$ is a bounded set of formulas. In formulas (43) and (44) $\{\mathcal{H}_i\}_{i \in I}$ is an arbitrary non-empty family of sets of formulas such that its union is bounded. The disjunction in the consequent of (43) and the conjunction in the antecedent of (44) extend over all elements $\{F_i\}_{i \in I}$ of the Cartesian product of the family $\{\mathcal{H}_i\}_{i \in I}$.

The inference rules are the introduction and elimination rules for the propositional connectives

$$(\wedge I) \quad \frac{\Gamma \Rightarrow H \quad \text{for all } H \in \mathcal{H}}{\Gamma \Rightarrow \mathcal{H}^{\wedge}}$$

$$(\wedge E) \quad \frac{\Gamma \Rightarrow \mathcal{H}^{\wedge}}{\Gamma \Rightarrow H} \quad (H \in \mathcal{H})$$

$$(\vee I) \quad \frac{\Gamma \Rightarrow H}{\Gamma \Rightarrow \mathcal{H}^{\vee}} \quad (H \in \mathcal{H})$$

$$(\vee E) \quad \frac{\Gamma \Rightarrow \mathcal{H}^{\vee} \qquad \Delta, H \Rightarrow F \quad \text{for all } H \in \mathcal{H}}{\Gamma, \Delta \Rightarrow F}$$

$$(\to I) \quad \frac{\Gamma, F \Rightarrow G}{\Gamma \Rightarrow F \to G}$$

$$(\to E) \quad \frac{\Gamma \Rightarrow F \quad \Delta \Rightarrow F \to G}{\Gamma, \Delta \Rightarrow G},$$

where $\mathcal{H}$ is a bounded set of formulas, and the weakening rule

$$(W) \ \frac{\Gamma \Rightarrow F}{\Gamma, \Delta \Rightarrow F}.$$

The set of *theorems of the extended system* is the smallest set of sequents that includes the axioms of the system and is closed under the application of its inference rules. We say that formulas F and G are *equivalent in the extended system* if $\Rightarrow F \leftrightarrow G$ is a theorem of the extended system.

Theorem *For any set $\mathcal{H}$ of formulas, if formulas F and G are equivalent in the extended system then $\mathcal{H} \cup \{F\}$ and $\mathcal{H} \cup \{G\}$ have the same stable models.*

References

Ferraris, P., and Lifschitz, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming* 5:45–74.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence* 175:236–263.

Ferraris, P. 2005. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 119–131.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R., and Bowen, K., eds., *Proceedings of International Logic Programming Conference and Symposium*, 1070–1080. MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

Gelfond, M. 2002. Representing knowledge in A-Prolog. *Lecture Notes in Computer Science* 2408:413–451.

Harrison, A.; Lifschitz, V.; and Truszczynski, M. 2014. On equivalence of infinitary formulas under the stable model semantics. *Theory and Practice of Logic Programming*. To appear.

Hosoi, T. 1966. The axiomatization of the intermediate propositional systems S_n of Gödel. *Journal of the Faculty of Science of the University of Tokyo* 13:183–187.

Lee, J., and Meng, Y. 2012a. Stable models of formulas with generalized quantifiers. In *Working Notes of the 14th International Workshop on Non-Monotonic Reasoning (NMR)*.

Lee, J., and Meng, Y. 2012b. Stable models of formulas with generalized quantifiers (preliminary report). In *Technical Communications of the 28th International Conference on Logic Programming (ICLP)*, 61–71.

Lee, J., and Meng, Y. 2012c. 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)*.

Lifschitz, V., and Yang, F. 2013. Lloyd-Topor completion and general stable models. *Theory and Practice of Logic Programming* 13(4–5).

Pearce, D., and Valverde, A. 2004. Towards a first order equilibrium logic for nonmonotonic reasoning. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 147–160.

Truszczynski, M. 2012. 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.