The Semantics of Gringo and Proving Strong Equivalence

Amelia Harrison
Department of Computer Science
The University of Texas at Austin
2317 Speedway, 2.302
Austin, Texas 78712
Internal Mail Code: D9500
E-mail: ameliaj@cs.utexas.edu

submitted 12 May 2013; accepted 5 June 2013

Abstract

Manuals written by the designers of answer set solvers usually describe the semantics of the input languages of their systems using examples and informal comments that appeal to the user's intuition, without references to any precise semantics. We would like to describe a precise semantics for a large subset of the input language of the solver GRINGO based on representing GRINGO rules as infinitary propositional formulas. To prove strong equivalence of programs in this language we need a system of natural deduction for infinitary formulas, similar to intuitionistic propositional logic.

KEYWORDS: answer set programming, strong equivalence, semantics of aggregates.

1 Introduction

Answer set programming (ASP) is a powerful declarative paradigm for the design and implementation of knowledge intensive applications. It has been used in many areas of science and technology (Lifschitz 2008; Brewka et al. 2011). Its success is largely due to the expressivity of its modeling language and its efficient computation methods. The first ASP solvers were created more than ten years ago. One of their attractive features was that their input language had a simple, mathematically elegant semantics, based on the concept of a stable model (Gelfond and Lifschitz 1988).

Unfortunately, this cannot be said about the best, most powerful and efficient ASP solvers available today. Many constructs added over the years to the language of ASP because programmers found them useful cannot be explained in terms of stable models in the sense of the original definition of this concept and its straightforward generalizations. Consider for instance a rule with a conditional literal in the body:

$$p \leftarrow q : r.$$

It can be viewed as the nested implication

$$(r \rightarrow q) \rightarrow p,$$

written in logic programming notation. Stable models for such formulas can be defined using equilibrium logic (Pearce 1997) or in terms of reducts in the sense of (Ferraris 2005). However,

both of these definitions are quite different from the original definition of the stable model (Gelfond and Lifschitz 1988). When we look at manuals written by the designers of ASP solvers[1], we see that they explain the meaning of ASP programs using examples and informal comments that appeal to the user's intuition, without references to any precise semantics.

Without such a semantics, it is impossible to put the study of many important issues, such as the correctness of ASP solvers, programs, and optimization methods, on a firm scientific foundation. The absence of a precise semantics makes it difficult also to verify the correctness of ASP-based implementations of action languages and the relationship between input languages of different ASP solvers.

This note is a preliminary report regarding our work in the direction of describing a precise semantics for a large subset of the input language of the solver GRINGO. Our approach is based on representing GRINGO rules as infinitary propositional formulas (Truszczynski 2012).

We say that ASP programs A and B are *strongly equivalent* if for any set R of rules, the programs obtained by adding R to A and by adding R to B have the same stable models (Lifschitz et al. 2001). We would like to develop methods for proving strong equivalence of GRINGO programs. To this end, we define and study a system of natural deduction for infinitary formulas, similar to intuitionistic propositional logic.

2 Background: Stable Models of Infinitary Formulas

One of the reasons why infinitary formulas are an attractive formalism for defining the semantics of ASP languages is that they can be used to describe the semantics of aggregates. The semantics of aggregates proposed in (Ferraris 2005, Section 4.1) treats a ground aggregate as shorthand for a propositional formula. An aggregate with variables has to be grounded before that semantics can be applied to it. For instance, to explain the precise meaning of the expression $1\{p(X)\}$ ("there exists at least one object with the property p") in the body of an ASP rule we first rewrite it as

$$1\{p(t_1),\dots,p(t_n)\},$$

where $t_1,\dots,t_n$ are all ground terms in the language of the program, and then turn it into the propositional formula

$$p(t_1)\vee\cdots\vee p(t_n). \tag{1}$$

But this description of the meaning of $1\{p(X)\}$ implicitly assumes that the Herbrand universe of the program is finite. If the program contains function symbols then an infinite disjunction has to be used instead of (1). [2] [3]

[1] See, for instance, `http://sourceforge.net/projects/potassco/files/potassco_guide/` and `http://www.dlvsystem.com/dlvsystem/html/DLV_User_Manual.html`.

[2] This is not to say that there is anything exotic or noncomputable about ASP programs containing both aggregates and function symbols, however. For instance, the program

$$p(f(a))$$
$$q \leftarrow 1\{p(X)\}$$

has simple intuitive meaning, and its stable model $\{p(f(a)),q\}$ can be computed by existing solvers.

[3] References to grounding in other theories of aggregates suffer from the same problem. For instance, the definition of a ground instance of a rule in Section 2.2 of the ASP Core document (`https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.0.pdf`, Version 2.02) talks about replacing the expression $\{e_1;\dots;e_n\}$ in a rule with a set denoted by inst$(\{e_1;\dots;e_n\})$. But that set can be infinite.

The definitions of infinitary formulas and their stable models given below are equivalent to the definitions proposed in (Truszczynski 2012).

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

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

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

The definition of satisfaction familiar from classical propositional logic is extended to infinitary propositional formulas in a natural way.

The *reduct* F^I of a formula F with respect to an interpretation I is defined as follows:

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

The *reduct* $\mathscr{H}^I$ of a set $\mathscr{H}$ of formulas is the set consisting of the reducts of the elements of $\mathscr{H}$. An interpretation I is a *stable model* of a set $\mathscr{H}$ of formulas if it is minimal w.r.t. set inclusion among the interpretations satisfying $\mathscr{H}^I$.

3 Defining Semantics for Gringo Programs

In this note, we use Gringo to denote the input language of the solver GRINGO. The basis of this language is the language of logic programs with negation as failure, with the syntax and semantics defined in (Gelfond and Lifschitz 1988). In (Harrison et al. 2013b) we extend that semantics to a larger subset of Gringo. Specifically, we cover arithmetical functions and comparisons, conditions, and aggregates. Our proposal is based on the informal and sometimes incomplete description of the language in *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 key element of the semantics is a translation τ from Gringo into the language of infinitary propositional formulas described above. Like grounding in the original definition of a stable model (Gelfond and Lifschitz 1988), this translation is modular, in the sense that it applies to the program rule by rule.

The translation τ is defined first for literals, then for conditional literals, then for aggregate expressions, and then for rules. For example, the result of applying τ to the conditional literal

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

where X is a local variable, is the conjunction of the formulas

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

over all ground terms r. If the Herbrand universe of the program is infinite then the result of applying τ to a conditional literal will be an infinite conjunction.

A *stable model* of a Gringo program Π is a stable model, in the sense of the paper (Truszczynski 2012) reviewed in Section 2 above, of the set consisting of the formulas τR for all rules R of Π.

Instead of infinitary propositional formulas, we could have used first-order formulas with generalized quantifiers.[4] The advantage of 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 their corresponding infinitary formulas are equivalent in a natural deduction system, as discussed below.

4 Strong Equivalence of Infinitary Formulas

In (Harrison et al. 2013a) we define a basic infinitary system of natural deduction, analogous to an intuitionistic finite system of natural deduction. This system includes infinitary versions of the introduction and elimination rules for propositional connectives. For example, the rules

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

and

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

where $\mathscr{H}$ is a set of infinitary formulas, serve as infinitary analogs to the conjunction introduction and elimination rules of a finite system of natural deduction.

Theorem. *For any set $\mathscr{H}$ of (infinitary) formulas,*

(a) if a formula F is provable in the basic system then $\mathscr{H} \cup \{F\}$ has the same stable models as $\mathscr{H}$;
(b) if F is equivalent to G in the basic system then $\mathscr{H} \cup \{F\}$ and $\mathscr{H} \cup \{G\}$ have the same stable models.

This is a generalization of a well-known property of stable models for finite propositional formulas: intuitionistically equivalent formulas have the same stable models. The proof of this property presented in (Ferraris 2005), however, is based on ideas from equilibrium logic (Pearce 1997) and is very different from the proof for the infinitary case given in (Harrison et al. 2013a).

This theorem is useful because infinitary formulas can be used to precisely define the semantics of aggregates in ASP when the Herbrand universe is infinite. The following examples demonstrate how the theory described in (Harrison et al. 2013a) can be applied to prove equivalences between programs involving aggregates.

Example 1. The rule

$$p(Y) \leftarrow card\{X,Y : q(X,Y)\} \geq 1 \tag{2}$$

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

$$p(Y) \leftarrow q(X,Y) \tag{3}$$

[4] Stable models of formulas with generalized quantifiers are defined in (Lee and Meng 2012a; Lee and Meng 2012b; Lee and Meng 2012c).

within any program does not affect the set of stable models. To prove this claim, we need to calculate the result of applying τ to rule (2) and rule (3). The result of applying τ to (2) is

$$\bigwedge_t \left(\bigvee_u q(u,t) \ \rightarrow \ p(t) \right),$$

where t and u range over all ground terms. On the other hand, the result of applying τ to (3) is

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

These formulas are equivalent in the basic system.

Example 2. The rule

$$order(X,Y) \leftarrow p(X),\ p(Y),\ X < Y,\ not\ p(Z) : p(Z), X < Z, Z < Y \tag{4}$$

can be used for sorting.[5] It can be replaced by either of the following two simpler rules within any program without changing that program's stable models:

$$order(X,Y) \leftarrow p(X),\ p(Y),\ X < Y,\ \bot : p(Z), X < Z, Z < Y, \tag{5}$$

$$order(X,Y) \leftarrow p(X),\ p(Y),\ X < Y,\ not\ p(Z) : X < Z, Z < Y. \tag{6}$$

If we wish to prove this claim for rule (5), for example, by the theorem stated above, it is sufficient to show that the result of applying τ to (4) is equivalent to the result of applying τ to (5) in the basic system.

The result of applying τ to (4) is the conjunction of the formulas

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

for all numerals $i,\ j$. The result of applying τ to (5) is the conjunction of the formulas

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

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 (6) is similar. Rule (5), like rule (4), is safe; rule (6) is not.

Example 3. Consider the following rule from Example 3.7 of the *User's Guide* (see Footnote 1):

$$weekdays \leftarrow day(X) : day(X),\ not\ weekend(X). \tag{7}$$

Using the theorem above, we can show that (7) is strongly equivalent to *weekdays*. In other words, replacing this rule with the fact *weekdays* within any program would not affect its stable models.

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

5 Conclusion

Strong equivalence is an important notion in ASP. If a programmer knows that two rules are strongly equivalent then she may replace one rule with another, even in the context of a large program, and be assured that this change will not affect the stable models of the program. Two finite propositional formulas are strongly equivalent if and only if they are equivalent in the logic of here-and-there (Ferraris 2005, Proposition 2). The results in (Harrison et al. 2013a) are similar to the if part of that theorem. However, we don't know how to extend the only if part to infinitary formulas. It appears that axioms or inference rules not mentioned in that paper may be required, and identifying them is a topic for future work.

The project described in this note is directed towards defining a precise semantics for a subset of the input language of the solver GRINGO based on representing GRINGO rules as infinitary propositional formulas. A system of natural deduction for infinitary formulas, similar to intuitionistic propositional logic, allows us to prove strong equivalence of programs in this language.

Acknowledgements

My sincere thanks to Vladimir Lifschitz for the time he dedicated to editing this note and all the time he dedicates to teaching me. Thank you also to the anonymous reviewers who suggested a number of helpful improvements to a draft of this note.

References

BREWKA, G., NIEMELÄ, I., AND TRUSZCZYNSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM 54(12)*, 92–103.

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 *Proceedings of International Logic Programming Conference and Symposium*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

HARRISON, A., LIFSCHITZ, V., AND TRUSZCZYNSKI, M. 2013a. On equivalent transformations of infinitary formulas under the stable model semantics (preliminary report)[6]. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. To appear.

HARRISON, A., LIFSCHITZ, V., AND YANG, F. 2013b. On the semantics of Gringo[7]. In *Working Notes of the 6th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*. To appear.

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)*.

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

LIFSCHITZ, V. 2008. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*. MIT Press, 1594–1597.

LIFSCHITZ, V., PEARCE, D., AND VALVERDE, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2*, 526–541.

PEARCE, D. 1997. A new logical characterization of stable models and answer sets. In *Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, J. Dix, L. Pereira, and T. Przymusinski, Eds. Springer, 57–70.

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.