Formal Methods for Answer Set Programming

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 29 April 2015; accepted 5 June 2015

Abstract

Answer set programming is a logic programming paradigm that has increased in popularity over the past decade and found applications in a wide variety of fields. Even so, manuals written by the designers of answer set solvers usually described the semantics of the input languages of their systems using examples and informal comments that appeal to the users' intuition, without references to any precise semantics. We describe a precise semantics for the input language of the grounder GRINGO, which serves as the front end for several answer set solvers. The semantics represents GRINGO rules as infinitary propositional formulas. We develop methods for using this semantics to prove properties of GRINGO programs, such as verifying program correctness.

KEYWORDS: infinitary formulas, semantics of aggregates, stable models.

1 Introduction

Answer set programming (ASP) is a powerful declarative paradigm for the design and implementation of knowledge-intensive applications (Lifschitz 2008; Brewka et al. 2011). This paradigm leverages computational methods used in the design of fast satisfiability solvers. The first ASP solvers were created more than ten years ago. One of their attractive features was that their input language had a simple, fully specified semantics, based on the concept of a stable model (Gelfond and Lifschitz 1988). This semantics meant that ASP programs could be analyzed using formal methods.

As the number of ASP users increased, new constructs useful to programmers were added to input languages. This increase in expressivity, however, came at the expense of a fully specified semantics. Many of the new constructs lacked a precise mathematical formulation.

We have worked in close collaboration with the GRINGO system[1] designers to

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

bridge that gap. In (Gebser et al. 2015), we present a precise semantics which covers a large subset of the GRINGO input language. To the best of our knowledge, this semantics exactly matches the meaning of the input language for Version 4.5 of GRINGO, released on May 5, 2015. It covers a number of constructs not covered in our previous work, including intervals, pools, and lparse-style aggregates.

Our semantics is based on a translation from rules in the GRINGO input language to infinitary propositional formulas (formulas with infinite conjunctions and disjunctions). Previous work on the semantics of ASP used a translation to first-order formulas (Lee et al. 2008). However, such a translation is not applicable to aggregate expressions like

$$\#\mathtt{count}\{\mathtt{X} : \mathtt{p}(\mathtt{X})\} = \mathtt{Y}$$

where the result of applying the aggregate function is compared to a variable. In (Lee and Meng 2012) the authors propose a translation to logic with generalized quantifiers which can capture the meaning of the aggregate expression above.

With our semantics in place, we can ask questions regarding the correctness of ASP solvers, programs, and optimization methods, and we can develop general methods for answering such questions. The results presented in this note represent initial results on developing formal methods for approaching these kinds of questions.

Such a semantics is also a prerequisite for precisely describing the relationship between input languages of different ASP solvers. While this proposal deals with the input language of GRINGO in particular, it is worth noting that most ASP languages have very similar syntax. In this regard, formalizing a precise semantics and developing formal methods for the input language of GRINGO goes a long way towards doing the same for other ASP input languages.

In Section 2, we describe what we have accomplished in the direction of describing a precise semantics for the input language of GRINGO. In Section 3, we describe the infinitary logic which we will use to represent GRINGO programs, and in Section 4 we give a theoretical result which makes this logic an appealing representational choice. In Section 5, we show that the semantics can be used to establish the correctness of a GRINGO program. Finally, our plans for future work are documented in Section 6.

2 Defining Semantics for Gringo Programs

In this note we will use "Gringo" to denote the input language of GRINGO. In work currently in submission (Gebser et al. 2015), we extend the translational approach to defining the semantics of Gringo proposed in (Harrison et al. 2014b). In that note, we showed how rules containing arithmetical functions, comparisons, conditions, and aggregates can be translated into the language of infinitary propositional formulas described in Section 3. In (Gebser et al. 2015), we extend that semantics to cover intervals, pools, division of integers, aggregates with non-numeric values, and lparse-style aggregate expressions. That semantics serves as a specification for Gringo from Version 4.5 on.

The key element of the proposed semantics is a translation function from Gringo

into the language of infinitary propositional formulas. 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. For example, the result of applying this translation to the rule

$$\{\mathtt{q(1..n,1..n)}\} \tag{1}$$

is the formula

$$\bigwedge_{1\leq i,j\leq n} (q(i,j) \vee \neg q(i,j))\,.$$

(The expression `1..n` in rule (1) is an example of an interval.)

As another example, consider the aggregate expression

$$\#\mathtt{count}\{\mathtt{X} : \mathtt{q(X,1)}\} = \mathtt{1}. \tag{2}$$

If the ground atom $q(i,j)$ expresses that there is a queen at square (i,j) of a chessboard, then this aggregate expression says that there is exactly one queen in the first column. The translation of this aggregate expression can be written as

$$\bigvee_{t\in H} q(1,t) \wedge \bigwedge_{\substack{G\subseteq H\\ |G|=2}} \neg \bigwedge_{t\in G} q(1,t), \tag{3}$$

where H is the set of all ground terms, which may be infinite. This example illustrates the need for infinitary formulas. We will explain more precisely what is meant by "can be written as" in Section 4.

Extending the semantics proposed in (Harrison et al. 2014b) to cover more constructs is not entirely straightforward. To include intervals, for example, we have to modify the semantics from (Harrison et al. 2014b) in two ways. First, we have to say that an arithmetic term denotes, generally, a finite set of integers, not a single integer. (And it is not necessarily a set of consecutive integers, because the Gringo language allows us to write `(1..3)*2`, for instance. This expression denotes the set $\{2,4,6\}$.) Second, in the presence of intervals we cannot treat a choice rule `{A}` as shorthand for the disjunctive rule

```
A ; not A
```

as proposed in (Ferraris and Lifschitz 2005). Indeed, rule (1) has 2^{n^2} stable models; the rule

```
q(1..n,1..n) ; not q(1..n,1..n)
```

has only 2 stable models.

3 Review: Infinitary Formulas and their Stable Models

Infinitary formulas were originally introduced more than fifty years ago (Scott and Tarski 1958; Karp 1964). The definitions of infinitary formulas and their stable models given below are equivalent to those proposed in (Truszczynski 2012).

Let σ be a propositional signature, that is, a set of propositional atoms. For every nonnegative integer r, *(infinitary propositional) formulas (over σ) of rank r* are defined recursively, as follows:

- every atom from σ is a formula of rank 0,
- if $\mathcal{H}$ is a set of formulas, and r is the smallest nonnegative integer that is greater than the ranks of all elements of $\mathcal{H}$, then $\mathcal{H}^\wedge$ and $\mathcal{H}^\vee$ are formulas of rank r,
- if F and G are formulas, and r is the smallest nonnegative integer that is greater than the ranks of F and G, then $F \to G$ is a formula of rank r.

We will write $\{F, G\}^\wedge$ as $F \wedge G$, and $\{F, G\}^\vee$ as $F \vee G$. The symbol $\bot$ will be understood as an abbreviation for $\emptyset^\vee$ and $\neg F$ stands for $F \to \bot$. These conventions allow us to view finite propositional formulas over σ as a special case of infinitary formulas.

A set or family of formulas is *bounded* if the ranks of its members are bounded from above. 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 *interpretations* of σ. The satisfaction relation between an interpretation and a formula is defined recursively, as follows:

- For every atom p from σ, $I \models p$ if $p \in I$.
- $I \models \mathcal{H}^\wedge$ if for every formula F in $\mathcal{H}$, $I \models F$.
- $I \models \mathcal{H}^\vee$ if there is a formula F in $\mathcal{H}$ such that $I \models F$.
- $I \models F \to G$ if $I \not\models F$ or $I \models G$.

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

- For every atom p from σ, p^I is p if $p \in I$, and $\bot$ otherwise.
- $(\mathcal{H}^\wedge)^I = \{G^I \mid G \in \mathcal{H}\}^\wedge$.
- $(\mathcal{H}^\vee)^I = \{G^I \mid G \in \mathcal{H}\}^\vee$.
- $(G \to H)^I$ is $G^I \to H^I$ if $I \models G \to H$, and $\bot$ otherwise.

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 F^I of all formulas F from $\mathcal{H}$.

4 Strong Equivalence of Infinitary Formulas

About sets $\mathcal{H}_1$, $\mathcal{H}_2$ of infinitary formulas we say that they are *strongly equivalent* to each other if, for every set $\mathcal{H}$ of infinitary formulas, the sets $\mathcal{H}_1 \cup \mathcal{H}$ and $\mathcal{H}_2 \cup \mathcal{H}$ have the same stable models. About formulas F and G we say that they are *strongly equivalent* if the singleton sets $\{F\}$ and $\{G\}$ are strongly equivalent. Strong equivalence is an important property because if the translations of two rules are strongly equivalent, then we can replace one rule with the other within a Gringo program without changing the meaning of that program. Strong equivalence is also important because it allows us to simplify infinitary formulas that may be difficult to reason about. For example, recall aggregate expression (2) from Section 2. Applying our translation to that aggregate expression yields a formula that is syntactically complicated; however, that formula is strongly equivalent to (3).

It is well-known that finite propositional formulas are strongly equivalent if and

only if their equivalence is provable in a 3-valued logic called the logic of here-and-there (Lifschitz et al. 2001). In (Harrison et al. 2015), we define and axiomatize an infinitary version of that logic and show that infinitary propositional formulas are strongly equivalent to each other if and only if they are equivalent in the infinitary logic of here-and-there. Our axiomatization of this logic, called HT^∞, includes infinitary versions of the introduction and elimination rules for propositional connectives. For example, the rules

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

and

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

where $\mathcal{H}$ is a possibly infinite set of infinitary formulas and Γ is a finite set of infinitary formulas, serve as infinitary analogs to the conjunction introduction and elimination rules of a finite system of natural deduction. It also includes the following axiom schemas:

$$F \Rightarrow F,$$

$$F \vee (F \to G) \vee \neg G, \tag{4}$$

and

$$\bigwedge_{\alpha \in A} \bigvee_{F \in \mathcal{H}_\alpha} F \to \bigvee_{(F_\alpha)_{\alpha \in A}} \bigwedge_{\alpha \in A} F_\alpha \tag{5}$$

for every non-empty family $(\mathcal{H}_\alpha)_{\alpha \in A}$ of sets of formulas such that its union is bounded; the disjunction in the consequent of (5) extends over all elements $(F_\alpha)_{\alpha \in A}$ of the Cartesian product of the family $(\mathcal{H}_\alpha)_{\alpha \in A}$.

5 Example: The Correctness of a Gringo Program

In this section, we present a theorem, proved in (Gebser et al. 2015), that expresses the correctness of a simple, but nontrivial Gringo program with respect to the translation defined in that paper.

Table 1 shows an ASP encoding of the n-queens problem. It is similar to the most optimized of the solutions in the language of GRINGO Version 3 presented in (Gebser et al. 2011). We will call this program K.

The n-queens problem involves placing n queens on an $n \times n$ chess board such that no two queens threaten each other. We will represent squares by pairs of integers (i, j) where $1 \leq i, j \leq n$. Two squares (i_1, j_1) and (i_2, j_2) are said to be in the same row if $i_1 = i_2$; in the same column if $j_1 = j_2$; and in the same diagonal if $|i_1 - i_2| = |j_1 - j_2|$. A set Q of n squares is a *solution* to the n-queens problem if no two elements of Q are in the same row, in the same column, or in the same diagonal.

Let τK denote the result of applying the translation defined in our semantics to the program K. For any stable model I of τK, by Q_I we denote the set of pairs (i, j) such that $q(i, j) \in I$.

```
% place queens on the chess board
{ q(1..n,1..n) }.

% exactly 1 queen per row/column
:- X = 1..n, not #count{ Y : q(X,Y) } = 1.
:- Y = 1..n, not #count{ X : q(X,Y) } = 1.

% pre-calculate the diagonals
d1(X,Y,X-Y+n) :- X = 1..n, Y = 1..n.
d2(X,Y,X+Y-1) :- X = 1..n, Y = 1..n.

% at most one queen per diagonal
:- D = 1..n*2-1, 2 { q(X,Y) : d1(X,Y,D) }.
:- D = 1..n*2-1, 2 { q(X,Y) : d2(X,Y,D) }.
```

Table 1. *An ASP encoding of the n-queens problem.*

Theorem 1
For each stable model I of τK, Q_I is a solution to the n-queens problem. Furthermore, for each solution Q to the n-queens problem there is exactly one stable model I of τK such that $Q_I = Q$.

The proof of the theorem uses as lemmas some facts regarding how we may simplify the result of applying our translation to aggregate expressions involving the count aggregate function (Gebser et al. 2015, Section 5.3). It was shown previously (Harrison et al. 2014b) that the infinitary propositional formulas corresponding to the count aggregate can be simplified by strongly equivalent transformations using its monotonicity properties. In (Gebser et al. 2015), we show how these formulas can be further simplified under the assumption that intervals do not occur in some parts of the aggregate expression.

6 A Summary of Future Work

Another application of this work would be to prove the correctness of the computational methods used in GRINGO and in the ASP solvers that use this grounder as their front end. Currently, those solvers employ a number of "intelligent instantiation" algorithms which allow the grounder to function efficiently (Gebser et al. 2009). Using our semantics we will work with the designers of GRINGO to prove that these algorithms are correct.

As previously mentioned, a semantics for Gringo takes us a long way in the direction of a semantics for most ASP languages. Exploring how these languages compare and contrast is yet another direction of future work. An example of one such language is the ASP Core language (Calimeri et al. 2012).

Finally, there are a number of theoretical questions regarding the infinitary sys-

tem from (Harrison et al. 2014a) which we would like to investigate. For example, it is well-known that any finite propositional formula which begins with negation and is classically provable is also intuitionistically provable (Glivenko 1929). One question that we would like to address is whether or not an analogous theorem holds of the system HT^{∞}. Another is whether or not we can succinctly characterize safety of a Gringo program using properties of its translation. Answering these questions and others like them will help build the arsenal of formal methods at our disposal for addressing more immediately practical questions, for example, questions of program correctness.

Acknowledgements

My sincere thanks to my friend and mentor Vladimir Lifschitz. Thanks also to the anonymous reviewers. This research was partially supported by the National Science Foundation under Grant IIS-1422455.

References

Brewka, G., Niemelä, I., and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM 54(12)*, 92–103.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., and Schaub, T. 2012. ASP-Core-2: Input language format. Available at `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.0.pdf`.

Ferraris, P. and Lifschitz, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5,* 1–2, 45–74.

Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V., and Schaub, T. 2015. Abstract Gringo. In *Proceedings of International Conference on Logic Programming (ICLP)*. `http://www.cs.utexas.edu/users/vl/papers/AG.pdf`; to appear.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. 2011. Challenges in answer set solving. In *Logic programming, knowledge representation, and nonmonotonic reasoning*. Springer, 74–90.

Gebser, M., Kaminski, R., Ostrowski, M., Schaub, T., and Thiele, S. 2009. On the input language of ASP grounder gringo. In *LPNMR*.

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.

Glivenko, V. 1929. Sur quelques points de la logique de M. Brouwer. *Académie Royale de Belgique. Bulletins de la Classe des Sciences, série 5 15*, 183–188.

Harrison, A., Lifschitz, V., Pearce, D., and Valverde, A. 2014a. Infinitary equilibrium logic. In *Working Notes of the 7th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*.

Harrison, A., Lifschitz, V., Pearce, D., and Valverde, A. 2015. Infinitary equilibrium logic and strong equivalence. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. `http://www.cs.utexas.edu/users/vl/papers/iel_lpnmr.pdf`; to appear.

Harrison, A., Lifschitz, V., and Yang, F. 2014b. The semantics of Gringo and infinitary propositional formulas. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Karp, C. R. 1964. *Languages with expressions of infinite length.* North-Holland, Amsterdam.

Lee, J., Lifschitz, V., and Palla, R. 2008. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 472–479.

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

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.

Scott, D. and Tarski, A. 1958. The sentential calculus with infinitely long expressions. In *Colloquium Mathematicae*. Vol. 6. 165–170.

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*, E. Erdem, J. Lee, Y. Lierler, and D. Pearce, Eds. Springer, 543–559.