anthem: Transforming gringo Programs into First-Order Theories (Extended Abstract)

Vladimir Lifschitz[1], Patrick Lühne[2], and Torsten Schaub[2,3]

[1] University of Texas at Austin, USA
[2] University of Potsdam, Germany
[3] INRIA Rennes, France

Abstract. In a recent paper by Harrison et al., the concept of program completion is extended to a large class of programs in the input language of the ASP grounder GRINGO. We would like to automate the process of generating and simplifying completion formulas for programs in that language, because examining the output produced by this kind of software may help programmers to see more clearly what their program does and to what degree its set of stable models conforms with their intentions. If a formal specification for the program is available, then it may be possible to use this software, in combination with automated reasoning tools, to verify that the program is correct. This note is a preliminary report on a project motivated by this idea.

Harrison et al. [5] extended the concept of program completion [1] to a large class of nondisjunctive programs in the input language of the ASP grounder GRINGO [4]. They argued that it would be useful to automate the process of generating and simplifying completion formulas for (tight[4]) GRINGO programs, because examining the output produced by this kind of software may help programmers to see more clearly what their program does and to what degree its set of stable models conforms with their intentions. Furthermore, if a formal specification for a GRINGO program is available, then it may be possible to use this software, in combination with automated reasoning tools, to verify that the program is correct.

This note is a preliminary report on a software development project that follows up on this idea. ANTHEM is a translator that converts a GRINGO program into its completion and simplifies it. By *simplifying* we mean, in this case, not so much making formulas shorter as writing them in a form that is "readable"—natural from the perspective of a human who is accustomed to expressing mathematical ideas using propositional connectives, quantifiers, variables for objects of various types, the summation symbol, and other standard notation. The language of GRINGO and many other input languages of answer set solvers, including those of SMODELS [7] and DLV [6], classify variables into global and local, instead of using quantifiers to classify occurrences into free and bound, and that distinguishes them from traditional notation. The same can be said about assuming

[4] *Tightness* is a syntactic condition that guarantees the equivalence between the stable model semantics and the completion semantics of a logic program [3, 2].

that all variables range over the same universe, instead of using variables of different sorts or types (for points, lines, and planes; for integers and real numbers; or for sets and classes; etc.). Each of the two notational traditions has its advantages, and ANTHEM provides a bridge between them.

Besides generating and simplifying the completion of a program, ANTHEM "hides" auxiliary predicate symbols occurring in the program when possible. In the language of GRINGO, the fact that a predicate symbol is not considered an essential part of the output can be expressed by not including it in **`#show`** directives. To eliminate such predicate symbols from its output, ANTHEM replaces them by their completed definitions.

The input language of ANTHEM is a large part of the input language of GRINGO. Input programs are supposed to be nondisjunctive. They may use arithmetic operations, intervals, comparisons, singleton choice rules without lower and upper bounds, and constraints. Aggregates and conditional literals are not supported in the current version.

The output of ANTHEM is a list of first-order formulas with variables of two sorts—for arbitrary precomputed terms (that is, for all elements of the Herbrand universe) and for the precomputed terms that correspond to integers—as proposed by Harrison et al. [5, Sections 3 and 9]. Differences between atoms in GRINGO programs and atomic parts of formulas are related mostly to arithmetic expressions.

References

1. Clark, K.: Negation as failure. In: Logic and Data Bases, pp. 293–322. Plenum Press (1978)
2. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming **3**(4-5), 499–518 (2003)
3. Fages, F.: Consistency of Clark's completion and the existence of stable models. Journal of Methods of Logic in Computer Science **1**, 51–60 (1994)
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. Harrison, A., Lifschitz, V., Raju, D.: Program completion in the input language of GRINGO. Theory and Practice of Logic Programming **17**(5-6), 855–871 (2017)
6. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3), 499–562 (2006)
7. Niemelä, I., Simons, P.: Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In: Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97). Lecture Notes in Artificial Intelligence, vol. 1265, pp. 420–429. Springer-Verlag (1997)