spock - Detecting Errors in Logic Programs under the Answer-Set Semantics: System Description*

Martin Gebser[1], Jörg Pührer[2], Torsten Schaub[1],
Hans Tompits[2], and Stefan Woltran[2]

[1] Institut für Informatik, Universität Potsdam,
August-Bebel-Straße 89, D-14482 Potsdam, Germany
{gebser,torsten}@cs.uni-potsdam.de
[2] Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Vienna, Austria
{puehrer,tompits,stefan}@kr.tuwien.ac.at

Abstract. Answer-set programming (ASP) is an emerging logic-programming paradigm, which strictly separates the description of a problem from its solutions. Despite its semantic elegance, ASP suffers from a lack of comfort for software developers. In particular, tools are needed which support engineers in detecting errornous parts of their programs . Unlike in other areas of logic programming, applying tracing techniques for debugging logic programs under the answer-set semantics seems rather unnatural, since sticking to imperative solving algorithms would undermine the declarative flavor of ASP.
In this system description, we present the system spock, a debugging support tool for answer-set programs making use of ASP itself. The implemented techniques maintain the declarative nature of ASP within the debugging process and are independent from the actual computation of answer sets.

1 General Information

Answer-set programming (ASP) [3] has become an important logic-programming paradigm for declarative problem solving, which incorporates fundamental concepts of non-monotonic reasoning. The non-monotonicity of answer-set programs, however, is an aggravating factor for detecting sources of errors, since every rule of a program might significantly influence the resulting answer sets.

The system spock [2] supports developers of answer-set programs to detect and locate errors in their programs, independent of specific ASP solvers.

The theoretical background of the implemented methods was introduced in previous work [1], and exploits and extends a *tagging technique*, as used by Delgrande et al. [4] for compiling ordered logic programs into standard ones. In our approach, a program to debug, Π, is translated into another program, $\mathcal{T}_K[\Pi]$, equipped with specialized meta-atoms, called *tags*, serving two purposes. Firstly, they allow for controlling and

* This work was partially supported by the Austrian Science Fund (FWF) under project P18019.

manipulating the formation of answer sets of Π and secondly, tags occurring in the answer sets of the translated program reflect various properties of Π.

In addition to $\mathcal{T}_{\mathsf{K}}$, `spock` is devised to support supplementary translations for a program to debug Π, allowing an extrapolation of non-existing answer sets in combination with explanations why an interpretation is not an answer set of Π.

2 System Specifics

Our debugging system `spock` implements several transformations for debugging propositional normal logic programs, involving the tags ap, bl, ok, ko, ab_p, ab_c, and ab_l. The basic measure of tagging is to split up the rules of an original program, into their heads and bodies, separating the causal relation between the satisfaction of a rule body from the occurrence of the respective heads in an interpretation.

One task of tags is providing information about the program to debug Π. E.g., there is a one-to-one correspondence between the answer sets of Π and the answer sets of $\mathcal{T}_{\mathsf{K}}[\Pi]$, such that an answer set of the translated program contains either tag $\mathsf{ap}(l_r)$ or tag $\mathsf{bl}(l_r)$, for each rule $r \in \Pi$, where l_r is a unique label for r, thus providing information whether r is applicable or blocked in the related answer set of Π. As an example consider program $\Pi_{\mathsf{ex}} = \{r_1 = a \leftarrow b,\ r_2 = b \leftarrow \mathit{not}\ c,\ r_3 = c \leftarrow \mathit{not}\ a\}$, having the answer sets $\{a, b\}$ and $\{c\}$. Then the answer sets of the transformed program $\mathcal{T}_{\mathsf{K}}[\Pi_{\mathsf{ex}}]$ are given by $\{a, b, \mathsf{ap}(n_{r_1}), \mathsf{ap}(n_{r_2}), \mathsf{bl}(n_{r_3})\} \cup OK$ and $\{c, \mathsf{ap}(n_{r_3}), \mathsf{bl}(n_{r_1}), \mathsf{bl}(n_{r_2})\} \cup OK$, where $OK = \{\mathsf{ok}(n_{r_1}), \mathsf{ok}(n_{r_2}), \mathsf{ok}(n_{r_3})\}$. From these we gain the information that, e.g., the rule labeled n_{r_1} is applicable under answer set $\{a, b\}$, and blocked under $\{c\}$.

Apart from being used for analyzing Π and its answer sets, tagging provides a handle on the formation of answer sets of Π. By joining the translated program with rules involving control tags $\mathsf{ko}(l_r)$, and $\mathsf{ok}(l_r)$, rule $r \in \Pi$ can selectively be deactivated.

This feature can be utilized by more advanced debugging modules, based on our kernel transformation $\mathcal{T}_{\mathsf{K}}$, which are not restricted to analyzing actual answer sets of Π. In particular `spock` features program translations for investigating, why a particular interpretation I is not an answer set of program Π. Here, three sources of errors are identified on the basis of Lin-Zhao Theorem [6]. Reasons for I not being an answer set are either related to the program, its completion, or its non-trivial loop formulas, respectively. According to this error classification, we distinguish between three corresponding *abnormality* tags, ab_p, ab_c, and ab_l, which may occur in the answer sets of the transformed program. The program-oriented abnormality tag $\mathsf{ab}_p(n_r)$ indicates that rule $r \in \Pi$ is applicable but not satisfied with respect to the considered interpretation I, the completion-oriented abnormality tag $\mathsf{ab}_c(a)$ is contained whenever atom a is in I but all rules having a as head are blocked. Finally, the presence of a loop-oriented abnormality tag $\mathsf{ab}_l(a)$ indicates the possible existence of some loop Γ in Π, $a \in \Gamma$, that is unfounded with respect to I.

As the number of interpretations for a program grows exponentially in the number of occurring atoms, we use standard optimization techniques of ASP to reduce the amount of debugging information, focusing on answer sets of the tagged program, which involve a minimum number of abnormality tags.

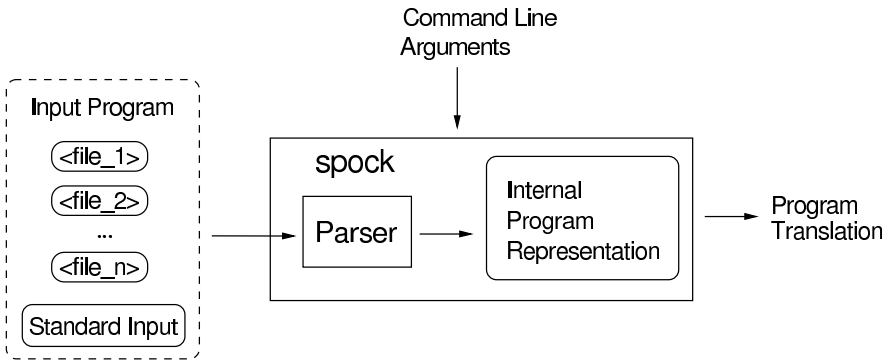

Fig. 1. Data flow of program translations.

The transformations to be applied are chosen by setting call parameters. Fig. 1 illustrates the typical data flow of program translations with `spock`. The tool is written in Java 5.0 and published under the GNU General Public License [7]. It can be used either with `DLV` [8] or with `Smodels` [5] (together with `lparse`) and is available at

`http://www.kr.tuwien.ac.at/research/debug`

as a jar-package including binaries and sources.

References

1. M. Brain, M. Gebser, J. Pührer, T. Schaub, H. Tompits, and S. Woltran. Debugging ASP programs by means of ASP. In C. Baral, G. Brewka, and J. Schlipf, editors, *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, number 4483 in Lecture Notes in Artificial Intelligence, pages 31–43. Springer, 2007.
2. M. Brain, M. Gebser, J. Pührer, T. Schaub, H. Tompits, and S. Woltran. "That is illogical captain!" – The debugging support tool spock for answer-set programs: System description. In M. De Vos and T. Schaub, editors, *Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07)*, pages 71–85, 2007.
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
4. Delgrande, J., Schaub, T., Tompits, H.: A framework for compiling preferences in logic programs. Theory and Practice of Logic Programming **3**(2) (2003) 129–187
5. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
6. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**(1-2) (2004) 115–137
7. Free Software Foundation Inc.: GNU General Public License - Version 2, June 1991 (1991) http://www.gnu.org/copyleft/gpl.html
8. 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) (2006) 499–562