Debugging Answer-Set Programs with Ouroboros – Extending the SeaLion Plugin*

Melanie Frühstück[1], Jörg Pührer[2], and Gerhard Friedrich[3]

[1] Siemens AG Österreich, Corporate Technology, Vienna, Austria
melanie.fruehstueck@siemens.com
[2] Technische Universität Wien,
Institut für Informationssysteme 184/3,
Favoritenstraße 9-11, A–1040 Vienna, Austria
puehrer@kr.tuwien.ac.at
[3] Alpen-Adria Universität, Klagenfurt, Austria
Gerhard.Friedrich@ifit.uni-klu.ac.at

Abstract. In answer-set programming (ASP), there is a lack of debugging tools that are capable of handling programs with variables. Hence, we implemented a tool, called `Ouroboros`, for debugging non-ground answer-set programs. The system builds on a previous approach based on ASP meta-programming that has been recently extended to cover weight constraints and choice rules. The main debugging question addressed is "given a program P and an interpretation I, why is I not an answer set of P". Our tool gives answers in terms of two categories of explanations: unsatisfied rules and unfounded loops. `Ouroboros` is a plugin of the `SeaLion` integrated development environment for ASP that is built on Eclipse. Thereby, `Ouroboros` complements and profits from `SeaLion`'s `Stepping` plugin, that implements a different debugging approach for ASP.

1 Introduction

Answer-set programming (ASP) is a well-known declarative problem-solving paradigm [1]. While a great deal of work on ASP implementations has been put into improving solver performance, comparably little effort has been spent on tools that support the development of answer-set programs, in particular, there is a lack of *debugging systems* for ASP. But, in recent years, methods for debugging have been explored theoretically [2–6]. Brain and De Vos [2] discussed what it means for answer-set programs to be incorrect and presented algorithms to locate bugs. Syrjänen [3] proposed to debug contradictory programs by means of ASP meta-programming. Gebser et al. [4] tried to find semantic errors of answer-set programs. The question is why an expected interpretation is not an answer set of a program (`spock` [7] implements this approach).

* This research has been funded by FFG FIT-IT (grant number 825071) within the scope of the RECONCILE project and by the Austrian Science Fund (FWF P21698).

P. Cabalar and T.C. Son (Eds.): LPNMR 2013, LNAI 8148, pp. 323–328, 2013.

However, these approaches are only able to deal with propositional programs which is clearly a limiting factor as far as practical applications are concerned. Therefore, Oetsch et al. [8] developed a meta-program for debugging non-ground programs in ASP. This approach is based on the meta-programming technique of Gebser et al. [4] for propositional programs. Recently, their method has been further extended to cover weight and cardinality constraints [9]. In this paper, we describe the debugging system `Ouroboros`, which implements this approach and explains why an expected interpretation is not an answer set of a given program. The system gives answers in terms of two categories of explanations: unsatisfied rules and unfounded loops. Intuitively, a rule is unsatisfied if its body is true but all literals in its head are false. Moreover, an unfounded loop is a set of atoms from the interpretation whose truth can only be derived from itself but is not founded in facts. Thus, an unfounded loop is reminiscent of the Ouroboros, a dragon biting its own tail, which our tool is named after. `Ouroboros`[1] is a plugin of `SeaLion` [10, 11], an integrated development environment (IDE) for ASP that is based on the Eclipse platform and supports developing answer-set programs using the *Potassco* and `DLV` solvers [12, 13]. Thereby, `Ouroboros` complements the *stepping-based debugging mechanism* [14] integrated in `SeaLion` [11]. It allows the user to interactively build up an interpretation by, stepwise, adding literals derived by a rule whose body is satisfied by the interpretation obtained in the previous step.

While, on the one hand, `Ouroboros` provides additional debugging functionality for `SeaLion`, on the other hand, it also profits from the `Stepping`-plugin which can help in building up the interpretation that is input to our approach. Another possibility to create an interpretation is to use the `Kara` plugin of `SeaLion` [15].

2 Backend

As mentioned in the introduction, `Ouroboros` makes use of ASP meta-programming to find explanations why a given interpretation I is not an answer set of the program P under development. The internal data flow of `Ouroboros` is depicted in Fig. 1. In a preprocessing step, all cardinality constraints of P are translated into standard rules [2]. Then, P and the expected interpretation I are reified, i.e., P and I are brought onto a meta-level (i.e. a fact `person(1)` would be presented by rule(r1). head(r1,r1h1). pred(r1h1,person). struct(r1h1,1,const,1).), represented by facts, and joined with rules for identifying the targeted explanations. Finally, the meta-program is fed to an ASP solver and the resulting answer sets get interpreted.

3 Usage and Graphical User Interface

The `Ouroboros` plugin itself comprises two graphical components, the *Debug Configuration Tab Group* for defining parameters for a debugging session and the

[1] The plugin is open source and available from `http://www.sealion.at`

[2] For a detailed description of these translations we refer the interested reader to a companion paper [9].

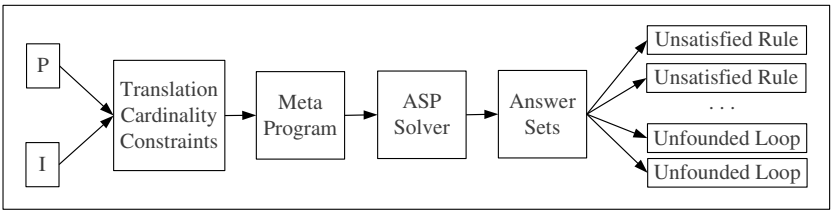

Fig. 1. Data Flow in Ouroboros

Debugging Explanation View that provides the explanations found to the user. In the following two subsections, both will be clarified by means of an example. The example is based on the original house problem [16] that is an abstraction of several configuration problems where entities may be contained in other entities and some additional requirements are defined. We considered a simplification of the modification of this problem [17, 18]. Given a set of cabinets, rooms, persons and objects, the problem consists of assigning objects to cabinets, cabinets to rooms and rooms to persons, such that following constraints hold: cabinets and rooms can contain only a specific number of objects and cabinets, respectively; objects belonging to different persons cannot be placed in the same cabinet; cabinets of different persons cannot be placed in the same room. Fig. 2 depicts the original program and the expected interpretation (in the interpretation view).

3.1 Debug Configurations

Debug configurations are similar to run configurations in Eclipse. They are used to start an application in the debug mode. When clicking on `Ouroboros` in the debug configurations, three tabs occur which the user can select, where one tab is the *Common* Eclipse tab. Let us assume that a user, called Benia, wants to debug the program given in Fig. 2. In the *Input Program/Interpretation* tab Benia selects the program file and the expected interpretation file. When clicking on the *add* button, a window occurs for selecting files from the Eclipse workspace. The currently opened file in the editor is preselected. As Benia wants to check for unsatisfied rules in the program, he selects the explanation type *Unsatisfiability*.

In the *Solver* tab, Benia chooses `Gringo/clasp` as solver configuration. If he had checked for unfounded loops instead unsatisfied rules, the solver would have to be able to deal with disjunctions as they are needed in the meta-program. In that case one can set the first check mark on the bottom of the tab to filter for solver configurations marked as `claspD` configurations. Using some other solver that is able to deal with disjunctions, requires the second check mark to be set.

After all required attributes of the debug configuration are set, Benia can start the debugging process. Now, all steps described in Fig. 1 are run through. When the final answer set is computed, the explanation why the given interpretation is not answer set of the given program is shown in the debugging explanation view.

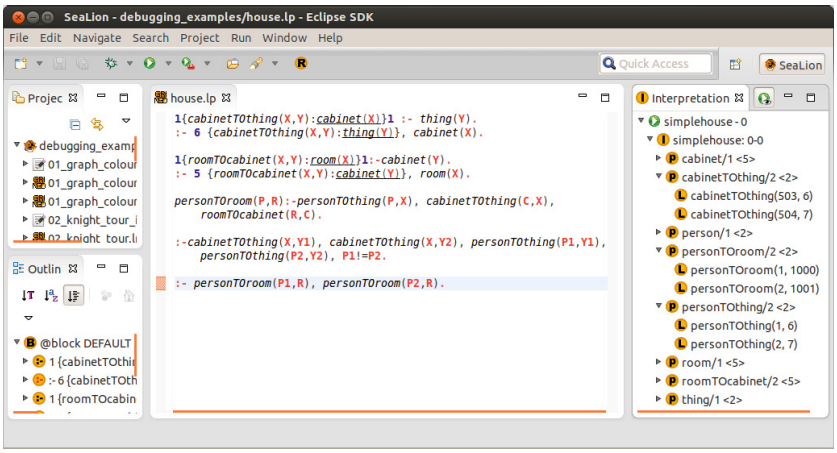

Fig. 2. The given program and the expected interpretation

3.2 Debugging Explanation View

The debugging explanation view consists of two columns. In the first column, the explanation is shown. This is either the rule that is unsatisfied with respect to the interpretation or an unfounded loop. In the second column the meta-programming predicates are shown. This can be either `guessRule/1` and `subst/2` (the former states about the unsatisfied rule and the latter about its substitution) or `inLoop/1` (all literals that form an unfounded loop). Even if the output of the debugger concerns a translation rule of a cardinality constraint, it is mapped back to the cardinality constraint itself. Additionally, all reasons of why the given interpretation is not an answer set of the program are given. In the case of Benia's program, the explanation represents the last constraint given in the program. In particular, the additional condition `P1!=P2` is missing, that means that the overall configuration does not allow a room belonging to two different persons. Thus, one room can only belong to one person. When Benia clicks on the explanation the corresponding rule is highlighted in the editor (see Fig. 3). If the explanation refers to a rule in a specific file, this file is automatically opened.

3.3 Additional Features in the Interpretation View

In addition to the main two components described above, the context menu of the interpretation view of `SeaLion` was extended. In general, the interpretation view provides a tree structure of each answer set of the executed program (cf. Fig. 2). In the context menu of the interpretation some new functionalities were added. The entry *Save as Facts* makes it possible to save the selected interpretation as facts. To do so, the user has to select a project in which the new file is inserted. The predefined name of the new file can be adapted. After finishing this process the file including the facts of the interpretation is opened in the editor.

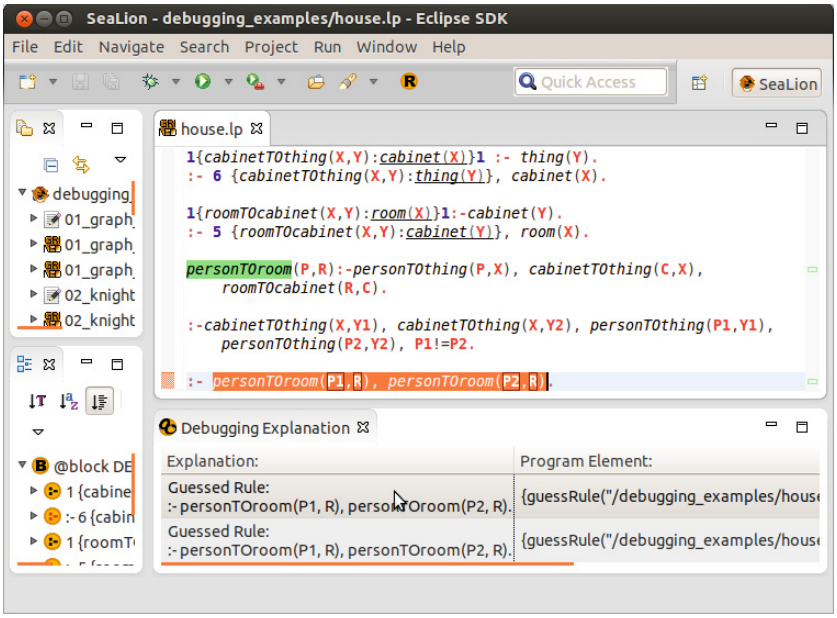

Fig. 3. Debugging Explanation View

Moreover, the user can select *Detect Unsatisfied Rules* if he or she wants to create a new debug configuration where the input interpretation is the one on which the context menu was opened and where the explanation type is automatically set to unsatisfied rules. *Detect Unsatisfied Rules for Launch* lets the user first select an existing debug launch configuration in which the interpretation file is substituted with the interpretation selected. Again, the explanation type is automatically set to unsatisfied rules. The entries *Detect Unfounded Loops* as well as *Detect Unfounded Loops for Launch* work analogously to the just described functionalities, except that the explanation type is set to unfounded loops.

If these four functionalities are used, the user has to be aware of the fact that the selected interpretations are just temporary files. That means, if the user exits Eclipse the interpretation files will get lost. To make them persistent the functionality *Save as Facts* can be used.

4 Conclusion

In this paper we described the debugging extension called `Ouroboros` in the `SeaLion` plugin for Eclipse. It provides debugging support of `Gringo` programs involving variables and cardinality constraints by explaining why a given interpretation is not an answer set of a given program. We concisely presented the the components and functionalities of the `Ouroboros` plugin and, by means of a debugging example, showed how the user can kick off the debugging process.

References

1. Gelfond, M., Leone, N.: Logic programming and knowledge representation - The A-Prolog perspective. Art. Intell. 138(1-2), 3–38 (2002)
2. Brain, M., De Vos, M.: Debugging logic programs under the answer set semantics. In: 3rd International Workshop on Answer Set Programming (ASP 2005). CEUR Workshop Proceedings, pp. 141–152 (2005)
3. Syrjänen, T.: Debugging inconsistent answer set programs. In: Proc. NMR 2006, pp. 77–83 (2006)
4. Gebser, M., Pührer, J., Schaub, T., Tompits, H.: A meta-programming technique for debugging answer-set programs. In: Proc. AAAI 2008, pp. 448–453. AAAI Press (2008)
5. Caballero, R., García-Ruiz, Y., Sáenz-Pérez, F.: A theoretical framework for the declarative debugging of datalog programs. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 143–159. Springer, Heidelberg (2008)
6. Pontelli, E., Son, T.C., Elkhatib, O.: Justifications for logic programs under answer set semantics. TPLP 9(1), 1–56 (2009)
7. Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: `spock`: A debugging support tool for logic programs under the answer-set semantics. In: Seipel, D., Hanus, M., Wolf, A. (eds.) INAP 2007. LNCS, vol. 5437, pp. 247–252. Springer, Heidelberg (2009)
8. Oetsch, J., Pührer, J., Tompits, H.: Catching the Ouroboros: On debugging non-ground answer-set programs. TPLP 10(4-6), 513–529 (2010)
9. Polleres, A., Frühstück, M., Schenner, G., Friedrich, G.: Debugging non-ground ASP programs with choice rules, cardinality and weight constraints. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. LNCS (LNAI), vol. 8148, pp. 452–464. Springer, Heidelberg (2013)
10. Oetsch, J., Pührer, J., Tompits, H.: The SeaLion has landed: An IDE for answer-set programming—Preliminary report. In: Proc. WLP 2011 (2011)
11. Busoniu, P., Oetsch, J., Pührer, J., Skočovský, P., Tompits, H.: `SeaLion`: An Eclipse-based IDE for answer-set programming with advanced debugging support (submitted draft, 2013)
12. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.T.: Potassco: The Potsdam answer set solving collection. AI Commun. 24(2), 107–124 (2011)
13. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Logic 7(3), 499–562 (2006)
14. Oetsch, J., Pührer, J., Tompits, H.: Stepping through an answer-set program. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011. LNCS (LNAI), vol. 6645, pp. 134–147. Springer, Heidelberg (2011)
15. Kloimüllner, C., Oetsch, J., Pührer, J., Tompits, H.: `Kara` - A system for visualising and visual editing of interpretations for answer-set programs. In: Proc. WLP 2011, pp. 152–164 (2011)
16. Mayer, W., Bettex, M., Stumptner, M., Falkner, A.: On solving complex rack configuration problems using CSP methods. In: Proc. IJCAI 2009 Workshop on Configuration (2009)
17. Friedrich, G., Ryabokon, A., Falkner, A., Haselböck, A., Schenner, G., Schreiner, H.: (re)configuration using answer set programming. In: Proc. IJCAI 2011 Workshop on Configuration, pp. 17–25 (2011)
18. Aschinger, M., Drescher, C., Vollmer, H.: LoCo—A logic for configuration problems. In: Proc. ECAI 2012, vol. 242, pp. 73–78 (2012)