Program Debugging and Validation Using Semantic Approximations and Partial Specifications

M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García

School of Computer Science
Technical University of Madrid, Spain
herme@fi.upm.es, http://www.clip.dia.fi.upm.es/~herme

(Extended Abstract)

The technique of Abstract Interpretation [11] has allowed the development of sophisticated program analyses which are provably correct and practical. The semantic approximations produced by such analyses have been traditionally applied to *optimization* during program compilation. However, recently, novel and promising applications of semantic approximations have been proposed in the more general context of program *validation* and *debugging* [3, 9, 7].

We study the case of (Constraint) Logic Programs (CLP), motivated by the comparatively large body of approximation domains, inference techniques, and tools for abstract interpretation-based semantic analysis which have been developed to a powerful and mature level for this programming paradigm (see, e.g., [23, 8, 18, 6, 12] and their references). These systems can approximate at compile-time a wide range of properties, from directional types to determinacy or termination, always safely, and with a significant degree of precision. Thus, our approach is to take advantage of these advances in program analysis tools within the context of program validation and debugging, rather than using traditional proof-based methods (e.g., [1, 2, 13, 17, 28]), developing new tools and procedures, such as specific concrete [4, 15, 16] or abstract [9, 10] diagnosers and declarative debuggers, or limiting error detection to run-time checking [28].

In this talk we discuss these issues and present a framework for combined static/dynamic validation and debugging using semantic approximations [7, 26, 21] which is meant to be a part of an advanced program development environment comprising a variety of co-existing tools [14]. Program validation and detection of errors is first performed at compile-time by inferring properties of the program via abstract interpretation-based static analysis and comparing this information against (partial) specifications written in terms of assertions. Such assertions are linguistic constructions which allow expressing properties of programs.

Classical examples of assertions are type declarations (e.g., in the context of (C)LP those used by [22, 27, 5]). However, herein we are interested in supporting a more general setting in which assertions can be of a much more general nature, stating additionally other properties, some of which cannot always be determined statically for all programs. These properties may include properties defined by means of user programs and extend beyond the predefined set which may be natively understandable by the available static analyzers. Also, in the proposed framework only a small number of (even zero) assertions may be present in the program, i.e., the assertions are *optional.* In general, we do not wish to limit the programming language or the language of assertions unnecessarily in order to make the validity of the assertions statically decidable (and, consequently, the proposed framework needs to deal throughout with *approximations*). We present a concrete language of assertions which allows writing this kind of (partial) specifications for (C)LP [25].

The assertion language is also used by the compiler to express both the information inferred by the analysis and the results of the comparisons performed against the specifications.[1] These comparisons can result in proving statically (i.e., at compile-time) that the assertions hold (i.e., they are validated) or that they are violated, and thus bugs have been detected. User-provided assertions (or *parts* of assertions) which cannot be statically proved nor disproved are optionally translated into run-time tests. Both the static and the dynamic checking are provably *safe* in the sense that all errors flagged are definite violations of the specifications.

We illustrate the practical usefulness of the framework by demonstrating what is arguably the first and most complete implementation of these ideas: CiaoPP, the Ciao system preprocessor [24, 20]. Ciao is a public-domain, next-generation (constraint) logic programming system, which supports ISO-Prolog, but also, selectively for each module, pure logic programming, functions, constraints, objects, or higher-order. Ciao is specifically designed to a) be highly extendible and b) support modular program analysis, debugging, and optimization. The latter tasks are performed in an integrated fashion by CiaoPP.

CiaoPP, which incorporates analyses developed by several groups in the community, uses abstract interpretation to infer properties of program predicates and literals, including types, modes and other variable instantiation properties, non-failure, determinacy, bounds on computational cost, bounds on sizes of terms in the program, etc. It processes modules separately, performing incremental analysis. CiaoPP can find errors at compile-time (or perform partial verification), by checking how programs call system libraries and also by checking assertions present in the program or in other modules used by the program. This allows detecting errors in user programs even if they contain no assertions. In addition, CiaoPP also performs program transformations and optimizations such as multiple abstract specialization, parallelization (including granularity control), and inclusion of run-time tests for assertions which cannot be checked completely at compile-time.

The implementation of the preprocessor is generic in that it can be easily customized to different CLP systems and dialects and in that it is designed to

[1] Interestingly, the assertions are also quite useful for generating documentation automatically using [19]

allow the integration of additional analyses in a simple way (for example, it has been adapted for use with the CHIP CLP(fd) system).

More info: For more information, full versions of selected papers and technical reports, and/or to download Ciao and other related systems please access `http://www.clip.dia.fi.upm.es/`.

Keywords: Global Analysis, Debugging, Verification, Parallelization, Optimization, Abstract Interpretation.

References

[1] K. R. Apt and E. Marchiori. Reasoning about Prolog programs: from modes through types to assertions. *Formal Aspects of Computing*, 6(6):743–765, 1994.

[2] K. R. Apt and D. Pedreschi. Reasoning about termination of pure PROLOG programs. *Information and Computation*, 1(106):109–157, 1993.

[3] F. Bourdoncle. Abstract debugging of higher-order imperative languages. In *Programming Languages Design and Implementation'93*, pages 46–55, 1993.

[4] J. Boye, W. Drabent, and J. Małuszyński. Declarative diagnosis of constraint programs: an assertion-based approach. In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 123–141, Linköping, Sweden, May 1997. U. of Linköping Press.

[5] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The Ciao Prolog System. Reference Manual. TR CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), August 1997. System and on-line version of the manual available at `http://clip.dia.fi.upm.es/Software/Ciao/`.

[6] F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla. Global Analysis of Standard Prolog Programs. In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

[7] F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla. On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs. In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 155–170, Linköping, Sweden, May 1997. U. of Linköping Press.

[8] B. Le Charlier and P. Van Hentenryck. Experimental Evaluation of a Generic Abstract Interpretation Algorithm for Prolog. *ACM Transactions on Programming Languages and Systems*, 16(1):35–101, 1994.

[9] M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Proving properties of logic programs by abstract diagnosis. In M. Dams, editor, *Analysis and Verification of Multiple-Agent Languages, 5th LOMAPS Workshop*, number 1192 in Lecture Notes in Computer Science, pages 22–50. Springer-Verlag, 1996.

[10] M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract diagnosis. *Journal of Logic Programming*, 39(1–3):43–93, 1999.

[11] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *4th. ACM Symp. on Principles of Programming Languages*, pages 238–252, 1977.

[12] M. García de la Banda, M. Hermenegildo, M. Bruynooghe, V. Dumortier, G. Janssens, and W. Simoens. Global Analysis of Constraint Logic Programs. *ACM Transactions on Programming Languages and Systems*, 18(5):564–615, September 1996.

[13] P. Deransart. Proof methods of declarative properties of definite programs. *Theoretical Computer Science*, 118:99–166, 1993.

[14] P. Deransart, M. Hermenegildo, and J. Maluszynski. *Analysis and Visualization Tools for Constraint Programming.* Number 1870 in LNCS. Springer-Verlag, September 2000.

[15] W. Drabent, S. Nadjm-Tehrani, and J. Małuszyński. The Use of Assertions in Algorithmic Debugging. In *Proceedings of the Intl. Conf. on Fifth Generation Computer Systems*, pages 573–581, 1988.

[16] W. Drabent, S. Nadjm-Tehrani, and J. Maluszynski. Algorithmic debugging with assertions. In H. Abramson and M.H.Rogers, editors, *Meta-programming in Logic Programming*, pages 501–522. MIT Press, 1989.

[17] G. Ferrand. Error diagnosis in logic programming. *J. Logic Programming*, 4:177–198, 1987.

[18] J.P. Gallagher and D.A. de Waal. Fast and precise regular approximations of logic programs. In Pascal Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming*, pages 599–613. MIT Press, 1994.

[19] M. Hermenegildo. A Documentation Generator for (C)LP Systems. In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 1345–1361. Springer-Verlag, July 2000.

[20] M. Hermenegildo, F. Bueno, G. Puebla, and P. López-García. Program Analysis, Debugging and Optimization Using the Ciao System Preprocessor. In *1999 International Conference on Logic Programming*, pages 52–66, Cambridge, MA, November 1999. MIT Press.

[21] M. Hermenegildo, G. Puebla, and F. Bueno. Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging. In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.

[22] P. Hill and J. Lloyd. *The Goedel Programming Language.* MIT Press, Cambridge MA, 1994.

[23] K. Muthukumar and M. Hermenegildo. Compile-time Derivation of Variable Dependency Using Abstract Interpretation. *Journal of Logic Programming*, 13(2/3):315–347, July 1992.

[24] G. Puebla, F. Bueno, and M. Hermenegildo. A Generic Preprocessor for Program Validation and Debugging. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 63–107. Springer-Verlag, September 2000.

[25] G. Puebla, F. Bueno, and M. Hermenegildo. An Assertion Language for Constraint Logic Programs. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

[26] G. Puebla, F. Bueno, and M. Hermenegildo. Combined Static and Dynamic Assertion-Based Debugging of Constraint Logic Programs. In *Logic-based Program Synthesis and Transformation (LOPSTR'99)*, number 1817 in LNCS, pages 273–292. Springer-Verlag, 2000.

[27] Z. Somogyi, F. Henderson, and T. Conway. The execution algorithm of Mercury: an efficient purely declarative logic programming language. *JLP*, 29(1–3), October 1996.

[28] E. Vetillard. *Utilisation de Declarations en Programmation Logique avec Contraintes.* PhD thesis, U. of Aix-Marseilles II, 1994.