Program Transformation for Program Verification

Alberto Pettorossi[1] and Maurizio Proietti[2]

[1] DICII, University of Rome Tor Vergata, Rome, Italy pettorossi@disp.uniroma2.it
[2] IASI-CNR, Rome, Italy maurizio.proietti@iasi.cnr.it

We present a transformational approach to program verification and software model checking that uses three main ingredients: (i) Constraint Logic Programming (CLP), (ii) metaprogramming and program specialization, and (iii) proof by transformation. (i) *Constraints* are used for representing in a compact way (finite or infinite) sets of values or memory states, and *logic* is used for expressing properties of program executions [2, 4, 5]. The least fixpoint semantics and negation allow us to denote both the least models and the greatest models of programs, and thus to reason about the (finite or infinite) behaviour of programs. (ii) *Metaprogramming* is used for getting a verification technique which is parametric with respect to the programming language in use. In particular, we introduce a CLP program I which defines the (meta)interpreter of the programming language in which the program P to be verified is written. Then, in order to gain efficiency, we remove this interpretation layer by *specializing* the interpreter I with respect to the given program P [1, 6, 7]. The property φ that should be proved (or disproved) about program P, is expressed through the CLP clauses that characterize the set of states in which φ holds (or does not hold, respectively). (iii) Having derived a CLP program $\widetilde{P}$ whose semantics represents the behaviour of the given program P and the property φ to be verified, we start a third phase which consists in the *proof by CLP program transformation.* This transformation is performed by using *unfold/fold rules* and also some generalization and goal replacement rules which all preserve the semantics [8]. By the generalization rule [3] one can derive the invariants which hold during program execution and are needed to verify the given property. Rules are applied according to some *strategies* with the objective of deriving from program $\widetilde{P}$ a new CLP program $\widetilde{P1}$ so that a selected atom, say *prop*, either belongs to $\widetilde{P1}$ (in which case φ holds) or no clause for *prop* belongs to $\widetilde{P1}$ (in which case φ does not hold). We have designed a few (semi)automatic strategies which make the transformation process to terminate. Obviously, they are all incomplete due to undecidability limitations, but they work well on many non-trivial examples.

References

[1] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying Programs via Iterated Specialization. In *Proc. ACM PEPM'13*, 43–52, New York, USA, 2013.

[2] G. Delzanno and A. Podelski. Model checking in CLP. In R. Cleaveland, ed., *TACAS'99*, LNCS 1579, 223–239. Springer-Verlag, 1999.

[3] F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Generalization strategies for the verification of infinite state systems. *Theory and Practice of Logic Programming.* 13(2):175–199, 2013.

[4] L. Fribourg. Constraint logic programming applied to model checking. In A. Bossi, ed., *Proc. LOPSTR'99*, LNCS 1817, 31–42. Springer-Verlag, 2000.

[5] S. Grebenshchikov, A. Gupta, N. P. Lopes, C. Popeea, and A. Rybalchenko. HSF(C): A Software Verifier based on Horn Clauses. In C. Flanagan and B. König, eds., *Proc. TACAS'12*, LNCS 7214, 549–551. Springer, 2012.

[6] M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialization. *Proc. LOPSTR'99*, LNCS 1817, 63–82. Springer, 2000.

[7] J. C. Peralta, J. P. Gallagher, and H. Saglam. Analysis of Imperative Programs through Analysis of Constraint Logic Programs. In G. Levi, ed., *Proc. SAS'98*, LNCS 1503, 246–261. Springer, 1998.

[8] A. Pettorossi and M. Proietti. Perfect model checking via unfold/fold transformations. In J. W. Lloyd, ed., *Proc. CL 2000*, Lecture Notes in Artificial Intelligence 1861, 613–628. Springer, 2000.