Software model checking using game semantics and CSP

Aleksandar Dimovski, Ranko Lazic
Departmant of Computer Science,
University of Warwick,
Coventry CV4 7AL,
United Kingdom
e-mail: {aleks, lazic}@dcs.warwick.ac.uk

Extended Abstract

Model checking is a system verification technique based on semantics: the verifier checks whether the semantics of a given system is a model for some formula [5]. It has gained industrial acceptance because, in contrast to the approaches of simulation, testing and theorem proving, model checking is automatic, and it also reports counter-examples.

The success of model checking has been mainly for hardware and protocols. More recently, applying model checking to software verification has become an active area of reserach [4, 6]. However, software systems do not seem to be as naturally suited to model checking as hardware systems, because they are often highly structured, and contain objects, higher-order computation, complex control mechanisams, pointers, concurency, and other feutures. The source of most problems specific to software model checking is the so called semantic gap, i.e. the absence of mechanically checkable models of software. The greiter difficulties are raised by the presence of non-local objects in the program to be verified, for example library function calls.

On the other hand, one of the main breakthroughs in theoretical computer science in the past decade has been the development of game semantics. It has been used to construct accurate i.e. fully abstract and fully complete models for a variety of programming languages and logical systems [1, 2, 12]. Still, although games based models are fully abstract, which means that in principle they correctly validate all program equivalences and inequaivalences, they at the same time seem complicated, they use complex and unwieldy notations so their application in practice is a very difficult task.

But in last few years, game semantics, restricted to yet non-trivial aspects of programming languages (without higher-order functions, recursion and infinite data types), is starting to be developed in a new much easier algorithmic way, with possibilities for its application in computer assisted verification and program analysis. In this direction, especialy important was work by Ghica and McCusker [8, 10, 11] who simplified representation of game semantics of a fragment of Idealized Algol using regular exressions, and have implemented a software model checking tool based on this representation of game semantics. In relation to the extensive current activity in model checking and computer assisted verification, this game semantics approach is distinctive, being founded on a highly-structured compositional semantic model, which means we can directly apply these methods to open program phrases i.e. terms-in-contex with free variables in a high-level language with functions, local variables and data types.

Bearing in mind the suitability of game semantics for software model checking, in this work we are giving a new interpretation of it, namely CSP interpretation. This should be the first step in building a new software model checking tool based on game semantics and CSP, a tool that in addition to all good game semantics features, will also benefit of very promising features of CSP language itself and model checking tool FDR. CSP is

an expressive language and program terms can be easily represented as CSP processes. Moreover this representation will be very elegant and clear, since composition of game strategies can be implemented as parallel composition plus hiding in CSP. That is shown here as well as that this interpretation is fully abstract, and that terms equivalence and verification of a wide range of program propreties are decidable with trace refinment, which is one of the main functions of FDR. Also, FDR contains a number of procedures for compositional state space reduction [7], which seem very promising for the software model checking tool.

References

[1] S.Abramsky and G.McCusker. Linearity, sharing and state: a full abstract game semantics for Idealized Algol with active expressions (extended abstract). In Proceedings of 1996 Workshop on Linear Logic, volume 3 of Electronic notes in Theoretical Computer Science. Elsevier, 1996.

[2] S.Abramsky and G.McCusker. Call-by-value games. In CSL: 11th Workshop on Computer Science Logic, LNCS 1414, pp 1-17. 1998.

[3] S.Abramsky. Algorithmic game semantics: A tutorial introduction. Lecture notes, Marktoberdorf International Summer School 2001.

[4] T.Ball and S.K.Rajamani. The SLAM Project: Debugging System Software via Static Analysis. In Proceedings of POPL, Jan, 2002.

[5] E.M.Clarke, O.Grumberg, and D.A.Peled. Model Checking. MIT Press. Jan, 2000.

[6] M.Dwyer, J.Hatcliff, R.Joehanes, S.Laubach, C.Pasareanu, H.Zheng, W.Visser. Tool-Supported Program Abstraction for Finite-State Verification. In Proceedings of 23rd ICSE, May 2001.

[7] Formal Systems (Europe) Ltd. Failures-Divergence Refinement: FDR2 Manual. 2000.

[8] D.Ghica and G.McCusker. Reasoning About Idealized Algol Using Regular Languages. In Proceedings of ICALP 2000, vol. 1853 of LNCS, Springer-Verlag, pp. 103-116, Geneva, Switzerland, 2000.

[9] D.Ghica and G.McCusker. The Regular-Language Semantics of First-order Idealized Algol. Theoretical Computer Science, 2001.

[10] D.Ghica. A Games-Based Foundation for Compositional Software Model Checking. PhD Thesis, Queen's University School of Computing, Kingston, Ontario, Canada, November 2002.

[11] D.Ghica. Game-based Software Model Checking: Case Studies and Methodological Considerations. Oxford University Computing Lab, Technical Report PRG-RR-03-11, May 2003.

[12] J.M.E.Hyland and C.H.Ong. On full abstraction for PCF: I, II and III. Information and Computation 163, 2000.

[13] A.W.Roscoe. The Theory and Practice of Concurrency. Prentice Hall, 1998.