

Consistent Composition and Testing of Components using Abstract State Machines

Elvinia Riccobene

Angelo Gargantini

Marianna Nicolosi Asmundo

Dipartimento di Matematica e Informatica, Università di Catania
{riccobene,gargantini,nicolosi}@dmi.unict.it

1 Introduction

Modelling complex systems in a coherent and rigorous way is a basic goal in system design. In particular, the *integration of specifications* could be a problematic operation having potentially severe side effects: when components of a system separately developed are combined together, they may intersect and overlap giving rise to inconsistencies and conflicts which have to be resolved in order to obtain a consistent system. Moreover, the substitution of old components with new ones or the introduction of new features in the system may affect the overall system behavior in an unpredictable way. For these reasons, the introduction of composition operations producing consistent systems is required.

Along the process of system development, testing is also of extreme importance to produce high-quality products. Testing allows to uncover development and coding errors, to assess system reliability and dependability, and to convince customers that the performance is acceptable. Although, software testing is extremely costly and time-consuming, *specification based testing* [11] offers an opportunity to significantly reduce the testing costs. In specification based testing, a specification can be used as "test oracle" [14], i.e. as authoritative font of the expected system behavior, and as way to assess correctness of implementations. Moreover, "test adequacy criteria" can be derived from a specification [15]. They determine if a test suite is adequate to test a system, whether enough testing has been performed or further tests are needed. A specification can also provide "selection criteria" of adequate test suites. Normally a selection criterion introduces some algorithms or techniques to actually generate test sequences from formal specifications.

We address both problems of components integration and specification-based testing of components using (sequential) Abstract State Machines (ASMs). The choice of this formal method as a platform to construct rigorous integration operations for partial specifications and to define methods for generating test suites from high-level specifications, is intentional and it is due to the fact that, besides having been successfully used in practice for design and analysis of complex hardware/software systems [3], ASMs have evident theoretical foundations, clear and precise semantics [8, 9, 1].

2 Consistent Integration of Sequential ASMs

When using components to describe a system behavior, components may represent different features of the same device, or different subsystems of a compound system. An ASM may be employed to describe the behavior of a system component, of a process to be integrated with other processes, etc. Having a rigorous operation for the integration of ASMs guarantees that we are able to construct the behavioral specification of a device from its features or to build a system from its components without flatten them and preserving the interconnections between them. In [13] we introduce two kinds of composition operations over sequential ASMs. The first one, called *feature composition* (\oplus), takes as input sequential ASMs representing features of a device and returns a sequential ASM being the required device. The second one, called *component composition* (\otimes), takes as input sequential ASMs representing components of a system. In case the components are synchronous, i.e. they have the same clock, the result of the operation is a multi-agent ASM with synchronous agents, otherwise, the result is a multi-agent ASM with asynchronous agents.

Operations \oplus and \otimes are both equipped with conditions of compatibility for the components to be integrated, and of tests to check the consistency of the compound system with respect to updates under the assumption that the components are already consistent with respect to updates. Such tests allow us to find and recognize possible update inconsistencies coming out from the integration process. We also shown how the composition operations can be applied to analyze and handle behavioral inconsistencies and to prove system properties.

The applicative aspects of the theoretical issues introduced in this paper are shown by means of three examples: the behavioral description of a telephone system presented in [5], the ASM specification of the Production Cell case study developed in [2], and the ASM solution of the railroad crossing problem given in [10].

3 ASM-based Testing and Automatic Tests Generation for ASMs

In [6] we show how to use an ASM as test oracle, we introduce and motivate several test criteria for ASMs, and for each coverage criterion, we provide a set of formulas, called *test predicates*, which determine the set of states that realize the coverage. These coverage criteria can be used as adequacy criteria to measure the degree of coverage achieved by the test set. We also explain how to use such test criteria to generate test suites exploiting the use of a model checker. In [7] we investigate further the use of model checkers for tests generation. We introduce a novel algorithm to translate (a particular class of) ASMs in PROMELA, the language of Spin [12] – this translation also allows using Spin to prove properties of ASM models–. Exploiting the counter example generation feature of a model checker (Spin, SMV), we present a method to automatically generate from ASM specifications test sequences which accomplish a desired coverage. Benefits and limitations in using model checkers for test generation are also discussed.

We have developed a prototype tool that implements the proposed method. Experimental results in evaluating our method are reported in [7]. For the Safety Injection System (SIS) [4], test sequences are generated, and the coverage of the SIS Java code provided by these tests is measured through a code coverage analyzer. Moreover, these tests generated from the ASM specification are compared with test sequences randomly generated.

References

- [1] A.Blass, Y.Gurevich. *Abstract State Machines Capture Parallel Algorithms*. ACM Trans. on Comp. Logic (to appear).
- [2] E.Boerger, L.Mearelli, L.*Integrating ASMs into the software Development Life Cycle*, JUCS vol. 3, n. 5, 1997.
- [3] E.Boerger.*The Origins and Development of the ASM Method for high level System Design and Analysis*, JUCS vol. 8, n. 1, 2002
- [4] P.Courtois, D.Parnas. Documentation for safety critical software. *Proc. ICSE '93*, 1993.
- [5] S.Easterbrook, B.Nuseibeh. *Using ViewPoints for Inconsistency Management*. Software Engineering Journal, 1996.
- [6] A.Gargantini, E.Riccobene. Asm-based testing: Coverage criteria and automatic test sequence generation. *Journal of Universal Computer Science*, 7(11), 2001.
- [7] A.Gargantini, E.Riccobene, S.Rinzivillo. Using Spin to Generate Tests from ASM Specifications. *ASM03*, LNCS 2589, 2003.
- [8] Y.Gurevich. *Evolving Algebras 1993: Lipari Guide*, in E. Boerger, editor, Specification and Validation Methods, pp. 9-36. Oxford University Press, 1995.
- [9] Y.Gurevich. *Sequential Abstract State Machines capture sequential algorithms*. ACM Trans. on Comp. Logic 1(1), 2000.
- [10] Y.Gurevich, J.Huggins. *The Railroad Crossing Problem: An Experiment with instantaneous Actions and Immediate Reactions*, Proc. CSL'95, LNCS 1092.
- [11] R.Hierons, J.Derrick. Special issue on specification-based testing. *Software testing, verification & reliability*, 10(4), 2000.
- [12] G.J.Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
- [13] M.Nicolosi, E.Riccobene. Consistent Integration for Sequential Abstract State Machines. *ASM03*, LNCS 2589, 2003.
- [14] D. J. Richardson, S. L. Aha, and T. O. O'Malley. Specification-based test oracles for reactive systems. *Proc. 14th International Conference on Software Engineering*, LNCS Springer, 1992.
- [15] Hong Zhu, Patrick A. V. Hall, and John H. R. May. Software unit text coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, December 1997.