Model Testing Asynchronously Communicating Objects using Modulo AC Rewriting

Olaf Owe, Martin Steffen, and Arild B. Torjusen[⋆]

University of Oslo, Norway

1 Motivation

Systematic testing [13] is indispensable to assure reliability and quality of software and systems. Hosts of different testing approaches and frameworks have been proposed and put to (good) use over the years. Traditionally testing has been perceived as a non-formal activity by many theoreticians, since it is based on *executing* code, rather than on formal reasoning. More recently there is an increasing tendency to see testing and formal methods as complementary [9]. Formal methods has proven valuable to render testing practice a more formal, systematic discipline (cf. e.g. [7,1]) and formal approaches to testing have gained momentum in recent years, as for instance witnessed by the trend towards model-based or specification-based testing [6,2], where an explicit formal model or specification of the behavior of the system is central. The formal model is used to generate test cases and as an *oracle* to evaluate output resulting from executing the test cases. In previous work [8] we presented a formal approach for black-box specification-based testing of asynchronously communicating components in open environments together with an implementation of a testing framework. In this paper we show how to extend the approach to *verification* of components and present experimental results that show the usefulness of our approach.

Creol Creol [5,11] is a high-level, object-oriented modelling language for distributed systems. Object-orientation is a natural choice, as object modelling is the fundamental approach to open distributed systems as recommended by RM-ODP [10]. In contrast with object-oriented languages based on multi-threading, such as *Java* or $C^\sharp$, Creol features *active objects*. The unit of activity is the object; every process belongs to an object, and activity does not cross object borders. Communication is based on exchanging messages *asynchronously*. This communication model is advantageous as it decouples caller and callee thus avoiding unnecessary waiting for method returns. On the downside, asynchronicity makes verifying and testing programs more challenging, communication delays due to the network or to queuing may lead to message overtaking and the resulting non-determinism leads to a state space explosion.

⋆ Corresponding author, and the one presenting the paper.

Specification-based testing Abstracting from internal executions, the black-box behavior of Creol components is given by interactions at their *interface*. We use a concise language over communication labels to specify components (cf. [8]). In the specification language, a clean separation of concerns between interaction under the control of the component or coming from the environment is central. This leads to an assumption-commitment style specification of a component's behavior by defining the valid observable output behavior, assuming a certain scheduling of the input. For input interactions, we ensure that the specified assumptions on the environment are fulfilled by *scheduling* the incoming calls in the order specified, while for output events, which are controlled by the component, we *test* that the events occur as specified.

The operational semantics of Creol is formalized in rewriting logic [12] and executable on the Maude rewriting engine [3]. We have developed an executable framework for testing Creol components which includes: an executable behavioral interface specification language; a method for composing Creol components and specifications for the purpose of testing; and a rewriting logic implementation of the method. Scheduling and testing of a component are done by synchronizing the communication between specification terms and objects. As a result, the scheduling is enforced in the execution of the object and the actual outgoing interactions from the object are tested against the output events in the specification. This gives a framework for testing whether an implementation of a component conforms to the interface specification. Incorrect or nonconforming behavior of the component under a given scheduling is reported as an error by the testing framework.

Due to message delays and overtaking, the order in which outgoing messages from a component are observed by an external observer does not necessarily reflect the order in which they were actually sent. Testing is based on behavior observable at the interface, and the order of outgoing communication should therefore not affect the test results. The operational semantics of the specification language takes this into account by treating certain reorderings of output events as observationally equivalent. This leads to a large increase in the reachable state space for the test cases. Reordering of output events can be expressed by defining sequences of output events as *associative* and *commutative* (AC). We argue that our testing framework is especially well suited to implement this since, using the rewriting logic system Maude, associativity and commutativity can be declared using *equational attributes* [4] which allows efficient evaluation of such specifications.

2 Results

This paper extends [8] which introduced and gave the formal basis for the approach we explore further here: We test whether an object/component conforms to its behavioral interface description, taking into account especially the asynchronous nature of the communication model. The main contributions are:

Verification We provide an implementation of the approach in the rewriter Maude and use Maude's *search* functionality for state exploration (for rewriting modulo AC) for verification of components and investigate how the support for AC reasoning built in into Maude contributes to state space reduction in verification of asynchronously communicating components.

Experimental results We present *experimental results* from using the Maude rewriting tool which give empirical evidence of the benefits of our method. We compare, in two series of experiments, the influence on the state space of using Maude's built in AC support against explicit representation of all possible reorderings of output events (with the same semantics). Using AC rewriting may considerably reduce the resource consumption when testing asynchronously communicating objects. AC rewriting significantly pays off in terms of time and the number of rewrites.

References

1. G. Bernot. Testing against formal specification: A theoretical view. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT '91, Vol. 1*, volume 493 of *LNCS*, pages 99–119. Springer, 1991.
2. M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*. Springer, 2005.
3. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In R. Nieuwenhuis, editor, *RTA 2003*, volume 2706 of *LNCS*, pages 76–87. Springer, June 2003.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *The Maude Manual (version 2.1.1)*. SRI International, Menlo Park, Apr. 2005.
5. The Creol language. `http://heim.ifi.uio.no/creol`, 2007.
6. S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 1999 International Conference on Software Engineering, 1999*, pages 285–294, 1999.
7. M.-C. Gaudel. Testing can be formal, too. In P. D. Mosses, M. Nielsen, and M. I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *LNCS*, pages 82–96. Springer, 1995.
8. I. Grabe, M. Steffen, and A. B. Torjusen. Executable interface specifications for testing asynchronous Creol components. In *Proceedings of the 3rd International Conference on Fundamentals of Software Engineering (FSEN'09), 15 - 17 April, Kish Island, Persian Gulf*, LNCS. Springer, 2009. To appear.
9. R. M. Hierons, J. P. Bowen, and M. Harman, editors. *Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers*, volume 4949 of *LNCS*. Springer, 2008.
10. International Telecommunication Union. Open Distributed Processing - Reference Model parts 1–4. Technical report, ISO/IEC, Geneva, July 1995.
11. E. B. Johnsen, O. Owe, and I. C. Yu. Creol: A type-safe object-oriented model for distributed concurrent systems. *Theoretical Computer Science*, 365(1–2):23–66, Nov. 2006.
12. J. Meseguer. Conditional rewriting as a unified model of concurrency. *Theoretical Computer Science*, pages 73–155, 1992.
13. R. Patton. *Software Testing*. SAMS, second edition, July 2005.