New developments around the μCRL tool set

Stefan Blom[a] Jan Friso Groote[b,a] Izak van Langevelde[a]
Bert Lisser[a] Jaco van de Pol[a,1]

[a] *Centrum voor Wiskunde en Informatica, Dept. of Software Engineering Amsterdam, The Netherlands*

[b] *Eindhoven University of Technology, Computer Science Group Eindhoven, The Netherlands*

Abstract

Some recent developments in the μCRL tool set are presented. New analysis techniques are a symbolic model checker, and a visualizer for huge state spaces. Also various transformations are presented. At symbolic level, theorem proving, data flow analysis, and confluence checking are used to obtain considerable state space reductions. At the concrete level, distributed implementations of state space generation and minimization are recent. We mention the successful application of the tools to the verification of large data-intensive distributed systems.

1 Introduction

The μCRL specification language [12] is used to specify data-intensive protocols and distributed systems. System behavior is described in process algebra, using non-determinism, sequential and parallel composition, synchronous communication, hiding and recursion. Data structures are modeled using abstract data types. The data is linked to the processes by means of parameterized actions and recursion, conditionals, and a choice operator.

The μCRL tool set[2] [3] can be used to generate the state space as a labeled transition system (LTS) of a μCRL system specification. In order to combat the state space explosion at the symbolic level, an intermediate format of *linear process* is introduced (Figure 1). This is a symbolic and concise description of the state space. Emphasizing new developments, we will review some transformation and analysis capabilities on linear processes (Section 2) and LTSs (Section 3). Some applications are mentioned in Section 4.

[1] Email: `Jaco.van.de.Pol@cwi.nl`
[2] URL: `http://www.cwi.nl/~mcrl`

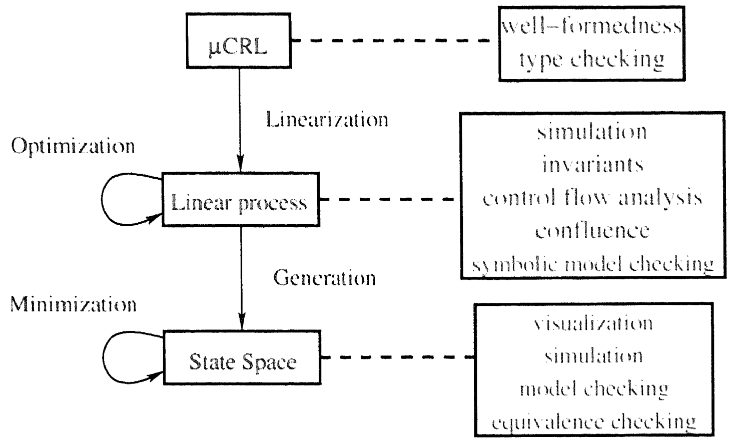

Fig. 1. Models, transformations, and analysis techniques in the μCRL tool set

2 Symbolic Manipulation and Analysis

The symbolic analysis and transformations are applied to linear processes. Linearization can be applied to any μCRL specification, although only a subset has been implemented. A linear process is a concise, symbolic description of a possibly infinite transition system. It consists of a vector of state parameters, and a number of summands. Each summand specifies a set of labeled transitions by means of a condition, an atomic action and a state update. A simulator is available that allows the user to step through the states, by choosing one of the enabled actions.

Data flow analysis. Various simplifications of linear processes have been implemented [9], e.g. the identification and substitution of constant parameters, and the elimination of unused parameters. These have proved to be widely applicable, and combining them results in an effective state space reduction. Recently, a tool has been developed to identify typical control flow parameters in a linear process, and to construct their control flow graph. These graphs can be visualized and animated via the simulator. The control flow is also the basis of a live-variable analysis. The state space is reduced by resetting temporarily unused parameters to a dummy value. Future developments will use control flows for automated invariant generation and confluence marking.

Invariants and Confluence. Owing to the linear process format, propositions like "ϕ is an invariant" can be expressed in terms of universally quantified data formulas. An automated theorem prover, based on extending BDDs with equations and interpreted functions, has been developed to solve such formulas. Invariants are useful for verifying desired properties, but they can also be used in combination with transformations, for instance to identify more constant state parameters, or to prove that certain summands are never enabled.

The theorem prover is also used to detect confluent summands [5], possibly using separately proven invariants. Confluent summands can be given priority, giving rise to enormous state space reductions in many cases, similar to partial-order reduction. This prioritization can be approximated at symbolic level, or applied during state space generation, using an on-the-fly algorithm for strongly connected components, to avoid problems with τ-loops.

Symbolic Model Checking. Recently, a prototype symbolic model checker has been built [13]. It can be used to check first-order modal μ-calculus properties on infinite state spaces, represented by linear processes. The model checking problem is reduced to boolean equation systems with parameterized recursion operators [10]. These are solved using the aforementioned extension of BDDs. In this approach, it is for instance possible to express and prove that an infinite-state merger process transforms increasing input streams to an increasing output stream.

3 Explicit State Space Minimization and Analysis

Within the μCRL toolset a library for "on-the-fly" state space exploration has been implemented. Using this library, we have implemented both single-threaded and distributed state space generation tools. This exploration library has also been used to implement the Open/Cæsar interface. Thus, we can use all Open/Cæsar based tools from the CADP toolset [3] [7]. The confluence-based partial-order reduction will soon be moved from the single-threaded state space generation tool to this exploration library, allowing both the Open/Cæsar tools and distributed state space generation tools to benefit from partial order reduction.

State Space Minimization. Model checking requires a significant amount of memory. As distributed generation can easily generate state spaces which do not fit on a single machine, it is essential to first reduce large LTSs modulo an equivalence which preserves the desired properties. To allow the reduction of very big state spaces, distributed strong and branching bisimulation reduction tools have been written for the μCRL tool set [4]. As usual, the reduction tools are based on partition refinement. The refinement strategy is quite unusual though: instead of refining a single block of the current partition in each iteration, *all* blocks are refined in parallel. In theory this leads to bad performance, but in practice the single-threaded versions of the tools are quite competitive with the `bcg_min` tool in the CADP toolset. The parallelism is of course also a major advantage for distributed implementations. Another unusual feature is that the distributed branching bisimulation reduction works on LTSs which contain τ-cycles. The framework used by the tools also allows

[3] URL: `http://www.inrialpes.fr/vasy/cadp/`

reduction modulo safety equivalence. Another direction for future work is the implementation of a distributed model checking tool in the style of XTL [7].

State Space Analysis. The LTS is generated in a format recognized by the CADP tool set. This makes the following analysis techniques available at the state space level: equivalence checking, explicit state model checking and visualization. Usually, visualization of state spaces is limited to small systems ($<$ 1K nodes). However, by using modern visualization techniques, such as clustering, very large graphs of $>$1M nodes can be visualized as cone trees [8]. The resulting drawings can be analyzed for symmetries, and individual parts can be inspected by zooming in. Also, various properties of the state vector of a distributed system can be translated to highlighted regions in the visualization.

4 Applications

Since its conception, the development of the μCRL tool set has been inspired and stimulated by its application in industrial case studies. Conversely, these industrial applications have benefitted from the analytical strength offered by the tools, which more than once revealed fatal bugs in the systems under scrutiny.

An optimization of *Transaction Capabilities Procedures* [2], lent itself well for verification through μCRL. The dimensions of the state spaces generated were relatively small, involving no more than a few thousand transitions. The success of this case study resulted in several more case studies in the telecommunication industry [1].

A *distributed lift system* [11], presented more of a challenge for the tool set, in that it was only possible to analyse systems of up to 5 lifts (or 6 on a cluster of 8 machines). Despite of this restriction, the analysis through μCRL revealed bugs that lead to possibly dangerous situations.

The *Splice coordination architecture* [6] was a killer for the tool set. The architecture involves a number of applications which have access to a distributed data space, but the systems for which the state space could be generated were limited to those consisting of 2 relatively simple applications; anything more complex hit the limit of several millions of transitions. Recent experiments show that confluence reduction is very effective on such architectures.

A *cache coherence protocol for Java* [14] crashed into the same limits. However, this verification greatly benefits from the young development of distributed tools for μCRL which opened new horizons by generating explicit state spaces of more than 800 million transitions.

The control system of the *Dutch railroad trajectory* Woerden-Harmelen is waiting to be verified. At the moment, its complexity grossly surpasses the limits of the μCRL tool set. As these limits are pushed further and further, it is realistic to expect that one day this case study can be successfully tackled.

References

[1] T. Arts, C. Benac Earle, and J. Derrick. Verifying Erlang code: A resource locker case-study. In L.-H. Eriksson and P.A. Linsday, editors, *Proc. of FME'02*, LNCS 2391, pages 184–203. Springer, 2002.

[2] T. Arts and I.A. van Langevelde. Correct performance of transaction capabilities. In *Proc. of ICACSD'01*, pages 35–42. IEEE, 2001.

[3] S.C.C. Blom, W.J. Fokkink, J.F. Groote, I.A. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In G. Berry etal, editor, *Proc. CAV'01*, LNCS 2102, pages 250–254. Springer, 2001.

[4] S.C.C. Blom and S.M. Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. In L. Brim and O. Grumberg, editors, *Electronic Notes in Theoretical Computer Science*, volume 68. Elsevier, 2002.

[5] S.C.C. Blom and J.C. van de Pol. State space reduction by proving confluence. In E. Brinksma and K.G. Larsen, editors, *Proceedings of CAV'02*, volume LNCS 2404, pages 596–609. Springer, 2002.

[6] P.F.G. Dechering and I.A. van Langevelde. The verification of coordination. In A. Porto and Roman G.-C., editors, *Proc. of COORDINATION'00*, LNCS 1906, pages 335–340. Springer, 2000.

[7] H. Garavel, F. Lang, and R. Mateescu. An overview of CADP 2001. *European Assoc. for Software Science and Technology (EASST) Newsletter*, 4:13–24, 2002.

[8] J.F. Groote and F.J.J. van Ham. Large state space visualization. In *Proceedings of TACAS'03*, volume LNCS 2619, pages 585–590. Springer, 2003.

[9] J.F. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. In M. Leuschel and U. Ultes-Nitsche, editors, *Proc. of VCL'02*, Technical Report DSSE-TR-2002-5, University of Southampton, 2002.

[10] J.F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In A.M. Haeberer, editor, *Proc. of AMAST'98*, LNCS 1548, pages 74–90. Springer, 1998.

[11] J.F. Groote, J. Pang, and A. Wouters. Analysis of a distributed system for lifting trucks. *J. of Logic and Algebraic Programming*, 55(1–2):21–56, 2003.

[12] J.F. Groote and M.A. Reniers. Algebraic process verification. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 17. Elsevier, 2001.

[13] J.F. Groote and T.A.C. Willemse. A checker for modal formulas for processes with data. Technical report, Eindhoven Univ. of Technology, Dept. of CS, 2002.

[14] J. Pang, W.J. Fokkink, R. Hofman, and R. Veldema. Model checking a cache coherence protocol for a Java DSM implementation. In *Proc. of IPDPS, FMPPTA*. IEEE, 2003.