μCRL: A Toolset for Analysing Algebraic Specifications

Stefan Blom[1], Wan Fokkink[1], Jan Friso Groote[2], Izak van Langevelde[1], Bert Lisser[1], and Jaco van de Pol[1]

[1] CWI, Department of Software Engineering
PO Box 94079, 1090 GB Amsterdam, The Netherlands
{sccblom,wan,izak,bertl,vdpol}@cwi.nl
[2] Eindhoven University of Technology, Department of Computing Science
PO Box 513, 5600 MB Eindhoven, The Netherlands
jfg@win.tue.nl

1 Introduction

μCRL [13] is a language for specifying and verifying distributed systems in an algebraic fashion. It targets the specification of system behaviour in a process-algebraic style and of data elements in the form of abstract data types. The μCRL toolset [21] (see `http://www.cwi.nl/~mcrl`) supports the analysis and manipulation of μCRL specifications. A μCRL specification can be automatically transformed into a *linear process operator* (LPO). All other tools in the μCRL toolset use LPOs as their starting point. The simulator allows the interactive simulation of an LPO. There are a number of tools that allow optimisations on the level of LPOs. The instantiator generates a labelled transition system (LTS) from an LPO (under the condition that it is finite-state), and the resulting LTS can be visualised, analysed and minimised.

An overview of the μCRL toolset is presented in Figure 1. This picture is divided into three layers: μCRL specifications, LPOs and LTSs. The rectangular boxes denote different ways to represent instances of the corresponding layer (for example, LPOs can be represented in a binary or a textual form). A solid arrow denotes a transformation from one instance to another that is supported by the μCRL toolset; keywords are provided to these arrows to give some information on which kinds of transformations are involved. Finally, the oval boxes represent several ways to analyse systems, and dashed arrows show how the different representations of LPOs and LTSs can be analysed. The box named BCG and its three outgoing dashed arrows actually belong to the CADP toolset (see Section 4). The next three sections are devoted to explaining Figure 1 in more detail.

The μCRL toolset was successfully used to analyse a wide range of protocols and distributed algorithms. Recently it was used to support the optimised redesign of the Transactions Capabilities Procedures in the SS No. 7 protocol stack for telephone exchanges [1,2], to detect a number of mistakes in a real-life protocol over the CAN bus for lifting trucks [10], to analyse a leader election protocol from the Home Audio/Video interoperability (HAVi) architecture [20], and to perform scenario-based verifications of the coordination language SPLICE [6].

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 250–254, 2001.

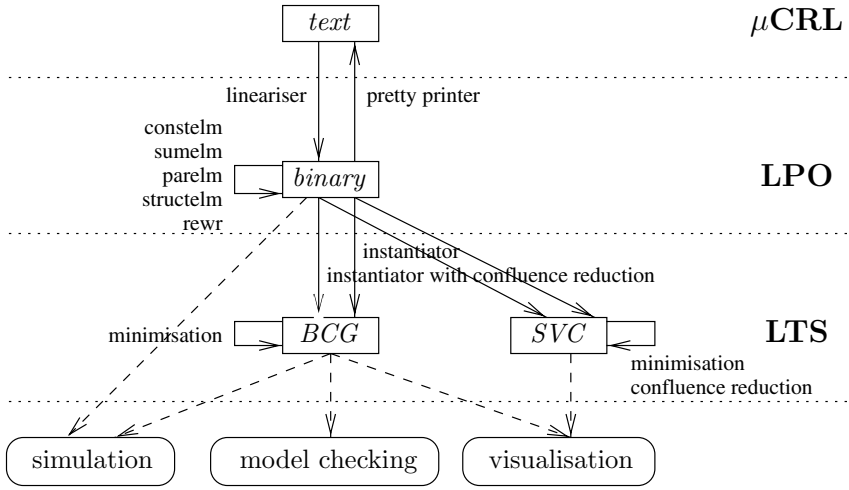

Fig. 1. The Main Components of the μCRL Toolset.

2 μCRL Specifications

The μCRL language is based on the process algebra ACP. It allows one to specify system behaviour in an algebraic style using atomic actions, alternative and sequential composition, parallelism and communication, encapsulation, hiding, renaming and recursive declarations. Furthermore, μCRL supports equationally specified abstract data types. In order to intertwine processes and data, atomic actions and recursion variables carry data parameters. Moreover, an if-then-else construct enables that data elements influence the course of a process, and alternative quantification chooses from a possibly infinite data domain.

3 Linear Process Operators

When investigating systems specified in μCRL, our current standard approach is to transform the μCRL specification under scrutiny to a relatively simple format without parallelism or communication, called an LPO. In essence this is a vector of data parameters together with a list of condition, action and effect triples, describing when an action may happen and what is its effect on the vector of data parameters. It is stored in a binary format or as a plain text file. From an LPO one can generate an LTS, in which the states are parameter vectors and the edges are labelled with parametrised actions.

In [14] it is described how a large class of μCRL processes can be transformed automatically to a bisimilar LPO. The resulting LPO and its data structures are stored as ATerms. The ATerm library [5] stores terms in a very compact way by minimal memory requirements, employing maximal sharing, and using a tailor-

made garbage collector. Moreover, the ATerm library uses a file format that is even more compact than the memory format.

The μCRL toolset comprises five tools (constelm, sumelm, parelm, structelm and rewr) that target the automated simplification of LPOs while preserving bisimilarity [8]. These tools do not require the generation of the LTS belonging to an LPO, thus circumventing the ominous state explosion problem. The simplification tools are remarkably successful at simplifying the LPOs belonging to a number of existing protocols. In some cases these simplifications lead to a substantial reduction of the size of the corresponding LTS.

Elimination of constant parameters. A parameter of an LPO can be replaced by a constant value, if it can be statically determined that this parameter remains constant throughout any run of the process.

Elimination of sum variables. The choice of a sum ranging over some data type may be restricted by a side condition to a single concrete value. In that case the sum variable can be replaced by this single value.

Elimination of inert parameters. A parameter of an LPO that has no (direct or indirect) influence on the parameters of actions or on conditions does not influence the LPO's behaviour and can be removed. Whereas the two reduction techniques mentioned above only simplify the description of an LPO, elimination of inert parameters may lead to substantial reduction of the LTS underlying an LPO. If the inert parameter ranges over an infinite domain, the number of states can even reduce from infinite to finite by this operation. This typically happens after hiding part of the system's behaviour.

Elimination of data structures. Sometimes, the operations above cannot be applied to single parameters, but they can be applied to parts of the data structures that these variables range over. For this to take place, such data structures must be partitioned into their constituents.

Rewriting data terms. The data terms occurring in an LPO can be rewritten using the equations of the data types. If a condition is rewritten to false, then the corresponding condition, action and effect triple in the LPO is removed.

Confluence is widely recognised as an important feature of the behaviour of distributed communicating systems. Roughly, a τ-transition from a state in an LTS, representing an internal computation that is externally invisible, is confluent if it commutes with any other transition starting in this same state. In [18] it was shown that confluence can be used in process verification. In [15] several notions of confluence were studied on their practical applicability, and it was shown that on the level of LPOs confluence can be expressed by means of logical formulas. In [4] it is shown that the presence of confluence within an LPO can be exploited at a low cost at the level of the instantiator, i.e., during the generation of the associated LTS. A prototype of this generation algorithm was implemented, and experience learns that this exploitation of confluence within an LPO may lead to the generation of an LTS that is several orders of magnitudes smaller compared to the standard instantiator. The detection of confluence in an LPO is performed using the automated reasoning techniques that are surveyed in Section 5.

4 Labelled Transition Systems

The SVC format [17] offers an extremely compact file format for storing LTSs. This format is open in its specification and implementation, and allows states to be labelled by ATerms. A prototype visualisation tool has been developed for the SVC format, dubbed Drishti. A reduction algorithm based on confluence and minimisation algorithms modulo equivalences such as bisimulation and branching bisimulation have been implemented, collapsing equivalent states.

Alternatively, LTSs belonging to μCRL specifications can be visualised and analysed using the Cæsar/Aldébaran Development Package (CADP) [7]. This toolset originally targets the analysis of LOTOS specifications. Cæsar generates the LTS belonging to a LOTOS specification, and supports simulation. Aldébaran performs equivalence checking and minimisation of LTSs modulo a range of process equivalences. XTL offers facilities for model checking formulas in temporal logics. The CADP toolset comprises the BCG format, which supports compact storage of LTSs. SVC files can be translated to BCG format and vice versa, given a CADP license (as the BCG format is not open source).

In [11] a reduction algorithm for LTSs is presented, based on priorisation of confluent τ-transitions. First the maximal class of confluent τ-transitions is determined, and next outgoing confluent τ-transitions from a state are given priority over all other outgoing transitions from this same state. For LTSs that do not contain an infinite sequence of τ-transitions, this reduction preserves branching bisimulation. An implementation of this algorithm is included in the μCRL toolset. In some cases it reduces the size of an LTS by an exponential factor. Furthermore, the worst-case time complexity of the reduction algorithm from [11] compares favourably with minimisation modulo branching or weak bisimulation equivalence. Hence, the algorithm from [11] can serve as a useful preprocessing step to these minimisation algorithms.

5 Symbolic Reasoning about Infinite-State Systems

For very large finite-state systems, a symbolic analysis on the level of LPOs may result in the generation of much smaller LTSs. For systems with an inherently infinite number of states the use of theorem proving techniques is indispensable.

The original motivation behind the LPO format was that several properties of a system can be uniformly expressed by first-order formulae. Effective proof methods for LPOs have been developed, incorporating the use of invariants [3] and state mappings [16]. Also the confluence property of an LPO can be expressed as a large first-order formula [15]. Using these techniques, large distributed systems were verified in a precise and logical way, often with the help of interactive theorem provers. See [9] for an overview of such case studies.

Since the confluence properties and the correctness criteria associated with state mappings for industrial-scale case studies tend to be rather flat but very large, we are developing a specialised theorem prover based on an extension of BDDs with equality [12]. A prototype tool has been implemented [19], which was used to detect confluence in a leader election protocol and in a Splice specification

from [6]. (This information on confluence was exploited using the method of [4]; see Section 3.) This tool can also check invariants and the correctness criteria associated with a state mapping between a specification and its implementation.

References

1. Th. Arts and I.A. van Langevelde. How μCRL supported a smart redesign of a real-world protocol. In *Proc. FMICS'99*, pp. 31–53, 1999.
2. Th. Arts and I.A. van Langevelde. Correct performance of transaction capabilities. In *Proc. ICACSD'2001*. IEEE, 2001.
3. M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In *Proc. 5th Conference on Concurrency Theory*, LNCS 836, pp. 401–416. Springer, 1994.
4. S.C.C. Blom. Partial τ-confluence for efficient state space generation. Technical Report, CWI, 2001.
5. M.G.J. van den Brand, H. de Jong, P. Klint, and P.A. Olivier. Efficient annotated terms. *Software – Practice and Experience*, 30(3):259–291, 2000.
6. P.F.G. Dechering and I.A. van Langevelde. The verification of coordination. In *Proc. COORDINATION'2000*, LNCS 1906, pp. 335-340. Springer, 2000.
7. J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP – a protocol validation and verification toolbox. In *Proc. 8th Conference on Computer-Aided Verification*, LNCS 1102, pp. 437–440. Springer, 1996.
8. J.F. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. Technical Report, CWI, 2001.
9. J.F. Groote, F. Monin, and J.C. van de Pol. Checking verifications of protocols and distributed systems by computer. In *Proc. 9th Conference on Concurrency Theory*, LNCS 1466, pp. 629–655. Springer, 1998.
10. J.F. Groote, J. Pang, and A.G. Wouters. Analysis of a distributed system for lifting trucks. Technical Report, CWI, 2001.
11. J.F. Groote and J.C. van de Pol. State space reduction using partial τ-confluence. In *Proc. MFCS'2000*, LNCS 1893, pp. 383–393. Springer, 2000.
12. J.F. Groote and J.C. van de Pol. Equational binary decision diagrams. In *Proc. LPAR'2000*, LNAI 1955, pp. 161–178. Springer, 2000.
13. J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Proc. ACP'94*, Workshops in Computing, pp. 26–62. Springer, 1995.
14. J.F. Groote, A. Ponse, and Y.S. Usenko. Linearization of parallel pCRL. To appear in *Journal of Logic and Algebraic Programming*.
15. J.F. Groote and M.P.A. Sellink. Confluence for process verification. *Theoretical Computer Science*, 170(1/2):47–81, 1996.
16. J.F. Groote and J. Springintveld. Focus points and convergent process operators: a proof strategy for protocol verification. In *Proc. ARTS'95*, 1995.
17. I.A. van Langevelde. A compact file format for labeled transition systems. Technical Report SEN R0102, CWI, 2001.
18. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
19. J.C. van de Pol. A prover for the μCRL toolset with applications – version 1. Technical Report SEN R0106, CWI, 2001.
20. Y.S. Usenko. State space generation for the HAVi leader election protocol. To appear in *Science of Computer Programming*.
21. A.G. Wouters. Manual for the μCRL toolset (version 1.11). Technical Report, CWI, 2001.