Truth/SLC — A Parallel Verification Platform for Concurrent Systems

Martin Leucker and Thomas Noll

Lehrstuhl für Informatik II, Aachen University of Technology
Ahornstr. 55, D–52056 Aachen, Germany
{leucker,noll}@informatik.rwth-aachen.de

1 Introduction

Concurrent software and hardware systems play an increasing rôle in today's applications. Due to the large number of states and to the high degree of non–determinism arising from the dynamic behavior of such systems, testing is generally not sufficient to ensure the correctness of their implementation. Formal specification and verification methods are therefore becoming more and more popular, aiming to give rigorous support for the system design and for establishing its correctness properties, respectively (cf. [2] for an overview).

In view of the inherent complexity of formal methods it is desirable to provide the user with tool support. It is even indispensable for the design of safety–critical concurrent systems where an *ad hoc* or conventional software engineering approach is not justifiable. There is one particularly successful automated approach to verification, called model checking, in which one tries to prove that (a model of) a system has certain properties specified in a suitable logic.

During the recent years several prototypes of model–checking tools have been developed, e.g., CWB [13], NCSU–CWB [4], SPIN [5], and the symbolic model checker SMV [9]. Most of these are tailored to a specific setting, choosing, e.g., the CCS process algebra with transition–system semantics as the specification language and offering model checking for the modal μ–calculus.

However, in the theoretical modeling and in the implementation of concurrent systems there exists a wide range of specification formalisms, semantic domains, logics, and model–checking algorithms. Our aim is therefore to offer a modular verification system which can be easily adjusted to different settings. We started out in 1998 with the development of an initial version of our tool, called TRUTH, which is described in Section 2. It was complemented later by rapid prototyping support for specification languages, provided by the SLC specification language compiler generator presented in Section 3. The most recent component of the TRUTH Verification Platform is a dedicated parallel version running on workstation clusters which is intended for high–end verification tasks, and which is briefly described in Section 4.

2 Truth: The Basic Tool

Here we give a short account of the actual TRUTH tool. For a more thorough presentation, the reader is referred to [6] and to [16], where different releases can be downloaded.

In its basic version TRUTH supports the specification and verification of concurrent systems described in CCS, a well–known process algebra [12]. To support the understanding of the system's behaviour, the specification can be graphically simulated in an interactive and process–oriented way. Figure 1 shows a screenshot for the simulation of a two–place buffer process `B2`, composed in parallel of two commicating instances of a unary buffer `B1`.

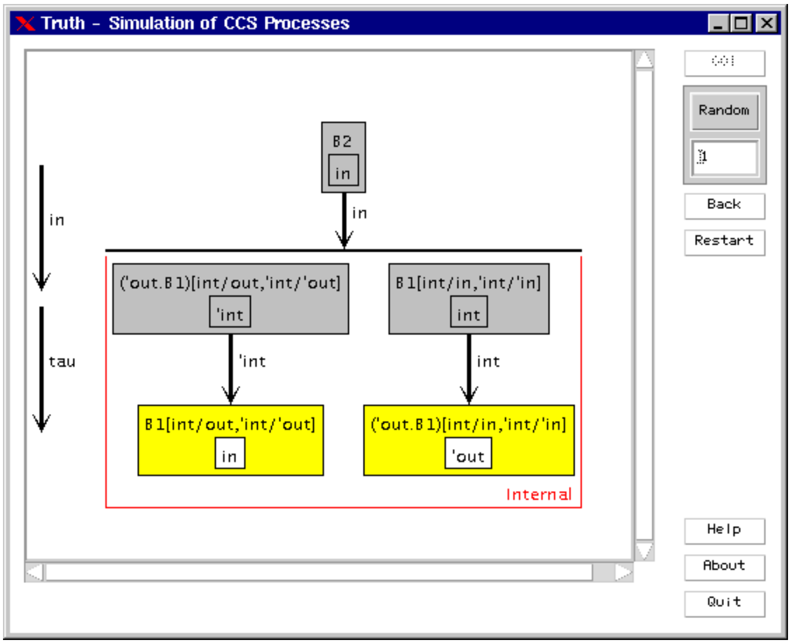

Fig. 1. A Process-Orientted Simulation of a Two-Place Buffer.

From the specification a labeled transition system is built. Its desired properties can be expressed using the μ–calculus, a powerful logic which allows to describe various safety, liveness, and fairness properties. It semantically subsumes the temporal logics CTL (whose operators are implemented as macros in TRUTH), CTL*, and LTL.

TRUTH offers several model checking algorithms, such as the tableau–based model checker proposed in [3]. It has fairly good runtime properties and supports the full μ–calculus. Furthermore, it is a local model checking algorithm, i.e., it has the advantage that in many cases only a part of the transition system has to be built in order to verify or to falsify a formula.

Additionally, a local game–based algorithm has been integrated which can be used to demonstrate the invalidity of a formula by means of an interactive construction of a counterexample [7,15]. Again, the process visualization component is used to play and visualize this game between the user and the TRUTH tool in order to support debugging of error–prone specifications.

As mentioned in the introduction, we have chosen a modular design that allows easy modifications and extensions of the system. In particular, this feature is exploited by a compiler–generator extension which will be described in the following section. Figure 2 gives an overview of the software architecture.

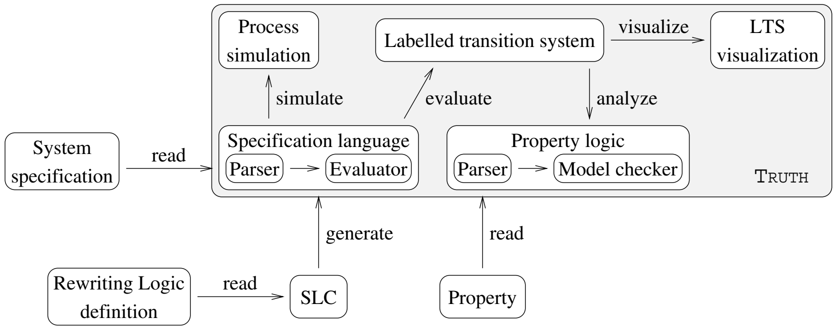

Fig. 2. Structure of TRUTH/SLC.

TRUTH is implemented in Haskell, a general–purpose, fully functional programming language. The choice of a declarative language serves a number of purposes. Changes to the system become easier when using a language which lacks side effects. Moreover many algorithms which are employed in the context of model checking have a very concise functional notation. This makes the implementation easier to understand. Furthermore, in principle it allows to prove the correctness of the implementation which is crucial for a model–checking tool to be used in safety–critical applications. By employing optimization techniques such as state monads for destructive updates we achieve a runtime behaviour which is competitive with other model–checking tools supporting process specifications in CCS.

3 SLC: The Specification Language Compiler Generator

A notable extension of TRUTH is the SLC Specification Language Compiler Generator which provides generic support for different specification formalisms [8]. Given a formal description of a specification language, it automatically generates a corresponding TRUTH frontend (cf. Figure 2).

More specifically, the syntax and semantics of the specification language has to be described in terms of Rewriting Logic, a unified semantic framework for concurrency [11]. From this definition a compiler is derived which is capable of parsing a concrete system specification and of computing the corresponding semantic object, such as a labeled transition system. This compiler is linked together with the TRUTH platform to obtain a model–checking tool which is tailored for the specification language in question.

The description of the specification language formalism consists of three parts. First, the syntax of the language has to be given in terms of a context free grammar (with typing information). The second part is a set of conditional rewrite rules defining the operational semantics.

Finally, the description contains a set of equations between process terms which identify certain states of the respective system, thus reducing the state space. Considering CCS for example, we can define equations like $x \parallel y = y \parallel x$ and $x \parallel \mathsf{nil} = x$. Then the resulting transition system is minimized with respect to symmetry, and, since "dead" nil processes are removed, it is often finite–state although the original semantics would yield an infinite system.

We have successfully developed an instance of Truth for a version of CCS respecting the previous equations. To verify that our approach is also applicable in connection with other models of concurrency than labeled transition systems, we constructed an implementation for Petri nets. Currently we employ our compiler generator to support the distributed functional programming language *Erlang*.

4 Truth: The Parallel Version

Despite the improvements of model checking techniques during the last years, the so–called *state space explosion* still limits its application. While *partial order reduction* [14] or *symbolic model checking* [10] reduce the state space by orders of magnitude, typical verification tasks still last days on a single workstation or are even (practically) undecidable due to memory restrictions.

On the other hand, cheap yet powerful parallel computers can be constructed by building Networks Of Workstations (*NOWs*). From the outside, a NOW appears as one single parallel computer with high computing power and, even more important, huge amount of memory. This enables parallel programs to utilize the accumulated resources of a NOW to solve large problems.

Hence, it is a fundamental goal to find parallel model checking algorithms which then may be combined with well–known techniques to avoid the state space explosion to gain even more speedup and further reduce memory requirements.

We developed a parallel model checking algorithm for the alternation–free fragment of the μ-calculus. It distributes the underlying transition system and the formula to check over a NOW in parallel and determines, again in parallel, whether the initial state of the transition system satisfies the formula.

Systems with several millions of states could be constructed within half an hour on a NOW consisting of up to 52 processors. We found out that the algorithm scales very well wrt. run–time and memory consumption when enlarging the NOW. Furthermore, the distribution of states on the processors is homogeneous.

While the demand for parallel verification procedures also attracted several other researchers (on overview can be found in [1]), Parallel Truth is—to our knowledge—the first parallel model checking tool that allows the validation of safety *and* liveness properties.

A thorough presentation of this algorithm and its runtime properties can be found in [1].

References

1. Benedikt Bollig, Martin Leucker, and Michael Weber. Local parallel model checking for the alternation free μ–calculus. Technical Report AIB-04-2001, RWTH Aachen, March 2001.
2. E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. Technical Report CMU-CS-96-178, Carnegie Mellon University (CMU), September 1996.
3. R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–748, 1990.
4. R. Cleaveland and S. Sims. The NCSU concurrency workbench. In *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397, 1996.
5. Jean-Charles Grégoire, Gerard J. Holzmann, and Doron A. Peled, editors. *The Spin Verification System*, volume 32 of *DIMACS series.* American Mathematical Society, 1997. ISBN 0-8218-0680-7, 203p.
6. M. Lange, M. Leucker, T. Noll, and S. Tobies. Truth – a verification platform for concurrent systems. In *Tool Support for System Specification, Development, and Verification*, Advances in Computing Science. Springer-Verlag Wien New York, 1999.
7. M. Leucker. Model checking games for the alternation free mu-calculus and alternating automata. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 77–91. Springer, 1999.
8. Martin Leucker and Thomas Noll. Rewriting logic as a framework for generic verification tools. In *Proceedings of the Third International Workshop on Rewriting Logic and its Applications (WRLA'00)*, volume 36 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 2000.
9. K. L. McMillan. The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University, 1992.
10. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.
11. José Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Seventh International Conference on Concurrency Theory (CONCUR '96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 331–372. Springer Verlag, August 1996.
12. R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.
13. F. Moller. *The Edinburgh Concurrency Workbench (Version 6.1).* Department of Computer Science, University of Edinburgh, October 1992.
14. Doron Peled. Ten years of partial order reduction. In *CAV, Computer Aided Verification*, number 1427 in LNCS, pages 17–28, Vancouver, BC, Canada, 1998. Springer.
15. Perdita Stevens and Colin Stirling. Practical model-checking using games. In B. Steffen, editor, *Proceedings of the 4th International Conference on Tools and algorithms for the construction and analysis of systems (TACAS '98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 85–101, New York, NY, USA, 1998. Springer-Verlag Inc.
16. The TRUTH verification tool. `http://www-i2.informatik.rwth-aachen.de/Research/MCS/Truth`.