The NCSU Concurrency Workbench*

Rance Cleaveland Steve Sims

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
e-mail: {rance@csc, stsims@eos}.ncsu.edu

Abstract. The NCSU Concurrency Workbench is a tool for verifying finite-state systems. A key feature is its flexibility; its modular design eases the task of adding new analyses and changing the language users employ for describing systems. This note gives an overview of the system's features, including its capacity for generating diagnostic information for incorrect systems, and discusses some of its applications.

1 Introduction

The NCSU Concurrency Workbench (NCSU-CWB) [1] supports the automatic verification of finite-state concurrent systems. The main goal of the system is to provide users with a tool that is flexible and easy to use and yet whose performance is competitive with that of existing special-purpose tools. In support of the former, and like its predecessor, the (Edinburgh) Concurrency Workbench [9, 15], the NCSU-CWB includes implementations of decision procedures for calculating a number of different behavioral equivalences and preorders between systems and for determining whether systems satisfy formulas written in an expressive temporal logic, the modal mu-calculus. In contrast with the Edinburgh CWB, and with other tools, however, the NCSU-CWB also supports the following.

- **Diagnostic information.** The NCSU-CWB provides appropriate diagnostic information for explaining why two systems fail to be related by a given semantic equivalence or preorder.
- **Language flexibility.** The design of the system exploits the language-independence of its analysis routines by localizing language-specific procedures (syntactic analyzers, semantic functions) in one module. This enables users to change the system description language of the CWB using the Process Algebra Compiler tool [8].

In order to enable the tool to handle large "real-world" systems we have also paid great attention to issues of time- and space-efficiency.

The remainder of this note provides an overview of the features of the NCSU-CWB and reviews some case studies to which the tool has been applied.

* Research supported by NSF grants CCR-9120995 and CCR-9402807, ONR Young Investigator Award N00014-92-J-1582, NSF Young Investigator Award CCR-9257963, and AFOSR grant F49620-95-1-0508.

2 System Overview

User Interface. The NCSU-CWB supports two user interfaces: one text-based, and hence capable of running on a variety of different platforms, and the other graphical, and hence easier to use. Each consists of a "command loop"; users enter commands either textually or by pushing buttons, and the system calculates and displays the result. To analyze a system, a user first creates a file containing the definition of the system in the language supported by the version of the NCSU-CWB at hand, invokes the tool, loads the file into the NCSU-CWB, and executes appropriate commands.

Commands. In addition to providing a system simulation facility, the NCSU-CWB can compute a number of behavioral equivalences and preorders and calculate whether or not systems satisfy mu-calculus formulas. The textual interface of the NCSU-CWB uses a shell-like syntax for commands corresponding to these procedures. For example, the command to check whether systems `Spec` and `ABP` are must equivalent is the following: `eq -Smust Spec ABP`. Here the qualifier for the `-S` flag indicates which semantic equivalence should be checked; other possible qualifiers include `sim` (simulation equivalence), `obseq` (observational equivalence), `trace` (trace equivalence), and several others. The general command for preorder checking, `le`, uses the same scheme for specifying which ordering to check. In any case, appropriate *diagnostic information* is returned to the user when the given systems are found to be unrelated. For example, two systems are must equivalent iff they must pass exactly the same tests (in a precisely defined sense). Thus if `Spec` and `ABP` are not must equivalent, there must be a test that distinguishes them; when this is the case the NCSU-CWB computes such a test and displays it to the user.

The NCSU-CWB includes two model checkers, each of which allow users to specify formulas in a particular subset of the modal mu-calculus; one accepts formulas in the alternation free fragment [10] while the other accepts formulas in the L_2 fragment [2].

The system also supports commands for *minimizing* systems with respect to certain equivalences; this proves to be very useful in fighting state explosion.

Implementation. The NCSU-CWB is implemented in SML of New Jersey and consists of approximately 18,000 lines of code.

3 System Design

As stated in the introduction, one of the main features of the NCSU-CWB is its flexibility; in particular, users should be able to 1) add new equivalences and preorders, and 2) alter the language used for defining systems. In support of the first goal, the design of the NCSU-CWB uses *generic* preorder/equivalence-checking algorithms in conjunction with *system and formula transformations* in order to compute equivalences and preorders. To illustrate the approach, we explain how the tool calculates its response to the command `eq -Smust Spec ABP` mentioned in the previous section.

1. `Spec` and `ABP` are "compiled" into automata.
2. These automata are transformed into a type of deterministic automata [6].
3. The transformed automata are fed into a generic equivalence checker, which calculates whether or not the two automata are bisimulation equivalent [14, 16]. If they are not equivalent, the checker produces a formula in a special logic that distinguishes the two automata [5].
4. If a formula is produced, a formula transformer converts it into a test, which is then returned to the user [4].

This general scheme is robust: numerous equivalences may be computed, and relevant diagnostic information generated, using appropriate transformations in tandem with bisimulation equivalence. Consequently, to add an equivalence relation to the NCSU-CWB, one need only define an appropriate process transformation and, if diagnostic information is desired, a formula transformation. A similar scheme is used for preorders.

In support of the second goal, all language-specific information has been localized within one module inside the NCSU-CWB. This module may be thought of as encapsulating the parse trees resulting from processing a user-supplied system description; it exports parsing and unparsing functions, and semantic routines (such as those for calculating single-step transitions), for these trees to the rest of the tool. To change the system description language, then, one need only alter routines in this module. The Process Algebra Compiler [8] greatly eases this task by providing users with a high-level notation in which to define the syntax and (operational) semantics of their languages. Using this tool, front ends for CCS, CSP [12], Basic Lotos [8], a version of CCS with priorities [7], and other languages have been generated for the NCSU-CWB.

4 Applications and Future Work

To date the NCSU-CWB has been applied to the analysis of several different systems, including the following.

- The connection phase of the UNI (Version 3.0) protocol used in ATM networks was formalized in CCS in [3] and verified. The largest finite-state machine handled in the course of the analysis contained about 60,000 reachable states.
- The timing behavior of an active-structure control system was analyzed in [11]. The system was formalized in a real-time variant of CCS and contained in excess of 10^{19} states. By minimizing the system in a componentwise manner, however, the analysis was carried to completion.
- The functional behavior of different variations of a railway signaling system was analyzed in [7]. The language used to define the system borrowed constructs from several different process algebras, while the system's requirements were specified using mu-calculus formulas.

As future work, we plan to investigate techniques for computing and displaying diagnostic information when systems fail to satisfy temporal formulas. We have also partially implemented on-the-fly versions of the preorder/equivalence-checking routines with a view to comparing the performance of global and on-the-fly algorithms.

References

1. The NCSU Concurrency Workbench home page. URL http://www4.ncsu.edu/eos/users/r/rance/WWW/ncsu-cw.html.
2. G. Bhat and R. Cleaveland. Efficient local model checking for fragments of the modal μ-calculus. In Margaria and Steffen [13], pages 107–126.
3. U. Celikkan. *Semantic Preorders in the Automated Verification of Concurrent Systems.* PhD thesis, North Carolina State University, 1995.
4. U. Celikkan and R. Cleaveland. Computing diagnostic tests for incorrect processes. In *Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification*, pages 263–278, Lake Buena Vista, Florida, June 1992. North-Holland.
5. R. Cleaveland. On automatically explaining bisimulation inequivalence. In E.M. Clarke and R.P. Kurshan, editors, *Computer Aided Verification (CAV '90)*, volume 531 of *Lecture Notes in Computer Science*, pages 364–372, Piscataway, NJ, June 1990. Springer-Verlag.
6. R. Cleaveland and M.C.B. Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Aspects of Computing*, 5:1–20, 1993.
7. R. Cleaveland, G. Luettgen, V. Natarajan, and S. Sims. Priorities for verifying distributed systems. In Margaria and Steffen [13], pages 278–297.
8. R. Cleaveland, E. Madelaine, and S. Sims. A front-end generator for verification tools. In E. Brinksma, R. Cleaveland, K.G. Larsen, and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '95)*, volume 1019 of *Lecture Notes in Computer Science*, pages 153–173, Aarhus, Denmark, May 1995. Springer-Verlag.
9. R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of finite-state systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.
10. R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2:121–147, 1993.
11. W. Elseaidy, J. Baugh, and R. Cleaveland. Verification of an active control system using temporal process algebra. *Engineering with Computers*, 12:46–61, 1996.
12. J. Gray. A CSP interface for the concurrency workbench. Undergraduate Honors Thesis, Department of Computer Science, North Carolina State University, May 1996.
13. T. Margaria and B. Steffen, editors. *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96)*, volume 1055 of *Lecture Notes in Computer Science*, Passau, Germany, March 1996. Springer-Verlag.
14. R. Milner. *Communication and Concurrency.* Prentice-Hall, London, 1989.
15. F. Moller and P. Stevens. *The Edinburgh Concurrency Workbench (Version 7.0).* University of Edinburgh, November 1994.
16. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, December 1987.