The Concurrency Factory: A Development Environment for Concurrent Systems*

Rance Cleaveland
Dept. of Computer Science
N.C. State University
Raleigh, NC 27695-8206
rance@csc.ncsu.edu

Philip M. Lewis, Scott A. Smolka, Oleg Sokolsky
Dept. of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794–4400
{pml, sas, oleg}@cs.sunysb.edu

Abstract. The *Concurrency Factory* supports the specification, simulation, verification, and implementation of real-time concurrent systems such as communication protocols and process control systems. While the system uses *process algebra* as its underlying design formalism, the primary focus of the project is practical utility: the tools should be usable by engineers who are not familiar with formal models of concurrency, and it should be capable of handling large-scale systems such as those found in the telecommunications industry.
This paper serves as a status report for the Factory project and briefly describes a case-study involving the GNU UUCP i-protocol.

1 Introduction

The *Concurrency Factory* is an integrated toolset for specification, simulation, verification, and implementation of real-time concurrent systems such as communication protocols and process control systems. Two themes underpin the work done on the project: the use of *process algebra* [Mil89, BK84, Hoa85] as a formal design notation, and the provision of *practical* support for formal design analysis. Our goal is to make the Factory usable by system engineers who may not be familiar with formal verification as well as applicable to problems of the size found in industrial applications.

In order to achieve these aims, the Factory includes the following major components.

- A graphical editor, VTView [Tre92], and a simulator, VTSim [Jai93], for for hierarchically structured networks of finite-state processes. The graphical language, GCCS, resembles informal design diagrams drawn by engineers but possesses a formal, process-algebra-based semantics. We are currently extending the GUI to allow processes to be embedded in states of other processes, thereby permitting compact specifications such as those found in statecharts [Har87].

* Research supported in part by NSF Grants CCR–9120995, CCR–9208585, CCR–9257963, and CCR–9402807, AFOSR Grants F49620-93-1-0250 and F49620-95-1-0508, and ONR Grant N00014-92-J-1582.

- Support for system designs expressed in a programming-language-inspired design notation, VPL [Sok96]. VPL is a simple language for concurrent processes that communicate values from a finite data domain; as is the case with GCCS, however, the language features an underlying process-algebraic semantics. A compiler translates VPL programs into networks of finite-state processes.
- A collection of *analysis routines* that currently includes linear-time local and global model checker for the alternation-free fragment of the modal mu-calculus [CS93, Sok96], a local model checker for a real-time extension of this logic [SS95], and strong and weak bisimulation checkers.
  Care is being taken to ensure that these algorithms are efficient enough to be used on real-life systems. For example, we are investigating how these algorithms can be parallelized [ZS92, lic94], and made to perform incrementally [SS94].
- A *compiler* for transforming VTView and VPL specifications into executable code. The current Factory prototype produces Facile [GMP89] code, a concurrent language that symmetrically integrates many of the features of Standard ML [Mil84] and CCS [Mil89]. We are also considering adding a concurrent extension of C++ as another target language. The compiler relieves the user of the burden of manually recoding their designs in the target language of their final system.

The Concurrency Factory is written in C++ and executes under X-Windows, using Motif as the graphics engine, so that it is efficient, easily extendible, and highly portable. It is currently running on SUN SPARCstations under SunOS Release 4.1.

The remainder of this note describes VTView and VTSim and briefly discusses the i-protocol study. A fuller account of the system may be found in [CGL$^+$94] and at URL `http://www.cs.sunysb.edu/~concurr/`.

2 VTView, GCCS and VTSim

The graphical user interface of the Concurrency Factory consists of the graphical editor, VTView [Tre92], and the graphical simulator, VTSim [Jai93]. VTView [Tre92] supports the design of hierarchically structured systems of communicating tasks expressed in GCCS, a graphical specification language. GCCS provides system builders with intuitive constructs (buses, ports, links, a subsystem facility, etc.) for concurrent systems, and it allows for both top-down and bottom-up development methodologies. The tool maintains an internal representation of systems as they are being created; this internal representation may then be manipulated by other tools in the system.

In contrast with other graphical languages [Har87, Mar89], GCCS is designed to model systems in which processes execute asynchronously (although communication between processes is synchronous). The language is equipped with two semantics: one involving a translation into Milner's CCS [Mil89], and another in the form of a structural operational semantics, à la Plotkin [Plo81]. The latter semantics has been "implemented" in the Factory as a collection of methods that compute the set of transitions that are possible for a system in a given state. encapsulating the semantics of VTView objects, all tools within the Factory, including the simulator and model checkers, are guaranteed to interpret GCCS systems.

VTSim [Jai93] permits users to simulate graphically the execution of GCCS systems built using VTView. The tool provides both interactive and automatic modes of operation, and it also includes features such as breakpoints and reverse execution. The user may view the simulated execution of a system at different levels in the structure; one can either choose to observe the simulation at the interprocess level and watch the flow of messages, or one can look at individual processes in order to see why messages are sent when they are.

3 A Case Study: The i-protocol

The most sophisticated case study undertaken to date involved the use of the Concurrency Factory's local model checker to uncover and correct a subtle livelock in the i-protocol, a bidirectional sliding-window protocol implemented in the GNU UUCP file transfer utility. We analyzed a version of the protocol whose window size was 2; in the course of the analysis, the model checker explored 1.079×10^6 states out of a total estimated global state space of 1.473×10^{12}.

One key to the successful outcome of the case study was the use of an abstraction to reduce the message sequence number space from 32 — the constant defined in the protocol's C-code — to $2W$, where W is the window size. This insight underscores a central feature of practical use of formal verification: user understanding of the system being analyzed is crucial.

4 Future Work

We plan to extend the Factory in several directions, including the generation of simulator-based diagnostic information for verification routines, the development of improved state-space management techniques based on the underlying process-algebraic model, the support of languages besides Facile by the design compiler, and broader support for real-time systems.

References

[BK84] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[CGL+94] R. Cleaveland, J. N. Gada, P. M. Lewis, S. A. Smolka, O. Sokolsky, and S. Zhang. The concurrency factory — practical tools for specification, simulation, verification, and implementation of concurrent systems. In G.E. Blelloch, K.M. Chandy, and S. Jagannathan, editors, *Proceedings of DIMACS*

Workshop on Specification of Parallel Algorithms, volume 18 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 75–90, Princeton, NJ, May 1994. American Mathematical Society.

[CS93] R. Cleaveland and B. U. Steffen. A linear-time model checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2, 1993.

[GMP89] A. Giacalone, P. Mishra, and S. Prasad. Facile: A symmertric integration of concurrent and functional programming. *International Journal of Parallel Programming*, 18(2), 1989.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, London, 1985.

[Jai93] S. Jain. VTSIM: A graphical simulator for finite-state networks. Master's thesis, Department of Computer Science, North Carolina State University, 1993.

[lic94] *Ninth Annual Symposium on Logic in Computer Science (LICS '94)*, Versailles, France, July 1994. Computer Society Press.

[Mar89] F. Maraninchi. Argonaute, graphical description, semantics and verification of reactive systems by using a process algebra. In *Proc. CAV '89*, volume 407 of *Lecture Notes in Computer Science*, pages 38–53, Grenoble, June 1989. Springer-Verlag.

[Mil84] R. Milner. A proposal for standard ML. Technical Report CSR-157-83, Department of Computer Science, University of Edinburgh, 1984.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Sok96] O. Sokolsky. *Efficient Graph-Based Algorithms for Model Checking in the Modal Mu-Calculus*. PhD thesis, Department of Computer Science, SUNY at Stony Brook, April 1996.

[SS94] O. Sokolsky and S. A. Smolka. Incremental model checking. In *Proceedings of the 6th International Conference on Computer-Aided Verification*. American Mathematical Society, 1994.

[SS95] O. Sokolsky and S. A. Smolka. Local model checking for real-time systems. In *Proceedings of the 7th International Conference on Computer-Aided Verification*. American Mathematical Society, 1995.

[Tre92] V. Trehan. VTVIEW: A graphical editor for hierarchical networks of finite-state processes. Master's thesis, Department of Computer Science, North Carolina State University, December 1992.

[ZS92] S. Zhang and S. A. Smolka. Efficient parallelization of equivalence checking algorithms. In M. Dias and R. Gros, editors, *Proceedings of FORTE '92 – Fifth International Conference on Formal Description Techniques*, pages 133–146, October 1992.