Università di Pisa

Dipartimento di Informatica

Technical Report : TR-96-29

DEMOS at CONCUR96

Corrado Priami, Editor

August 22, 1996

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

DEMOS at CONCUR96

A Collection of Abstracts

Pisa, 26-29 August 1996

EDITOR

Corrado Priami, Università di Pisa

Table of Contents

P E P - (Programming Environment based on Petri Nets)
Eike Best, Burkhard Bieber, Bernd Grahlmann 1

The Fc2tools Set (Tool Demonstration)
Amar Bouali, Annie Ressouche, Valerie Roy, Robert de Simone 3

The Circal System
Antonio Cerone, Alex J. Cowie and George J. Milne 4

The NCSU Concurrency Workbench
Rance Cleaveland, Steve Sims 6

An automata based verification environment for mobile processes
The PPCC Group 10

The Compositional Security Checker:
a Tool for the Automatic Compositional Verification of Some Security Properties
Riccardo Focardi, Roberto Gorrieri 12

TIPPtool — Performance Estimation
based on a Stochastic Process Algebra
Holger Hermanns, Vassilis Mertsiotakis, Michael Rettelbach 14

MuDiv: A Tool for Partial Model Checking
Henrik Reif Andersen, Jørn Lind-Nielsen 15

Games with the Edinburgh Concurrency Workbench
Perdita Stevens 17

`prod` 3.1 - An Advanced Tool for Efficient Reachability Analysis
Kimmo Varpaaniemi, Johan Lilius and Keijo Heljanko 18

DEMOS at CONCUR96

This is a collection of the abstracts of the demonstrations given at the CONCUR conference held in Pisa in August 1996. The demonstrations cover different aspects of theory of concurrency. They range from process algebras verifiers to model checkers as well as to Petri nets support tools. There are also demonstrations on the check of security properties and performance analysis based on process algebra descriptions. Finally, a demonstration of game aspects with the CWB tool is presented.

Corrado Priami - Università di Pisa

P E P
(Programming Environment based on Petri Nets)

Eike Best, Burkhard Bieber, Bernd Grahlmann

Institut fuer Informatik
Universitaet Hildesheim
Marienburger Platz 22
D-31141 Hildesheim, Germany
e-mail: `first-name@informatik.uni-hildesheim.de`
http://`www.informatik.uni-hildesheim.de`

The PEP environment (Programming Environment based on Petri Nets) is a comprehensive graphical window-based system supporting:

- *A Petri net editing and simulation tool.* This includes an editor for both low-level and high-level nets; an on-screen simulator which runs in single-step or in automatic mode; a display of cross-references; and a feature for generating PostScript files out of displayed Petri nets. High-level nets can optionally be provided with hierarchical structure.

- *A process algebraic component.* This includes an editor for a CCS-like process algebra and a translator allowing to compile expressions of the algebra into low-level Petri nets (which may subsequently be used as input for the Petri net editor).

- *A programming language component.* This allows the user to design concurrent algorithms in a simple imperative programming language incorporating both shared data and channel-based communication between processes. The PEP system can generate either low-level or high-level Petri nets automatically from such programs. The resulting nets may be further processed by the net editor and/or the verification component. The system retains knowledge about which net(s) belong to which program(s). There also exists a cross-reference component mapping parts of a net to part of the program it belongs to.

- *A formula editor.* This allows the user to input and edit a set of formulae of a simple temporal logic. The basic assertions of the logic refer to a given net. Its operators are the Boolean connectives and the "FOR ALL" temporal logic operator (and other derived operators). This yields a restricted branching-time CTL in which a large range of safety properties (such as "does there exist a possibility of executing two given transitions concurrently?") as well as a restricted number of liveness properties can be expressed.

- *A verification component.* This allows both specific properties such as boundedness and deadlock-freeness, as well as user-defined properties, i.e. formulae of the temporal logic, to be checked. The implemented techniques avoid the construction of the state space altogether, which renders the model checking algorithm very time-efficient (and which is also the main reason for the present restriction on branching-time rather than linear-time model checking). Using the cross-reference component, a user-defined property can be made to refer either to a program or to a (corresponding) net, and the model checking algorithm works equally for both.

  The dedicated algorithms available in PEP also avoid constructing the state space. For example, the reachability of a marking and deadlock-freeness can be checked by linear programming.

- *An interface for INA (Peter Starke's textual analysis tool).* This interface provides access to various algorithms implemented within INA. They include, among others, the calculation of invariants and the construction of stubborn set and/or symmetrically reduced reachability graphs.

PEP has been implemented on Solaris 2.x, SUN OS 4.1.x and Linux. A version for AIX is in preparation. Ftp-able versions and further informations are available via the URL `http://www.informatik.uni-hildesheim.de/~pep`.

The FC2TOOLS Set (Tool Demonstration)

Amar Bouali[1] *Annie Ressouche*[2]
Valerie Roy[1] *Robert de Simone*[2]

The AUTO/GRAPH toolset developed in our group was a pioneering software for analysis and verification of networks of communicating processes. We describe here the next-generation AUTO/GRAPH, consisting of a modular tool suite interfaced around a common file description format named FC2. The format allows representation of single reactive automata as well as combining networks. This format was developed in the scope of Esprit BRA project 7166:CONCUR2.

In the new implementation, most analysis functions are implemented with redundancy using both *explicit* classical representation of automata, and also *implicit* state space symbolic representation using *Binary Decision Diagrams.* The two alternative techniques offer drastically different performances in different cases, and having both at hand in a unified framework is a valuable thing.

Both FC2EXPLICIT and FC2IMPLICIT commands perform synchronised product and reachable state space search. They can minimize results w.r.t. *strong, weak, branching* bisimulation notions, and produce the result as an FC2 automaton. They can also *abstract* the system with a notion of "abstract actions", each synthesizing a set of sequences of concrete behaviours (in this sense behavioural abstraction can be seen as reverse from refinement). In addition FC2IMPLICIT has a fast checker for *deadlocks, livelock* or *divergent* states, for which it produces counterexample paths in case of existence, while FC2EXPLICIT allows *compositional* reduction techniques, mostly in case of "observational" bisimulation minimisations. Several extensions are still underway.

The tool suite is completed by the graphical editor AUTOGRAPH, which allows for graphical depiction of automata and networks as well as recollection of some form of results.

The toolset is available by anonymous ftp. Information can be obtained from the WWW page `http://cma.cma.fr/Verification/verif-eng.html`, or by e-mailing to `fc2team@cma.cma.fr`.

[1] CMA/Ecole des Mines de Paris, B.P. 207, F-06904 Sophia-Antipolis cedex
[2] INRIA , B.P. 93, F-06902 Sophia-Antipolis cedex

The Circal System

Antonio Cerone, Alex J. Cowie and George J. Milne
Advanced Computing Research Centre
School of Computer and Information Science
University of South Australia
Adelaide SA 5095
Australia
E-mails: {cerone,cowie,milne}@cis.unisa.edu.au

This toolset is based on Circal [3], a process algebra with the following characteristics

- Processes are guarded by set of actions that are always performed synchronously and an arbitrary number of processes can synchronize on the same action.
- There is no special action to describe a process that is evolving due to an internal (invisible) cause, but there are two different operators for an internal choice and an external choice.
- There is a particular concurrent composition operator which allows the synchronization of actions with the same name and prevents actions belonging to both of the components from occurring independently.

The Circal process algebra has been embedded in the high level language XTC in order to enhance the descriptive facilities available to the user. The resulting language, called XCircal, has many features derived from the programming language C and two new primitive types, the type "Event", describing a set of actions, and the type "Process". Data of both of these types can be structured in arrays, they can be used inside control structures such as "for-loops" and "if" statements and as arguments of functions, and they can be returned by functions. This makes it easier to build sets of event and processes, in particular when the system consists of identical replicated components (e.g. a buffer consisting of n identical cells).

The Circal System [3] is an implementation of XCircal existing in two versions: one uses an explicit representation of the state space, whereas the other uses an implicit representation in terms of OBDD's. The system also implements equivalence testing and several preorders on processes. The Circal System can be used for the specification and automatic verification of concurrent systems. Simulation can be carried out by composing test patterns (which can contain both input and output events) with a system behaviour. The capabilities of synchronising several processes on the same action and using sets of actions as events support the use of a constraint-based modelling style and a simple representation of time. Time can be modelled as tick actions which can be manipulated in the same way as other actions without any extension to the process algebra framework [2].

A graphical interface to a Circal model of an audio control communication protocol [1] will be demonstrated. The user interface allows the input of messages and specification of relative clock timings both in the simulation and the formal verification of the protocol operation.

WWW Information

http://cis0.levels.unisa.edu.au/www/projects/circal/circal_system.html

References

[1] D.J.B. Bosscher, I. Polak, and F.W. Vaandrager. Verification of an audio control protocol. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Development*, volume 2 of *AMAST Series in Computing*, pages 147–176. World Scientific, Singapore, 1994.

[2] G. J. Milne. The formal description and verification of hardware timing. *IEEE Transactions on Computers*, 40(7):811–826, July 1991.

[3] G. J. Milne. *Formal Specification and Verification of Digital Systems.* McGraw-Hill, 1994.

The NCSU Concurrency Workbench[1]

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`

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.

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.

[1]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. This work is reprinted from CAV96

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 component-wise 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.

An automata based verification environment for mobile processes[2]

The PPCC Group[3]

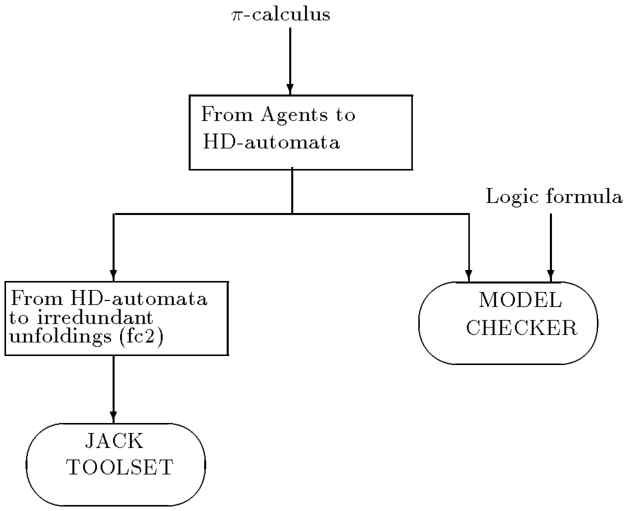
The π-calculus [5] is a process calculus able to describe 'mobile' processes, i.e. processes with a communication topology which is dynamically changing. In the π-calculus mobility is achieved by giving processes the possibility of communicating channel names. The π-calculus has greater expressive power than ordinary process calculi, but also a much more complicate theory. In particular, due to name-passing and name-creation during communications, the usual operational models are infinite-state. The techniques developed to obtain finite representations are often *ad hoc* and do not allow for re-usage of theory and algorithms [3, 7].

In [6] it has been presented a general technique which allows to associate ordinary finite state automata to a wide class of π-calculus agents, so that equivalent automata are associated to equivalent agents. The finite-state representation of π agents is obtained in two steps. In the first step an intermediate *finite-state* and *finite-branching* representation, called HD-Automaton, is produced. Names and name associations are semantic objects directly represented inside HD-automata. The second step produces, starting from a finite HD-automaton, an ordinary finite automaton.

We explore the ideas and the techniques of [6] to develop an automata-based verification environment for the π-calculus. The verification environment for the π-calculus agents has been implemented on top of an existing verification environment for process algebras: the JACK environment [2]. The JACK system includes general facilities to decide bisimulation equivalences. The current prototype version provides facilities to construct the HD-automata representation of π-agents, and to map HD-Automata into ordinary automata (represented in the standard `fc2` format [4]). Moreover, a model checker for verifying logical properties of mobile processes has been implemented. The model checker allows us to check behavioural properties (expressed by formulae of the Hennessy-Milner logic enriched with π-calculus actions) directly on HD-Automaton [1]. An extension of the model checker to more powerful logics is under investigation. The logical structure of the verification environment is illustrated in the following figure.

[2]Work partially founded by the *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione* CNR Integrated Project.

[3]The members of the PPCC group are: Gianluigi Ferrari, Ugo Montanari and Marco Pistore (Dept. of Computer Science, University of Pisa), Giovanni Ferro (IEI-CNR, Pisa), Stefania Gnesi (IEI-CNR, Pisa) and Gioia Ristori (Scuola Superiore di Studi Universitari e di Perfezionamento S. Anna, Pisa). Michele Boreale (University of Rome) collaborated with some PPCC members in the design and implementation of the Model Checker.



References

[1] M. Boreale, G. Ferro, G. Ristori. A model checking tool for the π-calculus based on π-automata. *I.E.I. - C.N.R. Tech. Rep. B4-52-12-95.*

[2] A. Bouali, S. Gnesi and S. Larosa. The integration project for the JACK environment. *EATCS Bullettin*, October 1994.

[3] M. Dam. On the decidability of process equivalences for the π-calculus. In *Proc. AMAST'95*, LNCS 936, 1995.

[4] A. Bouali, A. Ressouche, V. Roy, and R. de Simone. The FC2Tools set. *In Proc. CAV'96*, LNCS, 1996.

[5] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I and II. *Information and Computation*, 100(1):1–77, 1992.

[6] U. Montanari and M. Pistore. Checking Bisimilarity for Finitary π-calculus. In *Proc. CONCUR'95*, LNCS 962, 1995.

[7] B. Victor and F. Moller. The Mobility Workbench — A Tool for the π-calculus. In *Proc. CAV'94*, LNCS 818, 1994.

The Compositional Security Checker: a Tool for the Automatic Compositional Verification of Some Security Properties

Riccardo Focardi, Roberto Gorrieri
Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A. Zamboni 7, I – 40127 Bologna (Italy).
e_mail:{focardi,gorrieri}@cs.unibo.it

Security is a crucial property of system behaviour, requiring a strict control over the information flow among parts of the system. The main problem is to limit, and possibly to avoid, the damages produced by malicious programs, called *Trojan Horses*, which try to broadcast secret information. There are several approaches to solve this problem. An example is *Multilevel Security* [1]: every object is bounded to a security level, and so every subject is; information can flow from a certain object to a certain subject only if the level of the subject is greater than the level of the object. So a Trojan Horse, which operates at a certain level, has no way to *downgrade* information, and its action is restricted inside such a level. There are two access rules: *No Read Up* (a subject cannot read data from an upper level object) and *No Write Down* (a subject cannot write data to a lower level object).

However, these access rules are not enough. It could be possible to indirectly transmit information using some system *side effect*. For example, if two levels – 'high' and 'low' – share some finite storage resource, it is possible to transmit data from level 'high' to level 'low' by using the 'resource full' error message. For a high level transmitter, it is sufficient to alternatively fill or empty the resource in order to transmit a '1' or a '0' datum. Simultaneously, the low level receiver tries to write on the resource, decoding every error message as a '1' and every successful write as a '0'. It is clear that such indirect

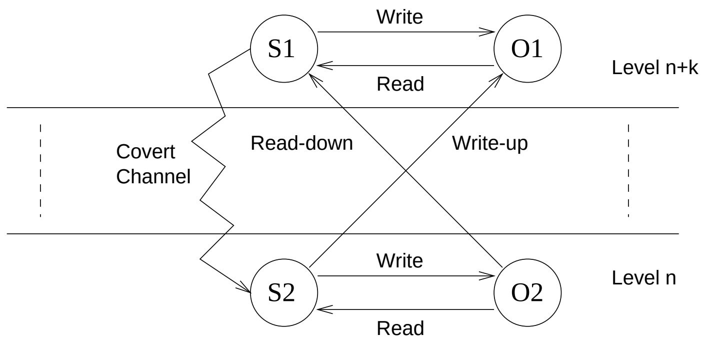


Figure 1: Information flows in multilevel security.

ways of transmission, called *covert channels*, do not violate the two multilevel access rules (see Figure 1). Therefore it is necessary to integrate a MAC discipline with a covert channel analysis (see e.g. [7]).

An alternative, more general approach to security requires to control directly the whole flow of information, rather than the accesses of subjects to objects. To make this, it is necessary to choose a formal model of system behaviour and to define the information flow on such a model. By imposing some information flow rule, we can control any kind of transmission, be it direct or indirect.

In the literature, there are many different definitions of this kind based on several system models (see e.g. [5, 8]). In [3] we have rephrased them in the uniform setting of *Security Process Algebra* (SPA, for short), then compared and classified. SPA is an extension of CCS [6] which permits to describe systems where actions belong to two different levels of confidentiality.

For some of the investigated information flow properties, we provided useful algebraic characterizations. They are all of the following form. Let E be an SPA process term, let X be a security property, let

$\approx$ be a semantic equivalence among process terms and let $\mathcal{C}_X$ and $\mathcal{D}_X$ be two SPA contexts for property X. Then, we can say: E is X-secure if and only if $\mathcal{C}_X[E] \approx \mathcal{D}_X[E]$.

Hence, checking the X-security of E is reduced to the "standard" problem of checking semantic equivalence between two terms having E as a subterm. In recent years a certain number of tools for checking semantic equivalence have been presented; among them, the Concurrency Workbench (CW for short) [2] is one of the most famous.

The tool we want to present is the Compositional Security Checker (CSC) which can be used to check automatically (finite state) SPA specifications against some information flow security properties. It has the same modular architecture of CW (Version 6.1), from which some modules have been entirely imported and some others only modified. When a user wants to check if an SPA process E is X-secure, CSC first provides operational semantic descriptions to the terms $\mathcal{C}_X[E]$ and $\mathcal{D}_X[E]$ in the form of two lts's; then verifies the semantic equivalence of $\mathcal{C}_X[E]$ and $\mathcal{D}_X[E]$ using their lts representations.

An interesting feature of CSC is the exploitation of the compositionality of some security property. The main problem in the verification of security properties is the exponential state explosion due to parallel composition. As an example consider two agents E_1 and E_2; the number of states of their parallel composition $E_1|E_2$ is equal to the product of the states of E_1 and E_2. Now if we have a compositional security property X, i.e. such that $F_1|F_2$ is X-secure whenever F_1 and F_2 are X-secure, then we can apply the following strategy: check the X-security of E_1 and E_2; if it is satisfied conclude that $E_1|E_2$ is X-secure, otherwise check the X-security of the whole agent $E_1|E_2$. Using this strategy for compositional security properties, CSC is able to avoid, in some cases, the exponential state explosion due to parallel composition operator.

A preliminary paper about the Compositional Security Checker is [4].

References

[1] D. E. Bell and L. J. La Padula. "Secure Computer Systems: Unified Exposition and Multics Interpretation". *ESD-TR-75-306, MITRE MTR-2997*, March 1976.

[2] R. Cleaveland, J. Parrow, and B. Steffen. "The Concurrency Workbench: a Semantics Based Tool for the Verification of Concurrent Systems". *ACM Transactions on Programming Languages and Systems*, Vol. 15 No. 1:36–72, January 1993.

[3] R. Focardi and R. Gorrieri. "A Classification of Security Properties for Process Algebras". *Journal of Computer Security*, 3(1):5–33, 1994/1995.

[4] R. Focardi and R. Gorrieri. "Automatic Compositional Verification of Some Security Properties". In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 167–186, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.

[5] J. A. Goguen and J. Meseguer. "Security Policy and Security Models". In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, April 1982.

[6] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[7] C. R. Tsai, V. D. Gligor, and C. S. Chandersekaran. "On the Identification of Covert Storage Channels in Secure Systems". *IEEE Transactions on Software Engineering*, pages 569–580, June 1990.

[8] J. T. Wittbold and D. M. Johnson. "Information Flow in Nondeterministic Systems". In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 144–161. IEEE Computer Society Press, 1990.

TIPPtool — Performance Estimation based on a Stochastic Process Algebra

Holger Hermanns, Vassilis Mertsiotakis, Michael Rettelbach

Universität Erlangen-Nürnberg, IMMD VII, Martensstr. 3, D–91058 Erlangen

There are many ways to incorporate a notion of time into process algebras in order to integrate functional design and performance analysis. One major research strand, *stochastic process algebras*, concentrates on the annotation of actions with exponentially distributed random variables.

We present a stochastic process algebra based modelling tool for functional analysis and performance evaluation. *TIPPtool* provides facilities for model specification, reachability analysis, as well as several numerical performance analysis algorithms. The input language of the tool is closely related to LOTOS, but enriched with stochastic timing information. An exponentially distributed random variable can be associated with each activity specifying its duration. Additionally, immediate actions and probabilistic branching are supported. The behaviour of a specification is defined by a structural operational semantics that maps a specification onto its semantic model, a (decorated) *labelled transition system* (LTS) [1].

Model descriptions are syntactically and semantically analyzed using a parser written in the functional programming language *Standard ML*. The parser has been generated using the tools *ML-YACC/LEXGEN* from the New Jersey ML toolkit. The operational semantics is also implemented in Standard ML and produces the underlying LTS which may subsequently be analyzed by other program modules written in *C*.

Performance analysis is based on the LTS. Every term in our language can be interpreted as a high level description of a Continuous Time Markov Chain (CTMC). For any model with finite state space the underlying CTMC can be derived directly by associating a state i with each node P of the LTS. Characteristic performance and dependability measures are extracted via the steady state solution of the CTMC using standard numerical techniques. The resulting values π_i can be interpreted as the probability to observe the behaviour corresponding to each process state i at any time in the steady state. The only sufficient condition for a unique equilibrium solution is strong connectivity of the LTS.

If the latter condition is violated or steady state measures do not give a sufficient insight into the system behaviour, it is possible to carry out transient analysis. Standard algorithms like randomization or calculation of higher moments may be used to find time-related properties.

The result of numerical analysis is usually a vector with state probabilities. In order to obtain more sophisticated and expressive results the user can specify measures. Experiment series are also supported by allowing rates to be symbolic variables. The specification of the model as well as the measures and experiments is supported through a graphical user interface. For more detailed explanations cf. [2]. More papers and information about the TIPP project are available at `http://www7.informatik.uni-erlangen.de/tipp/`.

References

[1] Norbert Götz, Ulrich Herzog, and Michael Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *Proc. of the 16th Int'l Symposium on Computer Performance Modelling, Measurement and Evaluation, PERFORMANCE '93*. Springer, 1993. LNCS 729.

[2] H. Hermanns and V. Mertsiotakis. A Stochastic Process Algebra Based Modelling Tool. In *Proc. of the 11th U.K. Perf. Eng. Workshop for Computer and Telecommunication Systems*. Springer, 1995.

MuDiv: A Tool for Partial Model Checking

Henrik Reif Andersen, Jørn Lind-Nielsen

Department of Information Technology, Technical University of Denmark, Bldg. 344, DK-2800 Lyngby.
Fax: +45 4588 4530, Denmark. E-mail: {hra,jl}@it.dtu.dk

Abstract

MuDiv is a tool for verifying concurrent systems. It is based on the technique of *partial model checking* described in [1]. The technique uses a variation of the modal μ-calculus, known as modal equations, to express the modal requirements and parallel compositions of finite labelled transition systems to express the models.

MuDiv is applicable to satisfaction problems of the form:

$$(t_1 \mid t_2 \mid t_3) \upharpoonright L \models A$$

Meaning "*transition system t_1 in parallel with t_2 and t_3, restricted to the actions in L, satisfies the assertion A*". The restriction defines the external communication which the transition systems may perform, and the assertion describes the requirements to the system as a sequence of modal equations.

MuDiv can be asked to solve two different kinds of problems. It can prove whether the assertion holds or not, returning true or false to indicate the result, or it can calculate the requirements A' for an unknown transition system x like this:

$$(x \mid t_1 \mid t_2) \upharpoonright L \models A$$

$$\Updownarrow$$

$$x \models A'$$

As an example, the three transition systems could be a sender, a receiver and a medium, modelling a communication protocol. The sender and the receiver are assumed known, but the medium is still undefined. MuDiv may then be asked to move the sender and receiver systems into the requirements, leaving the exact, necessary and sufficient requirements for the medium. If the sender and receiver are already inconsistent with respect to the requirement A, the result would be the assertion 'false.'

The name *partial model checking* is chosen to reflect that MuDiv transfers the transition systems, one by one, into the specifications, thereby building a new specification. So each of these intermediate specifications provides a partial answer to the original problem. Crucial to the success of applying partial model checking is a series of efficient implementable simplification strategies for modal equations. They vary in complexity from simple propagation of constant equations to symbolically determining equivalence of equations.

The implementation of MuDiv is performed in standard C++, and the result is a non-interactive batch program, where the input is provided as one or more input files, describing the model and the requirements. The output is the result of the model check and is presented on the standard output or written to a file. Some tests with MuDiv, using Milners Scheduler as an example, has shown promising results. Instead of an exponential explosion of the state space, the running time grows as a third degree polynomial of the input size.

Hardware and software requirements

A standard C++ compiler, preferably under Unix or Linux.

References

[1] Henrik R. Andersen. Partial Model Checking. In Proceedings of LICS95 (pp.398-407), IEEE Computer Society Press.

Games with the Edinburgh Concurrency Workbench

Perdita Stevens
Laboratory for Foundations of Computer Science University of Edinburgh

The Edinburgh Concurrency Workbench is a powerful and widely used tool for the analysis of concurrent systems defined in CCS (and variants). For example, it allows the user to check whether two processes are bisimilar, or whether a process satisfies a given modal mu-calculus formula. A major problem for users of the CWB, however, has been a lack of help from the CWB for the task of understanding *why* a process has certain properties. This has been a handicap both for students, trying to understand the concepts involved, and for experienced specifiers, trying to debug their specifications.

An interesting recent development in concurrency theory is the discovery of games which characterise equivalences or the satisfaction of modal mu-calculus properties. There is, for example, a rather simple two player game starting from a pair of processes, such that the player who starts has a winning strategy for the game exactly when the processes are not bisimilar. If the processes are bisimilar, then the other player will have a winning strategy. Moreover, the strategy itself exhibits a bisimulation, or the inevitable failure of the bisimulation conditions, as the case may be. Similar remarks hold for model-checking, although the game involved is more complex [Sti].

We have implemented the associated bisimulation and model-checking algorithms in the CWB, which can now calculate the appropriate winning strategy. Having found the strategy, we can use it to give feedback to the user which is much more useful than the old "true" or "false" answers (or even than modal distinguishing formulae). For example, a user who wrongly expected two processes to be bisimilar can now investigate why the answer was not as expected by actually playing the bisimulation game against the CWB. The CWB can be instructed to play the winning strategy against the user, who will certainly lose. We believe that by playing a succession of games against the CWB, and observing how and why these games are lost, the user will be able to come to a good understanding of just why the processes are not bisimilar. Of course, the information can be used in a variety of different ways; for example, the user can choose simply to examine a record of the whole strategy.

The motivation for adding these facilities to the CWB was mainly to provide this extra feedback to the user. It is, however, interesting to notice that the new implementations are (in practice, on real examples) considerably faster than the older ones. In the case of model-checking this difference is dramatic; for example, we have model-checking problems which, using the old optimised tableau-based implementation, had never finished even after weeks on our main compute-server, which are now solved in a few minutes on the same server. We hope that an added advantage of using game-based algorithms will be that they will extend in useful ways to value-passing processes; the clarity of the underlying basic theory seems helpful in thinking about the complications introduced by values.

We should like to demonstrate (at least):

- playing losing model-checking and bisimulation games against the CWB;
- the algorithms running in "answer only" mode;
- a prototype of a graphical interface to these capabilities.

References

[1] Stirling, C.,
Local model checking games,
Lecture Notes in Computer Science, **962**, 1-11, 1995.

`prod` 3.1
An Advanced Tool for Efficient Reachability Analysis

Kimmo Varpaaniemi, Johan Lilius and Keijo Heljanko

University of Technology, Digital Systems Laboratory

Otakaari 1, FIN-02150 Espoo, Finland

Kimmo.Varpaaniemi@hut.fi, Johan.Lilius@hut.fi, Keijo.Heljanko@hut.fi

The Pr/T-net reachability analysis tool `prod` [VHHP95], developed at the Digital Systems Laboratory at Helsinki University of Technology, implements several methods for efficient reachability analysis.

`prod` is a command-line driven tool, ie. the user interacts with the tool from a Unix-shell. Currently no graphical user-interface exists. `prod` supports the following reduced reachability graph generation methods that may also be combined:

- The *stubborn sets method* is based on the idea that for the verification of certain properties in the net it is unnecessary to generate all possible interleavings of independent transitions. Reachability graph generation using the stubborn set methods proceeds by calculating at each marking a stubborn set and the firing only enabled transitions from this set. In `prod` the calculation of the stubborn set can be done with two algorithms: (i) the incremental algorithm, and (ii) the deletion algorithm.
- The *CFFD preserving stubborn sets method* is a version of the stubborn set method that preserves the *chaos-free failures divergences* equivalence.
- *Sleep sets* are an alternative method that also utilizes the independence of transitions to reduce the reachability graph. Sometimes sleep sets used in conjunction with stubborn sets can increase the reduction.
- *Symmetries* are equivalence classes of markings. The calculation of symmetries can be very time-consuming, but it is well known that symmetries together with stubborn set can reduce the reachability graph radically.

`prod` supports verification of CTL properties from the reachability graph, and *on-the-fly verification* of LTL-properties using the different versions of the stubborn set algorithm.

`prod` is and has been used in several academic and industrial projects at the Digital system laboratory.

The analysis of the FSR: The Frame Synchronized Ring (FSR) is a high speed parallel data bus designed for high-throughput applications like ATM-switches. The following properties of the Medium Access Control Algorithm of the FSR were analyzed: (i) deadlock freeness, (ii) fairness, (iii) maximal waiting times.

The Emma project: The aim of the Emma (Extendible Multi-Method Analyzer) project is to develop a dynamic analyzer for TNSDL programs (TeleNokia SDL), which supports the detection of some errors related to parallelism and the investigation of their causes. The Emma analyzer is designed on top of Nokia's TNSDL translator and `prod` The user of Emma is working only on the level of TNSDL.

Teaching: `prod` has been used successfully as a teaching aid in a course on Parallel and Distributed systems. The students do a verification project in groups of 2 or 3. The course is taken by approximately 100 students yearly.

Other applications: Several other substantial models have been built and analyzed with `prod` including X.25, a video on demand system, an authentication protocol, and a specification of a simple telephone exchange (YXA).

`prod` is available over the Internet (during the last year `prod` has been downloaded to over 200 sites worldwide). The sources can be retrieved from `http://topos.hut.fi/~petrinet/prod.html`.

A bibliography with relevant literature about `prod`, and other Petri net research done at the Digital systems laboratory is available as `http://topos.hut.fi/~petrinet/publications.html`

References

[VHHP95] Kimmo Varpaaniemi, Jaakko Halme, Kari Hiekkanen, and Tino Pyssysalo. PROD reference manual. Technical Report B13, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, August 1995.