Verification of Infinite-state Systems

02.04.–07.04.2000

organized by

Ahmed Bouajjani, Javier Esparza

The development of our modern societies needs more and more involvement of computers in managing highly complex and (safety-)critical tasks, e.g., in telecommunication, chemical and physical process control, transportation systems, etc. It is essential to be able to produce reliable hardware and software systems, since any erroneous behaviour can have catastrophic (economical and human) consequences. This requires rigorous methods and techniques to conceive, analyze and validate these systems.
The verification problem consists in checking whether a system satisfies its specification. During the two last decades, significant achievements have been obtained in the case of finite-state systems (systems with finitely many states). One of the main actual challenges in the domain of automated verification is the conception of methods and tools allowing to deal with verification problems beyond the finite-state framework.
Such problems rise naturally as soon as we consider aspects like:

- real-time constraints: timed and hybrid systems,
- unbounded discrete data structures: counters, fifo-channels, stacks, etc.
- parametric reasoning about families of systems, e.g., networks of processes,
- process mobility, dynamic creation and destruction of processes (dynamic modification of the communication structure).

In the last two years the specification and verification of infinite-state systems have attracted the attention of more and more researchers belonging to a very broad range of research communities. Both process algebras (or term rewriting systems) and automata (or finite control machines) are being used as specification formalisms. Verification problems can be reduced to checking behavioural equivalence or implementation (simulation) relations, or to checking the satisfaction of properties described in temporal logics or fixpoint calculi (model checking problems). Verification methods can be deductive (based on the use of theorem provers), or algorithmic (based on decision or semi-decision procedures). Algorithmic methods can be based on fixpoint theory, automata theory or (constrained) logic programming.

Recent work has shown that different techniques can be combined with sometimes spectacular results. As a result, it is being acknowledged that only a combination of techniques can lead to methods and tools widely used in practice. The aim of this meeting is to bring together researchers working on different research fields in order to make a synthesis on the state of the art and evaluate the potential of combined methods. Concrete questions to be addressed are:

- Which results in logic, automata theory, rewriting systems, etc. are applicable to automatic verification?
- How should deductive and algorithmic techniques be combined?
- Which are the right techniques to deal with abstraction?
- Which are the most promising application fields (mobile systems, cryptographic protocols, static program analysis ...), and which are the most appropriate models, specification formalisms, and verification techniques to deal with them?

Contents

1 Regular Model Checking

Bengt Jonsson

Joint work with Marcus Nilsson, Parosh Abdulla, Ahmed Bouajjani, and Tayssir Touili.
We present *regular model checking*, a framework for algorithmic verification of infinite-state systems with, e.g., queues, stacks, integers, or a parameterized linear topology. States are represented by strings over a finite alphabet and the transition relation by a regular length-preserving relation on strings. Major problems in the verification of parameterized and infinite-state systems is to compute the set of states that are reachable from some set of initial state, or to compute the transitive closure of the transition relation. We present a direct automata-theoretic construction for computing the transitive closure, which uses elementary techniques adapted to our framework. The technique is incomplete in general, but we give sufficient conditions under which it works. The work has been presented in [1,2].

References

[1] B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite -state systems. In *TACAS 00*, LNCS, 2000.
[2] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV 00*, LNCS, 2000.

2 Model Checking Context-Free Properties

Markus Müller-Olm

Imperative programming languages typically offer a sequential composition operator which allows the straightforward specification of behavior proceeding in successive phases. Similar operators are provided by interval temporal logics, where they are called *chop*-operators. Important examples are Moszkowski's Interval Temporal Logic ITL [7] and the Duration Calculus [10]. Up to our knowledge, however, no point-based temporal logic and, in particular, no branching-time logic with a chop operator has been proposed up to now.
In the talk, which was based on [8], we presented some results on a logic called FLC (Fixpoint Logic with Chop) that extends the modal mu-calculus [6] by a chop operator ';' and *termination formulae* 'term'. In order to explain the meaning of the new types of formulas in a compositional way we utilize a 'second-order' interpretation of formulae. While (closed) formulae of usual temporal logics are

interpreted by sets of states, i.e. represent *predicates*, we interpret formulae by mappings from states to states, i.e. by *predicate transformers*. A similar idea has been used by Burkart and Steffen [2] in a model checking procedure for modal mu-calculus formulae and context-free processes. However, they rely on a second-order interpretation of *states* as property transformers, while we use here a second-order interpretation of *formulae*.

FLC-based model checking is an interesting alternative to modal mu-calculus-based model checking as FLC is strictly more expressive but still decidable for finite-state processes. We showed that FLC is undecidable for context-free processes, that satisfiability (and thus also validity) is undecidable, and that the logic does not have the finite model property. These results are inferred from the existence of formulae characterizing context-free processes up to bisimulation and simulation. In the talk we also discussed some connections to mu-calculus based model-checking of context-free and push-down processes [1,2,9] and to propositional dynamic logics with context-free programs [3,4,5]. We also indicated that the logic might be of use for casting interprocedural data-flow analysis as a model-checking problem on the so-called *supergraph* of the program in question.

References

[1] O. Burkart and B. Steffen. Model checking for context-free processes. In W. R. Cleaveland, editor, *CONCUR '92*, LNCS 630, pages 123–137. Springer-Verlag, 1992.

[2] O. Burkart and B. Steffen. Model checking the full modal mu-calculus for infinite sequential processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP '97*, LNCS 1256, pages 419–429. Springer-Verlag, 1997.

[3] D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *J. Comput. Syst. Sci.*, 26:222–243, 1983.

[4] David Harel and Danny Raz. Deciding properties of nonregular programs. *SIAM J. Comput.*, 22(4):857–874, 1993.

[5] T. Koren and A. Pnueli. There exist decidable context-free propositional dynamic logics. In *Logics of Programs*, LNCS 164, pages 290–312. Springer-Verlag, 1983.

[6] D. Kozen. Results on the propositional mu-calculus. *TCS*, 27:333–354, 1983.

[7] Ben Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[8] Markus Müller-Olm. A modal fixpoint logic with chop. In Christoph Meinel and Sophie Tison, editors, *STACS'99*, LNCS 1563, pages 510–520. Springer, 1999.

[9] I. Walukiewicz. Pushdown processes: Games and model-checking. In R. Alur and T. Henzinger, editors, *CAV'96*, LNCS 1102, pages 62–74. Springer, 1996.

[10] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

3 Alternating Automata Based Decision Procedures for Circuit Families

David Basin

(Following is joint work with Felix Klaedtke and Abdelwaheb Ayari)
We show how alternating automata provide decision procedures for the equivalence of inductively defined Boolean functions that are useful for reasoning about parameterized families of circuits. We use alternating word automata to formalize families of linearly structured circuits and alternating tree automata to formalize families of tree structured circuits. We provide complexity bounds and show how our decision procedures can be implemented using BDDs. In comparison to previous work, our approach is simpler, yields better complexity bounds, and, in the case of tree structured families, is more general.

4 Verification and Synthesis for Timed and Hybrid Systems

Oded Maler

In this talk I first explain the motivation for hybrid (discrete-continuous) systems and the main differences in the mathematical models underlying discrete and continuous dynamical systems. More details about a unifying model for discrete and continuous systems can be found in `http://www-verimag.imag.fr/~maler/cabst.html#unif`.
If one wants to extend verification methodology to treat such systems, the problem of (approximately) computing reachable states for continuous (and nondeterministic) systems should be solved. I describe an experimental system called **d/dt**, which combines numerical approximation with computational geometric techniques to verify and synthesize hybrid automata with linear differential inclusions. More about this work can be found in `http://www-verimag.imag.fr/~maler/jabst.html#procieee`.

5 Generic Type Systems for Mobile Processes

Barbara König

We introduce a generic type system for the synchronous polyadic π-calculus, allowing us to mechanize the analysis of input/output capabilities of mobile processes. The parameter of the generic type system is a lattice-ordered monoid, the elements of which are used to describe the capabilities of channels with respect to their input/output-behaviour. The type system can be instantiated in order to check process properties such as upper and lower bounds on the number of active channels and confluence.

6 Parametric Linear Temporal Logic

Salvatore La Torre

The results presented in the talk are mainly taken from [1].
Model checking has become a central methodology for automated verification of reactive systems. We extend the standard model checking paradigm of linear temporal logic, LTL, to a "model measuring" paradigm where one can obtain more quantitative information beyond a "Yes/No" answer. For this purpose, we define a *parametric temporal logic*, PLTL, which allows statements such as "a request p is followed in at most x steps by a response q", where x is a free variable. Given a formula $\phi(x_1, \ldots, x_k)$ of PLTL and a system model K, we are interested in $V(K, \phi)$, the set of valuations of $x_1, \ldots, x_k$ under which the system K satisfies the property ϕ. We show algorithms to check for emptiness, finiteness, and universality of $V(K, \phi)$, and we also show how to find valuations which satisfy various optimality criteria. The complexity of these algorithms is essentially that of ordinary LTL model checking: PSPACE in the formula size and polynomial-time in the size of the model. When all parameterized operators in the formula are of the same polarity, we can compute an explicit representation of $V(K, \phi)$ by symbolic constraints on parameter values. We have defined our logic with two restrictions: we do not allow equality subscripts (e.g. $\Diamond_{=x}$), and the same parameter cannot appear in association with two operators with different polarities (e.g., both $\Diamond_{\leq x}$ and $\Box_{\leq x}$). We show that our logic lies at the threshold of decidability for parametric temporal logics, in the sense that removing either of these two restrictions leads to an undecidable "model measuring" problem.

References

[1] R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric Temporal Logic for "Model Measuring". *Proceedings of ICALP'99*, LNCS 1644, pages 159–168. Springer-Verlag, 1999.

7 Automatic Abstraction by Syntactic Program Transformations

Kedar Namjoshi

(joint work with Bob Kurshan, Bell Laboratories)

We present an algorithm that constructs a finite state "abstract" program from a given, possibly infinite state, "concrete" program by means of a *syntactic* program transformation. Starting with an initial set of predicates from a specification, the algorithm iteratively computes the predicates required for the abstraction relative to that specification. These predicates are represented by boolean variables in the abstract program. We show that the method is *sound*, in that the abstract program is always guaranteed to simulate the original. We also show that the method is *complete*, in that, if the concrete program has a finite abstraction with respect to simulation (bisimulation) equivalence, the algorithm can produce a finite simulation-equivalent (bisimulation-equivalent) abstract program. Syntactic abstraction has two key advantages: it can be applied to infinite state programs or programs with large data paths, and it permits the effective application of other reduction methods for model checking. We show that our method generalizes several known algorithms for analyzing syntactically restricted, data-insensitive programs.

8 Unfoldings of Infinite State Systems

S. Purushothaman Iyer

Net unfoldings have attracted much attention as a powerful technique for combating state space explosion in model checking. The method has been applied to verification of 1-safe (finite) Petri nets, and more recently also to other classes of finite-state systems such as synchronous products of finite transition systems. We show how unfoldings can be extended to the context of infinite-state Petri nets. More precisely, we apply unfoldings to get an efficient symbolic algorithm for checking safety properties of unbounded Petri nets. We also discussed extensions of the notion of unfoldings to other classes of infinite state systems.

9 Provability in a Logic for Concurrent Objects is Well-structured!

Giorgio Delzanno

In the early 90's, Andreoli and Pareschi [2,3,4,5] introduced a logic for specifying concurrent objects. The logic captures the main features of object-oriented languages like classes, inheritance, objects with state, and dynamic updates. The logic provides two form of concurrency: OR-concurrency to model the internal evolution of objects, and AND-concurrency to model the interaction among objects. Furthermore, asynchronous communication is achieved via ask and tell operations on a common blackboard. The operational semantics of LO is given via a goal-driven sequent calculi. In this context, a proof is viewed as a goal-driven computation and an LO-program as a set of multiset rewriting rules. LO programs specify infinite-state concurrent systems. The following question naturally arises.

Can we use results developed for infinite-state verification to this special class of infinite-state systems?

To answer the question, we show that provability in the logic LO is a well-structured relation. Following from general results of Abdulla-Cerans-Jonsson-Tsay [1] and Finkel-Schnoebelen [7], this property implies that it is possible to construct effectively a finite representation of the set of all logical consequences of an LO-program P, i.e., of the set of goals that are provable in P. This construction is achieved building a proof bottom-up starting from the axioms of the logic. This proof-search strategy can also be viewed as a method for the bottom-up evaluation of LO-programs.

The connection between the decidability results for infinite-state systems and provability in LO is interesting for at least two reasons. Firstly, it can be considered as a first step towards the application of verification techniques like model checking to concurrent object-oriented systems. Secondly, LO is based on a fragment of linear logic [8], a constructive logic obtained as a sub-structural refinement of classical logic. Thus, our result on well-structuredness of LO-provability can be considered as a potentially useful technique to find new proof search strategies for larger fragments of linear logic.

The full paper is available as technical report [6].

References

[1] P. A. Abdulla, K. Cerans, B. Jonsson and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In Proc. of LICS 96, pages 313–321, 1996.

[2] J.M. Andreoli. Logic Programming with focusing proofs in linear logic. Journal of Logic and Computation, 2(3):297–347, 1992.

[3] J.M. Andreoli. Coordination in LO. In Coordination Programming: mechanisms, models and semantics. Imperial College Press, London, 1996.

[4] J.M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with Built-In Inheritance. In Proc. of ICLP 90, pages 495–510, 1990.

[5] J.M. Andreoli and R. Pareschi. Communication as Fair Distribution of Knowledge. In Proc. of OOPSLA '91, pages 212-229, 1991.

[6] M. Bozzano, G. Delzanno, and M. Martelli. A Bottom-up Semantics from LO - Preliminary Results - Technical Report TR-00-06 DISI Univ. di Genova, March 2000. Available at `http://www.disi.unige.it/person/DelzannoG/papers`.

[7] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! Technical Report LSV-98-4, Laboratoire Specification et Verification, ENS Cachan, April 1998.

[8] J. Y. Girard. Linear logic. Theoretical Computer Science, 50:1–102, 1987.

10 On the Complexity of Bisimulation Equivalence

Richard Mayr

We give an overview on some recent results on the complexity of deciding bisimulation equivalence. Weak and strong bisimilarity between context-free processes and finite-state processes is decidable in polynomial time [1], in spite of the fact that there are cases where the bisimulation game must have exponential length. On the other hand, weak bisimilarity between pushdown automata and finite automata is PSPACE-hard, even for a small fixed finite automaton. The same problem for strong bisimulation is also PSPACE-hard, but only polynomial in the size of the pushdown automaton [3]. Finally, we show that strong bisimilarity of Basic Parallel Processes (BPP, also called communication-free Petri nets) is co-NP hard, while weak bisimilarity of BPP is Π_2^p-hard in the polynomial hierarchy [2].

References

[1] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*. Springer Verlag, 1999.

[2] R. Mayr. On the complexity of bisimulation problems for Basic Parallel Processes. In *Proc. of ICALP'2000*, *LNCS*. Springer Verlag, 2000, To appear.
[3] R. Mayr. On the complexity of bisimulation problems for pushdown automata. In *Proc. of IFIP TCS'2000*, *LNCS*. Springer Verlag, 2000, To appear.

11 Simulation and bisimulation over one-counter processes

Antonín Kučera

We present an overview of recent decidability and complexity results for checking bisimulation and simulation equivalence with processes of pushdown automata, one-counter automata, and one-counter nets. First we show that there is a general relationship between simulation problems and their bisimulation counterparts, which is valid (but generally not effective) for arbitrary finitely-branching processes. Then we show that this method can be effectively applied to processes of one-counter nets (which are 'weak' one-counter automata without an explicit test for zero). In this way we demonstrate that certain simulation problems (e.g., equivalence, regularity) for one-counter nets can be reduced to the corresponding bisimulation problems for one-counter automata and thus they are decidable.
Next, we present a summary of results concerning the complexity of simulation and bisimulation equivalence between pushdown (or one-counter) processes and finite-state processes. Some of the bounds are already tight (in particular, we show that bisimilarity between pushdown processes and finite-state processes is PSPACE-complete, while simulation equivalence is EXPTIME-complete).

12 Deciding first-order non-regular Properties of PA-Processes

Denis Lugiez, Philippe Schnoebelen

This paper gives decidability results for several first-order logic based upon the reachability predicate in Process Algebras, i.e. the model of parallelism with sequential and parallel composition, and process rules $X \xrightarrow{a} t$. The main idea is to show that the reachability relation is a recognizable relation. More precisely, this means that the set of terms $s \times t$ such that $s \xrightarrow{*} t$ is a regular set on the product

alphabet. Classical tree automata constructions are used to show that recognizable relations are closed under conjunction, disjunction, products, projection and cylindrification. Therefore the first order theory of recognizable relations is decidable. This classical closure properties and of the recognizability of the relation $\xrightarrow{*}$ imply that the first-order logic built from the classical connectives, quantifiers, and atoms $s \xrightarrow{*} t$ (as well as any other recognizable relation) is decidable.
We show similar results for constrained reachability (some constraints are set on paths) for several kind of constraints. These constraints are either time constraints (like reach t from s in less than 5 time units) or Presburger's arithmetic formula on the number of actions performed along a derivation (like reach t from s by performing as many actions a as actions b for s in a regular set of initial states and t belonging to a regular set of final states). The decision procedure for the first type of constraints relies on the notion of decomposable predicates. For the second type of constraints, we show that the resulting first-order logic is undecidable but that interesting fragments are decidable. In this case the decision procedure involve automata where a cost is associated to transition rules and we show how to embed these costs into a classical tree automaton or to compute with these costs.

13 Decision procedures for pushdown automata with ϵ-transitions

Colin Stirling

We examined decision procedures for pushdown automata with ϵ-transitions. First their graphs were introduced. We examined two transformations, a "collapsing" transformation and a determinatization transformation. We showed how these are very useful for understanding the DPDA equivalence problem.

14 Model Checking Pushdown Graphs

Igor Walukiewicz

A pushdown graph is the graph of configurations of a pushdown automaton. The model checking problem for some logic is: given a pushdown automaton and a formula α of the logic decide if α holds in the initial configuration of the pushdown graph.
First, we recall the EXPTIME-completeness result for model checking of linear time logics (LTL, mu-calculus). Next, we recollect the EXPTIME-completeness

result for the alternating reachability problem. Finally, we show a new result which is PSPACE completeness of the model checking problems for both EF-logic and CTL. EF-logic is a restriction of CTL to the formulas given by the grammar:

$$P \mid \neg\alpha \mid \alpha \wedge \beta \mid E \circ \alpha \mid EF\alpha$$

It turns out that there are two reasons for the model checking problem to be EXPTIME-complete. One is the ability to compare all pairs of adjacent intervals of size n (the case of linear time model checking). The other is unbounded alternation. Once both these features disappear the complexity becomes PSPACE.

15 Acceleration of infinite loops: very well-structured transition systems

Alain Finkel

We present a survey on the 10 different models related to well-structured transition systems (WSTS).
The importance of the choice of monotony (also called compatibility) and quasi-ordering is shown. Some new undecidability results on WSTS may help us to better understand the differences between models: in particular, we prove that boundedness and computation of a coverability set are both undecidable for WSTS (as defined in [1] and [2]). However, a coverability set has some advantages: it allows to decide some liveness properties, to build a coverability graph which simulates the original labelled transition system. To be able to define, to effectively construct and to insure termination of an algorithm computing a such coverability graph, we propose a new model, called very well-structured transition systems, which have these desired properties. Finally, we give some insights on how it is possible to accelerate infinite loops for reaching a cover of the reachability set.

References

[1] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Tsay Yih-Kuen. Algorithmic Analysis of Programs with Well Quasi-Ordered Domains. To appear in the Journal of Information and Computation.

[2] A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere!. Research Report LSV-98-4, Lab. Specification and Verification, ENS de Cachan, Cachan, France, April 1998. 29 pages. Theoretical Computer Science, 2000. To appear.

16 Efficient Algorithms for Model-Checking Pushdown Systems

Stefan Schwoon

The talk covers joint work with Javier Esparza, David Hansel, and Peter Rossmanith. The related paper is available as a technical report [1].
We study the model-checking problem for linear time logics on pushdown systems. We do not treat pushdown systems as language acceptors; instead, we regard them as infinite transition systems whose configurations are pairs of a control location and a stack content. Our motivation lies in the fact that pushdown systems can model sequential programs with procedures. We develop a strategy for solving the global model-checking problem, i.e. computing the set of all configurations which violate a given formula ϕ, and show that the problem can be solved in $O(|\mathbf{P}|^3|\mathbf{B}|^3)$ time and $O(|\mathbf{P}|^2|\mathbf{B}|^2)$ space if $|\mathbf{P}|$ is the size of the pushdown system and $|\mathbf{B}|$ is the size of a Büchi automaton corresponding to $\neg\phi$. If $\mathbf{P}$ is derived from a set of interprocedural flowgraphs, the time and space requirements become linear in $|\mathbf{P}|$.

References

[1] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. Technical Report TUM-I0002, Jan. 2000. Available at `http://www7.in.tum.de/gruppen/theorie/publications/2000.shtml`.

17 Heterogeneous Finite-state Representation Systems

Bernard Boigelot

We address the general problem of computing an exact representation of the set of reachable configurations of a system with finite control and unbounded data. In order to solve, even partially, this problem, one needs a representation system for potentially infinite sets of values, as well as a computation method for generating infinite sets of reachable states in finite time. We present a general approach to solving the former problem, based on finite-state automata recognizing encodings of states. The latter problem is solved thanks to meta-transitions, which are objects equivalent to iterations of cycles in the control graph of the system being analyzed. After presenting the results of a practical application of

this approach, we move to its main restriction which is that it can only be applied to systems whose data part is expressed with respect to this problem, we describe some restricted classes of systems for which one can lift the results obtained for individual data domains so as to make it possible to analyze systems with an heterogeneous data part. We then conclude with a list of open questions.

18 Temporal logics, flatness and counters

Hubert Comon

An automaton (or any finite state machine) is *flat* if there is at most one loop on each state. When such a machine is in a given state, it may stay in the same state for some time, performing some actions and either stay here forever or eventually move to a strictly smaller state.
We show that flatness is an important hypothesis: many problems which are undecidable in the general case, become decidable for flat machines. For instance, the transitive closure of the transition relation of a counter machine can be effectively defined in some decidable theory when the machine is flat (which is not the case otherwise).
Then we define *flat temporal logics* in the linear case and prove a correspondence between flat logics and flat automata. Such logics include, as atomic formulas, expressions which say something about the counters or the relations between counters, for instance $y' = x + 2$ says that the value of counter y after the transition is equal to the value of counter x before the transition plus two. Flatness imposes some restrictions on the left-nested "until" constructions. Because of the correspondence with flat automata, the satisfiability is decidable for such logics. Unfortunately, the logic is not closed under negation and it turns out that the validity problem is undecidable.
Finally, we extend the results to *piecewise flat logics* which include both flat logic and Propositional Linear Time Logic. We conclude by asking a question: to which extent is it possible to express parametrized properties within such a logic?
Most of the material of this talk is described in an abstract paper which is going to appear in the proceedings of Computer Science Logic 2000.

19 On the reachability problem in cryptographic protocols

Roberto Amadio

This paper is joint work with Denis Lugiez [1].
We study the verification of secrecy and authenticity properties for cryptographic protocols which rely on symmetric shared keys. The verification can be reduced to check whether a certain parallel program which models the protocol and the specification can reach an erroneous state while interacting with the environment. Assuming finite principals, we present a simple decision procedure for the reachability problem which is based on a 'symbolic' reduction system.

References

[1] R. Amadio, D. Lugiez. On the reachability problem in cryptographic protocols. INRIA Research Report, April 2000. Available at `http://www.cmi.univ-mrs.fr/~amadio/papers.html`.

20 Proof-Checking Bisimulations

Christine Röckl

In recent years, theorem proving has gained considerable relevance in the field of reactive systems. We survey how general-purpose theorem proving can be applied to tell whether two processes — one modelling the system, and the other the specification — are weakly bisimilar.
Applying a general-purpose theorem prover like Isabelle/HOL, the user specifies the goal he/she intends to prove, and then derives it in interaction with the prover. When formalizing a proof, the user can choose between various tactics, ranging from classical reasoning over simplification (i.e., algebraic reasoning) to fully automatic tactics. After a goal has been proven, it can be stored in the prover's theorem database as a new theorem to be referenced in further proofs.
There exist three main ways of showing that two processes are bisimilar: (1) exhibit a relation containing as a pair the two processes, and prove that it is a bisimulation; (2) use algebraic rules to transform one process into the other; and, (3) incrementally compute a bisimulation following the coinduction proof technique. The first and third proof methodologies can both be supported by "up to" techniques.

All three approaches have been applied in examples and case studies, formalized in theorem provers. The first technique deserves special interest: it follows a natural proof style, requiring only good insight in the behaviour of the system and its specification; also, the conceptually difficult part (exhibit the relation) and the simple but often tedious part (prove that it is a bisimulation) are clearly separated; finally, it works for all kinds of operational models.
The suitability of the approach wrt. theorem proving can be demonstrated by a range of case studies. We discuss infinite-state and parameterized (in the number of components) formalizations of communication and cache protocols: We prove that the Alternating Bit Protocol is observationally equivalent to a one-place buffer (length of proof, approx. 800 lines), and that a specification of the Sliding Window Protocol with window-size n is observationally equivalent to an n-place buffer (approx. 600 lines), and that a simple write-invalidate cache protocol keeps the cache coherent (approx. 1000 lines, applying a weak bisimulation up to expansion).

21 Semilinear witnesses for decidability

Petr Jančar

In the talk, the main ideas of two decidability proofs were explained. One concerns the boundedness problem for Petri nets with two resetable places, the other the simulation preorder for one-counter nets. Both have a common ingredient, namely exhibiting a 'semilinear' structure which implies the decidability.

22 On infinite-state systems

Didier Caucal

We present two general families of infinite-state systems over words: the family of recognizable graphs [1] and the family of rational graphs defined by Morvan [3]. A recognizable graph is a finite union of elementary graphs of the form

$$(U \xrightarrow{a} V)W = \{\, uw \xrightarrow{a} vw \mid u \in U, v \in V, w \in W \,\}$$

where U, V, W are rational languages. The recognizable graphs have a decidable monadic theory and their traces are the context-free languages. This family contains strictly the transition graphs of pushdown automata investigated by Muller and Schupp [4] and the equational graphs defined by Courcelle [2].
A rational graph is a graph such that for each label, its set of transitions is a rational relation. This family is very general: it contains the recognizable graphs

and the transition graphs of various systems (like the Petri nets). In particular, the first order theory of a rational graph is undecidable.
We show that these two families of graphs can be expressed naturally using the transition graphs of word rewriting systems. The recognizable graphs are the transition graphs of the prefix (or suffix) systems where a prefix (resp. suffix) system is a system such that a left hand side and a right hand side are overlapping only by prefix (resp. suffix). Furthermore, the rational graphs are the transition graphs of the right (or left) systems where a right (resp. left) system is a system such that a left hand side overlaps a right hand side only on the right (resp. left). Finally, we show that these four families of word rewriting systems are the maximal families of systems defined by overlapping between the left hand sides and the right hand sides, and such that the derivation (transitive closure by composition of the rewriting) is a rational relation.

References

[1] D. Caucal. On transition graphs having a decidable monadic theory. LNCS 1099, pages 194–205, 1996.

[2] B. Courcelle. Graph rewriting: an algebraic and logic approach. Handbook of TCS, Volume B, Elsevier, pages 193–242, 1990.

[3] C. Morvan. On Rational Graphs. Proceedings of FOSSACS 2000, LNCS 1784, pages 252–266, 2000.

[4] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. Theoretical Computer Science 37, pages 339–456, 1985.

23 Context–Representable Processes

Thomas Noll

(Joint work with Mads Dam, Swedish Institute of Computer Science.)
We introduce a new class of processes in the spirit of process rewrite systems [4,6]. Our investigation is motivated by a practical application of formal methods for handling infinite–state systems: the verification of distributed systems implemented in the Erlang programming language [3]. For this purpose a very general and powerful proof system has been developed [1,2]. It employs a tableau–based theorem–proving approach to establish system properties specified in a modal μ–calculus with Erlang–specific extensions.
However, due to the undecidability of the underlying model–checking problem, a proof generally requires a high degree of user interaction. Therefore an intimate knowledge of the theoretical foundations is indispensable to employ the system

in a meaningful way. It is thus our aim to increase its usability by providing more automatic support, in particular when dealing with fixed–point properties, which are handled by means of a rather complex discharge rule in the proof system.

One possible approach is to represent the states of the system under consideration in a symbolical way. For example, a CCS specification of a simple counter process is given as follows (in our examples we employ the CCS process algebra instead of Erlang to allow for more concise notations):

$$C = up.(C \parallel down.\mathsf{nil}).$$

That is, the execution of an *up* action adds a parallel component process of the form $down.\mathsf{nil}$ which can be later removed by a *down* action. Thus the states of the corresponding transition system can be symbolically represented by a "polynomial" of the form $C \parallel (down.\mathsf{nil})^n$. Here the exponent n ranges over the natural numbers and can be used as an inductive parameter for proving fixed–point properties.
However it is easy to find examples that can not be covered by such a polynomial form. In many data structures like e.g. stacks it is essential that the state representation reflects the order in which actions have occurred, which contradicts the commutativity that is implicitly assumed above. It is our idea to achieve this by considering every state of a system as being constructed by the iterated application of certain process context mappings to some initial state.
For instance, a stack over some set X is given by

$$S = \sum_{x \in X} push_x.(S[done \mapsto d] \parallel d.pop_x.S) \setminus \{d\} + \overline{done}.\mathsf{nil}.$$

It is straightforward to see that every state of the corresponding transition system can be obtained by iteratively applying context mappings of the form

$$c_x[\xi] = (\xi[done \mapsto d] \parallel d.pop_x.S) \setminus \{d\}$$

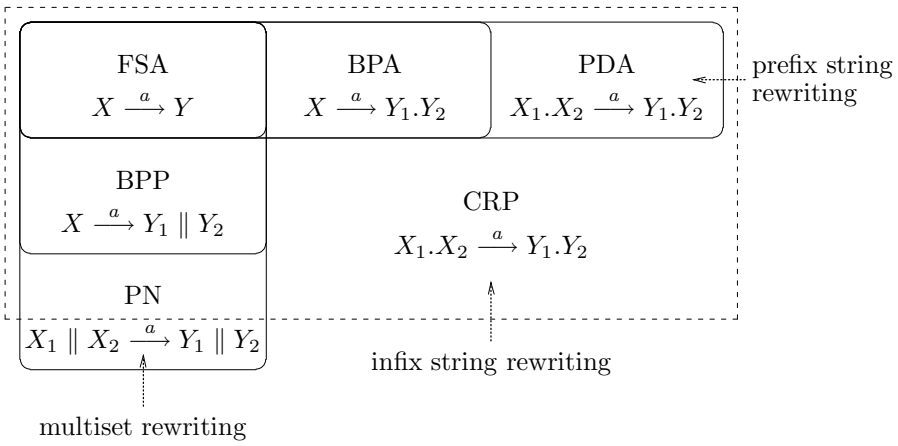
to the initial state S. In order to identify a state of the system it hence suffices to give the corresponding sequence of context indices $x \in X$, which coincides with the current stack content. A transition is represented now by a string rewriting operation on index words. Thus a transition system is described by a (finite) collection of labeled rewrite rules together with an initial state. In other words, we are dealing with process rewrite systems.

In this setting, a queue process can be specified by the rules

$$Q \xrightarrow{in_x} LO_xR \qquad L \xrightarrow{in_x} LO_x \qquad O_xR \xrightarrow{out_x} R.$$

Here L and R act as left and right markers indicating the positions where items are enqueued and dequeued, respectively.

The class of processes which can be described by such rewrite systems is called *context–representable processes (CRP)*; it generalizes the well–known class PDA of pushdown transition systems [5] in the following sense. While the syntactic form of the rewriting rules is identical in both cases, PDA rules are to be applied only in the prefix position of the current state. In contrast, the CRP interpretation allows rewriting steps at arbitrary positions. Thus we are dealing with arbitrary string rewriting rather than prefix string rewriting systems.
It is possible to show that, with respect to bisimulation, CRP subsumes the sequential classes (FSA, BPA, PDA) of the known hierarchy of process rewrite systems [4,6] as well as the class BPP of Basic Parallel Processes. Regarding Petri nets, we propose a candidate system which might be used to show that there are Petri net transition systems that are not bisimilar to any CRP system, thus establishing the incomparability of CRP and Petri nets with respect to bisimulation. The relationships are illustrated by the following diagram.



References

[1] T. Arts and M. Dam. Verifying a distributed database lookup manager written in Erlang. In *FM'99—Formal Methods, Volume I*, volume 1708 of *Lecture Notes in Computer Science*, pages 682–700. Springer, 1999.

[2] T. Arts, M. Dam, L. Fredlund, and D. Gurov. System description: Verification of distributed Erlang programs. In *Proc. of CADE'98*, volume 1421 of *LNAI*, pages 38–41, 1998.

[3] J.L. Armstrong, M.C. Williams, C. Wikström, and S.R. Virding. *Concurrent Programming in Erlang*. Prentice Hall, 2nd edition edition, 1995.

[4] O. Burkart and J. Esparza. More infinite results. *Bulletin of the European Association for Theoretical Computer Science*, 62:138–159, June 1997.

[5] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.

[6] F. Moller. A taxonomy of infinite state processes. *Electronic Notes in Theoretical Computer Science*, 18, June 1998.

24 Decidability of Reachability Problems for Classes of Two Counters Automata

Grégoire Sutre

We present a global and comprehensive view of the properties of subclasses of two counters automata for which counters are only accessed through the following operations: increment ($+1$), decrement (-1), reset ($c := 0$), transfer (the whole content of counter c is transfered into counter c'), and testing for zero. We first extend Hopcroft-Pansiot's result (an algorithm for computing a finite description of the semilinear set $post^*$) to two counters automata with only one test for zero (and one reset and one transfer operations). Then, we prove the semilinearity and the computability of pre^* for the subclass of 2 counters automata with one test for zero on c_1, two reset operations and one transfer from c_1 to c_2. By proving simulations between subclasses, we show that this subclass is the maximal class for which pre^* is semilinear and effectively computable. All the (effective) semilinearity results are obtained with the help of a new symbolic reachability tree algorithm for counter automata using an *Acceleration* function. When *Acceleration* has the so-called stability property, the constructed tree computes exactly the reachability set.

25 Abstracting WS1S Systems to Verify Parameterized Networks

Kai Baukus

The following is joint work with Y. Lakhnech, S. Bensalem, and K. Stahl.
Recently there has been much interest in the automatic and semi-automatic verification of parameterized networks, i.e., verification of a family of systems $\{\mathcal{P}_i \mid i \in \omega\}$, where each $\mathcal{P}_i$ is a network consisting of i finite-state processes. Apt and Kozen show in [2] that the verification of parameterized networks is undecidable. Nevertheless, automated and semi-automated methods for the verification of restricted classes of parameterized networks have been developed.

In [3] we first transform a given infinite family of networks of finite processes into a bisimilar single transition system whose variables are set variables and whose transitions are described in WS1S, the weak monadic second order logic of one successor. We call such systems WS1S transition systems.
The idea of representing sets of states of parameterized networks by regular languages is applied in [7], where additionally finite-state transducers are used to compute predecessors. The work presented in [1] extends the idea by considering the effect of applying infinitely often a transition that satisfies certain restrictions. Contrary, in our approach we do not try to compute the exact set of reachable states. We abstract the obtained WS1S transition system into a finite abstract system. The abstract system gives us an over-approximation of the set of reachable states, but also maintains some properties of the original control flow. These properties can be analyzed using model-checking techniques.
Since our abstraction is guaranteed to be conservative, i.e., the abstract system exhibits for every behavior of the WS1S system a corresponding abstract behavior, we are able to verify universal path quantified properties of the WS1S system. These properties also include liveness properties.
However, it is well known that an obstacle to the verification of liveness properties using abstraction is that often the abstract system contains cycles that do not correspond to paths in the concrete system. Therefore, we present an algorithm to add fairness conditions to the abstract system which are guaranteed to hold for the concrete system.
Moreover, in [4] we show how to deal with fairness requirements already given for the concrete system. Combining these techniques allows us to verify some interesting liveness properties of non-trivial parameterized networks.
The methods are implemented in a tool called PAX, that uses the decision procedures of MONA [5,6] to check the satisfiability of WS1S formulas. The first results obtained using our methods and PAX are very encouraging and can be found at `http://www.informatik.uni-kiel.de/~kba/pax/`.

References

[1] P.A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling Global Conditions in Parameterized System Verification. In N. Halbwachs and D. Peled, editors, *CAV '99*, volume 1633 of *LNCS*, pages 134–145. Springer, 1999.

[2] K. Apt and D. Kozen. Limits for Automatic Verification of Finit-State Concurrent Systems. *Information Processing Letters*, 22(6):307–309, 1986.

[3] K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In *TACAS 2000*. Springer, 2000.

[4] K. Baukus, Y. Lakhnech, and K. Stahl. Verifying Universal Properties of Parameterized Networks. Submitted to FTRTFT 2000, 2000.

[5] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic Second-Order Logic in Practice. In *TACAS '95*, volume 1019 of *LNCS*. Springer, 1996.

[6] N. Klarlund and A. Møller. MONA Version 1.3 User Manual. BRICS, 1998.

[7] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In O. Grumberg, editor, *Proceedings of CAV '97*, volume 1256 of *LNCS*, pages 424–435. Springer, 1997.

26 Symbolic techniques for parametric reasoning about counter and clock automata

Ahmed Bouajjani

We address the problem of automatic analysis of parametric counter and clock automata. We propose a semi-algorithmic approach based on using:

1. expressive symbolic representation structures called parametric DBM's, and
2. accurate extrapolation techniques allowing to speed up the reachability analysis and help its termination.

The extrapolation techniques we propose consist in guessing automatically the effect of iterating a control loop an arbitray number of times, and in checking that this guess is exact. Our approach can deal uniformly with systems that generate linear or nonlinear sets of configurations. We have implemented our techniques and experimented them on nontrivial examples such as a parametric timed version of the Bounded Retransmission Protocol.