Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Tools and Algorithms for the Construction and Analysis of Systems: An STTT special section

Preface by the section editor: Susanne Graf*

The date of receipt and acceptance will be inserted by the editor

Abstract. The papers in this special section appeared originally in the proceedings of the 2000 edition of the conference "Tools and Algorithms for the Construction and Analysis of Systems" (TACAS) hold in Berlin as a constituent event of the European joint conferences on Theory and Practice of Software. All papers present tools relevant in the context of validation of systems. The first three focus on extensions or particular applications of model-checking techniques, whereas the fourth one is about integration of design tools with validation tools, in particular theorem provers and model-checkers.

Key words:
Goals: conformance testing, model-checking, performance evaluation, formal verification
Techniques: static analysis, slicing, Markov chains, partial order reductions, theorem proving, tool integration
Application domains: asynchronous systems, probabilistic systems, security protocols, component based approaches

This special section on *Tools and Algorithms for the Construction and Analysis of Systems* presents papers which have originally been presented at the sixth edition of this international conference. TACAS is a forum for researchers, developers and users interested in rigorously based tools for the construction and analysis of systems. The conference intends to bridge the gaps between different communities - including but not limited to those devoted to formal methods, real-time, software engineering, asynchronous systems, hardware, theorem proving and programming languages - that traditionally had little interaction but share common interests in models and techniques at the basis of tool developments.

All four papers of this special section present a tool for the validation of systems. Each one has its particular application domain, such as conformance testing of asynchronous systems, model-checking of security protocols and probabilistic systems, but they share common formalisms, such as communicating finite state machines, temporal logic, labelled transition systems, and common techniques for mastering problems such as state space explosion. The first three papers present extensions or particular applications of model-checking techniques, whereas the last one presents a framework for integrating various verification tools, in particular theorem provers in a platform providing or facilitating the implementation of adequate interfaces.

The first paper on "*Using static Analysis to improve automatic test generation*" proposes a combination of static analysis and model-checking techniques in the context of automatic test case generation. Static analysis allows to extract a large spectrum of properties of programs with a relatively low computational complexity. Classic application domains are validation and code optimization. Here, static analysis methods are applied to particular programs representing communicating systems interacting with each other and an external (arbitrary) environment. The generated properties are used, not to optimize the executed code but to minimize the underlying semantic model (state graph) of the system, nevertheless fully preserving all its relevant properties. As a consequence, the proposed static analysis methods are useful for any tool based on model exploration, in particular model-checking tools such as SPIN[8], CADP[7] or IF[3,4]. Automatic test case generation is a particular application domain of model-checking where instead of an open system reacting to an arbitrary environment, only particular sequences of interactions between the environment and the system, as specified in *test objectives*,

* VERIMAG, 2 avenue de Vignate, F-38610 Grenoble-Gières, susanne.graf@imag.fr, http://www-verimag.imag.fr/ graf

are relevant. In this context the proposed slicing methods are particularly effectful, as sometimes large portions of the overall system can be eliminated before the actual model exploration. The reductions are obtained by means of syntactic transformations of the original system description given in the IF intermediate representation from which the tool TGV [6] generates tests.

Any tool that can handle systems described in IF can directly profit from the implemented simplifications. On the other hand, for systems described in high level design languages such as SDL[9] or UML, tests can be generated thanks to translators from these formalisms into the IF intermediate representation.

The second paper concerns "*A tool for model-checking Markov chains*". Here, model-checking techniques are extended for probabilistic models represented by discrete or continuous Markov chains, that is Kripke structures in which transitions are decorated with probabilities in the form of the *rate* of an exponential distribution. Properties are expressed in a very general temporal logic, an extension of CSL[2], in which the CTL[5] path quantifiers are replaced by probablity measures over paths, and which has a time bound version of the *until* operator as in TCTL[1]. The logic is enhanced with a *steady state* operator allowing to quantify the likelyhood of staying in a given set of states on the long run, which can be considered as a probabilistic generalisation of the *always* operator.

Model-checking of CSL formulae is an extension of the iterative model-checking algorithm of CTL requiring in addition the computation of the probability of (sets of) paths. The tool implements several direct and iterative methods for the computation of these probabilities, as their efficiency is quite example dependant.

The implemented model-checker has a modular structure, where a verification manager controls the order of the verification steps and the method to be applied at each step. The tool provides also interfaces to various high-level design languages, such as Queuing Networks or stochastic versions of Petri Nets, and in principle also SDL or UML. Nevertheless, for specifications described in high level formalisms, one has to pay the price of state explosion due to evaluation of all variables or agressive abstraction because of the low-level nature of the Markov chain model.

The third paper on "*Efficient Verification of Security Protocols using Partial Order Reductions*" is about specializing known model-checking methods to the particular domain of security protocols, where messages can be encrypted by means of keys. Verification is concerned with security related properties such as privacy of coded information, authentification, anonymity, but also with functional safety properties of the protocol itself. The approach is based on a tool taylored towards a particular framework of security protocols and their properties. In a security system, the communication medium is considered to be in the hand of an *intruder* who *knows* all sent messages and can use them to build new ones using the keys known to him. The protocol entities are modeled by a set of agents such that any bounded number of parallel interacting sessions can be modelled. Properties are expressed by means of a past time temporal logic with a set of atomic propositions adapted to security systems, including predicates expressing the fact that agents have *knowledge* of messages. This allows to express approximations of the ususal properties of this type of systems. Instead of using a general purpose model-checker, the authors made the choice to implement a particular one, exploiting the structure and laws of security systems in order to improve its complexity. In particular partial order reduction plays an important role, as at any time any process can accept a message. Thus, the particular structure of the considered systems, and the restriction to safety properties, allows to go beyond the standard partial order reductions by systematically taking send actions as early as possible, similar as in [11,10]. Nevertheless, their is also large amount of true non-determinism, as the messag that any protocol instance can receive at any time, is anyone which can be constructed from messages already sent by any instance, satisfying also the template of the expected message.

The fourth paper, describes the "*Prosper Toolkit*", a platform which allow to integrate a wide range of verification tools, including theorem provers, decision procedures and model-checkers by means of several application program interfaces (API). It also facilitates the incorporation of prove engines into existing application. Such an approach is highly interesting in order to make possible the integration of a large range of formal methods in any development process.

It is more ambitious compared to other tool-sets integrating functionalities of different tool-sets in the area of validation. The existing *open* validation platforms - such as CADP[7] and IF[4] - have API's for different representations of models; moreover, all inputs, outputs and any information useful for later use is stored in files in documented formats (shared between tools), and the interaction is possible in a blackbox commandline fashion with a large choice of commands or options. A collaboration platform like ETI[12] is based on the same blackbox view of tools and allows the user to define new functionalities by means of sequences of existing functions using the signatures of all functions and a topology for more abstract reasoning about the to be combined functions.

Theorem provers can in general not reasonably be combined using such a blackbox view, as the useful information is not necessarilly stored in an exploitable format. Existing validation environments combining theorem provers and model-checkers, are indeed embedded *within* a theorem prover by calling all external tools from within the theorem prover in a blackbox fashion.

The approach followed here allows to combine several tools without necessarily using blackbox interaction. The interaction between tools is defined via an interface lan-

guage taylored for exchanges between an application and a theorem prover allowing to communicate and manipulate items like terms, assertions, proofs and tactics. For each tool to be integrated, it is necessary to define an appropriate API defining the set of functionalities made visible to the outside.

The presented toolkit is much more focussed than standard component architectures such as CORBA, but it could profit from such an architecture for the actual communication between different tools. The presented approach is well suited for the collaboration between tools handling expressions and proofs. For the combination of tools working on different representations of models (which are of important size), there seems no way around translations between formats (or their standardisations).

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model checking for probabilistic real-time systems. In *18th International Colloquium on Automata, Languages and Programming (ICALP), LNCS 510*, July 1991.
2. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous markov chains. *ACM Transactions on Computational Logic*, 1(1), 2000.
3. M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, and L. Mounier. IF: An intermediate representation and validation environment for timed asynchronous systems. In *Proceedings of Symposium on Formal Methods 99, Toulouse*, number 1708 in LNCS. Springer Verlag, September 1999.
4. M. Bozga, L. Ghirvu, S. Graf, and L. Mounier. IF: A validation environment for timed asynchronous systems. In *Proceedings of Conference on Computer Aided Verification, CAV'00, Chicago*, number 1855 in LNCS. Springer Verlag, June 2000.
5. E.M. Clarke, E.A. Emerson, and E. Sistla. Automatic verification of finite state concurrent systems using temporal logic specification: a practical approach. In *10th ACM Symposium on Principles of Programming Languages (POPL83)*, 1983. Complete version published in ACM TOPLAS, 8(2):244–263, April 1986.
6. J.Cl. Fernandez, C. Jard, T. Jéron, and C. Viho. An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology. *Science of Computer Programming*, 29, 1997.
7. Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Radu Mateescu, Laurent Mounier, and Mihaela Sighireanu. Cadp (cæsar/aldebaran development package): A protocol validation and verification toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, volume 1102 of *LNCS*, pages 437–440. Springer Verlag, August 1996.
8. G. J. Holzmann. The model-checker SPIN. *IEEE Trans. on Software Engineering*, 23(5), 1999.
9. ITU. Sdl specification and description language. International standard, ITU, 1996.
10. J.P. Krimm and L. Mounier. Compositional State Space Generation with Partial Order Reductions for Asynchronous Communicating Systems. In S. Graf and M. Schwartzbach, editors, *Proceedings of TACAS'2000 (Berlin, Germany)*, volume 1785 of *LNCS*, pages 266–282. Springer, March 2000.
11. V. Shmatikov and U. Stern. Efficient finite-state analysis for large security systems.
12. B. Steffen, T. Margaria, and V. Braun. The electronic integration platform: concepts and design. *STTT (Software Tools and Technology Transfer)*, 1(1-2), December 1997.