Int J Softw Tools Technol Transfer (2016) 18:121–128
DOI 10.1007/s10009-015-0403-0

TACAS 2014

Some recent advances in automated analysis

Erika Ábrahám[1] · **Klaus Havelund**[2]

Published online: 13 October 2015

Abstract Due to the increasing complexity of software systems, there is a growing need for automated and scalable software synthesis and analysis. In the last decade, active research in the formal methods community brought interesting results and valuable tools. However, there are still challenges to face and hard problems that need to be solved. We briefly outline some recent trends, and review some of the latest achievements, introducing six papers selected from the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014).

Keywords Analysis · Parallel algorithms · Satisfiability modulo theories · Runtime verification · Probabilistic systems

1 Introduction

This special issue of the journal *Software Tools for Technology Transfer* (STTT) contains revised and extended versions of six papers selected out of 42 papers presented at the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14) [4], held in Grenoble, France during April 7–11, 2014, as part of the Joint European Conferences on Theory and Practice of Software (ETAPS). The peer-reviewed papers collected in this special issue have been invited by the guest editors amongst the top papers presented at TACAS'14 based on their relevance to STTT.

TACAS is a forum for researchers, developers, and users interested in rigorously based tools and algorithms for the construction and analysis of systems. The research areas covered by TACAS include, but are not limited to, formal methods, software and hardware specification and verification, static analysis, dynamic analysis, model checking, theorem proving, decision procedures, real-time, hybrid and stochastic systems, communication protocols, programming languages, and software engineering. TACAS provides a venue where common problems, heuristics, algorithms, data structures, and methodologies in these areas can be discussed and explored.

Due to the increasing complexity of software systems, there is a growing need for automated software synthesis and analysis. In the last decade, active research in the formal methods community brought interesting results and valuable tools. However, there are still challenges to face and hard problems that need to be solved. As the size of our software systems is increasing, the *scalability* of the automated synthesis and analysis techniques is a highly relevant issue.

The selected papers cover four domains, which we believe form trends within the formal methods community, and which are discussed below and organized as follows. Section 2 discusses parallel algorithms and their application to for example model checking. Section 3 discusses SAT and SMT solving. Section 4 discusses runtime verification. Section 5 discusses hybrid and probabilistic verification. Finally Sect. 6 concludes the paper.

The research performed by this author was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

✉ Klaus Havelund
Klaus.Havelund@jpl.nasa.gov

1 RWTH Aachen University, Aachen, Germany

2 Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA

2 Distributed and parallel algorithms

Nowadays, nearly all personal computers have many-core CPUs, the usage of cloud and grid computing is rising, and there are great advances in supercomputer architectures. The performance of the fastest supercomputers available today has reached the PetaFLOPS scale, i.e., they can execute 10^{15} floating point operations per second (FLOPS). The next generation of Exa-scale supercomputers with 10^{18} FLOPS performance is under development.

Distributed and parallel computing techniques make use of such hardware structures to solve computationally intensive problems. Typical areas for massively parallel applications are, e.g., weather forecasting, climate research, and simulations of chemical, biological and physical processes.

However, the *efficient* usage of these distributed and parallel computer architectures is not at all trivial. The computational effort might be unbalanced between the processes, such that one process might need to wait for a long time for results computed by another process, thereby wasting available computing resources. Last but not least, massive communication (e.g., message broadcasting in a massively parallel application) can be itself a bottleneck for efficiency.

For these reasons, achieving a linear speedup (in the number of used cores) for the computation time is hard to realize. Performance analysis techniques and tools help the developers to identify execution bottlenecks via monitoring the program execution, and computing and visualizing characteristic quantities like, e.g., average waiting times at certain control points. However, there is a strong need for further improvements. We still have problems to exploit the capabilities of Peta-scale supercomputers, and no one knows yet how to achieve this at the forthcoming Exa-scale, where, besides scalability issues, additional problems like increased failure rates must be faced [3].

Besides efficiency, a central problem is the *correctness* of parallel programs, which has different facets. Deadlocks can happen when threads or processes wait for each other in a cyclic manner, such that none of them can continue its execution. Furthermore, correct parallel programs should preferably (in the general case) yield the same result, independently of the temporal order of process executions. For example, in multi-threading, mutual exclusion must be used in a safe manner to assure atomic computation where needed. To achieve functional correctness, if a problem is decomposed into sub-problems, the result must be carefully synthesized from the sub-results.

To assure such correctness properties, formal methods can be used for the *verification* of parallel programs. Whereas the theoretical roots for the verification of parallel programs are historically relatively deep [12,31,62,66,71], current approaches are still far from being scalable at the super-computing level. New advances in this direction use parallel computing itself for verification, i.e., the verification algorithms themselves are parallel programs. Besides deductive techniques, powerful parallel model checking approaches have become available. However, to achieve scalability, also these techniques must reach an optimal load balance between the parallel running model checking processes.

Early attempts to overcome this problem include, e.g., techniques for partial order reduction and slicing [42,52], and randomization [19,75]. Several efforts have been made to parallelize the Spin model checker. An early attempt on distributed model checking in Spin is described in [64]. More recent work can be classified into two categories: multi-core approaches [40,53,55] where model checking is distributed on several cores on the same machine, and cloud approaches [54,56] where model checking is distributed on multiple machines in the cloud. Concerning multi-core approaches, [55] is a general method for safety verification, while [53] and [40] concern an algorithm for partial verification of liveness properties with parallel breadth-first search. Concerning cloud approaches, [54] describes how the use of massive parallelism in a cloud-computing context may deliver near real-time performance. This is furthermore an application of what is referred to a *swarm* approach, where multiple independent and different instances of the verification problem are launched in parallel.

In this volume we report on three latest developments on this research front. The paper *Concurrent Depth-First Search Algorithms based on Tarjan's Algorithm* [68], by Gavin Lowe, an extension of the TACAS'14 conference paper [67], deals with parallelization issues for some important graph-related problems: finding strongly connected components, cycles and lassos in graphs. Tarjan's algorithm is widely used in its sequential version; however, its efficient parallelization was still an open challenge. The proposed parallelized version may find application in model checking algorithms, for example to check which states are divergent, i.e., which states can lead to an unbounded number of internal steps.

The paper *FDR3—A Parallel Refinement Checker for CSP* [43], by Thomas Gibson-Robinson, Philip Armstrong, Alexandre Boulgakov, and A. W. Roscoe, extends the TACAS'14 publication [44]. It presents the FDR3 tool, which is a rewrite and of the FDR2 refinement checker with extended functionalities. Amongst its several improvements is a new parallel refinement-checking algorithm, able to achieve a near-linear speedup as a function of number of cores utilized (including clusters of cores), and a new algorithm used to construct the internal representation of CSP processes. FDR3 is furthermore able to efficiently make use of on-disk storage once main memory is exhausted. FDR3 relies on Tarjan's algorithm, and future work includes pursuing alternative methods of parallelizing divergence checking, including methods based on Gavin Lowe's work presented in this volume.

The paper *Many-Core On-the-Fly Model Checking of Safety Properties Using GPUs* [80], by Anton Wijs and Dragan Bošnački, presents another parallelization approach for model checking. A previous version of this paper was published in [79]. Despite advances in model checking, state space explosion is still a hindrance for scalability. Smart concurrent solutions can push forward the boundaries of applicability; however, the resources for concurrent execution are relatively limited on standard home computers. To best exploit these resources, this work makes use of General Purpose Graphics Processors (GPUs) for the computations. This is an extremely challenging path of research, to which this paper makes an important contribution.

3 SAT and SAT-modulo-theories solving

A further active research area is the *integration* of different techniques to form powerful and efficient tools. In this context, logical encoding of problems and the usage of SAT and SAT-Modulo-Theories (SMT) solving for satisfiability checking are frequently employed.

SAT solving aims at the automated check of propositional logic formulas for satisfiability. The technology development started around 1960. First approaches used enumeration and resolution [29]. The combination of enumeration with propagation led to the well-known DPLL algorithm [28]. A breakthrough regarding efficiency and scalability was achieved by combining enumeration and propagation with resolution to identify reasons to explain why certain (partial) assignments do not satisfy a formula. This resulted in the *conflict-directed clause learning* approach [69,82], whose impact is well reflected in the following citation from the zChaff webpage: "We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses. (Of course, it can't solve every such problem!)". After the pioneer solvers GRASP [69] and zChaff [82], a variety of other SAT solvers were developed with watched-literal techniques and smart heuristics, for e.g., clause learning and forgetting, dynamic variable ordering and restarts. Just to mention one of them, MiniSAT [38] is not only highly efficient but also small and, therefore, well suited for understanding and teaching the SAT mechanisms.

The scalability of SAT solvers opened the way to real-world applications. Besides academic applications in different research areas, nowadays also many companies use SAT solving, e.g., to solve huge combinatorial problems or for digital circuit design and verification.

The introduction of a standardized input language was a great achievement and an important milestone in the success of SAT solving. On the application side, it allows users to formalize their problems once and apply a wide range of solvers to them. On the development side, it enabled the collection of large benchmark sets and the start of competitions in 2002. In 2014, the SAT competition had an impressive number of 79 participants with 137 solvers. The SATLive! forum and dedicated conferences and journals further support the community with platforms for exchange.

The SAT developments showed promising results to apply similar technologies to more expressive logics, resulting in *SAT-Modulo-Theories* (*SMT*) *solving*. The idea is to use SAT solvers to get solutions for the Boolean skeleton of quantifier-free first-order logic problems, and use different *theory solvers* to check the corresponding sets of theory constraints for consistency. Some of the important milestones in this area were the development of decision procedures for combined theories [70,74], and the extension DPLL(T) [37] of the DPLL approach for SAT with theories.

One of the first theories considered for SMT solving were equalities and uninterpreted functions, bit-vectors and array theory. Later on, also solutions for linear real and integer arithmetic and fragments thereof were implemented. Latest developments address challenging extensions also for non-linear arithmetic theories. These theories are supported by a large and still increasing number SMT solvers, e.g., the tools AProVE [45], CVC [11], HySAT/iSAT [41], MathSAT [26], MiniSmt [81], OpenSMT [24], SMT-RAT [27], VeriT [23], Z3 [30], or Yices [36].

Due to the increased level of expressiveness, SMT finds application in a wide range of domains like, e.g., verification (model checking, static analysis, termination analysis), test case generation, controller synthesis, predicate abstraction, equivalence checking, scheduling, planning, or product design automation and optimization.

Also the SMT community profits from the SMT-LIB input standard and from competitions since 2005. In 2014, 20 solvers participated in 32 logical categories.

Where does the development go? Surely, a still major issue is efficiency and scalability. Whereas for easier theories already large problem instances can be solved, despite encouraging evolution, SMT solving for non-linear real and integer arithmetic is a yet upcoming area. To tackle those problems, we need dedicated SMT solvers for specific problem classes with further novel lemma generation and learning techniques, elegant ideas for the combination of decision procedures, and clever parallelization approaches. A big potential lies in learning from decision procedures and technologies used in symbolic computation [1]. Regarding functionalities, there is also a trend to increase applicability by generating unsatisfiable cores and interpolants, handling quantified formulas, and offering techniques for optimization.

In this volume two contributions are devoted to SAT and SMT solving. SATMC 3.0, a SAT-based bounded model checker for security-critical systems is presented in the paper

SATMC: a SAT-based Model Checker for Security Protocols, Business Processes, and Security APIs [7], by Alessandro Armando, Roberto Carbone, and Luca Compagna, as an extension of [6]. It is distinguished by combining techniques originally developed for planning with techniques developed for the analysis of reactive systems. SATMC has been applied in a variety of application domains, including security protocols, security-sensitive business processes, and cryptographic APIs. SATMC supports a powerful specification language, including rewrite rules, Horn clauses, and first-order LTL formulae. It leverages NuSMV to generate a SAT encoding for the LTL formulae and MiniSAT to solve the SAT problems.

The paper *Monitoring Modulo Theories* [33], by Normann Decker, Martin Leucker and Daniel Thoma, an extended version of the conference paper in [32], considers an SMT-based approach to runtime verification of temporal properties over first-order theories. It lifts monitor synthesis procedures for propositional temporal logics to a temporal logic over structures within a first-order theory, and proposes a first-order monitoring algorithm that combines SMT solving and classical monitoring of propositional temporal properties. The approach is here applied to LTL, and the Z3 SMT solver is used for solving data constraints. However, the approach is generic and can be applied to any suitable temporal logic, and any first-order theory can be chosen for which an SMT solver is available.

4 Runtime verification

The scalability issue often associated with formal methods is due to the desire to verify (analyze) all possible execution paths of the system being analyzed, and potentially for all possible inputs. This problem is in general NP-complete, and in practice becomes infeasible without relaxing on the kind of properties being proven or the confidence in the result. Testing is the practical less perfect alternative to full verification. Here test inputs are generated using a more or less automated strategy, and outputs are verified using more or less sophisticated test oracles (monitors). In industrial practice, test input generation is typically not automated (rather: test cases are manually created), and monitors are typically not very sophisticated, for example just comparing text files with a diff-command.

Runtime verification [39,50,65] (RV) is a subfield of computer science focusing on just analyzing *systems executions*, including collections thereof, either during test (the test oracle problem), or after deployment. The field is not concerned with test case generation, which is one of the main focuses of test research. The purpose of the field is to focus study how much we can get out of one or more execution traces, in other words: just by observing what the system does when executing. The field obviously intersects with testing by contributing to how to write advanced test oracles.

Runtime verification as a field covers various sub-fields. *Specification-based* monitoring is concerned with checking a program execution against a formal specification of one or more requirements. A program P to be monitored is instrumented to emit a sequence τ of observable events, which are fed into a monitor, which as a second input takes a specification ψ of expected behavior. The trace is then matched against the specification, also formalized as: $\tau \models \psi$. Instrumentation can, for example, be performed using aspect-oriented programming. Static analysis can be used to minimize the number of instrumentation points, a topic receiving increasing attention by the research community.

Events in practice carry data, as in: `open("file42")`, in contrast to propositional events, such as `openFile`, and it must be possible to refer to these data in specifications. Recent research has focused on efficient and low-impact monitoring of such data carrying events, referred to as *parametric monitoring*. Such data must be stored and especially searched efficiently as part of the monitor. The 1st Intl. Competition of Software for Runtime Verification (CSRV'14), in particular focusing on parametric monitoring, was held together with the RV conference in 2014 in Toronto, Canada.

Detection of a property violation can be used not only for testing an application, but also during operation in the field, to cause a change of behavior by triggering fault-protection code, which steers the application out of a bad situation. The extreme RV solution is planning and scheduling techniques, which continuously adapt to the current situation. For a survey relating verification and validation to planning and scheduling, see [21].

Over the last 15 years numerous specification-based runtime verification systems have been developed, only a few of which will be mentioned here. Initial specification-based systems could only handle propositional events. These include, for example, Temporal Rover [35], MaC [63], and Java PathExplorer [51]. The first systems to handle parameterized events appeared around 2004, and include [5,14,25,76]. Several parametric monitoring systems have appeared since then. RV systems usually implement specification languages which are based on formalisms such as state machines [13,25,46], regular expressions [5,25], temporal logic [17,18,25,32,48,76], variations over the μ-calculus [14], grammars [25], and rule-based systems [16,49]. A few of these logics incorporate time as a built-in concept, typically embedded in temporal logics, as for example in [17]. If no special concept of time is introduced, time observations can be considered as just data (time stamps).

Runtime verification systems are based on different algorithms. *Slicing-based* algorithms have shown very efficient

[5,13,25]. These algorithms conceptually slice a trace into projections, a projection for each parameter combination. The efficiency of these algorithms generally comes at the cost of lack of some expressiveness, as pointed out in [13]. Other monitoring systems represent data as *constraints*. A constraint-based system is the first-order linear temporal logic described in [18]. Some systems based on linear temporal logic apply rewriting of temporal formulas. These include, for example, [14,15,48,76]. Rule-based systems, such as [16,49] operate with a collection of facts, usually organized in an efficient data structure/network, which is modified by the rules.

Most of the logics mentioned above are so-called external DSLs, small languages with their own grammar and parser. However, also systems have been developed which offer APIs in programming languages (also referred to as internal DSLs), for writing monitors. These include [15,22,49].

Specifications can be written by humans, or they can be learned from nominal executions, also referred to as *specification mining* [57]. A form of runtime verification not requiring specifications is what we will refer to as *runtime analysis*, where program executions are analyzed with specialized *algorithms*. Examples include algorithms for detecting concurrency problems such as deadlock potentials [20] and data races [8,73]. Finally, *trace visualization* of execution traces supports human comprehension of what the system does [72]. Trace visualization is related to specification mining in that it produces an abstraction the system's behavior, although only for the eyes.

The focus of future runtime verification research will include continued studies of how to optimize monitoring algorithms, to use less time and less space. Static analysis can be combined with dynamic analysis to minimize the code instrumentation performed, thereby reducing the impact on the monitored program. Another way of looking at this problem is to use static analysis to prove as much as possible of a property, and then use runtime verification to monitor the remaining unproved proof obligations. There will be continued research in expressive and succinct logics, potentially merging well-known logical systems such as temporal logic, regular expressions, and state machines/rule systems. We may eventually see the emergence of such logics in programming languages as part of the design-by-contract paradigm. Specifications are hard to write, and specification mining and visualization may contribute a great deal to ease this task. Each time we run our program, we should learn from it. The ultimate proof of success of this field will be widespread deployment of monitoring against logic-based requirements in industrial applications.

This area of research is represented in this volume by the paper *Monitoring Modulo Theories* [33], by Normann Decker, Martin Leucker, and Daniel Thoma, already mentioned in Sect. 3.

5 Probabilistic systems

Probabilistic systems are systems with randomized behavior. Some examples are probabilistic algorithms which involve random values drawn from some probability distributions, computer systems with inherent randomization such as quantum computers or approximate computing, or biological systems whose evolution can be modeled probabilistically.

There are different languages to *model* probabilistic systems. Popular automata-based modeling formalisms for probabilistic systems are discrete- and continuous-time Markov chains, and variants thereof which exhibit non-determinism such as Markov decision processes or probabilistic automata. Probabilistic programs use, additionally to the standard programming constructs, probabilistic branching and probabilistically determined values in assignments, and are well suited for high-level modeling.

To describe the behavior of probabilistic models, probabilistic properties like "the (maximal) probability to reach a set of bad states is at most 0.1" can be formalized in different property specification languages. Probabilistic computation tree logic (PCTL) extends the logic CTL with probabilities and can be used to describe properties of discrete-time models. Continuous stochastic logic (CSL) is a PCTL extension supporting the specification of continuous-time properties. Last but not least, probabilistic linear-time temporal logic (PLTL), a probabilistic extension of LTL can be used to specify probabilistic liveness properties.

Efficient *model checking* algorithms for these models and logics have been developed, implemented in a variety of software tools, and applied to case studies from various application areas. The crux of probabilistic model checking [9,10,59,60] is to appropriately combine techniques from numerical mathematics and operations research with standard reachability analysis and model checking techniques. In this way, properties can be automatically checked up to a user-defined precision. Markovian models comprising millions of states can be checked rather fast by dedicated tools such as MRMC [58] and PRISM [61]. These tools are currently being extended with counterexample generation facilities to enable the possibility to provide useful diagnostic feedback in case a property is violated [2].

To be able to formalize and analyze systems with uncertain behavior or incomplete specification, also *parametric* modeling languages and probabilistic model checking techniques for them were investigated, resulting in tools like PARAM [47] and PROPhESY [34].

Despite this intensive and successful developments, there remain several challenging hard and practically relevant problems to be solved. There were some achievements on probabilistic hybrid systems, which have certain probabilistic components either in their discrete or in their continuous behavior. However, these techniques need to be strengthened to reach practical applicability. Also scalability is still an issue. Though model checking tools can handle huge models, novel symbolic approaches and abstraction techniques are needed to analyze probabilistic programs with large variable domains or large-scale parallelism. To mention a last challenge, probabilistic domain-specific languages and formal methods for their analysis would help to model and analyze applications from the area of high-performance computation and approximate computing.

The application of existing techniques and tools to case studies is extremely important, as it brings highly valuable insights to applicability, it highlights bottlenecks, drives research to important practical problems, and eases technology transfer to industry. In this volume, a report on an interesting case study is given in the paper *Probabilistic Verification and Synthesis of the Next Generation Airborne Collision Avoidance System* [78], by Christian von Essen and Dimitra Giannakopoulou, extending the TACAS'14 publication [77]. ACAS X, the next-generation airborne collision avoidance system considers probabilistic models to represent different types of uncertainty. The authors give a nice example of how the power of existing formal methods and frameworks can be bundled by integrating them into a tool dedicated to a special problem class.

6 Conclusion

Some recent advances in automated analysis have been discussed and related to selected papers from TACAS 2014, included in this volume. Four domains have been identified: the parallelization of algorithms—including algorithms for verifying systems, specifically model checking; SAT and SMT solving with a basis in first-order logic; runtime verification; and finally probabilistic systems. Parallel algorithms, SAT/SMT solving, and runtime verification illustrate different ways of dealing with the scalability problem of formal methods. Parallel algorithms and SAT/SMT solving can be considered successful techniques for solving the traditional verification problem, whereas runtime verification is an example of shifting the problem from verification of full models to analysis of single traces. Probabilistic systems modeling and verification is an example of a new domain, requiring new techniques all together.

Acknowledgments We are grateful to all authors for their contributions and to the reviewers of TACAS'14 and of this special issue for their thorough and valuable work.

References

1. Ábrahám, E.: Building bridges between symbolic computation and satisfiability checking. In: Proceedings of the 2015 ACM International Symposium on Symbolic and Algebraic Computation (ISSAC'15), pp. 1–6. ACM Press, New York (2015)
2. Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.-P., Wimmer, R.: Counterexample generation for discrete-time Markov models: an introductory survey. In: Formal Methods for Executable Software Models—14th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'14), Advanced Lectures. LNCS, vol. 8483, pp. 65–121. Springer, Berlin (2014)
3. Ábrahám, E., Bekas, C., Brandic, I., Genaim, S., Johnsen, E.B., Kondov, I., Pllana, S., Streit, A.: Preparing HPC applications for exascale: challenges and recommendations. CoRR. arXiv:1503.06974 (2015)
4. Ábrahám, E., Havelund, K. (eds.): Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413. Springer, Berlin (2014)
5. Allan, C., Avgustinov, P., Christensen, A.S., Hendren, L., Kuzins, S., Lhoták, O., de Moor, O., Sereni, D., Sittamplan, G., Tibble, J.: Adding trace matching with free variables to AspectJ. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05), pp. 345–364. ACM Press, New York (2005)
6. Armando, A., Carbone, R., Compagna, L.: SATMC: a SAT-based model checker for security-critical systems. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413, pp. 31–45. Springer, Berlin (2014)
7. Armando, A., Carbone, R., Compagna, L.: SATMC: a SAT-based model checker for security protocols, business processes, and security APIs. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-015-0385-y (2015)
8. Artho, C., Havelund, K., Biere, A.: High-level data races. Softw. Test. Verif. Reliab. **13**(4), 207–227. doi:10.1002/stvr.281 (2003)
9. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Performance evaluation and model checking join forces. Commun. ACM **53**(9), 76–85 (2010)
10. Baier, C., Katoen, J.-P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
11. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 171–177. Springer, Berlin (2011)
12. Barringer, H.: A Survey of Verification Techniques for Parallel Programs. LNCS, vol. 191. Springer, Berlin (1985)
13. Barringer, H., Falcone, Y., Havelund, K., Reger, G., Rydeheard, D.: Quantified event automata—towards expressive and efficient runtime monitors. In: Proceedings of the 18th International Symposium on Formal Methods (FM'12). LNCS, vol. 7436, pp. 68–84. Springer, Berlin (2012)
14. Barringer, H., Goldberg, A., Havelund, K., Sen, K.: Rule-based runtime verification. In: Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'04). LNCS, vol. 2937, pp. 44–57. Springer, Berlin (2004)
15. Barringer, H., Havelund, K.: TraceContract: a Scala DSL for trace analysis. In: Proceedings of the 17th International Symposium on Formal Methods (FM'11). LNCS, vol. 6664, pp. 57–72. Springer, Berlin (2011)
16. Barringer, H., Rydeheard, D.E., Havelund, K.: Rule systems for run-time monitoring: from Eagle to RuleR. J. Log. Comput. **20**(3), 675–706 (2010)
17. Basin, D.A., Klaedtke, F., Müller, S.: Policy monitoring in first-order temporal logic. In: Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10). LNCS, vol. 6174, pp. 1–18. Springer, Berlin (2010)

18. Bauer, A., Küster, J.-C., Vegliach, G.: From propositional to first-order monitoring. In: Proceedings of the 4th International Conference on Runtime Verification (RV'13). LNCS, vol. 8174, pp. 59–75. Springer, Berlin (2013)
19. Behrmann, G., Hune, T., Vaandrager, F.: Distributing timed model checking—how the search order matters. In: Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00). LNCS, vol. 1855, pp. 216–231. Springer, Berlin (2000)
20. Bensalem, S., Havelund, K.: Dynamic deadlock analysis of multi-threaded programs. In: Proceedings of the First Haifa International Conference on Hardware and Software Verification and Testing (HVC'05). LNCS, vol. 3875, pp. 208–223. Springer, Berlin (2006)
21. Bensalem, S., Havelund, K., Orlandini, A.: Verification and validation meet planning and scheduling. Softw. Tools Technol. Transf. **16**(1), 1–12 (2014)
22. Bodden, E.: MOPBox: A library approach to runtime verification. In: Proceedings of the 2nd International Conference on Runtime Verification (RV'11). LNCS, vol. 7186, pp. 365–369. Springer, Berlin (2011)
23. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Proceedings of the 22nd International Conference on Automated Deduction (CADE-22). LNCS, vol. 5663, pp. 151–156. Springer, Berlin (2009)
24. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT solver. In: Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10). LNCS, vol. 6015, pp. 150–153. Springer, Berlin (2010)
25. Chen, F., Roşu, G.: Parametric trace slicing and monitoring. In: Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09). LNCS, vol. 5505, pp. 246–261 (2009)
26. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13). LNCS, vol. 7795, pp. 93–107. Springer, Berlin (2013)
27. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Abraham, E.: SMT-RAT: an open source C toolbox for strategic and parallel SMT solving. In: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT'15). LNCS. Springer, Berlin (2015)
28. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
29. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960)
30. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). LNCS, vol. 4963, pp. 337–340. Springer, Berlin (2008)
31. de Roever, W.P., de Boer, F.S., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge Tracts in Theoretical Computer Science, vol. 54. Cambridge University Press, Cambridge (2001)
32. Decker, N., Leucker, M., Thoma, D.: Monitoring modulo theories. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413, pp. 341–356. Springer, Berlin (2014)
33. Decker, N., Leucker, M., Thoma, D.: Monitoring modulo theories. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-015-0380-3 (2015)
34. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.-P., Ábrahám, E.: Prophesy: a probabilistic parameter synthesis tool. In: Proceedings of the 27th International Conference on Computer Aided Verification (CAV'15). LNCS, vol. 9206, pp. 214–231. Springer, Berlin (2015)
35. Drusinsky, D.: The temporal rover and the ATG rover. In: Proceedings of the 7th International SPIN Workshop on Model Checking and Software Verification (SPIN'00). LNCS, vol. 1885, pp. 323–330. Springer, Berlin (2000)
36. Dutertre, B.: Yices 2.2. In: Proceedings of the 26th International Conference on Computer Aided Verification (CAV'14). LNCS, vol. 8559, pp. 737–744. Springer, Berlin (2014)
37. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06). LNCS, vol. 4144, pp. 81–94. Springer, Berlin (2006)
38. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03). LNCS, vol. 2919, pp. 502–518. Springer, Berlin (2004)
39. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Summer School Marktoberdorf 2012—Engineering Dependable Software Systems. IOS Press, Amsterdam (2013)
40. Filippidis, I., Holzmann, G.J.: An improvement of the piggyback algorithm for parallel model checking. In: Proceedings of the 2014 International Symposium on Model Checking of Software (SPIN'14), pp. 48–57. ACM Press, New York (2014)
41. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. J. Satisf. Boolean Model. Comput. **1**(3–4), 209–236 (2007)
42. Garavel, H., Mateescu, R., Smarandache, I.: Parallel state space construction for model-checking. In: Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN'01), pp. 217–234. Springer, Berlin (2001)
43. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W.: FDR3: a parallel refinement checker for CSP. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-015-0377-y (2015)
44. Gibson-Robinson, T., Armstrong, P.J., Boulgakov, A., Roscoe, A.W.: FDR3—a modern refinement checker for CSP. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413, pp. 187–201. Springer, Berlin (2014)
45. Giesl, J., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Proving termination of programs automatically with AProVE. In: Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14). LNAI, vol. 8562, pp. 184–191. Springer, Berlin (2014)
46. Goubault-Larrecq, J., Olivain, J.: A smell of ORCHIDS. In: Proceedings of the 8th International Workshop on Runtime Verification (RV'08). LNCS, vol. 5289, pp. 1–20. Springer, Berlin (2008)
47. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: a model checker for parametric Markov models. In: Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10). LNCS, vol. 6174, pp. 660–664. Springer, Berlin (2010)
48. Hallé, S., Villemaire, R.: Runtime enforcement of web service message contracts with data. IEEE Trans. Serv. Comput. **5**(2), 192–206 (2012)
49. Havelund, K.: Rule-based runtime verification revisited. Softw. Tools Technol. Transf. **17**(2), 143–170 (2014)
50. Havelund, K., Goldberg, A.: Verify your runs. In: Proceedings of the 1st IFIP TC 2/WG 2.3 Conference on Verified Software: Theories, Tools, Experiments (VSTTE'05), pp. 374–383 (2008)
51. Havelund, K., Roşu, G.: Efficient monitoring of safety properties. Softw. Tools Technol. Transf. **6**(2), 158–173 (2004)
52. Heyman, T., Geist, D., Grumberg, O., Schuster, A.: Achieving scalability in parallel reachability analysis of very large circuits. In: Proceedings of the 12th International Conference on Com-

puter Aided Verification (CAV'00), pp. 20–35. Springer, Berlin (2000)

53. Holzmann, G.J.: Parallelizing the SPIN model checker. In: Proceedings of the 19th International Workshop on Model Checking Software (SPIN'12). LNCS, vol. 7385, pp. 155–171. Springer, Oxford (2012)
54. Holzmann, G.J.: Proving properties of concurrent programs. In: Proceedings 20th International Symposium on Model Checking Software (SPIN'13). LNCS, vol. 7976, pp. 18–23. Springer, Berlin (2013)
55. Holzmann, G.J., Bošnački, D.: The design of a multicore extension of the SPIN model checker. IEEE Trans. Softw. Eng. **33**(10), 659–674 (2007)
56. Holzmann, G.J., Joshi, R., Groce, A.: Swarm verification techniques. IEEE Trans. Softw. Eng. **37**(6), 845–857 (2011)
57. Isberner, M., Howar, F., Steffen, B.: Learning register automata: from languages to program structures. Mach. Learn. **96**(1–2), 65–98 (2014)
58. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
59. Kwiatkowska, M.Z.: Model checking for probability and time: from theory to practice. In: Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS'03), pp. 351–360. IEEE Computer Society Press, Piscataway (2003)
60. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic model checking. In: Formal Methods for Performance Evaluation—7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'07), Advanced Lectures. LNCS, vol. 4486, pp. 220–270. Springer, Berlin (2007)
61. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591 (2011)
62. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Softw. Eng. **3**(2), 125–143 (1977)
63. Lee, I., Kannan, S., Kim, M., Sokolsky, O., Viswanathan, M.: Runtime assurance based on formal specifications. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), pp. 279–287. CSREA Press, Las Vegas (1999)
64. Lerda, F., Sisto, R.: Distributed-memory model checking with SPIN. In: Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking, pp. 22–39. Springer, Berlin (1999)
65. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Program. **78**(5), 293–303 (2008)
66. Levin, G.M., Gries, D.: A proof technique for communicating sequential processes. Acta Inform. **15**(3), 281–302 (1981)
67. Lowe, G.: Concurrent depth-first search algorithms. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413, pp. 202–216. Springer, Berlin (2014)
68. Lowe, G.: Concurrent depth-first search algorithms based on Tarjan's algorithm. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-015-0382-1 (2015)
69. Marques-silva, J.P., Sakallah, K.A.: Grasp: a search algorithm for propositional satisfiability. IEEE Trans. Comput. **48**, 506–521 (1999)
70. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. **1**(2), 245–257 (1979)
71. Owicki, S., Gries, D.: Verifying properties of parallel programs: an axiomatic approach. Commun. ACM **19**(5), 279–285 (1976)
72. Reiss, S.P., Tarvo, A.: What is my program doing? Program dynamics in programmer's terms. In: Proceedings of the 2nd International Conference on Runtime Verification (RV'11). LNCS, vol. 7186, pp. 245–259. Springer, Berlin (2011)
73. Savage, S., Burrows, M., Nelson, G., Sobalvarro, P., Anderson, T.: Eraser: A dynamic data race detector for multithreaded programs. ACM Trans. Comput. Syst. **15**(4), 391–411 (1997)
74. Shostak, R.E.: A practical decision procedure for arithmetic with function symbols. J. ACM **26**(2), 351–360 (1979)
75. Stern, U., Dill, D.L.: Parallelizing the Murϕ verifier. In: Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97), pp. 256–267. Springer, Berlin (1997)
76. Stolz, V., Bodden, E.: Temporal assertions using AspectJ. In: Proceedings of the 5th International Workshop on Runtime Verification (RV'05). ENTCS, vol. 144(4), pp. 109–124. Elsevier, Amsterdam (2006)
77. von Essen, C., Giannakopoulou, D.: Analyzing the next generation airborne collision avoidance system. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413, pp. 620–635. Springer, Berlin (2014)
78. von Essen, C., Giannakopoulou, D.: Probabilistic verification and synthesis of the next generation airborne collision avoidance system. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-015-0388-8 (2015)
79. Wijs, A., Bošnački, D.: GPUexplore: many-core on-the-fly state space exploration using GPUs. In: Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14). LNCS, vol. 8413, pp. 233–247. Springer, Berlin (2014)
80. Wijs, A., Bošnački, D.: Many-core on-the-fly model checking of safety properties using GPUs. Int. J. Softw. Tools Technol. Transf. doi:10.1007/s10009-015-0379-9 (2015)
81. Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In: Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16). LNAI, vol. 6355, pp. 481–500. Springer, Berlin (2010)
82. Zhang, L., Madigan, C.F., Moskewicz, M.H., Malik, S.: Efficient conflict driven learning in a Boolean satisfiability solver. In: Proceedings of the 2001 IEEE/ACM International Conference on Computer Aided Design (ICCAD'01), pp. 279–285. IEEE Computer Society Press, Piscataway (2001)