Abstract This special section contains the revised and expanded versions of eight of the papers from the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) held in March/April 2004 in Barcelona, Spain. The conference proceedings appeared as volume 2988 in the Lecture Notes in Computer Science series published by Springer. TACAS is a forum for researchers, developers and users interested in rigorously based tools for the construction and analysis of systems. The conference serves to bridge the gaps between different communities – including but not limited to those devoted to formal methods, software and hardware verification, static analysis, programming languages, software engineering, real-time systems, and communications protocols – that share common interests in, and techniques for, tool development. Other more theoretical papers from the conference are collected in a special section of the Theoretical Computer Science journal.

Keywords Tools · Verification · State spaces · Model-checking · Program analysis · Safety analysis · Error explanation

The development of today’s computer systems is a challenging and demanding task. Most systems possess concurrency and non-determinism, which means that the execution may proceed in many different ways, e.g. depending on whether messages are lost or delayed during transmission, the scheduling of processes, and the time at which input is received from the environment. This means that the systems have a very large number of possible execution sequences, and that it is extremely easy for a human designer to miss some important interaction patterns, leading to gaps or malfunctions. As a result computer systems are, by nature, complex and difficult to design, debug and test. Furthermore, many systems are safety-critical or at least very critical for the production and economy of the companies where they are used. To cope with the complexity of today’s computer systems – not to speak about tomorrow’s – it is therefore crucial to provide methods that enables modelling, debugging and testing of central parts of the system design.

The papers in this special section describe different ways to improve the design, debugging and testing of computer systems. The first paper [1] documents an invited talk at the TACAS conference. It uses the $2 \times 2 \times 2$ Rubik’s cube (with 3.67 million different states) for a very elegant and intriguing discussion of a variety of techniques to obtain efficient data structures and algorithms for the construction of state spaces. The case study shows that data structures picked from high-quality libraries may be very inefficient for a particular application. By designing an optimised application-specific data structure one can get a tremendous improvement in both speed and memory consumption at the same time. The paper also demonstrates the usefulness of an information-theoretic view both for the design of efficient data structures for large sets, and for understanding the fundamental limits of memory-efficiency. The basic idea is that, given a certain amount of memory, one can have only a limited number of bit combinations and one should try to exploit them in the best possible way. Finally, the paper develops a new very tight hash table data structure, and discusses the performance of perfect state packing and BDDs in this application.

The second paper [2] deals with a state space reduction technique called the sweep-line method. During the construction of a state space only a small part of the state space is kept in memory, while other parts may be garbage collected shortly after their construction and analysis. In the context of Petri nets, the paper proposes an automated construction of a progress measure which is an important pre-requisite for the sweep-line method. The construction
is based on linear-algebraic techniques concerning the transition vectors of the Petri net. A number of examples illustrates that an automated construction of progress measures may lead to similar performance as user-defined progress measures. The paper also discusses the possible combination of the sweep-line method with other state space reduction techniques, such as partial order reduction and reductions by means of symmetries.

The third paper [3] extends the status quo of timed automata abstractions by contributing several new abstractions. For verification purposes, one usually applies zone based abstractions with respect to the maximal constants to which the clocks of the timed automaton are compared. The paper shows that by distinguishing maximal lower and upper bounds, significantly coarser abstractions can be obtained. Soundness and completeness of the new abstractions with respect to reachability is shown. The paper also demonstrates that information about lower and upper bounds can be used to optimise the algorithm for bringing a difference bound matrix into normal form. Finally, it is experimentally demonstrated that the new techniques dramatically increases the scalability of the realtime model checker UPALL.

The fourth paper [4] compares two different approaches for transient analysis of stochastic systems: numerical analysis based on uniformisation and statistical techniques based on sampling and simulation. The paper compares the two techniques with respect to the verification of time-bounded until formulae in the temporal stochastic logic CSL. This is done both theoretically and through empirical evaluation on a set of case studies taken from the literature on performance evaluation and probabilistic model checking. The focus is on time-bounded properties as these are the type of properties most suitable for statistical methods. The paper observes that CSL model checking is a hypothesis testing problem rather than a parameter estimation problem. This makes it possible to rely on highly efficient sequential acceptance sampling tests, which enables statistical solution techniques to quickly return a result with some uncertainty. The paper also proposes a novel combination of the two solution techniques for verifying CSL queries with nested probabilistic operators.

The fifth paper [5] describes a (semi-)automated approach for assisting users in understanding and isolating errors in ANSI C programs. When a system does not satisfy a specification, a model checker will typically automatically produce a counterexample trace that shows a particular instance of the undesirable behaviour. Unfortunately, the important steps that follow the discovery of a counterexample are generally not automated. The user must first decide if the counterexample shows genuinely erroneous behaviour or is caused by improper specification or abstraction. In case the error is real, the difficult task of understanding the error well enough to isolate and modify the faulty aspects of the system remains. The approach taken in this paper is based on distance metrics for program executions and is derived from David Lewis’ counterfactual approach to causality. Experimental results show that the power of the model checking engine can be used to provide assistance in understanding errors and to isolate faulty portions of the source code.

The sixth paper [6] presents an automated and configurable technique for runtime safety analysis of multi-threaded programs which is able to predict safety violations from successful executions. Based on a formal specification of safety properties provided by a user, the technique automatically instruments a given program and creates an observer so that the program emits relevant state update events to the observer and the observer checks these updates against the safety specification. The events are stamped with dynamic vector clocks, enabling the observer to infer a causal partial order on the state updates. All event traces that are consistent with this partial order, including the actual execution trace, are then analysed online and in parallel. A warning is issued whenever one of these potential traces violates the specification. The technique is scalable and can provide better coverage than conventional testing but its coverage need not be exhaustive.

The seventh paper [7] presents a method for automatic verification of liveness properties for parameterised systems. The method is based on automatic computation of the assertions needed, by a deductive rule, according to the analysis of a small instance of the problem. Then, using a small model theorem, the verification conditions of the deductive rule are discharged using BDD techniques on a (sometimes not so) small instance of the parameterised system. Being able to discharge the verification conditions on a finite model has the advantage that the user never gets to see the assertions. The problem of uniform verification of parameterised systems is undecidable, thus it is impossible to develop a method that always succeeds. When the method does not work immediately, it may help to obtain tighter abstractions.

The eighth paper [8] describes how specifications written in the Java Modelling Language (JML) can be validated using a customised model checking framework, built using the Bogor model checker. Assertions provide a form of lightweight checkable specification that can be very effective in finding defects in programs and in guiding developers to the cause of a defect. The paper identifies the need for customised state-space representations and reduction strategies in model checking frameworks in order to effectively check the kind of strong behavioural specifications that can be written in JML. The paper discusses the advantages and tradeoffs of model checking relative to other specification validation techniques, and presents data that suggest that the cost of model checking strong program specifications is practical for several real programs.

References

1. Valman A (2005) What the small Rubik’s cube taught me about data structures, information theory, and randomisation