Editorial

Daniel Kroening¹, Tiziana Margaria² and Jim Woodcock³

¹ Department of Computer Science, University of Oxford, Oxford, UK. E-mail: kroening@cs.ox.ac.uk

² Chair Service and Software Engineering, Universität Potsdam, Potsdam, Germany

³ Department of Computer Science, University of York, York, UK

The importance of verification for software products is being increasingly appreciated in industry, although still not as much as necessary to become a standard development approach for industrial-scale high-quality software. In 2005, a global initiative was started by eminent researchers in both industry and academia, with the aim of establishing and disseminating a culture of software verification from the first principles by means of theories, tools and experiments. This special issue contains a selection of contributions originally presented at the 2008 Workshop on Tools at VSTTE 2008, the conference on Verified Software: Theories, Tools and Experiments in Toronto.

The VSTTE series of conferences and workshops focuses on the challenge of verifying software systems. Within VSTTE, the scope of the Tools workshop includes implementations and enabling techniques for program verifiers, which are important ingredients for the dissemination of principles and techniques among industrial practitioners. This special issue complements a sister special issue of the Journal on Software Tools For Technology Transfer (STTT) [STT10]. The FAC papers address the foundational aspects of tool-based verification, whereas the STTT selection focuses on practical aspects.

The general public perceives the quality of software products as a major issue. In fact, the cost of software construction is dominated by the process of debugging it and validating that the software meets the desired requirements. Due to the prohibitive cost of manual inspection, it is widely believed that computers themselves need to be part of the solution. To this end, Tony Hoare's Grand Challenge for computing research proposes the *Verifying Compiler*, that is, computer-implemented algorithms that validate the correctness of a given program [Hoa03].

In the Manifesto of the Grand Challenge, presented at VSTTE 2005 [MW08, Coo07], the first in the series of VSTTE conferences and workshops, Tony Hoare and Jay Misra directly recognise and appraise the importance of tools as vehicles for the transmission of knowledge to practitioners. In the second paragraph of the introduction, they write:

"This paper argues that the time is ripe to embark on an international Grand Challenge project to construct a program verifier that would use logical proof to give an automatic check of the correctness of programs submitted to it. Prototypes for the program verifier will be based on a sound and complete theory of programming; they will be supported by a range of program construction and analysis tools; and the entire toolset will be evaluated and evolve by experimental application to a large and widely representative sample of useful computer programs. The project will provide the scientific basis of a solution for many of the problems of programming error that afflict all builders and users of software today."

The paper also suggested that the achievement of this vision should be accelerated by a major international research initiative, modelled on a Grand Challenge, with specific measurable goals. The suggested measure was *one million lines of verified code*, together with its specifications, designs, assertions, and other artifacts.

Today, 6 years later, we are still far away from the fully automatic verification of a million lines of code, but we have achieved other important goals. They are beneficial to the community of software designers, but also, in our opinion more importantly, beneficial to software users. The verification community has identified benchmarks, tools and technologies that are useful for attacking a series of problems, and has started to compare those tools. It has started to provide libraries of code with verified properties, which are worth much more than the sum of the lines of code they comprise, since they are going to be used over and over again in a large number of artifacts, making them and the products in which they are included more reliable and thus of better quality.

The Grand Challenge spawned a series of Working conferences and workshops on Verified Software: Theories, Tools and Experiments, whose organisational structure follows the three research strands which have been identified as the three main ingredients of the grand challenge research programme [HMLS09]:

1. A comprehensive *theory* of programming.
2. A coherent set of *tools* that automate the theory and scale up to the analysis of industrial-strength software.
3. *Experiments* in verified software research, that is, a collection of realistic, verified programs.

From the VSTTE Workshop Tools 2008, we have selected two collections of papers that are now appearing as Special Sections of the *International Journal on Software Tools for Technology Transfer (STTT)* (four papers) [ABH⁺10, GC10, CRJ⁺10, Cok10] and the present issue of *Formal Aspects of Computing* (four papers). The papers in Formal Aspects of Computing focus on the theoretical and methodological aspects of the construction of verification tools, whereas the STTT issue presents leading work in the area of tools for verifying software.

Assuring property conformance of code generators via model checking

This paper was contributed by Sven Jörges et al. [JMS11]. The context of the paper is model-driven design. One of the key ideas of model-driven design is the automatic generation of the models at the lowest abstraction levels from the more abstract, high-level models. In particular, executables are typically obtained by means of automated code generators. The use of automated code generators is commonplace in numerous safety-critical domains including automotive, railways, and avionics. Code generators have to support a rich set of modeling constructs and employ sophisticated optimisations in order to deliver competitive execution performance and efficient usage of resources. The safety-critical nature of the application combined with the complexity of the code generators themselves naturally raises the issue of correctness of the generated code. The complexity of the code generators frequently prevents the application of verification technology to the code generators themselves; numerous safety arguments therefore fall back to verifying the *output* of the code generator, as opposed to verifying the code generator itself ([PSS98], as example).

The paper contributed by Jörges et al. addresses this issue in the context of the *Genesys* framework. This framework enables the creation of code generators from a library of well-defined building blocks. The key insight in the context of the present paper is that this well defined structure dramatically simplifies the verification of the code generators that are obtained in this way. The formal basis for representing instances of code generators produced by the framework are Kripke Transition Systems (KTSs). This formalisation enables the use of automated Model Checkers to verify that the code generators satisfy user-specified properties given in a temporal logic.

Building a push-button RESOLVE verifier: progress and challenges

This paper, contributed by Murali Sitaraman et al. [SAA⁺11], addresses the key problem of automation in software verification. The paper focuses on the RESOLVE verifier, which is maintained by the authors. In this framework, source code that is annotated with suitable specifications and contracts is given to a verification condition (VC) generator. The VC generator outputs a set of verification conditions, which are formulas that have to be shown to be valid in order to conclude that the code satisfies its specifications and contracts. Even a moderate-sized program can give rise to a substantial number of non-trivial VCs; the contributed paper addresses the question of how to minimise the effort of proving the VCs.

The paper first provides an overview of how VCs are generated from a given annotated code fragment. The RESOLVE framework supports a broad range of proof engines, including the Isabelle proof assistant [NPW02], the SMT solver Z3 [dMB08], and various custom-built decision procedures. The paper focuses the discussion on two of these, namely the use of Isabelle and an automated prover called SplitDecision.

A model checking-based approach for security policy verification of mobile systems

This paper addresses the growing domain of software for mobile systems, and was contributed by Braghin et al. [BSBA11]. The key feature of such systems is the distribution of program execution and therefore feature an explicit notion of *where* (which device) a thread is currently executed. The execution of code from a large and diverse number of untrusted sources is commonplace in this scenario. The paper therefore provides a formalisation of mobile systems as Labelled Kripke Structures together with a generic language for specifying security policies. These policies capture the actions that a software is permitted to perform when in a particular location.

The specification together with the program is handed to a prototypical framework for model checking the program against the security policy. The model checker is based on the SATABS framework [CKSY04] and implements the well-known counterexample-guided abstraction refinement approach together with predicate abstraction. Predicate abstraction has been promoted by the success of the SLAM project at Microsoft Research [BCLR04] and is typically applied to verify control-flow dominated properties of device drivers.

Verification of distributed systems with local–global predicates

The final paper, contributed by Mani Chandy et al. [CGM⁺11], focuses on a particular class of distributed systems: the basic assumption is that most state changes only affect few of the components. The paper first develops a theory that enables global guarantees such as system-level invariants and convergence. These guarantees are obtained by establishing local properties on individual components. The model includes low-level artifacts that are highly relevant to system engineers such as faults in communication that result in lost, delayed or re-ordered messages. The theorems presented in the paper have been formalised in the PVS theorem proving system. A number of examples presented in the paper were manually translated from PVS into JAVA; the automation of this step is suggested as future work.

References

- [ABH⁺10] Abrial J-R, Butler M, Hallerstede S, Son Hoang T, Mehta F, Voisin L (2010) Rodin: an open toolset for modelling and reasoning in Event-B. *Int J Software Tools Technol Trans (STTT)* 12(6):447–466
- [BCLR04] Ball T, Cook B, Levin V, Rajamani SK (2004) SLAM and static driver verifier: technology transfer of formal methods inside Microsoft. In: *Integrated Formal Methods (IFM)*, vol. 2999, pages 1–20. *Proceedings of Lecture Notes in Computer Science*. Springer, Berlin
- [BSBA11] Braghin C, Sharygina N, Barone-Adesi K (2011) A model checking-based approach for security policy verification of mobile systems. *Formal Asp Comput*
- [CGM⁺11] Mani Chandy K, Go B, Mitra S, Pilotto C, White J (2011) Verification of distributed systems with local–global predicates. *Formal Asp Comput*
- [CKSY04] Clarke Edmund M, Kroening D, Sharygina N, Yorav K (2004) Predicate abstraction of ANSI-C programs using SAT. *Formal Methods Syst Design* 25(2–3):105–127
- [Cok10] Cok D (2010) Improved usability and performance of SMT solvers for debugging specifications. *Int J Software Tools Technol Trans (STTT)* 12(6):467–481
- [Coo07] Cooke J (2007) Editorial (VSTTE special issue). *Formal Asp Comput* 19(2):137–138
- [CRJ⁺10] Chalin P, Robby, James P, Lee J, Karabotsos G (2010) Towards an industrial grade iver for java and next generation research platform for jml. *Int J Software Tools Technol Trans (STTT)* 12(6):429–446
- [dMB08] de Moura LM, Bjørner N (2008) Z3: an efficient SMT solver. In: Ramakrishnan CR, Rehof J (eds) *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29–April 6, 2008 proceedings*, vol. 4963, pp. 337–340. *Lect Notes Comput Sci*. Springer, Berlin
- [GC10] Gurfinkel A, Chaki S (2010) Combining predicate and numeric abstraction for software model checking. *Int J Software Tools Technol Trans (STTT)* 12(6):409–427
- [HMLS09] Hoare CAR, Misra J, Leavens Gary T, Shankar N (2009) The verified software initiative: a manifesto. *ACM Comput Surv* 41(4)
- [Hoa03] Hoare CAR (2003) The verifying compiler: a grand challenge for computing research. *J. ACM* 50(1):63–69
- [JMS11] Jörges S, Margaria T, Steffen B (2011) Assuring property conformance of code generators via model checking. *Formal Asp Comput*
- [MW08] Meyer Bertrand, Woodcock Jim (eds) (2008) Volume 4171 of *Lecture Notes in Computer Science*. Springer, Berlin
- [NPW02] Nipkow T, Paulson LC, Wenzel M (2002) Isabelle/HOL: a proof assistant for higher-order logic, vol. 2283. *LNCS*, Springer

- [PSS98] Pnueli A, Strichman O, Siegel M (1998) Translation validation for synchronous languages. In: Larsen KG, Skyum S, Winskel G (eds) Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13–17, 1998 proceedings, Volume 1443 of Lecture Notes in Computer Science, pages 235–246. Springer, Berlin
- [SAA⁺11] Sitaraman M, Adcock B, Avigad J, Bronish D, Bucci P et al (2011) Building a push-button RESOLVE verifier: progress and challenges. *Formal Asp Comput*
- [STT10] (2010) *Int J Software Tools Technol Trans (STTT)* 12(6)

Daniel Kroening, Tiziana Margaria, and Jim Woodcock

Published online 21 August 2011