Software Construction using Executable Constraints

Presented by: Viktor Kuncak
Joint work with: Ali Sinan Köksal, Ruzica Piskac, and Philippe Suter
Swiss Federal Institute of Technology Lausanne (EPFL)

Constructing software that satisfies the desired properties can greatly benefit from constraint solvers based on satisfiability modulo theories (SMT) paradigm. We propose a research program in which software construction and SMT technology become even more interconnected than today. Instead of checking properties of low-level legacy programs, new generations of SMT solvers could focus on constructing programs and values within compilers and run-time systems of declarative programming languages designed with constraing solving in mind. The result will be programming systems that support implicit programming, in which computations can be given as specifications and not only as directly executable code. This research agenda has implications on both theories and interfaces of future SMT solvers.

The theories of interest come in part from functional programming languages and data structure libraries. This includes algebraic data types (term algebras), arrays, maps, sets, and multisets. Specialized techniques for new theories will likely continue to be required in the future. While simple versions of set and multiset theories can be efficiently encoded into arrays, the addition of a size operator makes the theory much richer and leads to interesting algorithmic and complexity-theoretic questions. Non-disjoint theory combination will likely be necessary for expressive domains; one way to understand them are reductions to known decidable theories such sets or integer linear arithmetic.

New theories of interest arise by adding executable functions on data types, with recursive schemas such as map, fold, and their generalizations. For such executable functions we obtain immediately semi-decidability of satisfiability for quantifier-free formulas, and, under certain cases, decidability. I illustrate the effectiveness of this approach by outlining a verification system, Leon, which proves and disproves properties of recursive functional programs written in a subset of the Scala programming language. Leon is complete for counterexample generation as well as for proofs involving sufficiently surjective functions.

New interfaces for SMT solvers are driven by their extended role. In addition to yes/no satisfiability answers, we need synthesis of values and programs. To answer declarative queries at run-time, we need efficient generation and fair enumeration of models, as well as their mapping to programming language values. Models of interest may contain structured values such as sets, algebraic data types, and arrays. In contrast to applications in verification, model generation functionality is no longer merely for debugging or test generation, but becomes a core computation mechanism. As a result, it must be predictable and integrated into the programming language. I outline a system, named Kaplan, which incorporates SMT-backed constraint programming into the Scala programming language. The core of Kaplan is the counterexample generation mechanism of Leon. To integrate constraint solving into the programming language, Kaplan introduces constraints as first-class objects as well as logical variables that delay constraint solving until a concrete solution

is needed. Kaplan ensures a fair enumeration of solutions by ordering them with respect to a given measure function. Potential uses of Kaplan range from execution of declarative specifications to test generation and constraint optimization.

When an SMT solver is used to find values that satisfy given queries, it acts as an interpreter. An SMT solver could also be used as a compiler, if it can solve parameterized constraints. We demonstrate compilation of constraints through an algorithm for complete functional synthesis implemented in the Comfusy tool. Comfusy compiles implicit constraints into standard executable code. The algorithm is based on a constructive variant of quantifier elimination for Presburger arithmetic. To support constraints on sets, it extends the reduction approach from satifiability checking to synthesis.

To help applications such as Comfusy in the future, solvers need to support parameterized constraints and generation of terms as solutions, as in quantifier elimination and in simplification functionality similar to the one in computer algebra systems. Due to the dynamic nature of constraint solving and the essential role of learning, profile-guided and just-in-time compilation approaches are likely to be important for efficient constraint programming. To support such applications, SMT solvers need to become virtual machines for constraint programming, working with data structures directly manipulated by programs, and performing selective specialization of frequently invoked constraint solving paths. To exploit the power of such solvers, programming languages will need to favor data types that have good constraint solving properties.

These new directions suggest that engineering challenges for future SMT solvers are huge. If they can be tackled, however, SMT solvers will play as important role in the next few decades as compiler technology played in the past ones, and will lead to a qualitatively higher level of software development.

References

[1] Leonardo de Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *FMCAD*, 2009.

[2] Viktor Kuncak, Mikael Mayer, Ruzica Piskac, and Philippe Suter. Complete functional synthesis. In *PLDI*, 2010.

[3] Viktor Kuncak, Ruzica Piskac, Philippe Suter, and Thomas Wies. Building a calculus of data structures. In *VMCAI*, volume 5944 of *LNCS*, 2010.

[4] Ruzica Piskac and Viktor Kuncak. Linear arithmetic with stars. In *CAV*, volume 5123 of *LNCS*, 2008.

[5] William Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, 1992.

[6] Philippe Suter, Ali Sinan Köksal, and Viktor Kuncak. Satisfiability modulo recursive programs. In *Static Analysis Symposium (SAS)*, 2011.

[7] Thomas Wies, Ruzica Piskac, and Viktor Kuncak. Combining theories with shared set operations. In *FroCoS: Frontiers in Combining Systems*, 2009.