Report on NSF Workshop on Symbolic Computation for Constraint Satisfaction

November 14, 2008

Alex Aiken, Stanford University
Rajeev Alur, University of Pennsylvania
Clark Barrett, New York University
Martine Ceberio, University of Texas at El Paso
Werner Damm, Universitat Oldenburg, Germany
Leonardo de Moura, Microsoft Research
Sharad Malik, Princeton University
Ken McMillan, Cadence Labs
Jose Meseguer, University of Illinois at Urbana-Champaign
Andre Platzer, Carnegie Mellon University
Amir Pnueli, New York University
Thomas Reps, University of Wisconsin, and GrammaTech Inc.
Bart Selman, Cornell University
N. Shankar, SRI

1 Introduction

The workshop on **Symbolic Computation for Constraint Satisfaction** was held at the FDIC Building in Arlington, VA, on November 14, 2008. The workshop was focused on the role of symbolic decision procedures for constraint satisfaction problems. The workshop brought together computer science researchers in diverse areas such as program analysis, formal verification, AI planning, optimization, and hybrid systems. The workshop was organized by Rajeev Alur (University of Pennsylvania). Lenore Mullin, a program manager at NSF, opened the workshop by describing her long standing interest in numeric and symbolic computation. The morning session, titled **Computational Tools for Constraint Satisfaction Problems** consisted of a series of talks that reviewed the state of the art in symbolic decision procedures. Talks in this session were given by Clark Barrett (New York University), Martine Ceberio (University of Texas at El Paso), Sharad Malik (Princeton University, Jose Meseguer (University of Illinois at Urbana-Champaign), and Bart Selman (Cornell University). The afternoon session, titled **Role of Constraint Satisfaction in Computer Science Problems**, explored various problem domains where constraint satisfaction plays a central role. This session consisted of talks by Alex Aiken (Stanford University), Werner Damm (Universitat Oldenburg, Germany), Leonardo de Moura (Microsoft Research), Ken McMillan (Cadence Labs), and Andre Platzer (Carnegie Mellon University). The workshop concluded with a panel focused on challenges for future research. The panelists were Amir Pnueli (New York University), Thomas Reps (University of Wisconsin, Madison), and N. Shankar (SRI).

This report briefly summarizes the themes discussed in the talks, challenges and opportunities for future research, and the need for focused funding from NSF to support this area. The appendix includes talk titles and abstracts. Information about the workshop, along with the talk presentations, is available publicly at `http://www.cis.upenn.edu/~alur/nsfsymbolic08.html`.

2 State of the Art

The talks and discussion at the meeting touched upon a wide range of topics. The themes included the following:

Diversity of Constraint Satisfaction Problems The most common constraint satisfaction problems are *propositional satisfiability* SAT (deciding if a formula over boolean variables, formed using logical connectives, can be made True by choosing 0/1 values for the variables), and *linear programming* (optimizing a linear cost function over a set of real-valued variables while satisfying a conjunction of linear constraints). Participants discussed a number of more general forms of constraint problems. These included *Quantified Boolean Formulas* (QBF), first-order formulas with respect to specific logical theories (commonly known as *Satisfiability Modulo Theories* SMT), polynomial as well as non-linear constraints, interval constraints, and set constraints (boolean combinations of inclusion constraints over set-valued variables).

Algorithms and Tools The core computational complexity of most constraint satisfaction problems is high (e.g., SAT is NP-complete). As a result, many decision procedures for such problems involve manipulation of *symbolic* representation of the solution space. In recent years, there has been an enormous progress in the scale of problems that can be solved, thanks to innovations in core algorithms, data structures, as well as attention paid to implementation details such as caching. For example, modern SAT solvers can check formulas with

10,000s of variables and millions of clauses. Rapid progress has also been reported for SMT solvers for the more commonly occurring theories. The annual competition for such solvers is also a key ingredient in driving progress (see Figure 1 for progress in scale of problems). Both SMT and Rewriting are now emerging as unified frameworks for solving constraint satisfaction problems. The participants also reported on a number of new algorithmic ideas. For example, integrating numeric and interval-based symbolic techniques has led to new tools for solving non-linear constraints. For propositional formulas, algorithms based on probabilistic sampling have led to dramatic performance gains and new insights into the (#P-complete) problem of counting the number of satisfying assignments.

Applications Constraint satisfaction problems arise in many diverse areas of computer science. The application to analysis and formal verification of software and hardware was discussed by many speakers. Sample problems include symbolic model checking, generation of invariants for programs, (symbolic) testing of programs, type inference, compiler optimizations, and program analysis. The improvements in tools for constraint satisfaction problems have directly resulted in adoption of formal methods at companies such as Intel and Microsoft. Another key application area is *planning*. Contemporary AI planners formulate generation of a plan as a constraint satisfaction problem, and use modern SAT solvers for real-world applications. Constraints over real-valued variables arise in modeling and analysis of dynamical and hybrid systems. These problems have been applied to diverse areas ranging from air-traffic control to systems biology. We also got a nice perspective of how constraint satisfaction is a central technical theme in the Transregional Collaborative Research Center AVACS funded by the German Science foundation.

3 Research Challenges

Despite enormous progress in tools for, and practical applications of, constraint satisfaction problems, many challenges remain. The following is a partial list of some of the opportunities that were articulated at the workshop. The individual talk abstracts in the appendix discuss these opportunities in more details.

1. A number of heuristic strategies have been proposed, and shown to work on real-world problems, for problems such as SAT. However, we have a limited understanding of why some of these strategies work. A better understanding of the relationship between effectiveness of different strategies and problem structure, can also lead to algorithms that dynamically change strategies.

2. For SAT as well as SMT tools, the current tools cannot handle quantifiers well, and sustained effort is needed to improve the state of the art for solving formulas with nested quantifiers.

3. Handling non-linear constraints efficiently is a key challenge. Such constraints are essential in analyzing hybrid systems, and some recent techniques for combining numeric and symbolic methods show promise.

4. Integration of different decision procedures and tools requires significant efforts. Standardized libraries, common input formats for competitions, and unifying frameworks such as SMT and rewriting, are steps in the right direction.

5. Robustness of the tools remains a barrier for effective use of the tools by practicing engineers. Small perturbations in the input constraints can change running times of the tools dramatically. Understanding sources of numerical errors is also essential for robustness.

6. Researchers developing tools for symbolic analysis should also focus on *usability* of tools. This aspect is many times ignored by researchers, but is necessary for wider dissemination of this technology.

7. Symbolic computation should be viewed more broadly than (symbolic) decision procedures for specific logics. For example, many components of the abstract interpretation framework are essentially symbolic, and are necessary for simplifying the analysis problem before it is handed off to a decision procedure.

8. New algorithmic insights have been central to the advances in the tools for constraint satisfaction problems. Fundamental research in algorithms and complexity of constraints satisfaction problems should be continued. For example, the exact complexity of the entailment problem arising in program analysis is still unknown. The science of which instances of propositional satisfiability are "easy" is still emerging.

4 Conclusions

The diversity of the problems discussed at the workshop brought into sharp focus the fact that many computing problems, from verification, program analysis, and software testing, to planning, scheduling, and optimization, are indeed constraint satisfaction problems. Research in developing efficient symbolic tools for these problems is vibrant, with demonstrable technology transfer. We recommend that NSF should recognize the importance and foundational nature of this area, and provide sustained and focused funding dedicated to it.

Appendix: Talk Abstracts

Alex Aiken, Stanford University
Constraint Problems in Program Analysis

A great variety of constraint resolution problems arise in the area loosely referred to as *program analysis*. The general approach is to reduce some problem stated in terms of properties of a program represented in some particular way (e.g., as an abstract syntax tree or a control-flow graph) into a mathematical problem about the existence of solutions to a system of constraints. Many useful constraint theories and associated algorithms are known, but there are also a number of topics that are of central importance and that are also not well understood:

The ability to analyze programs in pieces and then compose the information derived from the pieces to construct information about the entire program is often called local or compositional analysis. Local analysis depends upon being able to express in some formalism all behaviors of a program fragment in all possible environments. We can see immediately that to be feasible the size of such descriptions (i.e., the constraints) cannot vastly exceed the size of the fragments of program being analyzed. Thus, there must be a great deal of sharing among the solutions (or very few distinct solutions) for local analysis to be practical. If instead there are many possible, and incomparable, behaviors, then one is better off doing whole-program analysis, and reanalyzing the program fragment once for each of the actual contexts in which the program uses it (which will generally be much, much smaller than the set of all possible contexts). The trade-offs between when one can do local and when one must do global analysis are not well-understood today; there are no guidelines for people designing analyses to tell them which they should prefer, or whether a local analysis is even possible.

A second topic that is under-investigated is efficient techniques for implementing constraint resolution algorithms in practice. Experience in many domains has shown that the fact that a constraint resolution problem is in theory intractable does not mean that there is not structure in practical instances that can be exploited. Exploitation of such structure is often the difference between a particular decision procedure being useful for realistic examples and being a curiosity.

A third topic is the need for a better understanding of entailment problems. While constraint solving is well-understood for many kinds of constraints that arise in program analysis problems, constraint entailment is much less well-understood. Entailment is always need to, for example, to optimize the representation of constraints or to ask queries about a system of constraints, but for many important program analysis problems there are open problems about the exact complexity of the entailment problem.

Clark Barrett, New York University
Satisfiability Modulo Theories: Successes and Challenges

Automatic analysis of computer hardware and software requires engines capable of reasoning efficiently about large and complex systems. A primary goal of research in Satisfiability Modulo Theories (SMT) is to create verification engines that can reason natively at a higher level of abstraction. This facilitates ease of use and in many cases also leads to substantial performance gains over the more naive approach of encoding problems as Boolean satisfiability (SAT) formulas.

The general SMT problem is that of checking the satisfiability of first-order formulas with respect to some logical theory T of interest. Several important theoretical and practical results

have led to systems that can efficiently integrate Boolean reasoning with reasoning in multiple theories.

Research in SMT has been gaining momentum in recent years. Important developments include a set of standards (the SMT-LIB standard), a yearly workshop, and a yearly competition. The performance of SMT solvers has increased significantly, and these solvers are now being used in an increasing number of industrial applications.

There remain many challenges. These include: exploring new theories and decision procedures; finding better ways of integrating SAT and theory reasoning; both theoretical and practical efforts to improve completeness when reasoning with quantifiers and non-linear arithmetic; engineering more mature tools; investigating parallelization of SMT algorithms; and improving reliability via proofs. SMT solvers are creating capabilities in verification that were completely unrealistic only a few years ago. It is critical that the momentum in this area be maintained and accelerated with funding for research to meet these challenges.

Martine Ceberio, University of Texas at El Paso
Symbolic-Numeric Algorithms for Constraint Solving

Continuous constraints are constraints whose variables are defined over continuous domains. This is the only notable difference between discrete and continuous constraints. However, this small difference results in the need for very different solving techniques. Indeed, when enumeration (and fancy, more efficient variations of it) is an option with discrete constraints, it is no longer possible when variables can take an infinite amount of values.

In this talk, I present several approaches to address this problem, and I focus on the approaches that use interval computations to model the continuity of the variables' domains. I also introduce the problem of computations performed by computers and their limitations to floating-point numbers. All of these limitations, along with these of interval computations (dependency of the computations), call for the combination of numeric and symbolic algorithms to solve continuous constraints.

I point out that the numeric part of the solving algorithms is indispensable, since the problems at hand are numeric in essence. The symbolic part of these same solving algorithms is essential if we seek efficiency. I show several attempts to speed up the constraint solving process using symbolic algorithms.

I conclude by pointing out future research directions, among which the use of a different interval arithmetic (namely, circular arithmetic) that proved to be efficient for interval computations, and whose use should be further investigated in the case of constraint solving algorithms. Besides, tensors also appear to constitute an interesting alternative approach for they provide a nice framework to keep track of variables' dependencies between constraints and are expected to speed up the domain narrowing process.

Werner Damm, Universitat Oldenburg, Germany
Automatic Verification and Analysis of Complex Systems: Key results of the transregional collaborative research project AVACS

The German Science Foundation runs as high-end programs so called Large Scale Collaborative Research Projects (*Sonderforschungsbereiche*, for short SFB), which may involve up to three participating sites (acronym SFB-TRR). The talk presented results of the first funding period of the SFB-

TRR AVACS on Automatic Verification and Analysis of Complex Systems, see `www.avacs.org`, and explained the principles of organization of AVACS projects in combining a rich class of symbolic methods (including Abstraction, Decision Diagrams, Constraint Solving, Heuristic Search, Linear Programming, Model Checking, Lyapunov Method, SMT, Decision Procedures) for the verification of benchmarks taken from the transportation domain.

Leonardo de Moura, Microsoft Research
(SAT & SMT in) Program Verification & Testing

Software verification has been the Holy Grail of computer science for many decades. Tony Hoare suggested the Verification Challenge as an effort to create a tool set that would, as automatically as possible, guarantee that programs meet given specifications. Decision procedures for checking satisfiability of logical formulas are an important component of this tool set. Of particular recent interest are solvers for Satisfiability Modulo Theories (SMT). SMT solvers are used in several projects at Microsoft, the main applications are: program verification, test case generation, white-box fuzz testing, and predicate abstraction. In this talk, I will cover these applications, and discuss the many challenges we face.

Sharad Malik, Princeton University
SAT and QBF: Trick or Treat?

In this talk I will cover the major advances in SAT and QBF solvers over the past decade, with the practical successes (the treat) and the challenges that remain, especially with QBF (the trick). I will highlight the diverse set of techniques from across computer science that has been brought to bear in modern solvers. These include deduction, search, caching, randomization, and data structures/algorithms. In particular, I will emphasize the alternating emphasis between deduction and search between succeeding generations of solvers.

Ken McMillan, Cadence Labs
Symbolic Model Checking

José Meseguer, University of Illinois at Urbana-Champaign
Rewriting Needs Constraints and Constraints Need Rewriting

Rewriting is a very general mechanism for symbolic computation. Solving of constraints is also a very essential part of symbolic computation. Therefore, research in both rewriting and constraint solving are important aspects of symbolic computation research. Furthermore, rewriting and constraint solving are intimately related, need each other, and can help each other in fundamental ways. This can be expressed by the slogan of the above talk title.

Rewriting needs constraints. The need for constraint solving in rewriting techniques is present almost everywhere. Three good example areas are:

1. rewriting modulo axioms A, which needs A-matching algorithms;
2. narrowing modulo axioms A, which needs A-unification algorithms; and
3. inductive reasoning in initial models, where a host of constraint solvers such as unification algorithms, tree automata, and decision procedures for special theories are used.

Constraints need rewriting. The need for rewriting techniques in constraint solving is also very frequent. Three good example areas are:

1. The formal specification of inference systems for constraint solving, where inference rules are rewrite rules, and alternative algorithms can be understood as alternative strategies to apply such rules. For example:

   - semantic unification algorithms for various theories;
   - abstract congruence closure;
   - Groebner basis computation;
   - combinations of decision procedures; and
   - SAT and SMT solvers

   can all be formally analyzed and understood this way. The advantage is to separate correctness issues (based solely on the rewrite rules) from efficiency, complexity, and implementation issues (based on the strategies).

2. A much wider range of applications can often be covered by combining theory-specific constraint solvers with theory-generic rewriting techniques. A good example is narrowing, where:

   - a theory-specific A-unification algorithm can be seamlessly combined with theory-generic narrowing modulo A with equations E to obtain a unification algorithm in the combined theory of A and E under suitable requirements on E.
   - narrowing modulo equations E can be used to perform complete symbolic reachability analyses for systems whose transitions are specified with rewrite rules modulo E. For example, cryptographic protocols can be verified this way with much stronger guarantees than in the syntactic Dolev-Yao model.

3. Using a high-performance rewrite engine, rewriting-based specifications of constraint solvers can be used for rapid prototyping, analysis, debugging, and fast experimental evaluation of such solvers. Furthermore, in a good number of cases the rewrite rules specifying the solver can be directly used for implementation purposes with competitive performance.

In summary, both constraint satisfaction and rewriting are essential methods for symbolic computation. Also, both sets of techniques have many practical applications which greatly broaden the applicability of symbolic computation techniques. Finally, as suggested in the talk title, rewriting and constraint solving need each other very intimately, so that research advances in both areas will benefit each other as well as substantially advancing symbolic computing research as a whole.

Andre Platzer, Carnegie Mellon University
Symbolic Computations in Hybrid Systems Verification

Verification and analysis of hybrid systems is important for several practical safety-critical systems ranging from railway technology over driver assistants in cars to aircraft collision avoidance control systems. Real applications have challenging non-linear dynamics. Contrasting numerical versus symbolic techniques, we analyze limits of approximation techniques for (non-linear) continuous image computation in model checking hybrid systems. In particular, we show that even a single step

of continuous image computation is not semi-decidable numerically even for a very restricted class of functions. Moreover, we show that symbolic insight about derivative bounds provides sufficient additional information for approximation refinement model checking. Finally, we prove that purely numerical algorithms can perform continuous image computation with arbitrarily high probability. We further contrast these numerical techniques with our fully symbolic analysis technique of differential invariants for hybrid systems. Using these results, we analyze the prerequisites for a safe operation of the roundabout maneuver in air traffic collision avoidance.

Amir Pnueli, New York University
A Customer View of Symbolic Computation and Constraint Satisfaction

In this discussion, I attempted to elaborate my view as a customer of Symbolic Decision Procedures and my expectations and requirements from tools that provide this technology, in order to carry out the task of Formal Verification of Reactive Systems.

In our earlier work, we emphasized the methods of Deductive Verification whose main advantage is that, unlike the algorithmic methods of model checking, it is not restricted to finite-state systems, and can verify, in principle, all temporal properties of reactive programs with arbitrarily complex data structures. As is well known, the task of a deductive verifier consists of two major subtasks:

1. The identification and creation of auxiliary constructs, such as inductive invariant assertions, and ranking functions, which are required in most deductive rules for temporal verification.

2. The formal verification of the verification conditions using a theorem prover or a corresponding decision procedure. These are usually first-order implications, using appropriate theories that cover the relevant data structures.

Our conclusions from extensive application of these methods, and continued experience with teaching the deductive technologies to students were that these two tasks or not easy to master. In particular, the more automation that can be introduced to the second sub-task, the better. In fact, in several courses that I have taught, I have intentionally restricted the data domain to cases where reasonable decision procedures exist, in order to emphasize the first sub-task which is the essential element in deductive verification.

At this first stage of activity, we also invested much efforts in the complete formalization for the deductive theory of temporal logic. These efforts required a very powerful and expressive logic in which notions of state sequences and temporal operators over them could be expressed and justified formally. This led to the development of tools such as TLPVS (constructed over the high-order PVS system) which enables us to verify an arbitrary temporal property of a reactive system by deductive rules that reduce the verification task into establishing a set of verification conditions that are first-order implications over a variety of theories.

A more recent trend in the temporal verification of infinite-state reactive system places greater reliance on abstraction methods, in particular finitary abstraction approaches such as predicate and ranking abstractions. According to this method, the original verification problem is reduced by abstraction to the verification of propositional temporal formula over a finite-state system, which can be accomplished by algorithmic methods such as model checking. The role of symbolic methods within this approach to temporal verification is mainly in the computation of the abstracted system. This radically changes the set of requirements we now expect from the symbolic methods. For example, the feature of quantifier elimination becomes highly valuable because it allows the

computation of an abstraction in a complexity that is much better than the exponential complexity that is usually required in cases we do not have quantifier elimination within our theory. On the other hand, we may relax somewhat the restrictive demand of completeness. This is because, even in cases of an incomplete decision procedure, we can always obtain a sound abstraction.

As a last reminder, the increasing recent interest in the verification of Hybrid Systems poses new challenges for the new symbolic computation methods, raising higher our interest in algebraic and semi-algebraic methods as relevant techniques for dealing with this new family of systems.

Bart Selman, Cornell University
Planning as Satisfiability: Progress and Challenges

Thomas Reps, University of Wisconsin and GrammaTech, Inc.
Some Possible Research Challenges

I presented a few ways in which logical/constraint-based/symbolic techniques relate to issues that arise in program analysis – in particular, to abstraction. In particular, I described how abstract domains and logics could be connected via two operations: (1) $\hat{\gamma}$, which converts an abstract element a to a formula $\hat{\gamma}(a)$ that describes exactly the state-set that a represents, and (1) $\hat{\alpha}(\phi)$, which gives the most-precise over-approximation in the abstract domain of the state-set that a formula ϕ represents. These operations provide a starting point for (i) using symbolic techniques in abstract interpretation, (ii) using abstraction in conjunction with symbolic techniques, and (iii) communicating information between abstract values of different abstract domains (by using formulas of the logic as the *communication medium*). This should provide fertile ground for future research. I also made the case that when designing a research program in the area of symbolic computation, NSF should support work not just on satisfiability checking (e.g., for various logics), but should interpret "symbolic computation" more broadly. Examples of other operations of interest include (i) symbolic composition, needed for instance to compose summary functions or summary relations for modular analysis, (ii) widening and other generalization operations, needed for enforcing termination, and (iii) finding loop invariants. In addition, work on logics that are based around operations for expressing changes in state (*structure-update operations*), such as dynamic logic and dynamic descriptive complexity, is particularly worthy of support.

Finally, based on my experience working on analysis and verification of machine code, I expressed the opinion that it was unfortunate that recent work on designing decision procedures for separation logic had chosen to abandon several key features of the original formulation of separation logic, namely, the ability to treat address values as integers and to perform arithmetic on addresses. Such features are particularly important when reasoning about machine code.

N. Shankar, SRI International Computer Science Laboratory
Research Challenges in Constraint Solving

All problem solving is constraint solving. Constraints include systems of equalities and inequalities; planning, scheduling, and optimization problems; geometric constructions; database query processing; computer-aided design; and for creating and solving puzzles like Sudoku and games like Chess and Go. If we had constraint solvers that could handle complex real world constraints, we might expect to see such solvers embedded in everything from vacuum cleaners to airplanes. We now have extremely fast solvers for a useful class of constraints. The research challenges for

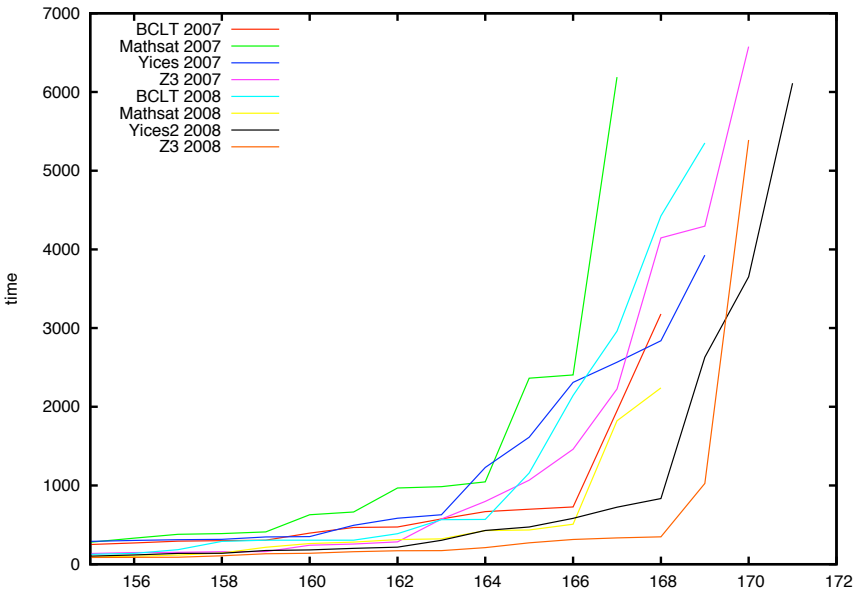

Figure 1: Comparing SMT solvers on the QF_RDL benchmarks from SMT-LIB

constraint solving are in expanding the range of constraints, extracting richer information from these constraints, increasing the efficiency of constraint solvers, and exploring novel applications.

A typical constraint solving problem has the form: find an $\overline{x}$ such that $C(\overline{x})$ holds. Either there is a solution for $\overline{x}$ or we need to demonstrate that there is no such solution. Some problems can be unsolvable or infeasible. We might also require an enumeration of all solutions or the identification of the best solution according to some metric. The constraints $C(x)$ can also be a mixture of hard constraints that must hold of $\overline{x}$ and soft constraints that can be relaxed. Constraints can also involve probability estimates such as the likelihood that $\overline{x} = \overline{a}$ given $C(\overline{x})$, where $C(\overline{x})$ contains some facts and rules. Constraint solving can also include solving *meta-constraints* that involve finding winning strategies for games, ruler-and-compass constructions, programs, program invariants, ranking controllers, functions, Lyapunov functions, or interpolants for the conjunction of two sets of constraints.

Many industrial applications use numeric techniques that compromise on either soundness (good solutions might be missed) or completeness (proposed solutions might not be valid), or both. Symbolic techniques attempt to exactly or approximately represent the set of solutions, and can be used to generate exact solutions if needed. Symbolic constraint solving techniques include search, propagation, inference, saturation, and abstract interpretation. Constraint solving can either indicate solvability, construct a solution, claim the absence of a solution, or provide evidence or proof for the absence of a solution. Constraint solvers can also be incremental and support incremental assertion and retraction.

Constraint solvers have been growing rapidly in efficiency over the last few years. Propositional satisfiability (SAT) solvers can now handle problems with millions of Boolean variables and con-

straints. Solvers for satisfiability modulo theories (SMT) can handle constraints in combinations of theories such as integer and real linear arithmetic equalities, disequalities, and inequalities, arrays, bit vectors, uninterpreted (i.e., unconstrained) function symbols, datatypes, and quantified rules. The performance of SMT solvers has been increasing quite dramatically over recent years. Figure 1 shows the progress from 2006 to 2008. The graph shows the performance for each of eight systems on the quantifier-free real difference logic benchmarks from the SMT-LIB library. For each system, the graph shows its performance on the benchmarks sorted in increasing order of time (in seconds). The Yices 2007 system is essentially the same as the one that took part in the 2006 SMT-COMP competition, and it can be contrasted with Z3 2008 which completes nearly all of the benchmarks in small amount of time.

Ongoing research in SAT and SMT is aimed at efficient algorithms and representations, support for parallelism, richer interfaces, support for nonlinear arithmetic constraints, approaches for combining multiple theories, quantified Boolean formulas, combining hard and soft constraints, probabilistic inference, interpolant generation, and approximation.

Large-scale constraint solving will have a big impact on science and engineering. Applications in hardware and software analysis lead to more robust, reliable, and secure computer systems. Constraint solving can be used in the computer-aided design and manufacturing of electronic and cyber-physical systems. These techniques can also be applied to problems in medical technology, energy, environment, and transportation. Finally, constraint solving can be used in educational software to assist in problem construction and problem solving.