Design and Results of the 1st Satisfiability Modulo Theories Competition (SMT-COMP 2005)

Clark Barrett
Department of Computer Science
New York University

Leonardo de Moura
Computer Science Laboratory
SRI International

Aaron Stump
Department of Computer Science and Engineering
Washington University in St. Louis

Abstract. The Satisfiability Modulo Theories Competition (SMT-COMP) is intended to spark further advances in the decision procedures field, especially for applications in hardware and software verification. Public competitions are a well-known means of stimulating advancement in automated reasoning. Evaluation of SMT solvers entered in SMT-COMP took place while CAV 2005 was meeting. Twelve solvers were entered, 1352 benchmarks were collected in seven different divisions.

Keywords: satisfiability modulo theories, decision procedures, competition

1. Introduction

Decision procedures for checking satisfiability of logical formulas are crucial for many verification applications (e.g., [35, 32, 19, 13, 29, 16, 12, 11]). Of particular recent interest are solvers for Satisfiability Modulo Theories (SMT). SMT solvers decide logical satisfiability (or dually, validity) of formulas in classical multi-sorted first-order logic with equality, with respect to a background theory. The success of SMT for verification applications is largely due to the suitability of supported background theories for expressing verification conditions. These theories include: the empty theory, which gives rise to the so-called logic of equality and uninterpreted functions (EUF) [18, 25]; real or integer arithmetic; and theories of program or hardware structures such as bitvectors [22] and arrays [3, 33]. It is usually necessary to adopt some syntactic restriction on the input formulas to be checked in order to ensure efficient decidability. For example, formulas are often required to be quantifier-free. For arithmetic, more efficient algorithms are known for *difference formulas*, where atomic formulas consist of difference constraints of the form $x - y \leq c$, with x and y variables and c a numeric constant [28, 2]. Many solvers further increase expressivity by taking

the background theory to be a combination of several individual theories, and some solvers include limited support for quantified formulas, further increasing expressive power.

The Satisfiability Modulo Theories Competition (SMT-COMP) is intended to spark further advances in the SMT field, especially for applications in verification. Public competitions are a well-known means of stimulating advancement in automated reasoning. Examples include the CASC Competition [24] for first-order reasoning, the SAT Competition for propositional reasoning, and the Termination Competition for checking termination of term rewriting systems [24, 10]. Participants report that competitions fuel significant improvements in tool capabilities from year to year.

The idea of holding SMT-COMP came out of discussions of the SMT-LIB initiative at the 2nd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR) at IJCAR 2004. SMT-LIB is an initiative of the SMT community to build a library of SMT benchmarks in a proposed standard format. SMT-COMP aims to serve this goal by contributing collected benchmark formulas used for the competition to the library, and by providing an incentive for implementors of SMT solvers to support the SMT-LIB format. See `http://combination.cs.uiowa.edu/smtlib/` for more information on SMT-LIB.

The 1st SMT-COMP was held July 6 - 8, 2005, as a satellite event of the 17th International Conference on Computer-Aided Verification (CAV). The primary goals of SMT-COMP at CAV 2005 were:

- To collect benchmarks in a common format, namely the SMT-LIB format [26].
- To jump-start definition of SMT theories, again using the proposed SMT-LIB format.
- To spur development of SMT solver implementations, in particular, support for SMT-LIB format.
- To connect implementors of SMT solvers with potential users in the verification community.

Evaluation of SMT solvers entered in SMT-COMP took place while CAV 2005 was meeting, in the style of CASC. Intermediate results were posted periodically on the SMT-COMP website, `http://www.csl.sri.com/users/demoura/smt-comp/`, as SMT-COMP proceeded. Final results were announced in a special session on the last day of CAV. The CAV organizers arranged for SMT-COMP to have exclusive access to a group of GNU Linux machines (detailed in Section 5 below), which were used to run the competition.

Even before evaluation began, SMT-COMP was already a success. Six months before the competition, there were no solvers parsing SMT-LIB format, and there were no benchmarks collected. 1352 benchmarks in SMT-LIB format were collected for SMT-COMP, in seven different divisions (called *logics* in SMT-LIB terminology). Twelve solvers were entered supporting the SMT-LIB format. Comments from some of the participants, particularly of less mature systems, suggest that the competition was a significant motivator to make progress on their implementations.

The rest of this paper describes the rules and competition format for SMT-COMP, which were designed by the authors (Section 2); the benchmarks collected (Section 3); the scripts and actual execution of the solvers (Section 5); and of course, the final results (Section 6).

2. Rules and Competition Format

The rules and competition format for SMT-COMP draw substantially on ideas from the design and organization of CASC.

2.1. ENTRANTS

Participants were allowed to enter SMT solvers into SMT-COMP in either source code or binary format. The organizers reserved the right to submit their own systems, and indeed did so in the form of CVC, CVC Lite, and Yices; as well as systems SVC and Simplics, of which organizers were co-implementors. For solvers submitted in source code form, the organizers stated they would take reasonable precautions to ensure that the source code was not viewed by anyone other than the organizers. No solver was entered, however, where concerns over intellectual property were an issue. Participants were encouraged to be physically present at SMT-COMP, but were not required to be so to participate or win. The organizers committed to making reasonable efforts to install each system, but reserved the right to reject an entrant if its installation process was overly difficult. In several cases, solvers as submitted did not quite conform to some of the requirements below. Fortunately, with some extra effort by the submitters and the organizers, these conformance problems were repaired before the competition began. Finally, each entrant to SMT-COMP was required to include a short (1-2 pages) system description which can be found on the SMT-COMP web page.

2.2. SOLVER INTERFACE

Each SMT-COMP entrant, when executed, was required to read a single SMT-LIB benchmark file presented on its standard input channel. These benchmark files were given in the concrete syntax of the SMT-LIB format, version

```
(benchmark int_incompleteness2.smt
  :source { Leonardo de Moura }
  :notes "Check completeness of the integer DP."
  :status unsat
  :logic QF_LIA
  :extrafuns ((x1 Int) (x2 Int) (x3 Int) (x4 Int))
  :formula
  (and (> (+ x1 x2) 0)
          (< (+ x1 x2) 3)
          (= x1 (* 3 x3))
          (= x2 (* 6 x4))))
```

Figure 1. Example Benchmark in SMT-LIB Format

1.1 [26]. An example SMT-LIB benchmark is shown in Figure 1. The format requires the name of the SMT-LIB *logic* (a restriction on the set of formulas considered, together with a background theory) for the formula. It contains the status of the formula; for the competition, the status was, of course, always listed as "unknown". Extra function and predicate symbols beyond those provided by the logic can be declared. A single formula is then given, in a LISP-like prefix syntax. In standard mathematical notation, the formula in the Figure is:

$$(x_1 + x_2 > 0) \wedge (x_1 + x_2 < 3) \wedge (x_1 = 3 * x_3) \wedge (x_2 = 6 * x_4)$$

The SMT-LIB format also includes a simple sort system. SMT-COMP entrants were allowed to operate under the assumption that they would be given only well-sorted formulas. For a given input formula, each SMT-COMP entrant was then expected to report on its standard output channel whether the formula is satisfiable or unsatisfiable. An entrant could also report "`unknown`" to indicate that it could not determine satisfiability of the formula. Aborts, timeouts, other output, and exhaustion of memory were all treated as if the tool had reported "`unknown`".

2.3. JUDGING AND SCORING

Scoring was done using the system of points and penalties in Figure 2. Unsound or incomplete solvers were penalized, but not disqualified. The motivation for this requires some explanation. First, benchmarks in SMT-LIB format did not exist prior to the competition. The authors were able to make some benchmarks available in early April, 2005, but many more were being collected (and made available) into June. Hence, implementors did not have a long time to stress test their tools on the competition's benchmarks. Furthermore, it is relatively difficult to achieve a mature, bug-free SMT solver

Reported	Points for correct response	Penalty for incorrect response
`unsat`	+1	-8
`sat`	+1	-4
`unknown`	0	0
timeout	0	0

Figure 2. Points and Penalties

implementation. This is due to the fact that the reasoning required is more specialized (and thus, more logically elaborate) than for more universal automated reasoning domains (e.g., first-order theorem proving). Those domains can certainly have implementations that are at least as complex, but their complexity is due to sophisticated implementation of relatively simple inference rules. Finally, smaller penalties were assessed for incompleteness than for unsoundness, due to the belief that achieving completeness is more difficult for SMT solving than soundness. These rules were not uncontroversial, and they are likely to be modified for SMT-COMP 2006. But indeed, the organizers concerns proved to be justified: fully a third of the competition's field reported a wrong answer on at least one benchmark.

The organizers took responsibility for determining in advance whether formulas are satisfiable or unsatisfiable. This was done either using knowledge about how the benchmarks were generated, or by running several reasonably trusted existing solvers (with no time or memory limits) to get an answer. Although this is not as strong a guarantee as one might like, it appears no further validation mechanism was necessary: mature solvers all agreed on the competition benchmarks, and no incorrect classifications were reported before or after the competition. In the event of a tie in total number of points, the solver with the lower total CPU time on formulas for which it did not report "`unknown`" was considered the winner.

2.4. PROBLEM DIVISIONS

Definitions of the following SMT-LIB logics and their corresponding theories were made publicly available in advance of the competition on the SMT-LIB web page. The prefix "QF_" below means the formulas in the logic are quantifier-free.

– QF_UF: uninterpreted functions

– QF_RDL: real difference logic

- QF_IDL: integer difference logic
- QF_UFIDL: uninterpreted functions and integer difference logic.
- QF_LRA: linear real arithmetic
- QF_LIA: linear integer arithmetic
- QF_AUFLIA: arrays, uninterpreted functions and linear integer arithmetic.

3. Benchmarks

3.1. Collecting Benchmarks

One of the primary challenges for the first SMT-COMP was the collection of benchmarks in the SMT-LIB format. After the format had stabilized, a call for benchmarks was sent to the SMT community. The response was encouraging, but none of the benchmarks initially received were in the SMT-LIB format. Fortunately, many groups had benchmarks in CVC format [9]. Because its architecture can easily accommodate new input and output languages, the CVC Lite [7] system was chosen as a platform to support translation from CVC format to SMT-LIB format. This turned out to be fairly straightforward. A more challenging task was to automatically identify which division a benchmark belongs to. CVC Lite was instrumented to accomplish this task as well. Arithmetic posed a particular challenge because CVC Lite had to identify not just whether arithmetic symbols were used, but whether the use fell into the difference logic category or the more general linear arithmetic category. Most of the benchmarks collected for the first SMT-COMP were translated using the CVC Lite translator. The rest were translated and provided by Albert Oliveras. Each benchmark contains an attribute indicating its source. Most are from real applications. The benchmarks can be found on the SMT-LIB and SMT-COMP web pages.

In order to verify the syntactic and type correctness of the translated benchmarks, a separate parser and syntax checker was written in OCaml. The syntax checker checked that each benchmark was well-formed and well-typed. It also parsed the theory and logic SMT-LIB files and checked that the sorts and symbols used in the benchmarks were defined there. Finally, the CVC Lite translator, which can also accept input in SMT-LIB format, was re-run on each generated SMT-LIB benchmark to check that reprocessing it would return the same (SMT-LIB) benchmark. This turned up several subtle problems in the benchmark suite.

Both the CVC Lite translator and the OCaml syntax checker were made available on the SMT-COMP web page. The strategy of careful translation and checking of the benchmarks paid off as there were no problems with benchmark syntax during the competition. There were some complaints that the benchmarks were not in their most natural representation, but as far as known, the syntax and categorization was correct in all instances. The excellent work done by Cesare Tinelli and Silvio Ranise to define an expressive yet precise format for SMT-LIB was also very conducive to a clean set of benchmarks.

3.2. Selection of Competition Benchmarks

There was not enough time during the competition to run all the solvers on all the benchmarks. This was not necessarily desirable anyway because using all the benchmarks in a division might result in a suboptimal distribution in terms of satisfiable versus unsatisfiable and difficult versus easy.

It was decided that the competition benchmarks would consist of 50 benchmarks from each division. The criteria for selection were as follows. There should be a spectrum of difficulty ranging from easy to difficult. As much as possible, each benchmark source should be equally represented. Since most of the benchmarks collected are unsatisfiable, it was not possible to put an equal number of satisfiable and unsatisfiable benchmarks in each division, subject to the other constraints. Each division had approximately 42 unsatisfiable benchmarks and 8 satisfiable benchmarks in each division. Together with the scoring system, this distribution at least ensured that there was no expected benefit to guessing: with 8 satisfiable benchmarks and a score of -8 for each incorrect answer on those benchmarks, guessing unsatisfiable would be expected to result in a negative score. Nevertheless, if there were any solvers that were much stronger on satisfiable than unsatisfiable benchmarks, they were at a significant disadvantage with this distribution. Collecting more satisfiable benchmarks is an important priority for next year's competition.

The benchmark selection strategy worked fairly well, but could be improved in the future. The fact that there was a wide distribution of scores and no solver was able to solve every benchmark in any division shows that there was a reasonable diversity of difficulties. However, some solvers got near-perfect scores in some divisions, indicating that more difficult benchmarks will be needed in the future. Adopting a rating system such as that used by CASC or the SAT competition, where problems are deemed harder if fewer systems can solve them, would help improve the selection of benchmarks.

The strategy for mixing satisfiable and unsatisfiable benchmarks worked well except in the QF_UFIDL division where only two satisfiable benchmarks were available. As a result, a tool which guessed "unsatisfiable" for every

benchmark in this division would have placed third. Indeed, one tool which answered incorrectly on both of the satisfiable benchmarks still did quite well.

For divisions based on the theory of integers, it was also included at least one hand-crafted benchmark which was unsatisfiable over the integers but satisfiable over the reals. The rationale for this was that many benchmarks are equisatisfiable over the reals and the integers, but integer reasoning is much more challenging. The idea was to make sure that some effort was made to test this integer reasoning. It is significant to note that this "integer completeness" benchmark did trip up one solver in the QF_LIA division. More than anything, this reveals the need for better benchmarks for exercising integer reasoning. In the future, it would be preferable to have more realistic benchmarks for debugging integer reasoning and for differentiating the ability of solvers to handle integers versus reals.

4. Participants

There were twelve entries in the inaugural SMT-COMP. Here, a brief description of each of these systems is provided.

ARIO. ARIO was submitted by Hossein M. Sheini and Karem A. Sakallah from the University of Michigan. ARIO is implemented in C++ and combines an advanced Boolean SAT solver with special-purpose modules for reasoning about arithmetic. Ackermann's method is used to eliminate uninterpreted function symbols. ARIO competed in every division except for QF_AUFLIA. More information can be found in [31, 30] and at:

`http://www.eecs.umich.edu/~ario`.

BarcelogicTools. BarcelogicTools was submitted by Robert Nieuwenhuis and Albert Oliveras from the Technical University of Catalonia, Barcelona. BarcelogicTools is a C implementation of SMT for uninterpreted functions and difference logic based on the DPLL(T) framework for SMT [17]. BarcelogicTools uses heuristics to determine whether to use Ackermann's method or congruence closure for reasoning about uninterpreted functions. BarcelogicTools competed in the following divisions: QF_UF, QF_IDL, QF_RDL, and QF_UFIDL. More information can be found in [23] and at:

`http://www.lsi.upc.edu/~oliveras/bclt-main.html`.

CVC. CVC [34] is a legacy system developed at Stanford University by Aaron Stump, Clark Barrett, and David Dill. An updated version capable of parsing SMT-LIB format was submitted by Aaron Stump. CVC is implemented in C++ and implements a general framework for combining first order theories based on the Nelson-Oppen method [4]. CVC uses the Chaff SAT solver for Boolean reasoning [5]. CVC competed in all divisions. More information can be found at:

`http://cl.cse.wustl.edu/CVC/`.

CVC Lite. CVC Lite [7] is a new implementation of CVC developed primarily by Clark Barrett at New York University and Sergey Berezin at Stanford University. CVC Lite is implemented in C++ and is based on the framework for cooperating decision procedures found in Clark Barrett's PhD thesis [6]. CVC Lite has a custom SAT solver and is capable of producing independently-checkable proofs for valid queries. CVC Lite competed in all divisions. More information can be found at:

`http://verify.stanford.edu/CVCL/`.

HTP (Heuristic Theorem Prover). HTP was developed by Kenneth Roe. It is based on similar systems like SVC and CVC but incorporates new decision heuristics. HTP competed in all divisions.

MathSAT. A version of MathSAT 3 capable of parsing SMT-LIB was contributed by the MathSAT team (see `http://mathsat.itc.it`). MathSAT uses the MiniSAT solver for Boolean reasoning [14]. Uninterpreted functions are handled by either the Ackermann reduction or congruence closure. Support for arithmetic is *layered* with faster, less general solvers run first, followed by slower, more complete solvers. MathSAT competed in all divisions except for QF_AUFLIA. More information can be found in [21] and at:

`http://mathsat.itc.it/`.

Sammy. Sammy was submitted by Michael DeCoster, George Hagen, Cesare Tinelli, and Hantao Zhang from the University of Iowa. Sammy is written in OCaml and C and is based on the DPLL(T) framework for SMT [17]. Sammy uses a tool derived from SATO [36] for propositional reasoning and CVC Lite [7] for theory reasoning. Sammy competed in all divisions. More information can be found at:

`http://goedel.cs.uiowa.edu/Sammy/`.

Sateen. Sateen was submitted by Hyondeuk Kim, HoonSang Jin, and Fabio Somenzi from the University of Colorado at Boulder. Sateen is written in C and combines efficient Boolean reasoning with a layered approach to arithmetic. Sateen competed only in the division QF_IDL.

SBT. SBT (SatBox with Theories) was submitted by Hantao Zhang, Haiou Shen, and John Wheeler from the University of Iowa. SBT is written in C and is built on top of the SatBox toolbox for propositional reasoning (see `http://www.cs.uiowa.edu/~hzhang/satbox/`). It also incorporates some code from Albert Oliveras (one of the authors of the Barcelogic-Tools system). SBT competed in the following divisions: QF_UF, QF_IDL, QF_UFIDL, QF_LIA.

Simplics. Simplics was submitted by Bruno Dutertre and Leonardo de Moura from the Computer Science Laboratory at SRI International. Simplics is a recent successor to ICS [15], written mostly in OCaml. Simplics uses a core real-linear arithmetic solver based on an enhanced version of the simplex algorithm [27]. Simplics competed in the following divisions: QF_RDL,

QF_LRA. More information can be found at:
`http://fm.csl.sri.com/simplics/`.

SVC. SVC [8] is a legacy system developed at Stanford University by Clark Barrett, Jeremy Levitt, and David Dill. An updated version capable of parsing SMT-LIB format was submitted by Clark Barrett. SVC is implemented in C++ and implements a framework for combining decision procedures described in Jeremy Levitt's PhD thesis [20]. SVC competed in all divisions. More information can be found at:
`http://verify.stanford.edu/SVC/`.

Yices. Yices was submitted by Leonardo de Moura from the Computer Science Laboratory at SRI International. Yices is implemented in C++ and is based on the Nelson-Oppen method for combining decision procedures. Yices can produce proof objects for valid queries. Yices competed in all divisions. More information can be found at:
`http://fm.csl.sri.com/yices/`.

5. Scripts and Execution

SMT-COMP used 14 identical machines with 2.6Ghz Pentium 4 processors, 512Kb of cache and 512Mb of RAM, running GNU/Linux version 2.4.20. Solvers submitted in source code format were compiled using GCC version 3.2.2.

The program `TreeLimitedRun`, developed for the CASC competition, was used to monitor the execution of each solver. It watches the CPU usage of a process and its subprocesses, and kills them if they exceed the defined limits for CPU time (600 seconds), wall clock time (800 seconds), or memory usage (450Mb). The `ulimit` command was not used to enforce these limits because it does not take in consideration the time and memory consumed by subprocesses. The difference between CPU time and wall clock time was motivated by the fact that, despite our best efforts, the machines generously provided by University of Edinburgh Department of Informatics, but outside our total control, had some other processes running on the background. It was observed a difference of up to 20% between the actual CPU time and wall clock time reported for solving a formula. Although the physical amount of memory of each machine is 512Mb, the limit 450Mb was used to minimize the number of page faults.

Each solver was assigned to a different machine. A *controller* program was responsible for executing the solver on all formulas in a given division. The controller used `TreeLimitedRun` to monitor the execution, and the results were stored in a local *log* file and sent by email to a special account at SRI. At SRI, `procmail` was configured to filter the messages containing SMT-COMP results, and store them in a textual database. From time to time,

a script used the textual database to update on the fly the competition website with partial results. The local *log* file had two purposes. First, it was a backup for SMT-COMP results, just in case an email message with results was lost or corrupted. Second, and more importantly, it allowed the execution of a solver in a given division to be resumed when a machine crashed or was accidentally rebooted. This was useful as several crashes occurred.

Several solvers did not conform with the output format specified by the competition. These solvers basically were displaying extra information in the standard output besides `sat`, `unsat`, and `unknown`. Therefore, a wrapper script was used to execute each solver. This script filtered the standard output, and redirected the standard error to `/dev/null`.

6. Results

Figure 3 contains the number of problems in each division, the total number of solved problems by sound solvers, and the maximum number of solved problems by a single sound solver. The results for each division are summarized in Figures 4, 5, 6, 7, 8, 9, and 10. The first-place tools are Barcelogic Tools, Simplics, and Yices. Other tools placing second are CVC and MathSAT. Other tools placing third are Ario and CVC Lite. More detailed results are available on the SMT-COMP web site. The curves are used to show the behavior of the solvers in each division; they show how many problems (in abscissa) were solved when time (in ordinate) is increasing. Wrong answers and timeouts are not shown in these curves. In the tables, the column *Time* has the accumulated time, in seconds, used by each solver. This column does not include the time spent in problems where the solver produced the *unknown* result. The column *Unknown* contains the number of *unknown* results, and the column *Wrong* the number of wrong answers, due to unsoundness and incompleteness produced by each solver.

7. Observations on the State of the Art

The results described in Section 6 measure mainly implementations of SMT solvers. Nevertheless, some general observations are formulated from these results and system descriptions submitted by the participants, which should be helpful in studying and improving SMT solvers.

Tightly integrated SAT solvers. Most participants used the *lazy integration* approach [5] where a SAT solver is combined with a decision procedure for some theory T (DP_T). Modern SAT solving techniques must be used in the implementation of a competitive SMT solver, but it does not appear to be essential to incorporate a true state of the art (SOTA) SAT solver, since this is

Division	Num. of problems	Solved	Max. by a single solver
QF_UF	50	40	39
QF_IDL	51	49	47
QF_RDL	50	42	41
QF_UFIDL	49	46	45
QF_LIA	54	45	41
QF_LRA	50	49	49
QF_AUFLIA	52	50	49

Figure 3. Number of solved problems in each division

not the case for the best performing solvers. It seems is more important to use a tightly integrated SAT solver than a loosely integrated SOTA SAT solver.

Eager theory notification. In the *lazy integration* approach, a solver is said to implement lazy theory notification when the decision procedure DP_T is only notified after the SAT solver produces a complete boolean assignment for the input formula. In contrast, an *eager* notification [5] policy would notify DP_T immediately of every decision made by the SAT solver. All best performing solvers used eager theory notification.

Theory propagation. It is said a solver implements theory propagation when it detects literals of the input formula that are consequences of the partial model that is being explored. Suppose a hypothetical problem which contains the atoms $\{x = y, y = z, x = z\}$, and during the search the atoms $x = y$ and $y = z$ are assigned to true, then the atom $x = z$ is a consequence of this partial assignment, and can be assigned to true by theory propagation (transitivity). All best performing solvers implemented some form of theory propagation.

Minimizing "explanations" of conflicts. In the *lazy integration* approach, when a decision procedure for some theory *T* detects a conflict, it must provide an "explanation", that is, an inconsistent subset of the asserted literals in the branch being explored. The "explanation" is used to build a (learned) clause that will prevent the conflict from occurring again. The set of all literals asserted in the current branch is a valid but imprecise explanation. The main difficulty with this naïve approach is that the clauses added can be highly redundant. All best performing solvers used techniques to minimize the size of explanations. The main idea is to keep track of which facts were used to derive an inconsistency.

Difference logic. For arithmetic, more efficient algorithms are known for difference formulas. The SMT-COMP results provide empirical evidence that

general purpose linear arithmetic procedures are not competitive in this fragment. Simplex and Fourier-Motzkin are examples of general purpose linear arithmetic procedures. All the best performing solvers in the difference logic divisions (QF_RDL, QF_IDL, and QF_UFIDL) used specialized algorithms and/or data-structures for this fragment.

Ackermann's reduction. Ackermann's reduction [1] is a well known technique used to eliminate uninterpreted function symbols. This approach is generally considered inefficient in practice. Surprisingly, three good performing solvers (Ario, BarcelogicTools, and MathSat) made use of this technique, at least in some special situations.

Producing proofs and models. In recognition of the importance of exporting the results of SMT solvers for the benefit of systems such as proof assistants or applications such as proof-carrying code, SMT-COMP tried to stimulate entrants to produce suitable evidence for the results they report. Unfortunately, few participants were capable of producing proofs and models.

8. SMT-COMP 2006

SMT-COMP 2006 will be held as a satellite event of CAV 2006 as part of the Federated Logic Conference (FLoC) 2006. The intention is for SMT-COMP 2006 to continue its work of encouraging adoption of the SMT-LIB format and the collection of benchmarks. Several specific new goals have also been proposed:

– **New theories.** In his invited talk at the 3rd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR 2005), Eli Singerman of Intel called for SMT solvers to add support for logics like the combination of EUF and fixed-width bitvectors. Specifying this and related logics with bitvectors, and collecting suitable benchmarks in SMT-LIB format, is an important goal for SMT-COMP 2006.

– **More exchange among implementors.** One minor disappointment of the first SMT-COMP (almost lost in the very positive overall response to the competition) was that not enough discussions on technical and engineering matters among SMT solver implementors seemed to take place. To improve this, SMT-COMP 2006 participants will be invited to give short presentations on their solvers, followed by discussion, in a short FLoC workshop session. The intention is for this addition to improve on SMT-COMP 2005 by providing a formal mechanism for exchange of ideas among SMT solver implementors.

9. Acknowledgements

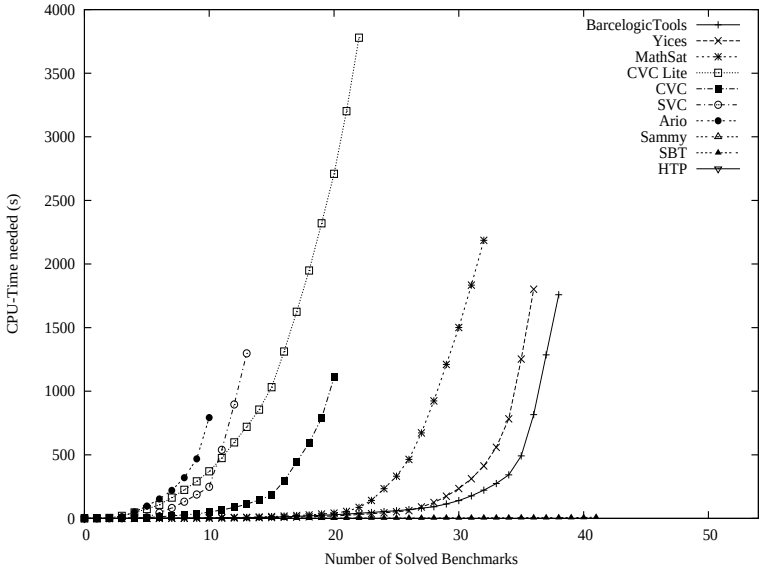
SMT-COMP is the result of the hard work and support of the entire SMT community, for which the authors would like to express their sincere thanks. Thanks are due to Kousha Etessami and Sriram Rajamani, the program chairs for CAV 2005, for helping make SMT-COMP at CAV possible. For their contributions to effort to collect benchmarks for SMT-COMP, thanks are also due first especially to Albert Oliveras, who helped translate many benchmarks into SMT-LIB format; and also Bruno Dutertre, Pete Monolios, Lee Pike, Jiae Shin, Sudarshan Srinivasan, Ofer Strichman; as well as members of the TSAT++ team, the MathSAT team, the SAL group at SRI, the UCLID project, and the Wisconsin Safety Analyzer project. Thanks also to Mike Decoster and Michael Schidlowsky for making available an SMT-LIB parser in OCaml. A special debt of gratitude is owed to Cesare Tinelli and Silvio Ranise for their work on the specification of the SMT-LIB format and the descriptions of the SMT-LIB theories and logics used by the competition. The authors wish to thank the anonymous reviewers of this paper for helpful criticisms. Finally, thanks go to all the people who entered solvers in SMT-COMP, whose hard work implementing solvers and adding support for the SMT-LIB format is greatly appreciated.

References

1. W. Ackermann. Solvable cases of the decision problem. *Studies in Logic and the Foundation of Mathematics*, 1954.
2. A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints. In *The 7th International Conference on Theory and Applications of Satisfiability Testing*, 2004.
3. A. Armando, S. Ranise, and M. Rusinowitch. A Rewriting Approach to Satisfiability Procedeures. *Information and Computation*, 183(2):140–164, June 2003. Special Issue on the 12th International Conference on Rewriting Techniques and Applications (RTA 2001).
4. C. Barrett, D. Dill, and A. Stump. A Framework for Cooperating Decision Procedures. In D. McAllester, editor, *17th International Conference on Automated Deduction*, pages 79–97. Springer-Verlag, 2000.
5. C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In *14th International Conference on Computer-Aided Verification*, 2002.
6. Clark Barrett. *Checking Validity of Quantifier-Free Formulas in Combinations of First-Order Theories*. PhD thesis, Stanford University, 2003.
7. Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the cooperating validity checker. In Rajeev Alur and Doron A. Peled, editors, *Proceedings of the* 16^{th} *International Conference on Computer Aided Verification (CAV '04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 515–518. Springer-Verlag, July 2004. Boston, Massachusetts.

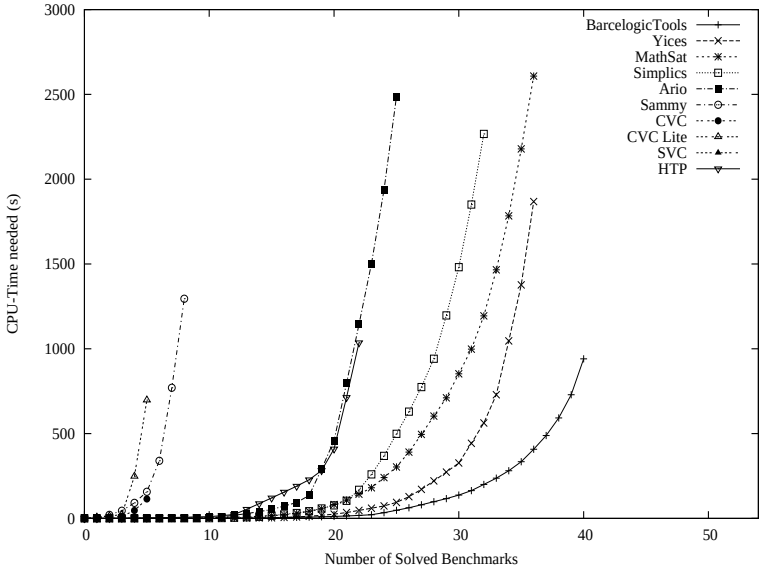
8. Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. Validity checking for combinations of theories with equality. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the* 1^{st} *International Conference on Formal Methods In Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, November 1996. Palo Alto, California.
9. CVC Lite website. `http://verify.stanford.edu/CVCL`.
10. D. Le Berre and L. Simon. The essentials of the SAT 2003 competition. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 452–467. Springer-Verlag, 2003.
11. P. Bjesse, T. Leonard, and A. Mokkedem. Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers. In G. Berry, H. Comon, and A. Finkel, editors, *13th Conference on Computer-Aided Verification*. Springer-Verlag, 2001.
12. E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1), 2001.
13. Satyaki Das and David L. Dill. Counter-example based predicate discovery in predicate abstraction. In *Formal Methods in Computer-Aided Design*. Springer-Verlag, November 2002.
14. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, May 2003.
15. J. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: integrated canonizer and solver. In G. Berry, H. Comon, and A. Finkel, editors, *13th International Conference on Computer-Aided Verification*, 2001.
16. C. Flanagan, K. Leino, M. Lillibridge, G. Nelson, J. Saxe, and R. Stata. Extended Static Checking for Java. *SIGPLAN Notices*, 37, 2002.
17. Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.
18. S. Lahiri, R. Bryant, A. Goel, and M. Talupur. Revisiting Positive Equality. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *LNCS*, pages 1–15. Springer-Verlag, 2004.
19. S. Lerner, T. Millstein, and C. Chambers. Automatically Proving the Correctness of Compiler Optimizations. In R. Gupta, editor, *In ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003. received best paper award.
20. Jeremy Levitt. *Formal Verification Techniques for Digital Systems*. PhD thesis, Stanford University, 1999.
21. M.Bozzano, R.Bruttomesso, A.Cimatti, T.Junttila, P.v.Rossum, S.Schulz, and R.Sebastiani. The MathSAT 3 system. In *Proceedings of the* 20^{th} *International Conference on Automated Deduction*, July 2005.
22. M. Möller and H. Rueß. Solving Bit-Vector Equations. In *Formal Methods in Computer-Aided Design*, pages 36–48, 1998.
23. Robert Nieuwenhuis and Albert Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, Lecture Notes in Computer Science. Springer, 2005.
24. F.J. Pelletier, G. Sutcliffe, and C.B. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.

25. A. Pnueli, Y. Rodeh, and O. Strichman. Range allocation for equivalence logic. In *21st Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2245 of *LNCS*, pages 317–333. Springer-Verlag, 2001.
26. Silvio Ranise and Cesare Tinelli. The SMT-LIB standard, version 1.1, 2005. Available from the "Documents" section of http://combination.cs.uiowa.edu/smtlib.
27. Harald Rueß and Natarajan Shankar. Solving linear arithmetic constraints. Technical Report SRI-CSL-04-01, SRI International, 2004.
28. S. Seshia and R. Bryant. Deciding Quantifier-Free Presburger Formulas Using Parameterized Solution Bounds. In *Logic in Computer Science*. IEEE, 2004.
29. N. Shankar. Little Engines of Proof. In *Invited Paper at Formal Methods Europe*, 2002.
30. Hossein M. Sheini and Karem A. Sakallah. A sat-based decision procedure for mixed logical/integer linear problems. In Roman Barták and Michela Milano, editors, *CPAIOR*, volume 3524 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2005.
31. Hossein M. Sheini and Karem A. Sakallah. A scalable method for solving satisfiability of integer linear arithmetic logic. In Fahiem Bacchus and Toby Walsh, editors, *SAT*, volume 3569 of *Lecture Notes in Computer Science*, pages 241–256. Springer, 2005.
32. I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. Debugging Overconstrained Declarative Models Using Unsatisfiable Cores. In *18th IEEE International Conference on Automated Software Engineering*, 2003. received best paper award.
33. A. Stump, C. Barrett, D. Dill, and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *16th IEEE Symposium on Logic in Computer Science*, pages 29–37. IEEE Computer Society, 2001.
34. Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A cooperating validity checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the* 14^{th} *International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer-Verlag, July 2002. Copenhagen, Denmark.
35. M. Velev and R. Bryant. Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors. *Journal of Symbolic Computation*, 35(2):73–106, February 2003.
36. H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 272–275. Springer, July 1997.



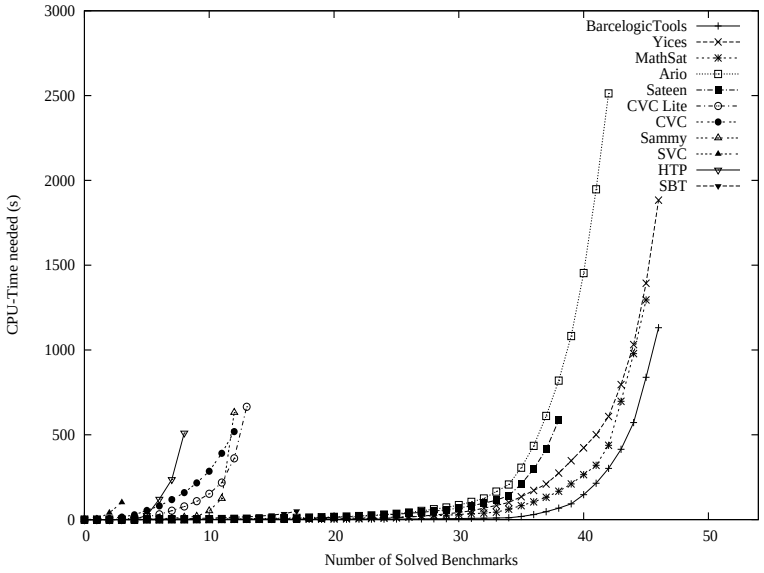
Solver	Score	Time	Unsat	Sat	Unknown	Wrong
BarcelogicTools	39	1758.2	31	8	11	0
Yices	37	1801.4	29	8	13	0
MathSat	33	2186.2	26	7	17	0
CVC Lite	23	3779.3	16	7	27	0
CVC	21	1108.8	16	5	29	0
SVC	14	1297.0	11	3	36	0
Ario	11	792.5	10	1	39	0
Sammy	1	0.3	1	0	49	0
SBT	-22	10.7	50	0	0	8
HTP	-43	567.5	0	12	38	11

Figure 4. Results for QF_UF



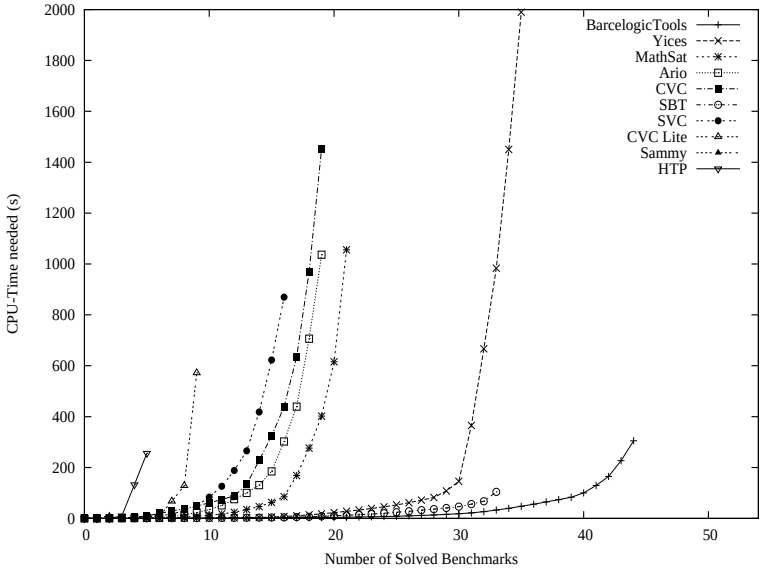
Solver	Score	Time	Unsat	Sat	Unknown	Wrong
BarcelogicTools	41	940.8	36	5	9	0
Yices	37	1868.0	32	5	13	0
MathSat	37	2608.0	32	5	13	0
Simplics	33	2267.0	30	3	17	0
Ario	26	2487.5	22	4	24	0
Sammy	9	1295.6	9	0	41	0
CVC	6	115.3	6	0	44	0
CVC Lite	6	697.6	6	0	44	0
SVC	1	0.3	1	0	49	0
HTP	-5	1390.4	25	3	22	5

Figure 5. Results for QF_RDL



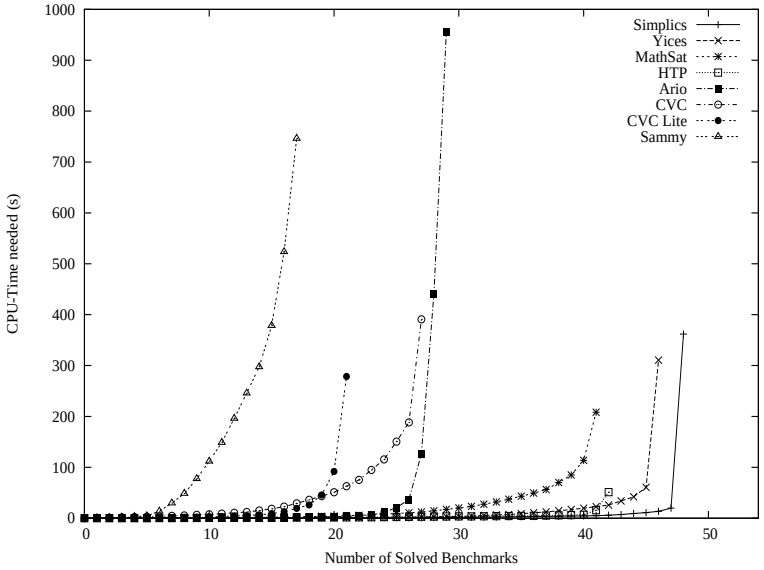
Solver	Score	Time	Unsat	Sat	Unknown	Wrong
BarcelogicTools	47	1131.2	38	9	4	0
Yices	47	1883.2	38	9	4	0
MathSat	46	1295.4	35	11	5	0
Ario	43	2513.0	34	9	8	0
Sateen	39	586.2	33	6	12	0
CVC Lite	14	665.4	12	2	37	0
CVC	13	519.9	13	0	38	0
Sammy	13	631.2	13	0	38	0
SVC	4	102.0	4	0	47	0
HTP	-43	1655.8	6	16	29	13
SBT	-90	109.6	19	21	11	22

Figure 6. Results for QF_IDL



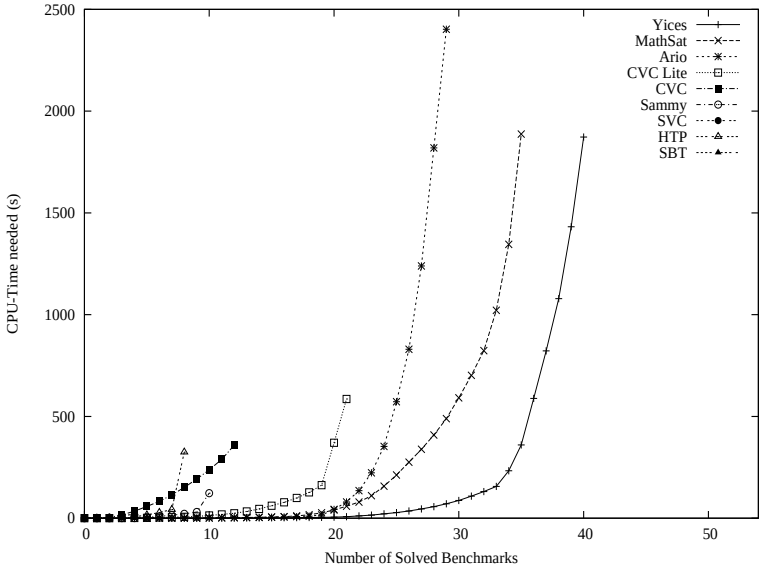
Solver	Score	Time	Unsat	Sat	Unknown	Wrong
BarcelogicTools	45	305.2	43	2	4	0
Yices	36	1989.8	34	2	13	0
MathSat	22	1055.5	20	2	27	0
Ario	20	1036.3	18	2	29	0
CVC	20	1454.0	20	0	29	0
SBT	18	104.9	36	0	13	2
SVC	17	869.5	17	0	32	0
CVC Lite	10	571.9	10	0	39	0
Sammy	-1	21.6	3	1	45	1
HTP	-42	519.8	5	13	31	12

Figure 7. Results for QF_UFIDL



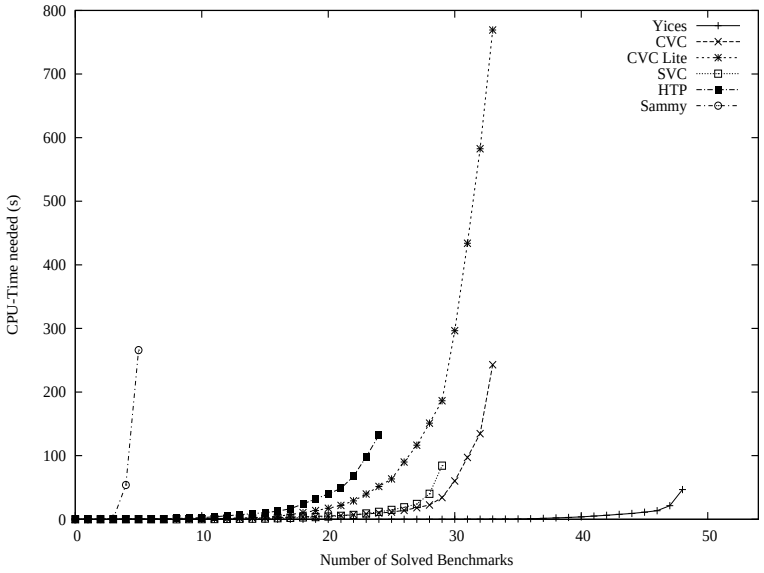
Solver	Score	Time	Unsat	Sat	Unknown	Wrong
Simplics	49	361.8	42	7	1	0
Yices	47	310.6	42	5	3	0
MathSat	42	208.3	41	1	8	0
HTP	35	51.9	42	2	6	1
Ario	30	955.7	27	3	20	0
CVC	28	391.2	25	3	22	0
CVC Lite	22	278.7	22	0	28	0
Sammy	6	824.8	12	8	30	2
SVC	0	0.0	0	0	50	0

Figure 8. Results for QF_LRA



Solver	Score	Time	Unsat	Sat	Unknown	Wrong
Yices	41	1873.0	28	13	13	0
MathSat	32	1887.2	23	14	17	1
Ario	30	2402.4	18	12	24	0
CVC Lite	22	585.6	15	7	32	0
CVC	13	359.2	13	0	41	0
Sammy	11	123.1	11	0	43	0
SVC	5	20.1	5	0	49	0
HTP	-31	325.2	10	5	39	6
SBT	-77	524.6	5	20	29	18

Figure 9. Results for QF_LIA



Solver	Score	Time	Unsat	Sat	Unknown	Wrong
Yices	49	46.8	35	14	3	0
CVC	34	243.0	34	0	18	0
CVC Lite	34	769.3	28	6	18	0
SVC	30	84.3	30	0	22	0
HTP	21	132.0	25	1	26	1
Sammy	-38	344.6	10	3	39	7

Figure 10. Results for QF_AUFLIA