Design and Results of the 2nd Annual Satisfiability Modulo Theories Competition (SMT-COMP 2006)

Clark Barrett
Department of Computer Science
New York University

Leonardo de Moura
Microsoft Research

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

Abstract

The Satisfiability Modulo Theories Competition (SMT-COMP) arose from the SMT-LIB initiative to spur adoption of common, community-designed formats, and to spark further advances in satisfiability modulo theories (SMT). The first SMT-COMP was held in 2005 as a satellite event of CAV 2005. SMT-COMP 2006 was held August 17 - 19, 2006, as a satellite event of CAV 2006. This paper describes the rules and competition format for SMT-COMP 2006, the benchmarks used, the participants, and the results.

1 Introduction

Decision procedures for satisfiability modulo theories (SMT) are of continuing interest for many verification applications (e.g., [1, 3, 5, 7, 8, 9]). SMT solvers are typically used for verification as backends: a verification problem or subproblem is translated into an SMT formula and submitted to the SMT solver. The solver then attempts to report satisfiability or unsatisfiability of the formula. The advantage SMT solvers are usually considered to have over pure SAT solvers, which are also often used as verification backends (e.g., for bounded model checking [4]), is the higher level of abstraction at which they can operate. By implementing theories like arithmetic, arrays, and uninterpreted functions directly, SMT solvers have the promise to provide higher performance than SAT solvers working on encodings of such structures to the bit level.

The additional promise of SMT over pure SAT is balanced by additional challenges. Since SMT deals with first-order (most commonly quantifier-free) formulas instead of purely propositional ones, creation and widespread adoption of a common input language is more difficult than for SAT. It is at the same time more important, since the more expressive setting of SMT potentially allows more room for variation in the exact

details of the logic (e.g., sorted or unsorted, total or partial functions, etc.). Hence, translations between input formats of different tools are more complex, and in some cases it may not even be clear what such a translation should be. This makes the issue of input format critical. For combination with other tools like skeptical proof assistants (requiring a proof of every theorem validated by an external tool), common output formats for objects like proofs and models are also necessary for the adoption of SMT.

The Satisfiability Modulo Theories Competition (SMT-COMP) arose from the SMT-LIB ("Satisfiability Modulo Theories Library") initiative to spur adoption of common, community-designed SMT-LIB formats, and to spark further advances in SMT, especially for verification. See `http://www.smtlib.org/` for more information on SMT-LIB. The first SMT-COMP was held in 2005 as a satellite event of the 17th International Conference on Computer-Aided Verification (CAV), in Edinburgh, Scotland. The experience with SMT-COMP 2005 confirmed the community's expectations that a public competition would indeed motivate implementors of SMT solvers to adopt the common SMT-LIB input format [2]. The second SMT-COMP, described in the present report, provides further evidence that such a competition can stimulate improvement in solver implementations: solvers entered in SMT-COMP 2006 were significantly improved over the winning implementations of SMT-COMP 2005.

SMT-COMP 2006 had two additional goals beyond the original ones of encouraging adoption of SMT-LIB input format and sparking implementation improvements. First, in his invited talk at the 3rd International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR '05), Eli Singerman of Intel called for SMT solvers to add support for logics like the combination of EUF ("equality with uninterpreted functions") and fixed-width bitvectors. Specifying this and related logics with bitvectors, and collecting suitable benchmarks in SMT-LIB format, was an important goal for SMT-COMP 2006, which the SMT community succeeded in achieving. Second, SMT-COMP 2005 did not provide a setting for exchange of technical and engineering ideas among solver implementors. So SMT-COMP 2006 included a special evening session where implementors had the opportunity to give brief (10 minute) presentations about their tools, and discuss implementation issues.

SMT-COMP 2006 was held August 17 - 19, 2006, as a satellite event of CAV 2006. The competition ran while CAV 2006 was meeting, in the style of the CADE ATP System Competition (CASC) for general first-order theorem provers [10]. Intermediate results were posted periodically on the SMT-COMP website, with final results announced in a special session on the last day of CAV. Solvers were run on a cluster of computers at SRI International. As discussed below, great efforts were made to make the running of the competition as transparent and repeatable as possible. So all scripts, solvers, and benchmark formulas were made available on the web before the competition began, and so anyone could (and some did) compute the results of the competition independently. The web site for SMT-COMP 2006 is `http://www.smtcomp.org/2006/`.

The rest of this paper describes the rules and competition format for SMT-COMP 2006 (Section 2), the new benchmarks collected (Section 3), the participants (Section 4), the scripts and execution of the solvers (Section 5), and the final results (Section 6).

2 Rules and Competition Format

This Section explains the rules and competition format for SMT-COMP. These draw substantially on ideas from the design and organization of CASC [10].

2.1 Entrants

SMT solvers were submitted to SMT-COMP via the EasyChair conference management system in either source code or binary format. The submission deadline was August 8, as close as the organizers could allow to the competition start date while still leaving enough time to ensure that all solvers interoperated correctly with the competition scripts. Submitted source code was kept private, but binary executables for all solvers were made available on the SMT-COMP website. SMT-COMP 2006 participants were not required to attend the sponsoring conference, although most did so, and most made presentations at the SMT-COMP evening session. Entrants to SMT-COMP were also required to include a short (1-2 pages) system description, stating, among other things, in which *problem divisions* (see Section 2.4 below) the entrant should participate.

2.2 Solver Interface

As for SMT-COMP 2005, each SMT-COMP 2006 entrant was executed by presenting a single SMT-LIB benchmark file to its standard input channel. These benchmark files were given in the concrete syntax of the SMT-LIB format, version 1.1 [11]. This format states the *logic* of the benchmark (a background theory together with any syntactic restrictions on formulas), declares the sorts of any uninterpreted symbols, and then gives the formula in a prefix syntax. The competition explicitly included only well-sorted formulas. SMT-COMP entrants were then expected to attempt to determine satisfiability or unsatisfiability of the input formula, and report either "`sat`" or "`unsat`" via their standard output channel. Solvers could also report "`unknown`", which is useful for solvers that are known to be, for example, incomplete on some subset of the formulas of a given problem division. 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 scoring system of Figure 1. Unlike for SMT-COMP 2005, SMT-COMP 2006 tolerated at most three wrong answers in any division. More than three wrong answers in any division resulted in disqualification from the entire competition. There were occasional wrong answers in several divisions, although many fewer than in SMT-COMP 2005. There were two disqualifications: the ExtSat tool in the QF_LIA division, with 5 wrong answers out of around 100 formulas; and the Jat tool in the QF_RDL division, with 59 wrong answers out of around 100 formulas. For comparison, the largest number of wrong answers in SMT-COMP 2005 was 22 out of around 50 formulas (the SBT solver in the QF_IDL division).

Reported	Correct?	Point/penalty
`unsat`	yes	+1
`unsat`	no	-8
`sat`	yes	+1
`sat`	no	-8
`unknown`	n.a.	0
timeout	n.a.	0

Figure 1: Points and Penalties

The organizers took responsibility for determining in advance whether formulas are satisfiable or not, using existing tools. 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 timeout 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: integer difference logic with uninterpreted functions
- QF_LRA: linear real arithmetic
- QF_LIA: linear integer arithmetic
- QF_UFLIA: linear integer arithmetic with uninterpreted functions
- QF_AUFLIA: linear integer arithmetic with uninterpreted functions and arrays
- QF_UFBV32: 32-bit fixed-width bitvectors with uninterpreted functions
- AUFLIA: quantified linear integer arithmetic with uninterpreted functions and arrays
- AUFLIRA: quantified linear mixed integer/real arithmetic with uninterpreted functions and arrays

3 Benchmarks

One of the main reasons for creating SMT-COMP was to provide a concrete incentive for collecting benchmarks in the SMT-LIB format. The collection of 1352 benchmarks for the 2005 competition represented an important milestone for the SMT-LIB initiative. For the 2006 competition, 40782 new benchmarks were collected for a total of 42134 benchmarks. The benchmark collection effort for the second SMT-COMP built upon the first effort in two natural ways: collecting additional benchmarks for existing divisions and creating new divisions. An encouraging sign was that many of the new benchmarks were provided in SMT-LIB format directly (whereas in 2005, most of the benchmarks had to be translated into SMT-LIB format by the organizers). This is further evidence that SMT-COMP is successfully promoting the adoption of the SMT-LIB standard format. After a few remarks on benchmark organization, we describe the new benchmarks collected in existing divisions, and then we describe the benchmarks collected in new divisions.

3.1 Organization of Benchmarks

In 2005, the number of benchmarks was small enough that a simple organization by division was sufficient. However, with the addition of many new benchmarks, we determined to further organize the benchmarks in three concrete ways. First, benchmarks were organized according to *families*. A benchmark family contains problems that are similar in some significant way. Typically they come from the same source or application, or are all output by the same tool. The rationale is that family information can be used to help ensure a sufficiently diverse set of competition benchmarks by limiting the number of benchmarks chosen from the same family (see Section 3.4 below).

Second, each benchmark was assigned a *difficulty*: an integer between 0 and 5 inclusive. The difficulty for a particular benchmark was assigned by running several SMT solvers from the 2005 competition on it and using the formula:

$$\text{difficulty} = 5(1 - \frac{\text{solved}}{\text{total}}),$$

where solved is the number of SMT solvers that could solve the problem in 10 minutes and total is the total number of SMT solvers tried. The following solvers from the 2005 competition (whichever ones were applicable for each benchmark) were used to compute this attribute: Ario 1.1, Barcelogic, CVC, CVC Lite 2.0, MathSAT 3.3.1, Sateen, Simplics, and Yices. Two of the new divisions, QF_UFBV32 and AUFLIRA, were not supported by any of the 2005 solvers, so the difficulty in these divisions was computed using prototype implementations available to the organizers. In addition, some benchmarks in the quantifier divisions were identified as being unusually trivial and were marked with difficulty -1.

Finally, each benchmark was assigned a *category*. There are four possible categories:

- *check.* These benchmarks are hand-crafted to test whether solvers support specific features of each division. In particular, there are checks for integer completeness (i.e. benchmarks that are satisfiable under the reals but not under the

integers) and big number support (i.e. benchmarks that are likely to fail if integers cannot be represented beyond some maximum value, such as $2^{31} - 1$). The rationale for these checks is that the divisions as defined in the SMT-LIB standard do include these as legal benchmarks. In order to encourage everyone to offer truly complete solvers for each division, it seemed reasonable to enforce that every solver be capable of solving these corner cases of the logic.

- *industrial.* These benchmarks come from some real application and are produced by tools such as bounded model checkers, static analyzers, extended static checkers, etc.
- *random.* These benchmarks are randomly generated.
- *crafted.* This category is for all other benchmarks. Usually, benchmarks in this category are designed to be particularly difficult or to test a specific feature of the logic.

The family information is stored implicitly in the SMT-LIB directory structure (all benchmarks from the same family are in the same sub-directory). The difficulty and category attributes are stored as special user-defined attributes in the benchmarks themselves. Section 3.4 describes how this information was used during the benchmark selection process.

3.2 New Benchmarks for Existing Divisions

New benchmarks were obtained in every division except the QF_UF division. The new benchmarks came from a variety of sources, primarily verification applications. Other benchmarks were crafted in various ways: either by hand, or from known hard problems. Figure 2 lists the new benchmark sets collected for each existing division (a benchmark set comes from a single source and contains one or more families), together with the number of benchmarks in the set and the category of the benchmark set. For comparison, the previous number of benchmarks (from 2005) is also given for each division. The sum gives the total number of benchmarks for 2006. Note that in the QF_UFIDL division, one benchmark was deleted from the uclid benchmark set. This is because it was simpler than the others–in fact it belonged to the simpler QF_IDL logic. Rather than create a benchmark set containing a single benchmark in that division, we opted to just remove the benchmark.

3.3 New Divisions

Four new benchmark divisions were added for SMT-COMP 2006: QF_UFLIA, QF_UFBV32, AUFLIA, and AUFLIRA. As has been our custom, the added divisions were based on available benchmarks and the expectation that more than one solver would support each new division. In fact, there were some additional benchmarks offered, in particular a set in the AUFNIRA logic that includes some non-linear arithmetic, that we did not include because it seemed unlikely that more than one solver would support this division. Figure 3 lists the new benchmark sets collected for these new divisions together with the number of benchmarks in the set and the category of the benchmark set.

Division	Benchmark Set	Number of Benchmarks	Benchmark Category
QF_AUFLIA	check	2	check
QF_AUFLIA	ios	30	crafted
QF_AUFLIA	piVC	21	industrial
QF_AUFLIA	qlock2	52	industrial
QF_AUFLIA	storecomm	2030	crafted
QF_AUFLIA	storeinv	172	crafted
QF_AUFLIA	swap	1368	crafted
QF_AUFLIA	2005 Benchmarks	54	check, industrial
QF_AUFLIA	Total	3729	
QF_IDL	Averest	252	industrial
QF_IDL	cellar	14	industrial
QF_IDL	check	2	check
QF_IDL	job_shop	120	crafted
QF_IDL	planning	45	industrial
QF_IDL	qlock	72	industrial
QF_IDL	queens_bench	297	crafted
QF_IDL	2005 Benchmarks	343	check, industrial, random, crafted
QF_IDL	Total	1145	
QF_LIA	Averest	19	industrial
QF_LIA	check	3	check
QF_LIA	2005 Benchmarks	182	check, industrial
QF_LIA	Total	204	
QF_LRA	check	2	check
QF_LRA	clock_synchro	36	industrial
QF_LRA	sc	144	industrial
QF_LRA	tta_startup	72	industrial
QF_LRA	uart	73	industrial
QF_LRA	2005 Benchmarks	174	industrial
QF_LRA	Total	501	
QF_RDL	check	2	check
QF_RDL	skdmxa2	32	industrial
QF_RDL	2005 Benchmarks	170	industrial, crafted
QF_RDL	Total	204	
QF_UF	2005 Benchmarks	152	crafted
QF_UF	Total	152	
QF_UFIDL	check	2	check
QF_UFIDL	RDS	28	industrial
QF_UFIDL	uclid	-1	industrial
QF_UFIDL	pete3	6	industrial
QF_UFIDL	UCLID-pred	79	industrial
QF_UFIDL	2005 Benchmarks	170	check, industrial
QF_UFIDL	Total	399	
All Existing	Total	6334	

Figure 2: New Benchmarks in Existing Divisions

Division	Benchmark Set	Number of Benchmarks	Benchmark Category
AUFLIA	Burns	14	industrial
AUFLIA	check	1	check
AUFLIA	misc	28	industrial, crafted
AUFLIA	piVC	42	industrial
AUFLIA	RicartAgrawala	14	industrial
AUFLIA	simplify	833	industrial
AUFLIA	Total	932	
AUFLIRA	misc	7	crafted
AUFLIRA	nasa	26504	industrial
AUFLIRA	Total	26511	
QF_UFBV32	bench_a	343	industrial
QF_UFBV32	crafted	22	crafted
QF_UFBV32	egt	7882	industrial
QF_UFBV32	Total	8247	
QF_UFLIA	check	2	check
QF_UFLIA	wisas	108	industrial
QF_UFLIA	Total	110	
All New	Total	35800	

Figure 3: New Benchmarks in New Divisions

3.4 Selection of Competition Benchmarks

For each division, the following algorithm was used to select benchmarks.

1. First, all benchmarks in the *check* category are automatically included.

2. The remaining benchmarks are put into a selection pool as follows: for each family, if the family contains more than 200 benchmarks, then 200 randomly selected benchmarks are put into the pool. Otherwise all of the benchmarks from the family are put into the pool.

3. Slots are allocated for 100 benchmarks to be selected as follows: 85 slots are for industrial benchmarks; 10 are for crafted; and 5 are for random. If there are not enough crafted or random benchmarks, then more industrial slots are allocated. If, on the other hand, there are not enough industrial benchmarks, then more crafted slots are allocated (all divisions had sufficient numbers of either industrial or crafted benchmarks).

4. In order to fill the allocated slots, the pool of benchmarks created in step 2 is consulted and partitioned according to category (i.e. industrial, random, crafted). Within each category, the benchmarks are further partitioned into four sub-categories: easy-sat, easy-unsat, hard-sat, and hard-unsat. A benchmark is easy if it has difficulty 0, 1, or 2 (benchmarks with difficulty -1 are ignored) and hard if it has difficulty 3, 4, or 5. A benchmark is "sat" or "unsat" based on its *status* attribute.

An attempt is made to randomly fill the allocated slots for each category with the same number of benchmarks from each sub-category (i.e. if there are 85 industrial slots, then there should be roughly 21 in each sub-category). If there are not enough in a sub-category, then its allotment is divided among the other sub-categories.

The main purpose of the algorithm above is to have a balanced and complete set of benchmarks. The one built-in bias is towards industrial rather than crafted or random benchmarks. This reflects a desire by the organizers and agreed upon by the SMT community to emphasize problems that come from real applications.

It should be noted that when applying the above algorithm to select the competition benchmarks, we used a random seed obtained by computing the sum of "magic numbers" provided by the contestants. This random seed was fed into a script which automatically ran the above algorithm. The script was made available on the SMT-COMP website prior to the submission of the magic numbers. In this way, we hoped to ensure that the benchmark selection process was as transparent as possible.

4 Participants

There were twelve entries in SMT-COMP 2006. This is, coincidentally, the same number as for SMT-COMP 2005, although only eight of these entrants also participated in SMT-COMP 2005. Four tools from SMT-COMP 2005 did not run in SMT-COMP 2006 (Sammy, SBT, Simplics, and SVC), and four new tools ran in SMT-COMP 2006 that did not run in SMT-COMP 2005 (ExtSat, Jat, NuSMV, and STP). The following gives brief descriptions of the SMT-COMP 2006 participants, drawn from their system descriptions. For more information, including, in many cases, references to papers with detailed descriptions of novel algorithms employed by the solvers, the interested reader is referred to the system descriptions, available on the SMT-COMP web site.

Ario 1.2. Ario 1.2 was submitted by Hossein M. Sheini and Karem A. Sakallah from the University of Michigan. Ario 1.2 is implemented in C++ and implements a hybrid online/offline approach to combining theory solvers with a CNF SAT solver. Problem divisions: QF_UF, QF_RDL, QF_IDL, QF_UFIDL, QF_LRA, QF_LIA, QF_UFLIA.

Barcelogic 1.1. Barcelogic 1.1 was submitted by Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio from the Technical University of Catalonia. Barcelogic 1.1 is a C++ implementation of the DPLL(T) framework [6]. Problem divisions: QF_UF, QF_IDL, QF_RDL, and QF_UFIDL, QF_AUFLIA.

CVC. CVC is a legacy system developed at Stanford University by Aaron Stump, Clark Barrett, and David Dill, and submitted to SMT-COMP 2006 by Aaron Stump. This version is essentially the same as the version from SMT-COMP 2005. Problem divisions: QF_LRA, QF_UF, QF_AUFLIA.

CVC3. CVC3 was submitted by Clark Barrett from New York University, with development credits also going to Yeting Ge at New York University; Cesare Tinelli, Alexander Fuchs, and George Hagan at University of Iowa; and the implementors of CVC Lite, a predecessor system. CVC3 is implemented in C++ and is based on Clark Barrett's framework for cooperating decision procedures. CVC3 competed in all problem divisions.

ExtSAT 1.1. ExtSat 1.1 was submitted by Paulo Matos of the Instituto Superior Técnico, Portgual. ExtSat is implemented in C++, and combines a boolean enumerator based on MiniSAT with arithmetic solvers. Problem divisions: QF_RDL, QF_IDL, QF_LRA, QF_LIA.

HTP (Heuristic Theorem Prover). HTP was developed by Kenneth Roe. HTP implements novel preprocessing algorithms before handing formulas off either Yices or Barcelogic from SMT-COMP 2005, or MinSat. Problem divisions: QF_UF, QF_IDL, QF_RDL, QF_LRA, QF_LIA, QF_UFLIA.

Jat Jat was submitted by Scott Cotton from Verimag, and developed by the submitter under the supervision of Oded Maler. Jat is written entirely in Java, and employs novel techniques for exhaustive theory propagation for difference logic. Problem divisions: QF_RDL.

MathSAT 3.4. MathSAT 3.4 was submitted by Roberto Bruttomesso, Alessandro Cimatti, Anders Franzen, Alberto Griggio, and Roberto Sebastiani from ITC-IRST and Università di Trento, Italy. MathSAT is written in C++, and combines a propositional reasoner based on MiniSAT with theory solvers, using the Delayed Theory Combination scheme or Ackermann's reduction. Problem divisions: QF_UF, QF_RDL, QF_IDL, QF_UFIDL, QF_LRA, QF_LIA, QF_UFLIA, QF_UFBV32.

NuSMV. NuSMV was submitted by R. Bruttomesso, R. Cavada, A. Cimatti, A. Franzen, S. Semprini, M. Roveri, and A. Tchaltsev, from ITC-IRST, Italy. NuSMV is written in C, and uses a pre-processing step to reduce input problems in the QF_UFBV32 division to problems that can be solved using routines from the NuSMV symbolic model checker. Problem divisions: QF_UFBV32.

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 All SAT Enumeration with a layered theory solver. Problem divisions: QF_IDL.

STP. STP was submitted by Vijay Ganesh and David Dill from Stanford University. STP preprocess and then translates input formulas into purely propositional formulas, which are then dispatched to MiniSAT. An abstraction-refinement technique is used for handling array read expressions. Problem divisions: QF_UFBV32.

Yices 1.0. Yices was submitted by Bruno Dutertre and Leonardo de Moura from SRI International. Yices is implemented in C++, and features a novel architecture where a SAT solver is integrated with a core theory solver, as well as satellite solvers. Yices competed in all problem divisions.

5 Scripts and Execution

SMT-COMP used 19 identical machines at SRI International with 3.0Ghz Pentium 4 processor 2Mb of cache and 2Gb of RAM, running GNU/Linux version 2.6.14. Solvers submitted in source code format were compiled using GCC version 4.0.2.

In order to ensure that no system received an advantage or disadvantage due to specific presentation of the benchmarks in the SMT-LIB format, a *benchmark scrambler* was used. This tool renames all predicate and function symbols, removes comments, and randomly reorders the arguments of AC operators. The scrambler is available for download on the competition website.

A *controller* (load balancer) was used to distribute tasks between the many computers used in the competition, and to consolidate the results produced by the solvers. Each task is a small set of benchmarks that is executed by all solvers in the same machine. This approach had two advantages with respect to the one used in SMT-COMP 2005, where each solver was assigned to a different machine. First, it minimizes unfairness due to potential differences among the benchmarking machines. Second, it is more effective in balancing the workload. In SMT-COMP 2005, machines executing fast solvers remained idle for long periods of time.

The execution of each solver was monitored by a program called `TreeLimitedRun`. This program, developed for the CASC competition [10], watches the CPU usage of a process and its subprocesses, and kills them if they exceed the defined limits for CPU time (1200 seconds), or memory usage (1.5Gb). The `ulimit` command was not used to enforce these limits because it does not take into consideration the time and memory consumed by subprocesses. Although the physical amount of memory of each machine is 2.0Gb, the limit 1.5Gb was used to minimize the number of page faults.

SMT-COMP results were stored in a textual database. From time to time, a script used the textual database to update the competition website with partial results.

6 Results

The results for each division are summarized in the figures following the bibliography. More detailed results are available on the SMT-COMP website, `http://www.smtcomp.org/2006/`. 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 considered in these curves. The column *Time* has the accumulated time, in seconds, used by each solver. This column does not include the time spent in instances where the solver produced the *unknown* result. A solver is considered to have produced the *unknown* result when it times out, crashes, or outputs a result different from `sat` or `unsat`. The column

Unknown contains the number of *unknown* results, and the column *Wrong* the number of wrong answers (due to unsoundness or incompleteness) produced by each solver.

7 Acknowledgements

SMT-COMP would not have been possible without the invaluable support, feedback, and participation of the entire SMT community, with special thanks to Cesare Tinelli and Silvio Ranise, the leaders of the SMT-LIB initiative. The organizers would also like to thank SRI International for use of the cluster on which the competition was run. Thanks also to Thomas Ball and Robert Jones, the program chairs of CAV 2006, for their support of SMT-COMP 2006 as a satellite event. Finally, the organizers wish to acknowledge the support of the U.S. National Science Foundation, under contract CNS-0551697, for SMT-COMP 2007 and (anticipated) 2008.

References

[1] M. Barnett, B.-Y. Chang, R. DeLine, B. Jacobs, and K. Leino. Boogie: A Modular Reusable Verifier for Object-Oriented Programs. In F. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Fourth International Symposium on Formal Methods for Components and Objects (FMCO'05), Post-Proceedings*, 2006.

[2] C. Barrett, L. de Moura, and A. Stump. Design and Results of the 1st Satisfiability Modulo Theories Competition (SMT-COMP 2005). *Journal of Automated Reasoning*, 35(4):373–390, 2005.

[3] Clark Barrett, Yi Fang, Ben Goldberg, Ying Hu, Amir Pnueli, and Lenore Zuck. TVOC: A translation validator for optimizing compilers. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the* 17^{th} *International Conference on Computer Aided Verification (CAV '05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 291–295. Springer-Verlag, July 2005. Edinburgh, Scotland.

[4] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1), 2001.

[5] Satyaki Das and David L. Dill. Counter-example based predicate discovery in predicate abstraction. In M. Aagaard and J. O'Leary, editors, *4th International Conference on Formal Methods in Computer-Aided Design*. Springer-Verlag, 2002.

[6] 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-Verlag, 2004.

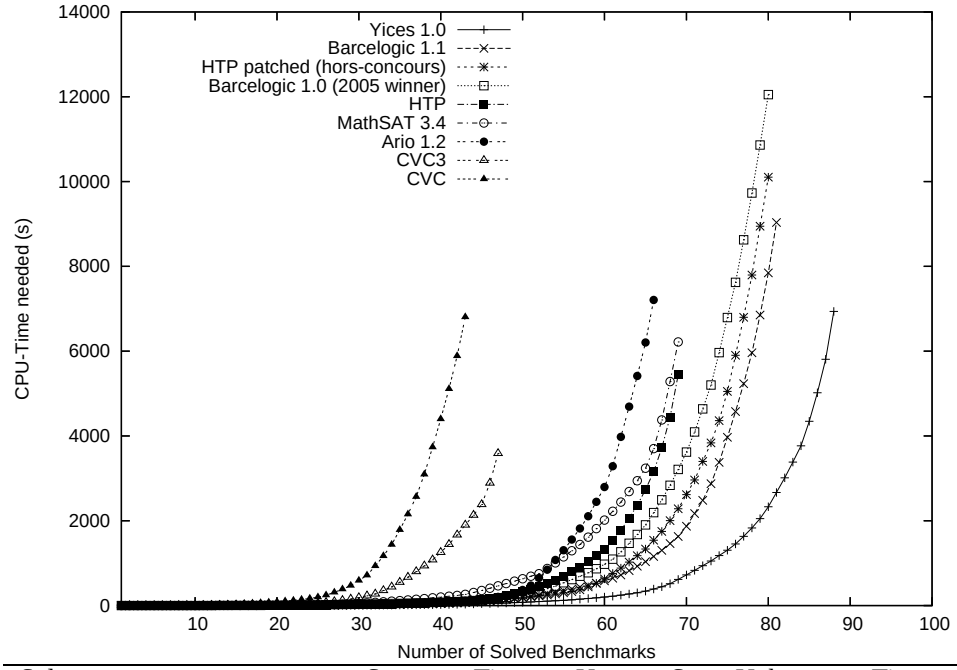
[7] S. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT Techniques for Fast Predicate Abstraction. In *18th International Conference on Computer-Aided Verification*, pages 424–437. Springer-Verlag, 2006.

[8] 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.

[9] S. McPeak and G. Necula. Data Structure Specifications via Local Equality Axioms. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer-Aided Verification*, pages 476–490. Springer-Verlag, 2005.

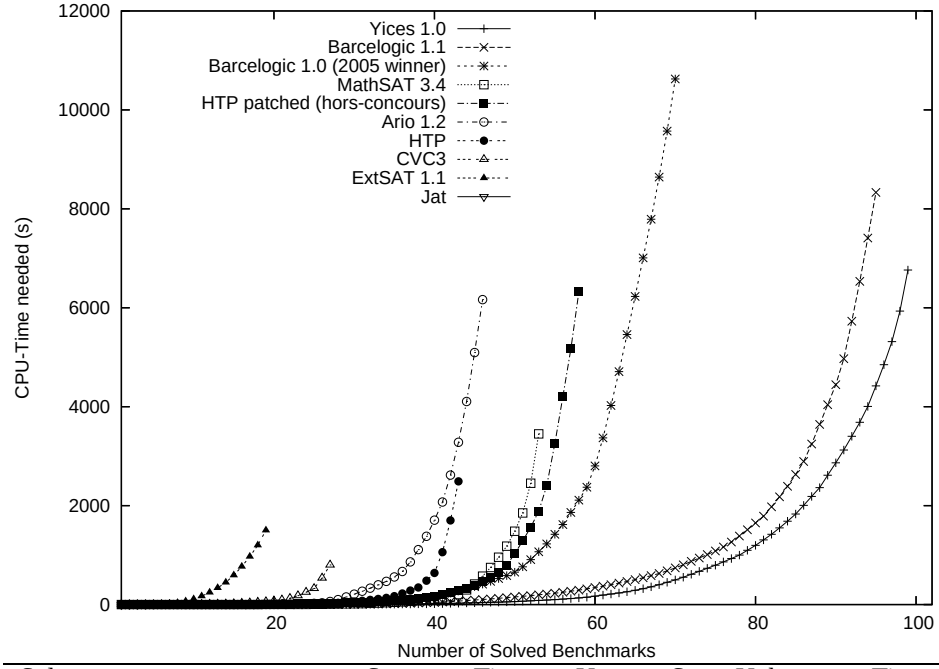
[10] F.J. Pelletier, G. Sutcliffe, and C.B. Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.

[11] 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.



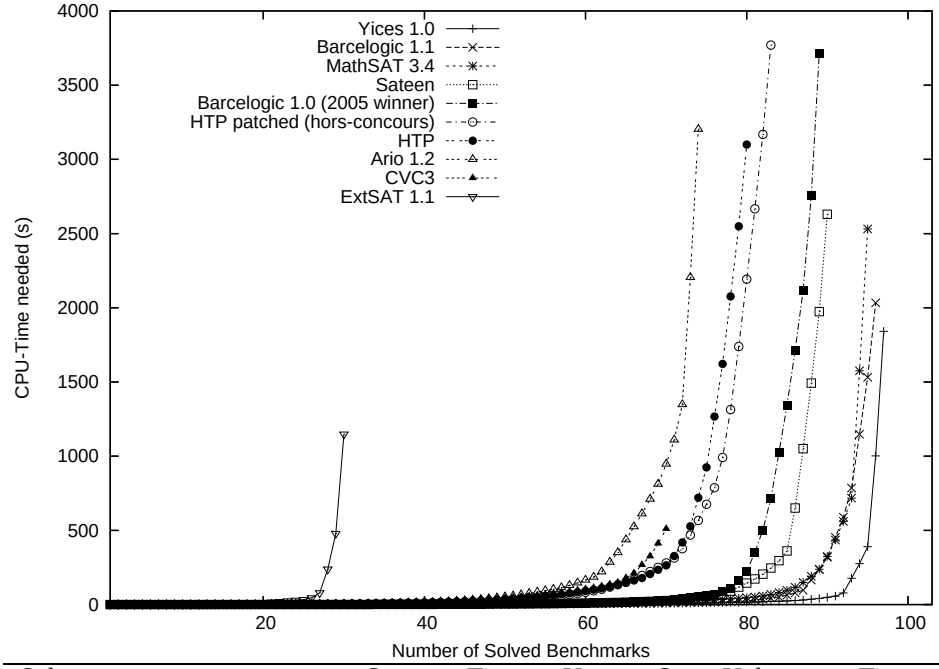
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	88	6937.1	76	12	0	12	0
Barcelogic 1.1	81	9035.2	69	12	0	19	0
HTP patched (hors-concours)	80	10104.3	68	12	2	18	0
Barcelogic 1.0 (2005 winner)	80	12050.5	68	12	0	20	0
HTP	69	5444.4	57	12	2	29	0
MathSAT 3.4	69	6216.8	58	11	0	31	0
Ario 1.2	66	7208.5	54	12	0	34	0
CVC3	47	3586.8	36	11	51	2	0
CVC	43	6805.2	32	11	14	43	0

Figure 4: Results of QF_UF



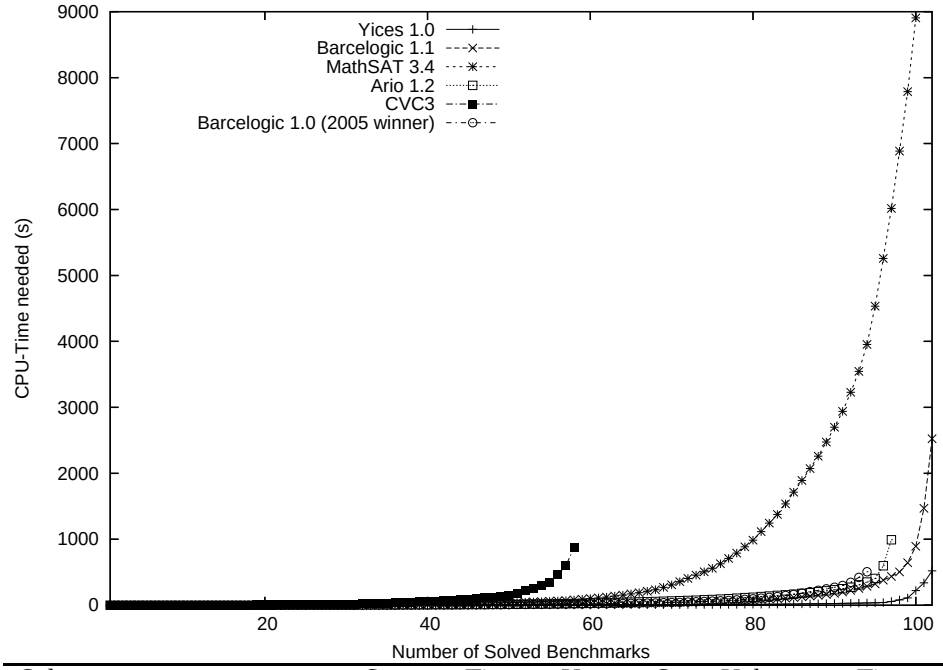
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	99	6761.8	76	23	0	3	0
Barcelogic 1.1	95	8332.3	72	23	0	7	0
Barcelogic 1.0 (2005 winner)	70	10624.1	56	14	2	30	0
MathSAT 3.4	53	3451.2	49	4	0	49	0
HTP patched (hors-concours)	50	6321.1	53	5	35	8	1
Ario 1.2	46	6164	43	3	2	54	0
HTP	35	2489.5	40	3	35	23	1
CVC3	27	800.8	26	1	75	0	0
ExtSAT 1.1	19	1502.2	19	0	2	81	0
Jat	-466	28.9	2	4	14	23	59

Figure 5: Results of QF_RDL



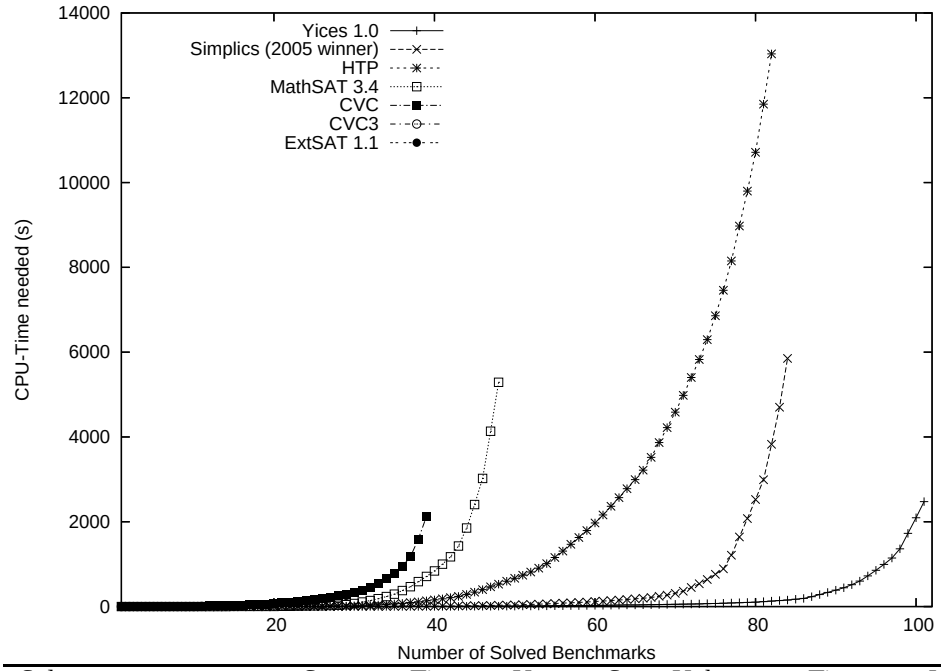
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	97	1841.1	45	52	0	6	0
Barcelogic 1.1	96	2033.2	44	52	0	7	0
MathSAT 3.4	95	2530.9	44	51	0	8	0
Sateen	90	2629.9	42	48	0	13	0
Barcelogic 1.0 (2005 winner)	89	3716	40	49	2	12	0
HTP patched (hors-concours)	83	3770	40	43	13	7	0
HTP	80	3099.9	39	41	13	10	0
Ario 1.2	74	3202.8	29	45	9	20	0
CVC3	70	509.9	32	38	31	2	0
ExtSAT 1.1	6	1148.2	30	0	2	68	3

Figure 6: Results of QF_IDL



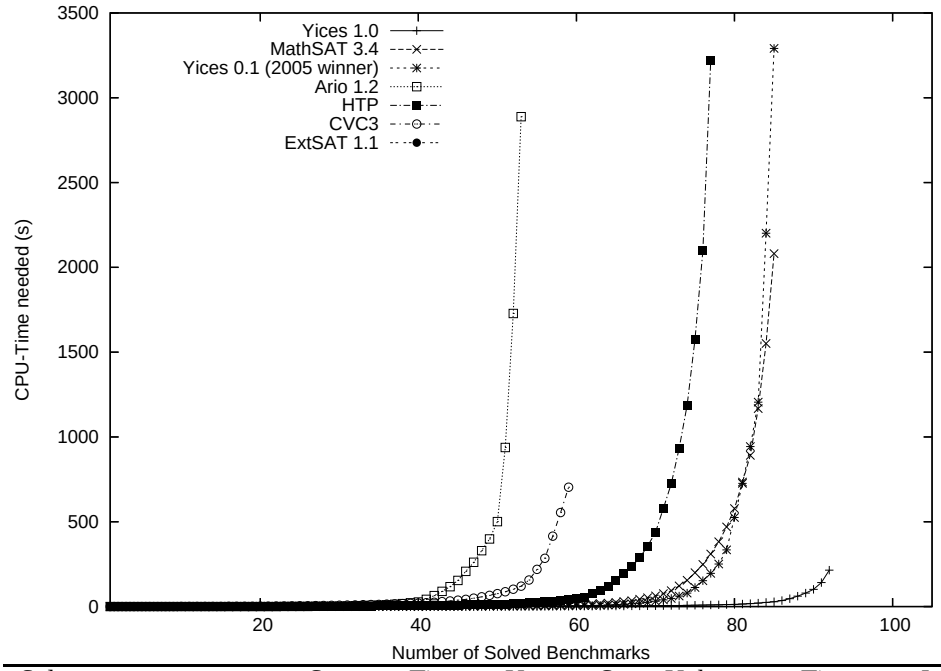
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	102	522.3	67	35	0	0	0
Barcelogic 1.1	102	2524	67	35	0	0	0
MathSAT 3.4	100	8905.8	65	35	0	2	0
Ario 1.2	97	991.9	62	35	2	3	0
CVC3	58	872.2	24	34	44	0	0
Barcelogic 1.0 (2005 winner)	54	506.6	59	35	2	1	5

Figure 7: Results of QF_UFIDL



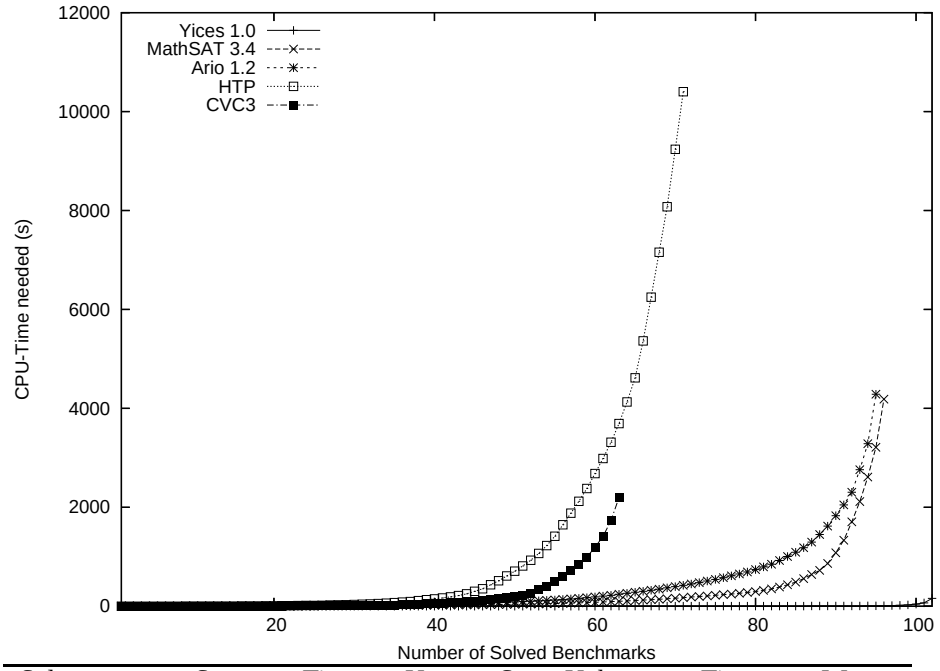
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	101	2475.5	50	51	0	1	0
Simplics (2005 winner)	84	5852.4	38	46	0	18	0
HTP	82	13034.5	38	44	1	19	0
MathSAT 3.4	48	5290.1	29	19	0	54	0
CVC	39	2122.3	22	17	54	9	0
CVC3	29	264.9	21	8	73	0	0
ExtSAT 1.1	2	0.9	2	0	67	33	0

Figure 8: Results of QF_LRA



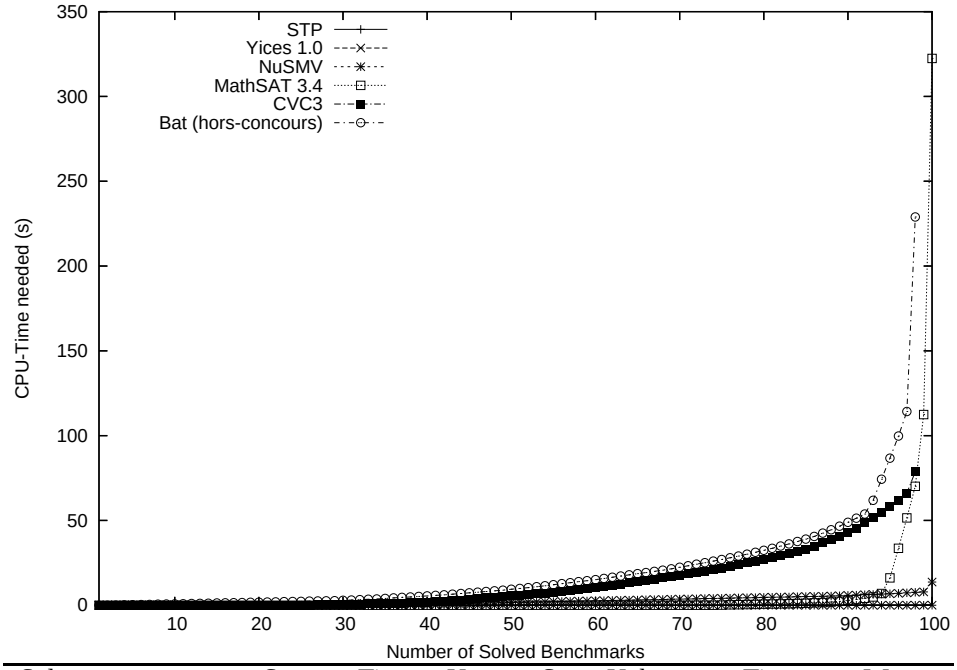
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	92	214.4	48	44	0	13	0
MathSAT 3.4	85	2080.8	46	39	0	20	0
Yices 0.1 (2005 winner)	77	3291.1	45	40	0	19	1
Ario 1.2	53	2888.3	29	24	22	30	0
HTP	53	3220.2	44	33	7	18	3
CVC3	43	703.7	33	26	43	1	2
ExtSAT 1.1	-34	2.5	6	0	58	36	5

Figure 9: Results of QF_LIA



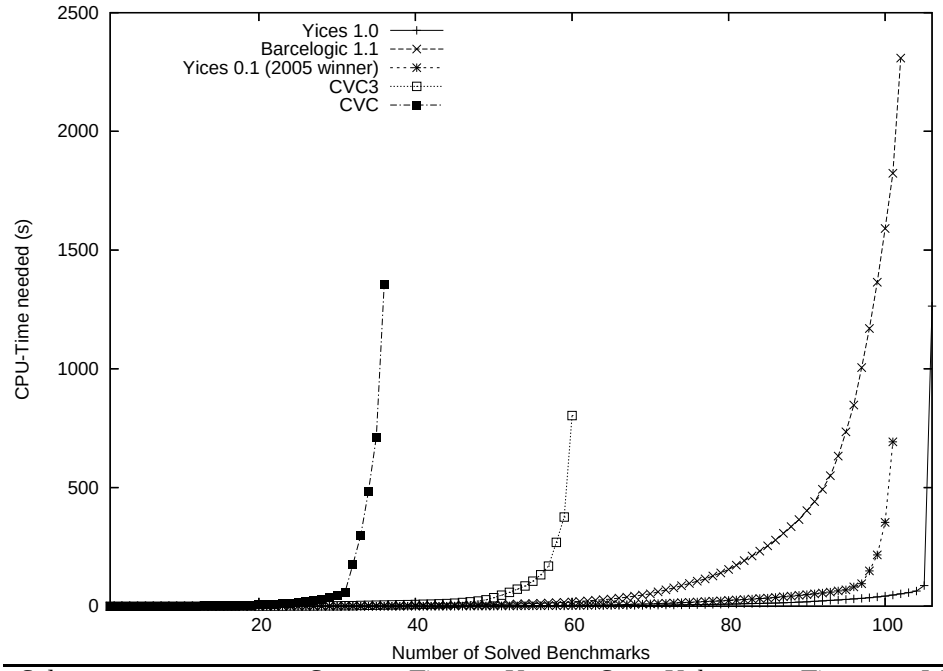
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	102	159.8	27	75	0	0	0
MathSAT 3.4	96	4185.1	22	74	0	6	0
Ario 1.2	95	4284.7	22	73	1	6	0
HTP	71	10400.6	21	50	0	31	0
CVC3	63	2197	18	45	39	0	0

Figure 10: Results of QF_UFLIA



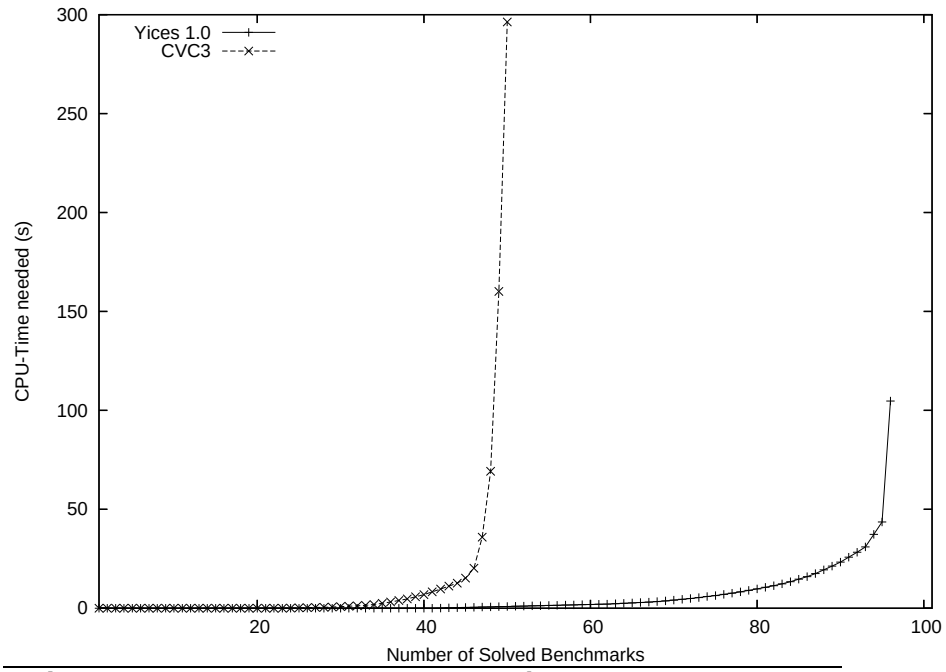
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
STP	100	0	51	49	0	0	0
Yices 1.0	100	0	51	49	0	0	0
NuSMV	100	13.7	51	49	0	0	0
MathSAT 3.4	100	322.4	51	49	0	0	0
CVC3	98	78.7	49	49	2	0	0
Bat (hors-concours)	82	228.9	50	48	0	0	2

Figure 11: Results of QF_UFBV[32]



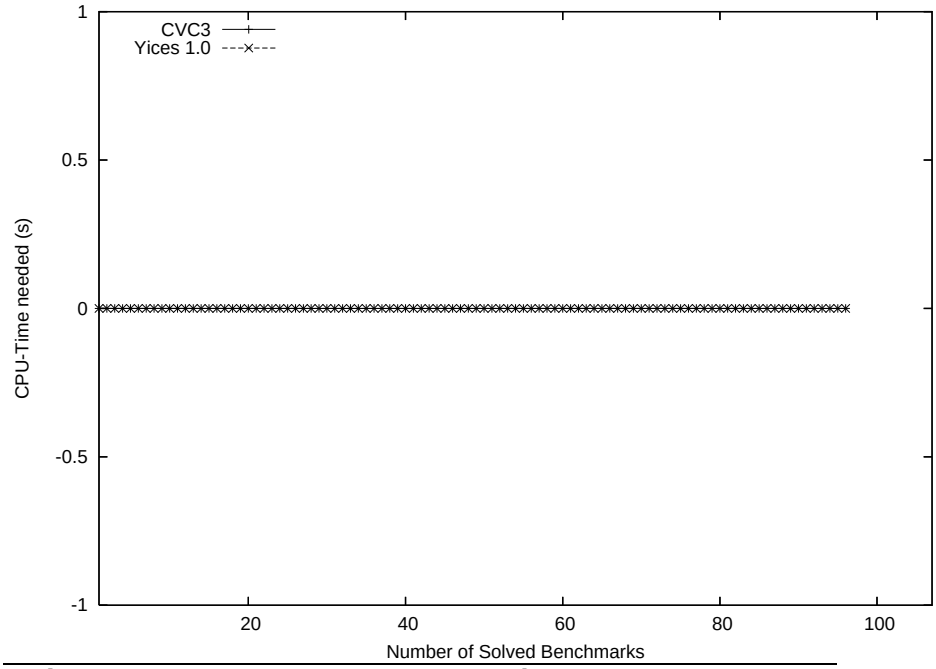
Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	106	1263.8	50	56	0	0	0
Barcelogic 1.1	94	2308.7	48	54	0	3	1
Yices 0.1 (2005 winner)	93	692.6	45	56	0	4	1
CVC3	60	802.9	36	24	46	0	0
CVC	36	1354.6	36	0	51	19	0

Figure 12: Results of QF_AUFLIA



Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
Yices 1.0	96	104.7	96	0	2	3	0
CVC3	50	296.3	50	0	51	0	0

Figure 13: Results of AUFLIA



Solver	Score	Time	Unsat	Sat	Unknown	Timeout	Wrong
CVC3	96	0	96	0	11	0	0
Yices 1.0	96	0	96	0	7	4	0

Figure 14: Results of AUFLIRA