jar manuscript No.
(will be inserted by the editor)

6 Years of SMT-COMP

**Clark Barrett · Morgan Deters ·
Leonardo de Moura · Albert Oliveras ·
Aaron Stump**

Received: date / Accepted: date

Abstract The annual Satisfiability Modulo Theories Competition (SMT-COMP) was initiated in 2005 in order to stimulate the advance of state-of-the-art techniques and tools developed by the Satisfiability Modulo Theories (SMT) community. This paper summarizes the first six editions of the competition. We present the evolution of the competition's organization and rules, show how the state of the art has improved over the course of the competition, and discuss the impact SMT-COMP has had on the SMT community and beyond. Additionally, we include an exhaustive list of all competitors, and present experimental results showing significant improvement in SMT solvers during these six years. Finally, we analyze to what extent the initial goals of the competition have been achieved, and sketch future directions for the competition.

Keywords SAT Modulo Theories · Competition · Experimental evaluation

1 Introduction

Domain-specific reasoning has emerged in the past decade or so as crucial for the success of automated reasoning tools for real-world applications [Sha02]. Consider problems involving arithmetic constraints, to take one example. Applying dedicated algorithms for solving arithmetic problems (e.g., Simplex in the case of linear arithmetic) allows one to exploit structural information that might be lost if one reduces the problem to a more general-purpose logic. If, in addition, we know that these arithmetic constraints come from software verification, we probably want to use modular arithmetic methods, since this is the type of arithmetic that computers indeed implement [Wan06]. In software verification, it is typical to generate problems where arithmetic terms appear as indices of arrays. Thus, the solver needs to have some means for reasoning in the theory of arrays. Finally, to make things even worse, a large skeleton of Boolean operators is usually present to express the relationships between the hundreds of constraints a medium-sized problem might have.

Satisfiability Modulo Theories (SMT) [BSST09] allows one to express such problems in a very natural way, using first-order formulas whose satisfiability is to be

New York University · New York University · Microsoft Research · Tech. University of Catalonia · The University of Iowa

checked modulo a background theory. The choice of the theory depends on the nature of the problem. Classic examples are linear arithmetic, arrays, bit-vectors, recursive data structures and uninterpreted functions. Using SMT to express such problems has important advantages over using more generic logics. If one uses purely propositional logic (SAT), for example, then data must be encoded into a Boolean representation: a bit-vector must be represented as just its individual bits, for example. In contrast, an SMT encoding can represent the bit-vector directly, and may be able to reason more efficiently at the bit-vector level of abstraction, without resorting to bit-level reasoning (though this may sometimes be necessary). And if one uses pure first-order logic (even with equality), one loses out on the myriad specialized algorithms that have been developed for many particular theories. Formulating problems with SMT keeps them at a reasonable level of abstraction. SMT tools are then free to tackle those problems using a rich variety of techniques: instantiation-based techniques for dealing with quantifiers [dMB07, GBT09], specialized Simplex-based algorithms for linear arithmetic [DdM06], graph-based algorithms for fragments of linear arithmetic [CM06, WIGG05], and even reductions to SAT if this is considered to be the best solving method [GTG07].

Devising a common input language for SMT solvers has been a highly non-trivial task for the SMT community, due to the need to accommodate a variety of different background theories, and express (at least informally) their syntax and semantics. This task has been addressed by the SMT-LIB initiative (`http://www.smtlib.org`), co-organized at first by Silvio Ranise and Cesare Tinelli, and later by Clark Barrett, Aaron Stump, and Cesare Tinelli. The goal of the SMT-LIB initiative is to promote the field of SMT through the development of standard formats for SMT solvers, standard definitions of SMT theories with respect to a precisely defined underlying logic, and the collection of a library of benchmarks in SMT-LIB format. The latter was inspired by the TPTP library of benchmarks for first-order theorem provers, curated by Geoff Sutcliffe [Sut09]. After a lengthy community process, SMT-LIB released the first version of the standard input language, named SMT-LIB v1.0, in July 2004. This included languages for describing theories and logics (logics modify the language that may be used, for example to contain only quantifier-free formulas, or to restrict arithmetic terms to be linear), and a standard input language for SMT solvers. It was an exciting day for the SMT community, and would prove to be so also for applications developers who would soon come to rely on this common format for SMT solvers. But there were as yet no solvers that supported the format, and no benchmarks in the library.

This is where the Satisfiability Modulo Theories Competition (SMT-COMP), which is the subject of this paper, made its first and most important contribution: jump-starting adoption of the SMT-LIB format and collection of benchmarks in that format. In just one year, from the release of the standard in July 2004 to the first SMT-COMP in the summer of 2005, the SMT community went from having its brand new standard input language but no compliant solvers and no benchmarks, to 12 compliant solvers and over 1300 benchmarks. When the SMT-COMP organizers announced these results to the audience at the Conference on Computer-Aided Verification (CAV), with whom SMT-COMP was affiliated in 2005, they received several rounds of spontaneous applause during their presentation. This warmly expressed the broader verification field's excitement about what SMT-COMP had been able to accomplish. Indeed, since two of the co-authors of this paper are also SMT-LIB co-organizers (Clark Barrett and Aaron Stump), we can say that the success the initiative has enjoyed has been highly dependent on SMT-COMP. For example, benchmark collection has always been driven by the competition. The competition organizers issue a call for benchmarks to the SMT community, and solicit them from applications developers. While new SMT benchmarks are certainly being created at other times, they have primarily been collected and added to the library in conjunction with finalizing the benchmarks for the annual competition. The implementation in SMT solvers of the SMT-LIB v1.0 standard, and its successor SMT-LIB v2.0, released March 2010, have also been strongly driven by the competition. Entering the competition requires the ability to read benchmarks in SMT-LIB format, and so in order to compete, solver developers invest the moderate but non-negligible effort required to write a parser for the format. Without SMT-COMP, we doubt (particularly in our capacity as SMT-LIB co-organizers), that adoption of the standard language would have occurred so quickly. This adoption was really the primary role of SMT-COMP.

An important secondary role of SMT-COMP has been to raise the profile of the SMT field in the broader research community. The competition has been held annually since 2005, collocated with either the Conference on Computer-Aided Verification (CAV) or the Conference on Automated Deduction (CADE). These are premier conferences in their respective fields of verification and automated reasoning, and the presence of SMT-COMP at these venues has significantly raised the profile of SMT within these communities. The SMT-COMP organizers learned a lot from the success of other competitions in automated theorem proving, such as the CASC competition for first-order theorem provers [SS06], which started in 1996, and the SAT competition for SAT solvers [SBH05], which started in 2002. From CASC, SMT-COMP adopted the practice of running the competition live during the affiliated conference, and announcing results at a special conference session. While running the competition "live" in this sense can be stressful for the organizers, the benefits in excitement from solver implementers and interest from members of the broader research communities makes this well worth the effort. SMT has gone from a minor field of automated reasoning to one of the most active and essential for applications. While this is, of course, due to a complex conjunction of factors, the role played by the competition is significant among them.

The competition has also served as a forum where implementers and users of SMT technology have met to express their needs and concerns, and has allowed the community to see how the performance of SMT systems has improved year after year. Indeed, a tertiary role for SMT-COMP has been to help spur development in the field, by giving new implementers a proving ground to establish themselves, and helping bring implementers and application developers together. This has been done so far in the context of the SMT Workshop (originally called Pragmatics of Decision Procedures in Automated Reasoning, PDPAR), with which each edition of the competition is always closely connected. As we will see below (Section 5.2), the competition has certainly seen significant improvement in solver performance and capabilities over the years. One need not find a direct causal connection to the competition in order to appreciate the role SMT-COMP has played in highlighting improvements to SMT solvers, and giving newcomers to the field an opportunity to prove themselves.

In 2011, the competition began its transition to a new set of organizers. This paper summarizes the six editions of the SMT-COMP which the authors organized. In contrast with previous SMT-COMP papers, we do not only focus on describing the technical settings of the competitions like its infrastructure or rules, but we also summarize the main characteristics of all SMT-COMP entrants over these years, analyze the impact the competition has had on the research community, judge to what ex-

Edition	Location	# Participants	# Divisions	# New Solvers
2005	Edinburgh	12	7	12
2006	Seattle	12	11	5
2007	Berlin	9	12	4
2008	Princeton	13	15	6
2009	Montreal	12	19	2
2010	Edinburgh	10	19	6

Table 1 Summary of the six SMT-COMP editions.

tent the initial goals have been achieved and present some conclusions that are worth considering for future editions.

The paper is organized as follows. In Section 2 we quickly overview some data about the different SMT-COMP editions. Section 3 concerns the benchmarks used for the competition, focusing on their organization and the language used. Section 4 describes the design of the competition, highlighting the changes it has undergone over the years. Section 5 describes the solvers that have entered SMT-COMP and presents some performance comparisons among winners of different editions. After that, Section 6 analyzes to what extent the initial goals of SMT-COMP have been achieved and presents some possible negative effects of the competition. We conclude in Section 7.

2 SMT-COMP Editions

SMT-COMP started in 2005, affiliated with CAV in Edinburgh. Since then, the number of participants has been quite stable, independent of whether it was affiliated with CAV (the first four and the sixth editions) or with CADE in 2009. Every year, new solvers entered the competition while some old entries decided not to participate. What has been increasing since the first edition is the number of different divisions, i.e. different types of problems, as we will explain in Section 3, starting with only 7 in 2005 and ending with 19 in 2010. All these data can be seen in detail in Table 1. For information about which are the different divisions, see Table 2.

The winners of different divisions have changed over the years, as can be seen in Table 3, where we summarize the winners of every division in the first six editions of SMT-COMP. A total of 13 different systems have won some division some year, which indicates that the competition is usually tough due to the quality of all entrants. In the 2006 edition, Yices 1.0 and STP tied on the QF_BV division, and Yices 1.0 and CVC3 tied on the AUFLIRA division. This was was due to all three systems solving all benchmarks in a negligible amount of time.

3 Benchmarks and their Language

3.1 Benchmarks

As has been stated, one of the primary goals of SMT-COMP has been to facilitate the collection and use of a large library of benchmarks in the standard SMT-LIB format. While a tremendous amount of effort was required to collect, translate and check the quality of these benchmarks, the effort has been very successful: the benchmark library

Logic	Description
QF_RDL	real difference logic
QF_IDL	integer difference logic
QF_LRA	linear real arithmetic
QF_LIA	linear integer arithmetic
QF_NIA	non-linear integer arithmetic
QF_NRA	non-linear real arithmetic
QF_UF	uninterpreted functions
QF_UFIDL	uninterpreted functions and integer difference logic
QF_UFLIA	uninterpreted functions and linear integer arithmetic
QF_UFLRA	uninterpreted functions and linear real arithmetic
QF_UFNRA	uninterpreted functions and non-linear real arithmetic
QF_AX	(extensional) arrays
QF_AUFLIA	arrays, uninterpreted functions, and linear int. arithm.
QF_BV	bit-vectors
QF_ABV	arrays and bit-vectors
QF_AUFBV	arrays, uninterpreted functions, and bit-vectors
LRA	quantifiers, linear real arithmetic
UFNIA	quantifiers, uninterpreted functs., non-linear int. arithm.
AUFLIA	quantifiers, arrays, unint. functs., linear int. arithm.
AUFLIRA	quantifiers, arrays, unint. functs., linear int. and real arithm.
AUFNIRA	quantifiers, arrays, unint. functs., non-lin. int. and real arithm.

Table 2 Description of competition divisions.

has grown from the 1360 benchmarks collected for the 2005 competition to more than 93,000 benchmarks available for the 2010 competition; nearly all SMT solvers support the format; and the SMT-LIB benchmarks are used in most serious papers reporting experimental results for SMT solvers. As the format has become more accepted, SMT users have also been more willing to support the format directly, making it easier to collect new benchmarks.

In the library, benchmarks are organized by *logic*, a formal notion in the SMT-LIB language standard (see below). Benchmarks are further divided into *families* (subdirectories under each logic). Typically, each family contains a set of benchmarks of varying difficulty from the same application. This same organization is used in the competition. For each logic containing a sufficient number and diversity of benchmarks, and for which there exist solvers supporting that logic, there is a *division* of the competition in which solvers compete on the benchmarks from that logic (occasionally divisions have been excluded for lack of interesting benchmarks or solvers). Table 4 shows, for each division and every year, the number of benchmarks in the library.[1] It is important to note that the changes from year to year are not only because new divisions and benchmarks were added, but also because occasionally divisions were split into new ones or existing benchmarks were removed (because they were trivial or redundant) or reclassified (into a more appropriate division). The addition of new divisions often coincided with new capabilities in the competing SMT solvers. For example, in 2006, quantified formulas were included; in 2006, bit-vectors (in the form of the QF_UFBV32 division) were new to the competition; and in 2009 and 2010, non-linear arithmetic divisions were run as part of the competition.

[1] Note that the table reports the availability of benchmarks in 21 divisions for SMT-COMP 2010. However, some of these benchmark divisions had too few benchmarks to run as a competition division. The benchmarks are still available as part of the benchmark library.

Edition	Winners	Divisions
2005	Barcelogic	QF_UF, QF_RDL, QF_IDL, QF_UFIDL
	Simplics	QF_LRA
	Yices 0.1	QF_LIA, QF_AUFLIA
2006	CVC3	AUFLIRA
	STP	QF_BV
	Yices 1.0	QF_UF, QF_RDL, QF_IDL, QF_UFIDL, QF_LRA, QF_LIA, QF_UFLIA, QF_UFBV32, QF_AUFLIA, AUFLIA, AUFLIRA
2007	CVC3 1.2	AUFLIRA
	Spear 1.9	QF_BV
	Yices 1.0.10	QF_RDL, QF_UFIDL, QF_AUFLIA, QF_UFLIA, QF_LRA, QF_LIA
	Z3 0.1	QF_UF, QF_IDL, QF_AUFBV, AUFLIA
2008	Barcelogic 1.3	QF_IDL, QF_AX
	Boolector	QF_BV, QF_AUFBV
	Yices 2	QF_UF, QF_LRA
	Z3 1.0	QF_RDL, QF_UFIDL, AUFLIA+p, AUFLIA-p, AUFLIRA, QF_AUFLIA, QF_UFLRA, QF_UFLIA, QF_LIA
2009	Barcelogic-QF_NIA	QF_NIA
	CVC3 2.0	AUFLIA-p, AUFLIA+p, AUFLIRA, UFNIA+p, QF_UFNRA, LRA, AUFNIRA
	MathSAT 4.3	QF_BV, QF_UFIDL
	Sateen-3.5	QF_IDL, QF_LIA
	Yices 2	QF_AX, QF_UFLRA, QF_AUFLIA, QF_UFLIA, QF_UF, QF_RDL, QF_LRA
2010	MathSAT 5	QF_UF, QF_UFLRA, QF_UFLIA, QF_LRA, QF_LIA
	OpenSMT-1.0-alpha	QF_RDL, QF_IDL, QF_UFIDL
	simplifyingSTP	QF_BV
	CVC3 2.3	QF_AUFLIA, QF_UFNRA, QF_ABV, QF_AX, AUFLIA+p, AUFLIA-p, AUFLIRA, AUFNIRA, UFNIA+p
	MiniSMT	QF_NIA, QF_NRA

Table 3 Winners of the different SMT-COMP editions.

3.2 The SMT-LIB Standard

The SMT-LIB standard includes a concrete syntax for first-order terms and formulas using many-sorted first-order logic. It also provides for a number of annotations, both within the logical structure of terms and formulas as well as in addition to it. Each benchmark consists of a set of formulas whose conjunction needs to be checked for satisfiability, preceded by several attributes with meta-data about that benchmark: *name*, *source*, *status*, *category*, *difficulty*, and *logic*.

The *name* of a benchmark is used only as additional documentation - it is not used in the competition (the family and filename are used to identify benchmarks in the competition). The *source* attribute gives information about where the benchmark came from. The *category* attribute identifies one of three categories to be associated with the benchmark: *crafted* - meaning hand- or machine-crafted benchmarks that serve no purpose other than to test SMT solvers; *random* - randomly generated from some distribution; and *industrial* - a bit of a misnomer since most are from academia, not industry, but used to indicate that the benchmark was generated from some piece of application software (interpreted broadly). The *difficulty* is a numeric value from 0 to 5, calculated for each year's SMT-COMP using the results of previous years' solvers (see

Division	2005	2006	2007	2008	2009	2010
QF_RDL	170	204	204	204	204	255
QF_IDL	343	1145	1145	1673	1672	1670
QF_LRA	174	501	501	543	543	634
QF_LIA	182	237	203	3277	3970	5259
QF_NIA					470	344
QF_NRA						166
QF_UF	152	152	6556	6656	6655	6647
QF_UFIDL	285	399	400	432	431	428
QF_UFLIA		110	110	564	564	562
QF_UFLRA				900	900	900
QF_UFNRA					26	26
QF_AX				1485	552	551
QF_AUFLIA	54	3729	3729	2244	1140	1140
QF_BV			2011	2505	31582	31046
QF_ABV						14335
QF_AUFBV		8247	8168	8461	8644	
LRA					374	374
UFNIA					1813	1799
AUFLIA		932	4534	4566	6445	6402
AUFLIRA		26511	27836	28034	25544	19917
AUFNIRA					1167	989
TOTAL	1360	42167	55397	61544	92696	93480

Table 4 Number of available benchmarks per division and year.

below). Finally, the *logic* attribute consists of the name of one of the logics available in the SMT-LIB standard. Each logic specifies a background theory and often also includes a set of syntactic restrictions on the first-order language associated with that logic. For example, the logic QF_UFLRA requires formulas to be quantifier-free and in a language mixing uninterpreted (free) function symbols and linear real arithmetic. Following the attributes, each benchmark contains one or more formulas (all but one labeled as *assumptions* and the final one labeled as the *formula*) whose conjunction is to be tested for satisfiability within the theory specified by the logic.

The competitions from 2005 through 2009 used version 1.2 of the SMT-LIB standard, which is described in detail in a document available online at `http://www.smtlib.org`. In 2010, a new version of the standard, version 2.0, was released [BST10]. The purpose of the new version was (among other things) to simplify syntax, make it easier to specify parametric theories and combinations of theories within logics, and to create a standard command language for SMT applications (benchmarks in version 2.0 are simply scripts in the command language).

The SMT-LIB standard continues to grow and adapt in response to the needs of the community. Ongoing efforts aim to refine and improve version 2.0, to introduce standard API's, to standardize a format for proofs, and, of course, to add new theories, logics, and benchmarks.

4 Design of the Competition

Naturally enough, the design of SMT-COMP has changed and developed over the six years the competition has run. This section discusses the evolution of the competition, in particular its rules and policies, approach to benchmark selection, and computing

Division	2005	2006	2007	2008	2009	2010
QF_RDL	10	8	5	6	6	3
QF_IDL	11	8	5	7	7	3
QF_LRA	9	6	6	5	4	4
QF_LIA	9	6	5	4	4	2
QF_NIA					2	3
QF_NRA						2
QF_UF	10	7	5	6	5	4
QF_UFIDL	10	5	4	4	4	3
QF_UFLIA		5	5	4	3	2
QF_UFLRA				4	3	2
QF_UFNRA					1	1
QF_AX				3	3	1
QF_AUFLIA	6	4	3	3	3	1
QF_BV			3	7	8	3
QF_ABV						1
QF_AUFBV		5	2	3	4	
LRA					1	
UFNIA					1	1
AUFLIA		2	4	3	1	1
AUFLIRA		2	3	3	1	1
AUFNIRA					1	1
TOTAL	12	12	9	13	12	10

Table 5 Number of solvers per division and year.

resources and infrastructure. As is perhaps not too surprising, the competition has moved towards more sophisticated infrastructure and more elaborate benchmark selection algorithms. At the same time, the competition's approach to correctness of solvers has developed more slowly, perhaps an indication of how challenging it is to be sure that systems as complex as modern SMT solvers are absolutely bug-free.

The basic format of the competition has remained constant over its history: for each division of the competition, submitted solvers are invoked on selected benchmarks in the division, subject to a fixed time limit per benchmark. Solvers attempt to classify benchmarks as either `sat` (satisfiable) or `unsat` (unsatisfiable). They may also report `unknown` for benchmarks which they are unable to solve (for whatever reason). Results are posted incrementally to the competition website, and finally announced in a short (15-20 minutes) presentation at the sponsoring conference. We first consider issues which arose historically in implementing this basic scheme, and then turn to changes in the competition's computing infrastructure.

4.1 Scoring and Penalties

Scoring system. The basic scoring system is very simple and has not changed since the first edition. One point is awarded for each correct `sat` or `unsat` answer that is given within the time limit. The winner of the division is determined by score. In 2005, ties were broken by comparing the total time taken for benchmarks that were correctly solved.

The time limit has always been between 10 and 30 minutes and is decided by the organizers depending on the number of entrants and the available computing resources. The goal is to let the competition run over the course of several days during the sponsoring conference, but make sure that the results are available by the last day of the conference so that they can be presented.

Penalties. There have been several changes over SMT-COMP's history with respect to the penalties for wrong answers. For the first SMT-COMP (2005), answering `unsat` for a benchmark that is, in fact, satisfiable was penalized by -8 points, while answering `sat` for an unsatisfiable benchmark received -4 points. The rationale for having different penalties was to penalize solvers which are unsound (viewed as a proof system) with respect to the model-theoretic semantics of SMT more heavily than those which are incomplete. The choice of -8 and -4 was somewhat arbitrary. Recall that unsatisfiability naturally corresponds to validity, and satisfiability to invalidity. Hence reporting `unsat` for a satisfiable formula corresponds to reporting that a formula is a theorem when it really is not, while reporting `sat` for an unsatisfiable formula corresponds to failing to report that a formula is a theorem when it really is.[2]

At the time, the organizers agreed to penalize incompleteness less than unsoundness, on the grounds that completeness is generally harder to achieve. This assessment was based on the organizers' personal experience developing SMT solvers: it was quite rare to perform an unsound inference in practice, but it was not rare to fail (due to bugs in the solver's code) to perform some inferences required for completeness. Put another way: to achieve the highest possible performance, there is significant pressure on the solver's implementation to optimize as much as possible. Such optimizations are intended to reduce the amount of work the solver performs. It was, at the time, the organizers' experience that it was more common to reduce (erroneously) the deductions performed to the point of incompleteness, rather than to perform deductions unsoundly. No limit was imposed on the number of wrong answers.

Scoring and penalties revised. The competitions after SMT-COMP 2005 all dropped the distinction between soundness and completeness errors: any wrong answer receives -8 points. The organizers' initial motivation was not sufficient to justify retaining the distinction in the face of criticism from members of the SMT community, who advocated using the same penalty in each case. The competitions after 2005 explicitly treated both crashing and exhaustion of memory like a timeout. For 2006 and 2007, ties were broken by comparing the total time taken on benchmarks for which the solver did not crash or report `unknown`. Starting in 2008, this was modified slightly to consider just the total time taken on benchmarks for which the solver reported the correct answer. The only difference, of course, is that this does not include time solvers took to report an incorrect answer. We judge now that it is the purpose of point penalties to account for wrong answers, and thus the simpler statement of the tie-breaking rule is preferable.

Disqualification. SMT-COMP 2005 revealed several issues with the scoring scheme. First, due no doubt to the fact that the competition was just starting, there were several entrants with rather large numbers of wrong answers in certain divisions (for example, in the `QF_IDL` division: see the results section for 2005 on `www.smtcomp.org`). SMT-COMP 2006 through SMT-COMP 2008 added a rule disqualifying, from the whole competition, any solver that gives more than three wrong answers in a single division. Starting with SMT-COMP 2009, however, the competition dropped this requirement, in response to perceptions that the barrier to entry for the competition was too high. In particular, one solver (Alt-Ergo), which was entered for the first time in 2008,

[2] While other possible definitions of soundness and completeness are reasonable in this context, our usage is consistent with the norms in the SMT and verification research communities.

suffered from a superficial bug which led it to give many wrong answers. This bug was not in the actual solver algorithms, but rather was a misunderstanding by the implementers who assumed for quantified formulas, an answer of `sat` would be treated the same as an answer of `unknown`. Once the problem was detected, the solver implementers quickly fixed the problem, and the solver did not exhibit any incompleteness. The revised solver ran *hors concours* with the competition. But the damage was done, so to speak, because the originally submitted tool was identified as disqualified for wrong answers on the competition web site. This kind of identification was viewed as potentially discouraging to new competitors. So for SMT-COMP 2009, the disqualification rule was removed. Solvers getting many wrong answers will certainly not be competitive. It is overkill, we now believe, to highlight their problems by disqualifying them.

In 2009, a paper was presented at the SMT workshop demonstrating that by running a sufficient number of randomly generated benchmarks (generated using a "fuzzer" tool), it was possible to crash or elicit incorrect answers from many existing SMT solvers [BB09a]. In 2010, in order to encourage implementers to make their solvers more robust, a small number of benchmarks generated by the fuzzer of [BB09a] were used in every division of the competition. Since these were the only benchmarks used in the competition that were not present in the SMT-LIB library, we considered that their presence would encourage participants to use the fuzzer to debug their systems.

Discussion. Of course, the field is certainly striving for totally correct solver implementations. In the absence of formal verification for solver implementations themselves, however, this is difficult to achieve and impossible to certify. Indeed, one might argue that the experience of fuzzing SMT solvers, which was able to produce incorrect answers from 6 well-known SMT solvers for bit-vector reasoning, confirms this [BB09a]. SMT solver implementations tend to be much larger than those for SAT or first-order provers. A typical number one hears from solver implementers is on the order of 40kloc-60kloc of C/C++, and some solvers have much more (the open source CVC3 solver, version 2.2, for example, has over 130kloc of C++). Furthermore, as noted above, the organizers frequently hear that the barrier to entry for the competition is too high, stifling development of new solvers. In this direction, we decided to be more tolerant of the occasional error in an SMT solver, rather than stigmatize tools as incorrect if they demonstrate an error. An alternative to formal verification of SMT solvers is to have solvers that produce an independently certifiable proof for unsatisfiable instances. While some solvers already do this to some extent, SMT solvers will not enjoy the same degree of usability for certified verification as they do for uncertified verification, until they can produce proofs in a common format. The challenging problem of devising such a format is an active research area in the SMT community.

4.2 Solver Requirements

Open vs. closed source. The SMT community has often discussed the question of whether or not solvers entering SMT-COMP should be required to release their source code. SMT-COMP has always adopted what we view as a compromise position: solver binaries are made publicly available on the SMT-COMP web site following the competition (and remain available thereafter), but source code need not be provided or released by solver implementers in order to compete. This is in contrast with at least the 2009 version of the SAT Competition, where any solvers provided only in executable form are relegated to a demonstration division [LBS06]. The competition division requires solver implementers to allow their source code to be posted on the competition website.

Due, it seems, to the higher barrier of entry for SMT, there are usually significantly fewer SMT solvers in SMT-COMP than there are entrants to the SAT Competition. Other causes might be that SMT technology is more recent than SAT or simply that SAT is more attractive to users and developers. For example, the main SAT track had over 25 competitors in 2009, while the largest SMT-COMP division in 2009 (`QF_BV`) had 8 competitors (see Table 5 for more details about number of competitors). The smaller number of competitors makes it more difficult for SMT-COMP to insist (as the 2009 SAT Competition did) on what is surely the scientifically most progressive rule: all solvers must be submitted in source form, and all sources will be posted following the competition.

Among the 35 solvers that have entered the competition at least once, only 15 of them have been closed source (see Table 6). The bad news is that, perhaps surprisingly, many important SMT solvers are closed source. Solvers like Yices, Barcelogic, Z3, and MathSAT, to name just a few, all have performed at the highest levels in various divisions over the history of the competition, and all are closed source, though their implementers freely provide binaries for academic use. In some cases, there are commercial reasons for remaining closed source. For example, the Barcelogic solver is the central technology of a start-up company of the same name, spun out in 2009 from the research group (including the fourth co-author of this paper) that develops the solver [BNO+08].

Over the history of the competition, the SMT-COMP organizers have been persuaded that making binaries public but not requiring open source represents a reasonable compromise between the needs of various research teams. Certainly, the academic SMT community would benefit from more open-source systems. Indeed, open-source solvers like CVC3 and OpenSMT are eagerly used in some projects where access to source code is required. Of course, many other properties of open-source software are required to make solvers easy to adapt or modify for application needs: the source code should have a well-documented in-memory interface, should be simply yet flexibly designed, and for some users, should be modifiable or usable for commercial purposes. But requiring open source would shut out solvers that are being applied commercially, since their implementers have a business need to protect their intellectual property from competitors. These implementers would simply not participate in the competition, thus depriving the community of innovation-driving competitors, and possibly discouraging such implementers from submitting papers about their technology. Such papers, submitted by all the above-mentioned closed-source groups, have revealed essential algorithmic ideas that have benefited many other solvers, e.g. the use of theory propagation [NO05] or the Simplex algorithm of [DdM06]. Finally, providing binaries helps application developers, who wish to utilize SMT solvers as back-ends for solving other problems.

Wrapper tools. In 2006, a solver called HTP was entered in SMT-COMP, which, as its system description documented, executed a preprocessing step and then passed along the resulting formula to one of several SMT solvers entered in SMT-COMP 2005. While this was perfectly appropriate according to the SMT-COMP 2006 rules, the organizers for 2007 felt that rules should be put in place to help make clear what value was added by the wrapped tool over and above the tool that it wrapped. Starting in 2007, wrapper tools were recognized by the SMT-COMP rules, which require a wrapper tool to list the wrapped tool's name as part of its name. The rules further require an implementer of a wrapper tool to obtain permission from the implementer of the wrapped tool if a version of the wrapped tool is being used which was released after the previous year's SMT-COMP. This rule is to prevent competitors from taking too much advantage of the good performance of another tool, particularly performance achieved after the previous competition. Finally, wrapper tools which do not best their wrapped tools in a given division are disqualified from that division.

Unlike for SAT, where an entire track of the 2009 competition is dedicated to modifications of the award-winning MiniSAT solver, wrapper tools have not been common in SMT. Indeed, with the exception of HTP, there have been none. Thus, the administrative investment of devising new rules for wrapper tools has yielded few returns so far. Nevertheless, it seems appropriate to have a codified policy for systems that seek to build on others' work.

Organizer entries. Another sensitive point for competitions like SMT-COMP and others is the participation of the competition organizers. The safest policy is, naturally enough, to exclude competition organizers from competing. The SAT Competition 2007, for example, did not allow organizers to enter the competition division, but only the demonstration division. In 2009, the SAT Competition allowed organizers to compete in both divisions, provided they released the MD5 checksums of their solvers before submission of other solvers began. This rule makes sense in the context of an open-source competition, where it would otherwise be very difficult to establish that organizers had not cheated by taking good ideas from other systems' source code.

SMT-COMP has always allowed organizer submissions. The rationale is that, as noted above, there are typically a much smaller number of SMT solvers competing in SMT-COMP than in the SAT competition. Excluding the organizers in any given year would have excluded 2 or 3 solvers, a non-trivial percentage of the total field. Furthermore, since SMT solvers are typically submitted for competition in binary form only, the organizers have not felt it necessary to put policies in place to be able to prove that they are not unfairly inspecting competitors' submissions. Since 2008, (enabled by a change to the competition's computing infrastructure–see Section 4.4 below), the MD5 checksums for all solvers have been made public just after the close of solver submission. This helps ensure that the solvers which are executed during the competition are the intended ones.

4.3 Benchmark Selection

A reasonable number of benchmarks are drawn randomly from SMT-LIB for the competition. The final number of benchmarks has varied somewhat based on the expected execution time of the competition. This is more than simply a matter of available computing resources; as SMT-COMP has always run *live* during CAV or CADE, a spirited competition, with adequate suspense and excitement, and a proper conclusion, have always been among the design goals. In 2008, for example, some divisions of the competition were run with different time limits, to ensure timely completion of the competition.

Using the full SMT-LIB as a benchmark source has consequences. As SMT-LIB aims to be a repository for as many available benchmarks as possible, some open problems are included in the library. Without a trustworthy status of the satisfiability of a benchmark, a benchmark is inadequate for scoring in a competition of SMT-COMP's design.[3] Further, many large families of benchmarks in SMT-LIB are all fundamentally encodings of the same problem with different parameters. Inclusion of all of these parameterizations in the library is in the library's interest, but not in the best interest of the competition. To provide diversity among the benchmarks and ensure that winners of the competition have a breadth of capabilities, we have taken care not to over-represent these large families in SMT-COMP.

The current benchmark selection algorithm takes into account:

Benchmark difficulty: Benchmark difficulties are recomputed for each competition year using the previous competition year's solvers. In years 2005-2009, a benchmark's difficulty was simply a function of the fraction of solvers that could solve the benchmark within ten minutes. Due to popular demand from the community, the 2010 difficulty calculation took time into account as well. The selection process tries to include approximately equal numbers of benchmarks at different difficulties within each division; this is not always possible, as the distribution of difficulties within a division is not uniform.

Benchmark category: Many benchmarks are encodings of problems that arise in industrial settings in scheduling, planning, and verification. These are favored by SMT-COMP's selection procedure over hand-crafted and randomly-generated benchmarks. (However, simple hand-crafted benchmarks checking for particular features of note, such as support for large integers, are considered "check" benchmarks and are always included in the competition.)

Benchmark status: The selection procedure for SMT-COMP tries to include equal numbers of satisfiable and unsatisfiable benchmarks at different levels of difficulty. Benchmarks with unknown status are not included.

Benchmark family: The selection procedure does not require all benchmark sources and families to be represented; however, the effect of large families is reduced to allow for smaller benchmark families to be represented in competition.

In order to provide transparency, and to ensure that no unfair advantage is given to competition organizers who may also be competitors, the benchmark selection procedure is designed to be verifiable (repeatable) by any interested party during or after the competition.

The benchmark selection algorithm is published with the rules, and an implementation is released in source form. The set of available benchmarks for selection (input to this software) is known before the competition deadline (there are no secret benchmarks). The other input to the selection algorithm, a random seed, is computed by combining contributed integers from all entrants and a published opening stock index the week of the competition.

Selected benchmarks are slightly scrambled to discourage problem recognition; this is done by providing the competition random seed as input to a publicly available benchmark scrambler. Scrambling consists of reordering arguments to commutative Boolean connectives like conjunction and disjunction, renaming variables, and similar shallow syntactic changes. Note that the performance of SAT solvers has been shown to be quite sensitive to such changes [Nik10].

[3] As SMT-COMP entrants are not required to provide evidence of (un)satisfiability, there is no way to judge a SAT or UNSAT response to a benchmark with unknown status.

4.4 Computing Infrastructure

Certain things have remained constant throughout the history of the competition with regard to the computing infrastructure used to run the competition. Naturally enough, solvers have always been run on machines without other user-processes running, and all solvers in a division run on machines with the same hardware and operating system versions and configurations. Results have always been reported incrementally; as soon as a solver completes a benchmark, its reported answer and run-time are incorporated into the results section of the competition website.

Another important fact is that, in order to check that no solver is benefited by the computing infrastructure, anyone can, in principle, rerun the competition on their own machines to confirm the results. In the 2006 competition, the authors of the STP solver did rerun the `QF_UFBV32` division of the competition, and found a discrepancy with the reported results: the competition reported STP and Yices 1.0 as tied for first place taking 0 measurable seconds, while the STP authors reported STP ahead, with 1.05 seconds for STP and 1.77 for Yices. The source of the problem was, apparently, the trivial nature of the benchmarks for that division, which was new in 2006, coupled with differences in the specifications of the machines (the competition machines had significantly more cache than the machines used by the STP authors). In the end, the tools were declared as tied, and efforts were made to collect more challenging benchmarks for subsequent years.

The first computing infrastructure, in 2005, was a small (tens of nodes) cluster of Linux machines, kindly made available for the competition's exclusive use by the University of Edinburgh's School of Informatics. For 2006, the competition ran on a similar cluster hosted at SRI. Leonardo de Moura wrote the scripts to distribute the workload of the competition across the cluster nodes, and collect the results for web display. From 2007 on, the competition has been run on a web service called SMT-EXEC (`http://www.smtexec.org`), designed by Morgan Deters and Aaron Stump, and implemented by Morgan Deters. This web service allows registered users to upload and run solvers on the benchmarks from the SMT-LIB benchmark library, using a small compute cluster hosted at Aaron Stump's institution (initially Washington University in St. Louis and then the University of Iowa). This cluster was purchased with funds from a grant from the U.S. National Science Foundation (grant number CNS–0551697).

On SMT-EXEC, users submit jobs, consisting of a selection of solvers to be run on a selection of benchmarks. SMT-EXEC breaks each job up into chunks of tasks, where a task is to run all selected solvers on a single benchmark. Each task is then executed (via sequential execution of the solvers) on a single node of the cluster. The chunking mechanism helps to reduce latency for new jobs to begin execution, since chunks from different jobs are interleaved. SMT-EXEC builds on visualization scripts written by Leonardo de Moura for SMT-COMP 2006. These visualizations can be used to compare the whole field of solvers in a division, as well as to compare two solvers head-to-head. Such visualizations are quite helpful in understanding differences in solver performance, for example across different families of benchmarks. In addition to such improved interface features, SMT-EXEC uses a MySQL database to keep track of information about benchmarks, solvers, users, and jobs. This has helped SMT-EXEC scale to 65 registered users, and multiple revisions of the benchmark library.

With such an open computing infrastructure, one could ask whether there is still room for competitions, since one could, at any time, run one's own competition on SMT-EXEC. The first important difference is that SMT-COMP gives the possibility to all SMT developers to submit their latest versions. Hence, it is a more trustworthy snapshot of the current state of the art. Moreover, the social aspect of SMT-COMP should also be taken into account. The days of the competition are very intense in terms of discussions among SMT developers, something which is more difficult to achieve in ordinary conferences, with a smaller attendance of SMT implementers.

4.5 Concurrent Track.

In order to stimulate research on parallel SMT solvers, we ran a concurrent track in 2010. Only two systems entered this track: testp_mathsat, which competed in QF_UF, QF_LRA, QF_LIA, QF_UFLRA and QF_UFLIA; and simplifyingSTP, which competed in QF_BV. Results were not very conclusive as these two tools were still very initial prototypes: testp_mathsat ran several instances of the sequential MathSAT 5 solver, and simplifyingSTP converted the formula to propositional logic and sent it to the parallel SAT solver ManySAT.

5 Solvers

5.1 Competition Entrants

There have been 35 distinct entrants over the first six years of SMT-COMP. A summary of the main characteristics of all solvers is shown in Table 6. In Table 7, the reader can find the years each solver participated in some division and in Table 8 we summarize the divisions each solver ever entered. A more detailed description of each solver follows:

Alt-Ergo. (2008.) Alt-Ergo was submitted by Sylvain Conchon (LRI, Université Paris-Sud and INRIA Saclay Ile-de-France) and Evelyne Contejean (LRI, CNRS, and INRIA Saclay Ile-de-France). Alt-Ergo is an OCaml implementation of a generic congruence closure algorithm over an equational theory (for the submission, the equational theory was instantiated with linear arithmetic). It provides for a method of theory combination similar to Shostak's approach [CCKL08]. Alt-Ergo includes a custom-built SAT engine and supports quantifiers. In 2008, it competed in the quantified divisions (AUFLIA+p, AUFLIA$-p$, and AUFLIRA).

AProVE NIA. (v0.2.1, 2010.) AProVE NIA was submitted by Karsten Behrmann, Andrej Dyck, Fabian Emmes, Carsten Fuhs, Jürgen Giesl and Patrick Kabasci from RWTH Aachen University, Germany; Peter Schneider-Kamp from University of Southern Denmark and René Thiemann from University of Innsbruck, Austria. AProVE is a termination prover that uses non-linear integer arithmetic when using well-founded orders based on polynomial or matrix multiplication. It only competed in QF_NIA, using an eager encoding into propositional logic and giving the resulting formula to MiniSAT.

ArgoLib. (v3.5, 2007.) ArgoLib was submitted by Filip Marić and Predrag Janičić from the University of Belgrade, Serbia. ArgoLib is a C++ implementation of DPLL(T), coupling a rational reconstruction of MiniSAT [ES03] with two rational linear arithmetic solvers, one based on Fourier-Motzkin [Wil76] and another one based on the Yices

Solver	Institution	Source	Progr. Language	Appr.	SAT Solver
Alt-Ergo	INRIA Saclay ProVal	Op.	OCaml	L	Custom
AProVE NIA	RWTH Aachen	Cl.	Java	E	MiniSAT
ArgoLib	Univ. Belgrade	Op.	C++	L	Custom
Ario	Univ. Michigan	Cl.	C++	L	Custom
Barcelogic	Technical Univ. Catalonia	Cl.	C++	L	Custom
Beaver	UC Berkeley	Op.	OCaml	E	Rsat/NFLSAT
Boolector	Johannes Kepler Univ.	Op.	C	E	PrecoSAT
clsat	Washington Univ St Louis				
	Univ. of Iowa	Op.	C++	L	Custom
CVC	Stanford Univ.	Op.	C++	L	Chaff
CVC Lite	Stanford + New York Univ	Op.	C++	L	Custom
CVC3	New York Univ + U. Iowa	Op.	C++	L	MiniSAT
CVC4	New York Univ + U. Iowa	Op.	C++	L	MiniSAT
ExtSAT	Inst. Superior Técnico	Cl.	C++	L	MiniSAT
Fx7	Univ. of Wroclaw	Op.	Nemerle	L	MiniSAT
HTP	Formal Doc. Systems	C	Cl.	P	MiniSAT
Jat	Verimag	Op.	Java	L	Custom
MathSAT	Univ. Trento/ITC-IRST	Cl.	C++	L	MiniSAT
MiniSMT	Univ. of Innsbruck	Op.	OCaml	L	MiniSAT
NuSMV	ITC-IRST	Op.	C	P	MiniSAT
OpenSMT	Univ. Svizzera Italliana	Op.	C++	L	MiniSAT
Sammy	Univ. Iowa	Op.	OCaml/C	L	SATO
Sateen	Univ. Colorado Boulder	Cl.	C	L	CirCUs
SBT	Univ. Iowa	Op.	C	L	SatBox
Simplics	SRI International	Cl.	Ocaml	L	Custom
simplifyingSTP	Univ. of Melbourne	Op.	C++	E	cryptoMiniSAT
SONOLAR	Univ. Bremen	Cl.	C++	E	PicoSat
Spear	Univ. British Columbia	Cl.	C++	E	Custom
STP	Stanford University/MIT/				
	Univ. Melbourne	Op.	C++	E	MiniSAT
SVC	Stanford University	Op.	C++	L	Custom
SWORD	Univ. of Bremen	Cl.	C++	L	MiniSAT
veriT	LORIA/INRIA/				
	Univ. Rio Grande	Op.	C	L	MiniSAT
test_pmathsat	Univ. Trento	Cl.	C++	L	MiniSAT
Yices	SRI International	Cl.	C++	L	Custom
Yices2	SRI International	Cl.	C	L	Custom
Z3	Microsoft Research	Cl.	C++	L	Custom

Table 6 Summary of the entrants for the six SMT-COMP editions.. The letters in Source stand for **Op**en or **Cl**osed-source and in Appr(oach) they stand for **E**ager, **L**azy or **P**reprocessing.

Simplex algorithm [DM06]. In 2007, ArgoLib competed in quantifier-free, real linear arithmetic divisions (QF_RDL and QF_LRA).

Ario. (Unversioned, 2005; v1.2, 2006.) Ario was submitted by Hossein M. Sheini and Karem A. Sakallah from the University of Michigan. Ario is implemented in C++ and combines a DPLL-style SAT solver with special-purpose modules for reasoning about arithmetic. Ackermann's method is used to eliminate uninterpreted function symbols. Ario competed in real and integer arithmetic divisions, uninterpreted functions, and their combination (QF_UF, QF_RDL, QF_IDL, QF_UFIDL, QF_LRA, QF_LIA, QF_UFLIA.) Ario did not compete in quantified divisions or divisions involving arrays.

2005	2006	2007	2008	2009	2010	Solver
			●			Alt-Ergo
					●	AProVE NIA
		●				Argolib
●	●					Ario
●	—	—	—	●		Barcelogic
			●	●		Beaver
			●	●		Boolector
			●	●		CLsat
●	●					CVC
●						CVC Lite
	●	—	—	—	●	CVC3
					●	CVC4
	●					ExtSAT
		●				Fx7
●	●					HTP
	●					Jat
●	—	—	—	—	●	MathSAT
					●	MiniSMT
	●					NuSMV
			●	—	●	OpenSMT
●						Sammy
●	—	—	—	●		Sateen
●						SBT
●						Simplics
					●	simplifyingSTP
					●	SONOLAR
		●	●			Spear
	●			●		STP
●						SVC
			●	●		SWORD
					●	test_pmathsat
				●	●	VeriT
●	—	●				Yices
			●	●		Yices v2
		●	●			Z3

Table 7 A view of entrants per year of SMT-COMP.

Barcelogic. (Unversioned, 2005; v1.1, 2006; v1.2, 2007; v1.3, 2008; unversioned special nonlinear arithmetic entry, 2009.) Barcelogic was submitted by Robert Nieuwenhuis and Albert Oliveras (TU Catalonia) in 2005; Miquel Bofill (University of Girona), Enric Rodríguez-Carbonell and Albert Rubio (TU Catalonia), along with Nieuwenhuis and Oliveras, in 2006 and 2007; Morgan Deters and Germain Faure (TU Catalonia) joined the other submitters in 2008; and the special entry in 2009 was by Cristina Borralleras (University of Vic), Nieuwenhuis, Oliveras, Rodríguez-Carbonell, and Rubio. Barcelogic was, in 2005 and 2006, a DPLL(T) SMT solver [GHN$^+$04], written in C. It featured a custom-built SAT solver and competed in QF_UF, QF_RDL, QF_IDL and QF_UFIDL. Later submissions, in C++ but keeping the overall design, supported full linear arithmetic (QF_LRA, QF_LIA, QF_UFLIA and QF_UFLRA) and arrays (QF_AX and QF_AUFLIA). A special submission in 2009, code-named *barcelogic-QF_NIA,* entered only the QF_NIA division and demonstrated the techniques published in [BLNM$^+$09b] for translating non-linear integer arithmetic to linear arithmetic.

	Quantifier-free (QF)										
Solver	RDL	IDL	LRA	LIA	NIA	NRA	UF	UFIDL	UFLIA	UFLRA	UFNRA
AProVE NIA					●						
ArgoLib	●		●								
Ario	●	●	●	●			●	●	●		
Barcelogic	●	●	●	●	●		●	●	●	●	
clsat		●									
CVC	●	●	●	●			●	●			
CVC Lite	●	●	●	●			●	●			
CVC3	●	●	●	●	●	●	●	●	●	●	●
CVC4			●								
ExtSAT	●	●	●	●							
HTP	●	●	●	●			●	●	●		
Jat	●										
MathSAT	●	●	●	●			●	●	●	●	
MiniSMT					●	●					
OpenSMT	●	●	●				●	●			
Sammy	●	●	●	●			●	●			
Sateen	●	●		●							
SBT		●		●			●	●			
Simplics	●		●								
SVC	●	●	●	●			●	●			
veriT	●	●					●	●			
test_pmathsat			●	●			●		●	●	
Yices	●	●	●	●			●	●	●		
Yices2	●	●	●	●			●	●	●	●	
Z3	●	●	●	●			●	●	●	●	

	Quantifier-free					Quantified		
						UFNIA		
						AUFNIRA		
Solver	AX	AUFLIA	BV	ABV	AUFBV	LRA	AUFLIA	AUFLIRA
Alt-Ergo							●	●
Barcelogic	●	●						
Beaver			●					
Boolector			●		●			
CVC		●						
CVC Lite		●						
CVC3	●	●	●	●	●	●	●	●
Fx7							●	
HTP		●						
MathSAT	●	●	●		●			
NuSMV			●					
OpenSMT			●					
Sammy		●						
simplifyingSTP			●					
SONOLAR			●					
Spear			●					
STP			●					
SVC		●						
SWORD			●					
Yices		●	●		●		●	●
Yices2	●	●	●		●			
Z3	●	●	●		●		●	●

Table 8 Summary of participants and the divisions they have competed in.

Beaver. (v1.0, 2008; unversioned, 2009.) Beaver was submitted by Susmit Jha, Rhishikesh Limaye, and Sanjit Seshia (UC Berkeley). It is an SMT solver, written in OCaml, for the theory of quantifier-free finite-precision bit-vector arithmetic (the SMT-LIB logic QF_BV). Beaver performs rewrites and simplifications to transform a bit-vector formula into a Boolean circuit. In 2008, it then converted the resulting circuit into CNF and ran an off-the-shelf SAT solver (RSat [PD07] for the competition); in 2009, it integrated NFLSAT [JC09] to obviate the need to convert the circuit into CNF.

Boolector. (v0.4, 2008; v1.2, 2009.) Boolector was submitted by Robert Brummayer and Armin Biere (Johannes Kepler University, Linz, Austria). It is a decision procedure for the quantifier-free theory of bit-vectors and for the quantifier-free extensional theory of arrays with bit-vectors and uninterpreted functions. Additionally, Boolector can be used as a model checker for word-level safety properties [BBL08]. Boolector is implemented in pure C; Picosat [Bie08] is used as the SAT solver for version 0.4; Picosat and PrecoSAT [Bie09] are each utilized in version 1.2. Boolector competed in QF_BV and QF_AUFBV.

clsat. (v0.17f, 2008; v1.0, 2009.) clsat was submitted in 2008 by Duckki Oe, Timothy Simpson, Aaron Stump, and Terry Tidwell (Washington University in St. Louis), and by Duckki Oe in 2009 (then at the University of Iowa). The clsat solver integrates a custom-built clause-learning SAT solver with a standard graph-based IDL solver. Started as a class project, it later incorporated modern SAT techniques, including watched literals, failure-driven assertions, and conflict clause simplification. Version 1.0 was capable of generating proofs for unsatisfiable formulas. clsat competed in QF_IDL.

CVC. (v1.0b, 2005; unversioned, 2006.) CVC [SBD02] 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 [NO79]. CVC uses the Chaff SAT solver [MMZ$^+$01] for Boolean reasoning. The 2006 version was largely unmodified from 2005, though it was ported to a more recent compiler. CVC competed in all divisions in 2005, but, as it did not support integers or bit-vectors, and had no special handling for difference logic, it only entered QF_UF, QF_AUFLIA, and QF_LRA in 2006.

CVC Lite. (Unversioned, 2005.) CVC Lite [BB04] is the successor to 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 Ph.D. thesis [Bar03]. CVC Lite has a custom SAT solver and is capable of producing independently checkable proofs for valid queries. CVC Lite competed in all divisions.

CVC3. (Unversioned, 2006; v1.2, 2007; v1.5, 2008; v2.0, 2009.; v2.3, 2010) CVC3 is a large, multiple-year effort, led by Clark Barrett (New York University) and Cesare Tinelli (University of Iowa). It was initially submitted in 2006 by Barrett and Tinelli, with development credits also going to Yeting Ge (New York University), Alexander Fuchs and George Hagen (University of Iowa); and the implementers of CVC Lite, its predecessor. It is implemented in C++ and is based on Clark Barrett's framework for cooperating decision procedures. 2007's version 1.2 credits Barrett, Fuchs, Ge, and Dejan Jovanović (New York University) with major code improvements, which expanded the feature set and improved performance. 2008's version 1.5 credits Barrett, Ge, Jovanović, Fuchs, Lorenzo Platania (University of Genoa), and Darren Kelley (New York University) with improvements in performance. 2009's version 2.0 credits Barrett, Chris Conway (New York University), Ge, Jovanović, and Fuchs with performance improvements. CVC3 competed in all problem divisions in 2006, 2008, 2009 and 2010. In 2007, it competed in QF_UF, QF_LIA, QF_LRA, QF_UFLIA, QF_AUFLIA, AUFLIA, and AUFLIRA.

CVC4. (v.1.0-α_0, 2010) CVC4 was submitted by Clark Barrett, Christoper Conway, Morgan Deters, Yeting Ge, Liana Hadarean, Dejan Jovanović and Timothy King from New York University and Cesare Tinelli from the University of Iowa. CVC4 is the fourth solver in the family of Cooperating Validity Checkers (CVC, CVC-Lite, CVC3 and CVC4), but it does not incorporate any code from its predecessors. It has been designed to increase the performance and reduce the memory overhead of the previous solvers. The submission to SMT-COMP 2010 was still a prototype and only competed in QF_LRA.

ExtSAT. (v1.1, 2006.) ExtSat was submitted by Paulo Matos of the Instituto Superior Técnico, Portugal. ExtSat is implemented in C++, and combines a Boolean enumerator based on MiniSAT [ES03] with arithmetic solvers. ExtSAT competed in QF_RDL, QF_IDL, QF_LRA, and QF_LIA.

Fx7. (Unversioned, 2007.) Fx7 was submitted by Michał Moskal from the University of Wrocław, Poland, with contributions from Jakub Łopuszański, from the same institution. Fx7 is implemented in Moskal's Nemerle language and was designed for software verification queries, which make heavy use of quantifiers. To deal with quantifiers, Fx7 implements two novel matching algorithms. It competed in AUFLIA.

Heuristic Theorem Prover (HTP). (Unversioned, 2005; unversioned, 2006.) HTP was developed by Kenneth Roe at Formal Documentation Systems, Redwood City, CA. It preprocesses benchmarks, then hands them off to Yices, Barcelogic, or MiniSAT. HTP competed in QF_UF, QF_IDL, QF_RDL, QF_LRA, QF_LIA, and QF_UFLIA in both years; it also competed in QF_AUFLIA in 2005.

Jat. (Unversioned, 2006.) Jat was submitted by Scott Cotton from Verimag, and developed by the submitter under the supervision of Oded Maler (also at Verimag). Jat is written entirely in Java, and employs novel techniques for exhaustive theory propagation for difference logic. Jat competed in QF_RDL.

MathSAT. (v3, 2005; v3.4, 2006; v4.0, 2007; v4.2, 2008; v4.3, 2009; v5, 2010.) A version of MathSAT 3 capable of parsing SMT-LIB was contributed by the MathSAT team: Roberto Bruttomesso (ITC-IRST[4], Trento, Italy), Alessandro Cimatti (ITC-IRST), Anders Franzén (ITC-IRST and the University of Trento, Italy), Alberto Griggio (University of Trento), and Roberto Sebastiani (University of Trento) from 2005–2010. MathSAT is a C++ implementation of the standard "online" lazy integration

[4] Later, ITC-IRST became FBK-IRST.

schema used in many SMT tools. It uses a customized version of MiniSAT [ES03] for Boolean reasoning. 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 was actively worked on between 2005 and 2010, resulting in new features and improved performance. In 2005, MathSAT competed in QF_UF, QF_RDL, QF_IDL, QF_UFIDL, QF_LRA, and QF_LIA. In 2006, it competed in QF_UFLIA and QF_UFBV32 as well. It didn't compete in the bit-vector division in 2007, though it did again in 2008 and 2009, as well as in the new QF_UFLRA division. In 2009, it also added support for arrays, and competed additionally in QF_AUFBV, QF_AX, and QF_AUFLIA. In 2010, it competed in QF_UF, QF_LRA, QF_LIA, QF_UFLIA and QF_UFLRA.

MiniSMT. (unversioned, 2010) MiniSMT was submitted by Harald Zankl and Aart Middeldorp from the University of Innsbruck, Austria. This open-source solver forms the core of the Tyrolean Termination Tool 2, an automatic analyzer for termination of rewrite systems. It is written in OCaml and reduces non-linear SMT formulas to propositional logic. It competed in QF_NIA and QF_NRA.

NuSMV. (Unversioned, 2006.) NuSMV was submitted by R. Bruttomesso, R. Cavada, A. Cimatti, A. Franzen, S. Semprini, M. Roveri, and A. Tchaltsev, from ITC-IRST, Trento, Italy. NuSMV is written in C, and uses a preprocessing step to reduce input problems in QF_UFBV32 to problems which could be solved using routines from the NuSMV symbolic model checker. NuSMV competed in QF_UFBV32.

OpenSMT. (v0.1, 2008; v0.2, 2009; v1.0-alpha, 2010.) OpenSMT was submitted by Roberto Bruttomesso and Natasha Sharygina (Università della Svizzera Italiana, Lugano, Switzerland). It was intended as a small and open-source SMT solver, written in C++, which provided a basic infrastructure for helping non-experts develop theory solvers without having to start from scratch. OpenSMT initially included a parser for the SMT-LIB language, a state-of-the-art SAT solver, and a core solver for uninterpreted functions, as well as an empty, template theory solver to facilitate the development of solvers for other logics. In 2008, OpenSMT competed only in QF_UF; however, 2009's submission, by Bruttomesso and Sharygina as well as Edgar Pek and Aliaksei Tsitovich (also from Svizzero Italiana), added support for QF_IDL, QF_RDL, QF_LRA, and QF_BV. In 2010, it did not compete in QF_BV but entered the QF_UFIDL division.

Sammy. (Unversioned, 2005.) 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. Sammy uses a tool derived from SATO [Zha97] for propositional reasoning and CVC Lite [BB04] for theory reasoning. Sammy competed in all divisions in 2005.

Sateen. (Unversioned, 2005; unversioned, 2006; unversioned, 2007; v2.2.1, 2008; v3.5, 2009.) Sateen was initially submitted by Hyondeuk Kim, HoonSang Jin, and Fabio Somenzi from the University of Colorado at Boulder. The 2009 submission added Hyojung Han to the list of credits, from the same institution. Sateen is written in C and combines efficient propositional reasoning with layered theory solvers, incorporating special solvers for difference logic and arithmetic with the CirCUs SAT engine [JHS05]. Sateen competed in QF_IDL from 2005–2007, in QF_RDL and QF_IDL in 2008, and in QF_RDL, QF_IDL, and QF_LIA in 2009.

SatBox with Theories (SBT). (Unversioned, 2005.) SBT was submitted by Hantao Zhang, Haiou Shen, and John Wheeler from the University of Iowa. SBT is written in C and built on top of the SatBox toolbox for propositional reasoning. It also incorporates some code from Albert Oliveras (one of the authors of Barcelogic). SBT competed in QF_UF, QF_IDL, QF_UFIDL, and QF_LIA.

Simplics. (Unversioned, 2005.) Simplics was submitted by Bruno Dutertre and Leonardo de Moura from the Computer Science Laboratory at SRI International. Simplics is a successor to ICS [FORS01], written mostly in OCaml. Simplics uses a core real-linear arithmetic solver based on an enhanced version of the simplex algorithm [RS04]. Simplics competed in QF_RDL and QF_LRA.

simplifyingSTP. (unversioned, 2010.) simplifyingSTP was submitted by Trevor Hansen, Peter Schachte and Harald Søndergaard from the University of Melbourne, Australia. It is a variant of the STP solver that adds extra simplification rules before encoding the problem into propositional logic. The resulting formula is then processed by CryptoMinisat or by ManySAT in the parallel division. It competed in QF_BV.

SONOLAR. (r252, 2010.) SONOLAR was submitted by Florian Lapschies from the University of Bremen, Germany. It is targeted for automatic test data generation in the field of model-based and C/C++ unit testing. It uses the eager approach to convert the SMT formula into a propositional one, which is then passed to PicoSAT. It competed in QF_BV.

Spear. (v1.9, 2007; unversioned, 2008.) Spear was submitted by Domagoj Babić from the University of British Columbia. Spear is a theorem prover for bit-vector arithmetic that translates the input formula into a propositional one, then sends it to a simple lightweight DPLL SAT solver. It was designed chiefly for software verification, but also for other industrial problems, like bounded hardware model checking. Spear competed in QF_BV.

STP. (Unversioned, 2006; #101, 2009.) STP was submitted in 2006 by Vijay Ganesh and David Dill from Stanford University. STP preprocesses and then translates input formulas into purely propositional formulas, which are then solved by MiniSAT [ES03]. An abstraction-refinement technique is used for handling array read expressions. Version #101, submitted in 2009 by Trevor Hansen (University of Melbourne) and Vijay Ganesh (then at MIT) incorporates a better CNF converter, and was improved to handle the updated SMT-LIB bit-vector theory. STP competed in QF_UFBV32 in 2006 and QF_BV in 2009.

SVC. (Unversioned, 2005.) SVC [BDL96] is a legacy system developed at Stanford University by Clark Barrett, Jeremy Levitt, and David Dill. An updated version capable of parsing the SMT-LIB format was submitted by Clark Barrett. SVC is implemented in C++ and features a framework for combining decision procedures described in Jeremy Levitts Ph.D. thesis [Lev99]. SVC competed in all divisions.

SWORD. (v0.2, 2008; v1.0, 2009.) SWORD was submitted in 2008 by Robert Wille, André Sülflow, and Rolf Drechsler (University of Bremen), and in 2009 by Jean Christoph Jung (University of Bremen) along with Wille, Sülflow, and Drechsler. The bit-vector solver SWORD makes use of *word-level* information. Logical bit-vector operations are handled as clauses in an incorporated SAT solver, but arithmetic operations are handled by adjoining *modules*, which are permitted to affect the search heuristics. SWORD uses MiniSAT [ES03] as a SAT engine, and competed in QF_BV.

veriT. (v200907, 2009; v201007, 2010.) veriT was submitted by Thomas Bouton (LORIA-INRIA), Diego Caminha B. de Oliveira (LORIA-INRIA), David Déharbe (Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil), and Pascal Fontaine (LORIA-INRIA). It is mainly written in C, is open-source, and supports Nelson-Oppen theory combination [NO79] over uninterpreted functions and difference logic over reals and integers. It also supports quantifiers and can produce proofs. veriT competed in 2009 and 2010 in QF_UF, QF_RDL, QF_IDL, and QF_UFIDL.

test_pmathsat. (v0.0.5, 2010.) test_pmathsat was submitted by Alberto Griggio from FBK-IRST, Trento, Italy. It is a simple parallel version of the MathSAT 5 solver which sends different instances of MathSAT with different search parameters to different CPU cores and terminates as soon as one of them returns a result. It consists of only 300 lines of C code and competed in QF_UF, QF_LRA, QF_LIA, QF_UFLRA and QF_UFLIA.

Yices. (v0.1, 2005; v1.0, 2006; v1.0.10, 2007.) The first submission of Yices in 2005 was by Leonardo de Moura while at the Computer Science Laboratory at SRI International. Yices is implemented in C++ and is based on the Nelson-Oppen method for combining decision procedures [NO79]. Yices produces proof objects for valid queries. Yices 1.0 was submitted in 2006 by Bruno Dutertre (SRI International) and de Moura, and incorporates significant work since the 2005 submission on the SAT solver, overall architecture, and theory solvers. In 2007, Yices 1.0.10 was submitted by Dutertre, incorporating many bug fixes, improvements to the bit-vector solver, and extensions to support new and updated logics of SMT-LIB. Yices competed in all divisions in all three years it was submitted.

Yices 2. (Unversioned prototype "c," 2008; unversioned prototype, 2009.) Yices 2 was submitted by Bruno Dutertre (SRI International, Menlo Park, California). It was a prototype of the successor to the Yices 1 SMT solver written entirely in C. It attempts to address several limitations of Yices 1, including type checking issues and limited functionality of the Yices API. The new solver supports a simpler specification language that can be statically type checked and it provides a full API to access all functions of the solver. It is intended to offer similar or better performance than Yices 1 on most benchmarks, while being more modular, extensible, and maintainable. Yices 2 competed in QF_UF, QF_RDL, QF_IDL, and QF_LRA in 2008. The 2009 version added support for bit-vectors and arrays, and competed in the same four divisions as in 2008, plus QF_BV, QF_AUFBV, QF_UFIDL, QF_AX, QF_AUFLIA, QF_UFLRA, QF_UFLIA, and QF_LIA.

Z3. (v0.1, 2007; v3.2$^{\alpha}$, 2008.) Z3 was submitted by Nikolaj Bjørner and Leonardo de Moura from Microsoft Research, Redmond. Z3 is written in C++ and is similar in spirit to Yices, but it also incorporates an E-matching abstract machine to deal with quantifiers and model-based theory combination techniques. Z3 supports linear real and integer arithmetic, fixed-size bit-vectors, extensional arrays, uninterpreted functions, and quantifiers. It can read problems in SMT-LIB and Simplify formats. Z3 competed in all competition divisions during both years it competed.

5.2 Evolution in Performance

To demonstrate the evolution in performance from 2005 to 2010, we ran the winners of each year's competition on the QF_LRA benchmark set used for the 2009 competition. The winners were Simplics (2005), Yices 1.0 (2006), Yices 1.0.10 (2007), Yices 2 "proto c" (2008), Yices 2 "proto" (2009) and MathSAT 5 (2010). The results are shown in Table 9. Note that the 2006 winner beat the 2007 winner, and the 2008 winner beat the 2009 and 2010 winners (each by a very small margin).

Simplics gave two incorrect answers (reporting SAT instead of the expected UNSAT) on the benchmarks *frame_prop.base.smt* and *reint_to_least.base.smt* from the *spider_benchmarks* suite. A bugfix version (version 0.1.1) of Simplics released after the competition fixed these problems.

A "cactus" plot of the comparison is shown in Figure 1. In this plot, for each solver, that solver's times on benchmarks are sorted in ascending order and plotted cumulatively—the best solver is then the one with its line furthest to the right of the plot (the most instances solved) and the lowest (the least CPU time used). As seen in the plot, considerable improvement all-around was made by 2006's entry (over 2005's). Later years showed a flattening of the curve at the 0-10 seconds region, indicating that more benchmarks were immediately solved by the winning solvers. Note that while the 2008 solver beat the 2009 solver overall in the test, owing to the fact that the 2009 solver timed out on one additional instance, the 2009 solver shows this curve-flattening improvement over 2008; although it lost in our test, it was able to immediately solve many instances that were reasonably easy (but not trivial) for the 2008 solver. This may indicate a practical advantage of the 2009 version, namely the ability to solve many relatively easy satisfiability problems very rapidly, an ability that is important in many domains. As benchmarks get harder, the 2008 and 2009 solvers seem to behave somewhat similarly performance-wise.

Finally, a scatter plot (Figure 2) shows a benchmark-by-benchmark comparison between the 2005 winner Simplics and the 2009 winner Yices 2 "proto". This version of Yices 2 shows considerable improvement over the state of the art in 2005.

Although we only provide data comparing the winners for the QF_LRA division, similar data could be given for other divisions. The situation is usually the same: a certain solver presents a key idea that improves the performance in a particular division, and this idea is implemented by most solvers in the following SMT-COMP editions. After that, only minor improvements are shown in that division, but still remarkable improvements appear in some others. The rationale behind that is that the main focus of SMT developers changes year after year. This is illustrated in Figure 3, where it can be seen that important improvements were made in 2006 (QF_LRA), 2007 (AUFLIA), 2008 (QF_BV) and 2010 (QF_LIA). We can easily identify the reasons behind such improvements: in 2006, Yices introduced a new Simplex-based algorithm for linear

Solver	Correct	Time	Unsat	Sat	Unk	T/O	Wrong
Yices2 “proto c” (2008)	186	4586.7 s	92 (0)	94 (0)	0	16	0
Yices 2 proto (2009)	185	3311.2 s	92 (0)	93 (0)	0	17	0
MathSAT 5 (2010)	185	4567.6 s	92 (0)	93 (0)	0	17	0
Yices 1.0 (2006)	183	2554.9 s	92 (0)	91 (0)	0	19	0
Yices 1.0.10 (2007)	183	2590.4 s	92 (0)	91 (0)	0	19	0
Simplics (2005)	166	10120.6 s	82 (2)	84 (0)	0	34	2

Table 9 Results for comparison of the SMT-COMP winners in QF_LRA from 2005 through 2010. The comparison was done on the SMT-Exec platform, on the 202 benchmarks of the QF_LRA 2009 competition set with a 30-minute timeout. The time reported here is for correctly-answered benchmarks only.

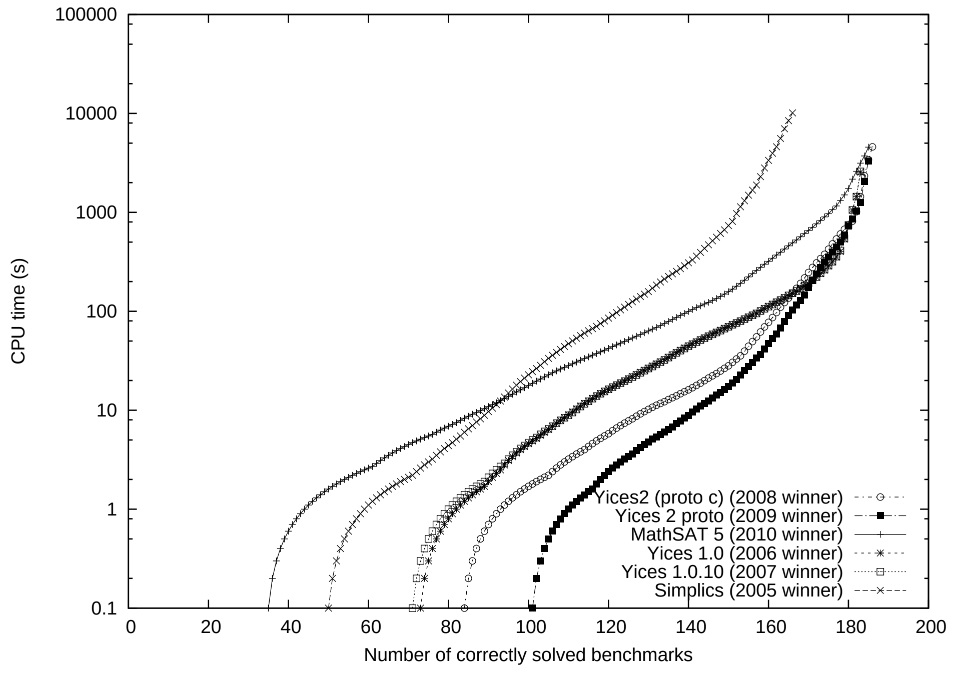

Fig. 1 A “cactus” plot showing the performance of the SMT-COMP winners from 2005 through 2010. Considerable improvement of the 2006 winner over the 2005 winner was unmatched in later years on this benchmark set, with important, but more modest, improvement overall.

arithmetic [DdM06]; in 2007, efficient E-matching techniques like the ones in [dMB07] turned out to be crucial for the performance of Z3; in 2008, several new ideas to deal with bit-vectors were implemented in Boolector [BB09b] and in 2010, MathSAT introduced specific techniques for reasoning over integer linear arithmetic [Gri10]. Hence, the current situation is that SMT improvements are mostly theory-specific.

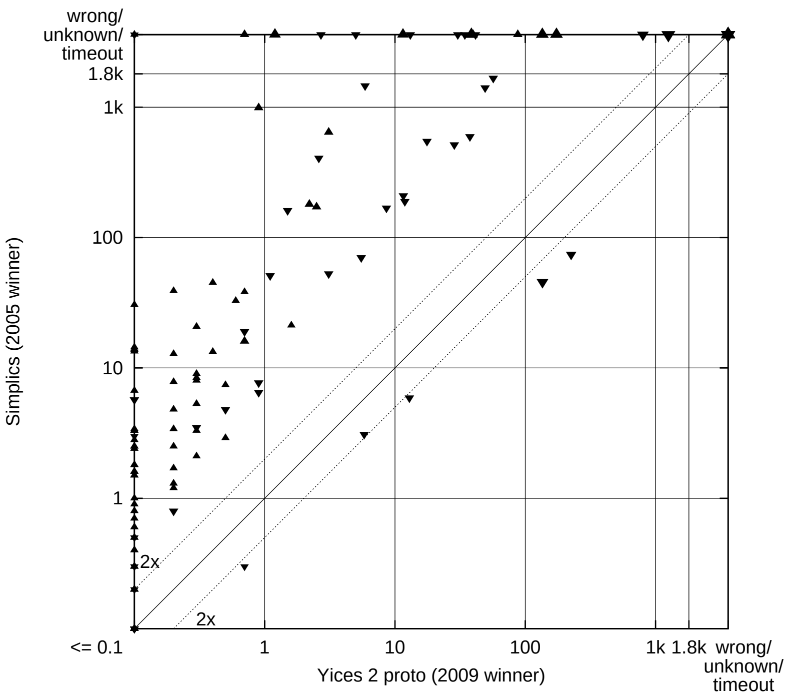

Fig. 2 A scatter plot showing a benchmark-by-benchmark comparison of the 2005 winner, Simplics, against the 2009 winner, Yices 2 "proto". Time limit was set to 1800 seconds. On many benchmarks, the 2009 winner is more than 10× faster than the 2005 one; overall, it performs better on satisfiable and unsatisfiable instances, and is much faster. Upward-pointing triangles indicate satisfiable instances; downward-pointing triangles unsatisfiable ones.

5.3 Underlying Techniques

Most SMT-COMP entrants implemented the so-called lazy approach to SMT, where a SAT solver is in charge of enumerating models for the Boolean part of the formula, and a theory solver checks whether they are consistent with the background theory. An important refinement where the theory also computes consequences of the current partial model, called theory propagation [NO05], was introduced by the Barcelogic team and was shown to improve performance across various divisions.

Although successful for most divisions, the lazy approach was not dominant in the bit-vector divisions, where the large majority of entrants converted the SMT formula to propositional logic before letting a SAT solver decide its satisfiability. The other divisions where an eager approach proved to be competitive were non-linear arithmetic divisions, where propositional encodings [FGM+07, ZM10] competed with ad-hoc techniques [BLNM+09b].

When the underlying theory consisted of the union of several theories, the combination mechanism chosen was usually Nelson-Oppen [NO79]. Two important improvements over the basic combination procedure need to be mentioned: the delayed theory combination [BBC+05] introduced by the MathSAT team, and the model-based theory combination [dMB08] presented by the Z3 implementers.

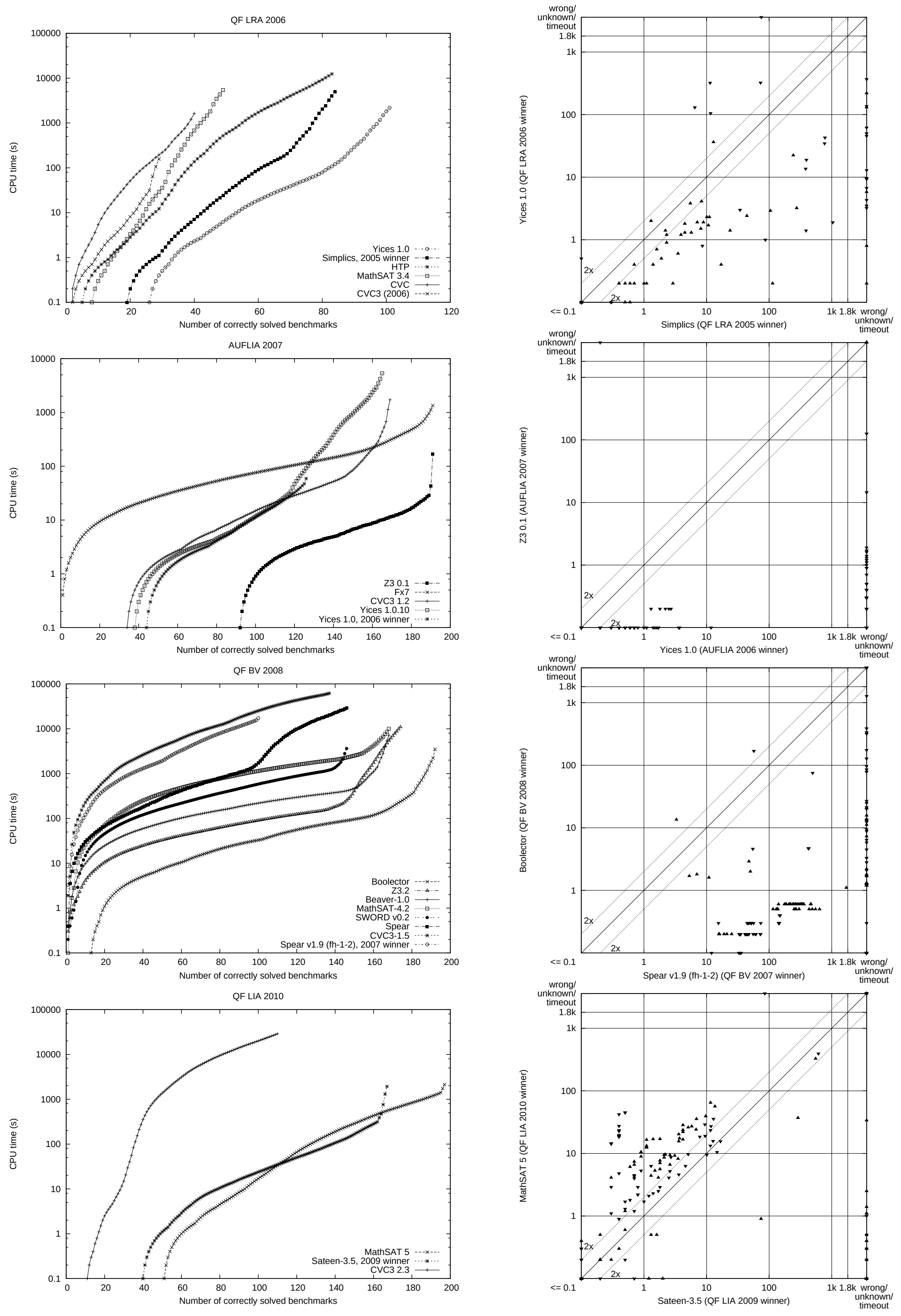

Fig. 3 Data about the divisions QF_LRA (2006), AUFLIA (2007), QF_BV (2008) and QF_LIA (2010). The cactus plot compares all entrants in the corresponding year with the winner of the previous edition, whereas the scatter plot only compares the winners of the two consecutive years.

Regarding linear arithmetic, several important developments were introduced. For dealing with the difference logic fragment, most solvers implemented variants of Bellman-Ford-like algorithms for negative cycle detection [NO05, WIGG05, CM06] or adaptations of the Floyd-Warshall algorithm when the resulting graph was very dense. For full linear arithmetic the first important algorithm was given by the Simplics people [RS04], but the ground-breaking algorithm by de Moura and Dutertre [DdM06] became the most common one. Later on, in [Gri10], dedicated techniques for the integer case were presented and proved to be successful in the competition.

Preprocessing techniques were also used by many entrants. This trend was especially common in the bit-vector divisions, where a large set of of rewrite rules was usually applied to the initial formula before encoding it into propositional logic. Another successful preprocessing technique was presented by the Sateen team. Their work [KSJ09] focused on the translation of formulas involving a large number of if-then-else terms; such formulas occur, for example, in many benchmarks in the QF_LIA division.

Finally, several techniques were used to deal with quantifiers. Building upon the pattern matching technique introduced by the Simplify solver, several new techniques were developed and used in the competition by different teams: Z3 [dMB07], CVC3 [GBT09] and Fx7 [MLK08].

This is by no means an exhaustive list of all the techniques used by the SMT-COMP competitors, but we believe is fairly representative of the ones that have had a significant impact.

6 Analysis of the Impact of SMT-COMP

Given the growing number of existing system competitions, it has become evident that system competitions are attractive for many research communities. Among others, we can find system competitions for first-order theorem provers, SAT solvers, Max-SAT solvers, Quantified Boolean Formula (QBF) solvers, Constraint Satisfaction Problems (CSP) solvers, termination systems, hardware model checkers, planning, timetabling, answer set programming (ASP) tools, and package/component installation and upgrade systems.

It is our belief that apart from determining which solver is the best performing according to the competition rules, a system competition can and should have other goals that benefit the research community. In the following list, we review the main goals that we had for SMT-COMP and analyze to what extent they have been realized:

– **Adoption of a common input language:** this was the original reason for organizing the competition. Already after the first edition, it was clear that the competition had been a success in this direction. Although some systems have their own input format, for backwards compatibility reasons or because it somehow best suits their needs, all SMT systems currently under development accept the SMT-LIB language.

– **Collection of benchmarks:** it is evident that, at least in term of quantity, the collection of benchmarks has been another success of the competition. Before the competition started, there were only a few hundred benchmarks being used by the community, and not all of them were in the same input language. The current situation, as explained in Section 3, is completely different.

Researchers familiar with the competition have a slight advantage over others trying to collect and make progress on benchmarks: users of SMT technology know that their submitted benchmarks will become part of next year's competition and that, because the library is publicly available, most implementers will work hard during the year to make their system fast on those problems. This situation encourages people to submit their benchmarks which is a laborious task, and is often neglected without something like a competition to motivate the effort required. However, we should say that in some divisions the progress in benchmark collection seems to have stopped, even though there seems to be ongoing research that could produce new benchmarks.

– **Enable system comparison:** before the competition started, it was not clear how different systems compared, either in performance or in expressivity of the input language supported (see the paper [dMR04]). With the results of the competition, one can get a reasonable approximation of the relative performance of various systems on different types of problems. Moreover, the SMT-Exec service allows one to compare, at any time of the year, any SMT solver with the ones submitted to the competition. The only drawback is that the comparison only considers the solvers' run-time and not other interesting parameters like, e.g., the number of decisions, which might help, for example, in understanding whether a system is better because it effectively prunes the search space or because it traverses it faster.

– **React to the research community:** it is important for a system competition to react to the changing demands of the research community. For that purpose, a mailing list was created and consulted for most delicate design decisions. Moreover, running the competition online and collocating it with major conferences like CAV or CADE and with the SMT workshop made it easier for users, implementers, and organizers to meet and exchange opinions about the current state of the competition.
The organization has also tried to react to the fact that hot research topics change from one year to another. Examples include the increased interest in bit-vectors and quantifiers during some years. On those occasions, special effort was made to collect representative benchmarks in these divisions so that the competition could fairly reflect the topics which were more attractive to the users of SMT technology. There remain some requests from the research community that have not yet been reflected in the competition. For example, there have been requests for solvers to compete on benchmarks that include features like incrementality, the ability to produce models, and the ability to produce interpolants. We are hopeful that future editions of the competition may address these requests.

– **Give recognition to system developers:** in areas where implementation plays an important role, it is accepted that system developers do not always get the recognition they deserve. The situation is even worse for Ph.D. students, who are usually in charge of implementations but are typically evaluated only according to the number of papers they publish. Competitions do give some recognition to system developers, but it is certainly true that this recognition is sometimes too local to the community and that systems performing very close to the winners get almost no recognition. Unfortunately, we have to admit that the success of SMT-COMP in addressing these issues has been very limited.

Besides analyzing the goals we had in mind and whether they have been fulfilled or not, it is also important to focus on possible negative aspects that SMT-COMP might have had on the community. Some aspects show up in any system competition and others are more specific to SMT-COMP.

– **Interesting new techniques abandoned too soon:** a system competition in which the only performance measure is the run-time encourages developers to optimize their systems to sometimes unexpected levels. First of all, just the fact that researchers have incentives to spend their valuable time on low-level implementation details may be cause for concern. Second, existing systems become highly optimized tools whose low-level details are usually either closed to other researchers or poorly documented. In either case, the result is a significant barrier to entry for newcomers. Finally, and perhaps most importantly, for the same reasons, techniques that are worth exploring might be abandoned too early because there is little hope that they can compete with existing tools in the short/mid-term.

– **Systems are trained on existing benchmarks:** because the benchmarks used in the competition are publicly available, system implementers optimize their tools by adjusting parameters according to the results they obtain on the benchmark library. In principle, this should not be a big issue if one assumes the library to be representative enough. However, this might not always be the case and systems could end up being optimized in the wrong direction. As a possible way of overcoming this problem, some competitions do not make all benchmarks publicly available. Then, systems that overtrain on given benchmarks instead of trying to be robust on all types of benchmarks can be easily detected since they underperform on the unpublished problems.

– **The scoring system encourages a focus on hard problems:** assuming that problems that the best solver can process in 60 seconds can be processed by any other solver in less than 15 minutes, as is usually the case, it is clear that the winning system will be the one able to solve a larger number of hard problems. Stated this way, this situation does not look too bad, but it overlooks the importance of solving easy problems as quickly as possible, which is crucial in some large verification environments where thousands of simple queries are made to an underlying SMT solver. In this case, spending one more second per query could be very costly and hence, heavy-weight techniques like thorough preprocessing algorithms should be abandoned in favor of light-weight methods. A change in the scoring system, where average time is also taken into account (e.g. the efficiency measure used in CASC) might be enough to fix this problem.
  Similarly, the competition gives almost no recognition for systems that provide additional capabilities, like having a rich and user-friendly API, proof and model production, incremental usage, etc. These are very interesting, sometimes even necessary, functionalities that users might want in an SMT solver, and it would be nice if this were rewarded somehow in the competition.

– **System developers might be afraid of bad performance:** it is probably true that for the winners, the time taken to prepare their system for the competition is worth it, but this might not be the case for other competitors. A bad performance by a given system might affect its reputation, and system developers might think twice before they submit the system to another edition. This is of course an understandable attitude, but it defeats one of the purposes of the competition, which is to give a snapshot of how current systems compare.

– **Special work needed to adjust solvers for the competition:** several strong solvers have not re-entered the competition after victories. One possible reason was stated above: they might be afraid of performing poorly and tarnishing their reputation. Another reason might be that in order to perform well in the competition, some (often significant) effort is required to ensure adequate performance on the competition benchmarks. Solvers that perform perfectly well in specific research projects might not, for various reasons, perform very well at the competition. This extra work might not be deemed worth it once a solver is already well-known by the community. One solution to this problem would be to make the competition as similar as possible to the real uses of SMT solvers (e.g. use them incrementally, asking for models, etc.) so that the extra work to enter the competition is minimized.

– **Allowing the organizers to compete:** this is always a delicate issue in any competition. In the case of SMT-COMP, since the organizers were allowed to compete, we tried to make the competition as transparent as possible, so that organizers would have no advantage with respect to other competitors. However, allowing the organizers to compete adds some restrictions to the competition (e.g. no private benchmarks can exist). Moreover, when delicate issues come up, like deciding whether a late patch for a simple bug can be accepted for a system submission, being an organizer and a competitor at the same time can lead to a potential conflict of interest.
  The easy solution is of course to forbid organizers to participate. However, it is difficult to find researchers who are willing to take on the work of organizing a system competition who are not system developers themselves. Moreover, we believe that entrants to the competition have a better view of whether the competition is fair or not, and which aspects of the competition need to be improved. Having an organizer/competitor thus helps in this direction.

7 Conclusions

The first six editions of SMT-COMP have undoubtedly helped the SMT community in various directions. First of all, the competition has been the main promoter of the SMT-LIB language, which is now accepted by all state-of-the-art SMT solvers. Second, SMT-COMP has witnessed how, year after year, solvers have improved their capabilities in order to handle harder and harder problems. Finally, the competition has become a venue that facilitates communication and the exchange of ideas among SMT developers and between developers and users.

Acknowledgments: SMT-COMP would have not been possible without the invaluable support, feedback and participation of the entire SMT community, with special thanks to Cesare Tinelli and Silvio Ranise, initial coordinators of the SMT-LIB initiative. We would like to thank the organizers and programme chairs of CAV'05, CAV'06, CAV'07, CAV'08, CADE'09 and CAV'10 for their support when hosting SMT-COMP as a satellite event. The execution of the competition would not have been possible without the help of Mark Bober of Washington University's Computing Technology Services (2007-2008) and the cluster provided by SRI International for running the 2006 edition. Hugh Brown and JJ Urich of The University of Iowa Computer Support

Group provided invaluable support for 2008-2010 editions of the competition. Finally, the organizers wish to acknowledge the support of the U.S. National Science Foundation, under contract CNS-0551697.

References

[Bar03] C. Barrett. *Checking Validity of Quantifier-Free Formulas in Combinations of First-Order Theories*. PhD thesis, Stanford University, 2003.

[BB04] C. Barrett and S. 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.

[BB09a] R. Brummayer and A. Biere. Fuzzing and Delta-Debugging SMT Solvers. In O. Strichman and B. Dutertre, editors, *7th International Workshop on Satisfiability Modulo Theories*, 2009.

[BB09b] R. Brummayer and A. Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In Stefan Kowalewski and Anna Philippou, editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'09*, volume 5505 of *Lecture Notes in Computer Science*, pages 174–177. Springer, 2009.

[BBC$^+$05] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient satisfiability modulo theories via delayed theory combination. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[BBL08] R. Brummayer, A. Biere, and F. Lonsing. BTOR: bit-precise modelling of word-level problems for model checking. In *SMT '08/BPR '08: Proceedings of the Joint Workshops of the 6th International Workshop on Satisfiability Modulo Theories and 1st International Workshop on Bit-Precise Reasoning*, pages 33–38, New York, NY, USA, 2008. ACM.

[BDL96] C. Barrett, D. L. Dill, and J. 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.

[Bie08] A. Biere. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 4(2-4):75–97, 2008.

[Bie09] A. Biere. P{re,i}coSAT@SC'09, 2009. http://fmv.jku.at/precosat/preicosat-sc09.pdf.

[BLNM$^+$09b] C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio. Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic. In Renate A. Schmidt, editor, *22nd International Conference on Automated Deduction , CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2009.

[BNO$^+$08] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. The Barcelogic SMT Solver. In A. Gupta and S. Malik, editors, *Computer Aided Verification (CAV)*, pages 294–298. Springer, 2008.

[BSST09] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[BST10] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB standard – version 2.0. In *Proceedings of the 8^{th} International Workshop on Satisfiability Modulo Theories (SMT '10)*, July 2010. Edinburgh, Scotland.

[CCKL08] S. Conchon, E. Contejean, J. Kanig, and S. Lescuyer. CC(X): Semantic Combination of Congruence Closure with Solvable Theories. *Electronic Notes in Theoretical Computer Science*, 198(2):51–69, May 2008.

[CM06] S. Cotton and O. Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In A. Biere and C. P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06*, volume 4121 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2006.

[DdM06] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In Thomas Ball and Robert B. Jones, editors, *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[DM06] B. Dutertre and L. De Moura. Integrating simplex with DPLL(T). Technical report, CSL, SRI International, 2006.

[dMB07] L. de Moura and N. Bjørner. Efficient E-Matching for SMT Solvers. In Frank Pfenning, editor, *21st International Conference on Automated Deduction, CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007.

[dMB08] L. de Moura and N. Bjørner. Model-based Theory Combination. *Electr. Notes Theor. Comput. Sci.*, 198(2):37–49, 2008.

[dMR04] L. de Moura and H. Ruess. An Experimental Evaluation of Ground Decision Procedures. In R. Alur and D. Peled, editors, *16th International Conference on Computer Aided Verification, CAV'04*, volume 3114 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2004.

[ES03] N. Eén and N. 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.

[FGM$^+$07] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT Solving for Termination Analysis with Polynomial Interpretations. In J. Marques-Silva and K. A. Sakallah, editors, *10th International Conference on Theory and Applications of Satisfiability Testing, SAT'07*, volume 4501 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 2007.

[FORS01] 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.

[GBT09] Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Ann. Math. Artif. Intell.*, 55(1-2):101–122, 2009.

[GHN$^+$04] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. 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.

[Gri10] A. Griggio. A Practical Approach to SMT(LA(Z)). In *8th International Workshop on Satisfiability Modulo Theories*, 2010.

[GTG07] M. K. Ganai, M. Talupur, and A. Gupta. SDSAT: Tight Integration of Small Domain Encoding and Lazy Approaches in Solving Difference Logic. *JSAT*, 3(1-2):91–114, 2007.

[JC09] H. Jain and E. M. Clarke. Efficient SAT Solving for Non-Clausal Formulas using DPLL, Graphs, and Watched Cuts. In *46th Design Automation Conference (DAC)*, 2009.

[JHS05] H. Jin, H. Han, and F. Somenzi. Efficient conflict analysis for finding all satisfying assignments of a boolean circuit. In *In TACAS05, LNCS 3440*, pages 287–300. Springer, 2005.

[KSJ09] H. Kim, F. Somenzi, and H. Jin. Efficient Term-ITE Conversion for Satisfiability Modulo Theories. In Oliver Kullmann, editor, *12th International Conference on Theory and Applications of Satisfiability Testing, SAT'09*, volume 4121 of *Lecture Notes in Computer Science*, pages 195–208. Springer, 2009.

[LBS06] D. Le Berre and L Simon. Preface, Special Issue on the SAT 2005 Competitions and Evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–4, 2006.

[Lev99] J. Levitt. *Formal Verification Techniques for Digital Systems.* PhD thesis, Stanford University, 1999.

[MLK08] M. Moskal, J. Lopuszanski, and J. R. Kiniry. E-matching for Fun and Profit. *Electr. Notes Theor. Comput. Sci.*, 198(2):19–35, 2008.

[MMZ$^+$01] M. Moskewicz, C. Madigan, Y. Zhaod, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *39th Design Automation Conference*, 2001.

[Nik10] M. Nikolić. Statistical Methodology for Comparison of SAT Solvers. In O. Strichman and S. Szeider, editors, *Thirteenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2010.

[NO79] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.

[NO05] R. Nieuwenhuis and A. 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-Verlag, 2005.

[PD07] K. Pipatsrisawat and A. Darwiche. Rsat 2.0: SAT solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007. `http://reasoning.cs.ucla.edu/rsat/papers/rsat_2.0.pdf`.

[RS04] H. Rueß and N. Shankar. Solving linear arithmetic constraints. Technical Report SRI-CSL-04-01, SRI International, 2004.

[SBD02] A. Stump, C. W. Barrett, and D. L. Dill. CVC: a cooperating validity checker. In *In 14th International Conference on Computer-Aided Verification*, pages 500–504. Springer, 2002.

[SBH05] L. Simon, D. Le Berre, and E. A. Hirsch. The SAT2002 competition. *Ann. Math. Artif. Intell.*, 43(1):307–342, 2005.

[Sha02] N. Shankar. Little Engines of Proof. In L. H. Eriksson and P. A. Lindsay, editors, *International Symposium of Formal Methods Europe, FME'02*, volume 2391 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2002.

[SS06] G. Sutcliffe and C. Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.

[Sut09] G. Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[Wan06] B. Wang. On the Satisfiability of Modular Arithmetic Formulae. In S. Graf and W. Zhang, editors, *4th International Symposium of Automated Techonology for Verification and Analysis, ATVA'06*, volume 4218 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 2006.

[WIGG05] C. Wang, F. Ivancic, M. K. Ganai, and A. Gupta. Deciding Separation Logic Formulae by SAT and Incremental Negative Cycle Elimination. In G. Sutcliffe and A. Voronkov, editors, *12h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2005.

[Wil76] H. P. Williams. Fourier-Motzkin elimination extension to integer programming problems. *Journal of Combinatorial Theory (A)*, 21:118–123, 1976.

[Zha97] 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-Verlag, July 1997.

[ZM10] H. Zankl and A. Middeldorp. Satisfiability of Non-linear (Ir)rational Arithmetic. In Edmund M. Clarke and Andrei Voronkov, editors, *16th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'10*, volume 6355 of *Lecture Notes in Computer Science*, pages 481–500. Springer, 2010.