The 2012 SMT Competition

David R. Cok (author and chair of organizing committee)$^1$, Alberto Griggio (co-organizer)$^2$, and Roberto Bruttomesso (co-organizer)$^3$

1 GrammaTech, Inc., dcok@grammatech.com
2 Fondazione Bruno Kessler (FBK), griggio@fbk.eu
3 Atrenta France, roberto@atrenta.com

12 June 2012

The 2012 SMT Competition continues the series of competitions in SMT solver capability and performance that began in 2005. The competition is held to spur advances in SMT solver implementations acting on benchmark formulas of practical interest. Public competitions are a well-known means of stimulating advancement in software tools. For example, in automated reasoning, the SAT and CASC competitions for first-order and propositional reasoning tools, respectively, have spurred significant innovation in their fields

As this summary is written prior to the deadline for 2012 submissions, we do not yet know the entrants or the results for 2012. Those will be reported at IJCAR’12. Information about the winners and results of the competition will be available online at www.smtcomp.org; information about previous years’ competitions is also available at that website.

The 2012 Competition. In planning the 2012 competition, the organizers desired to encourage breadth in the capability of SMT solvers. Previous years have challenged solvers to support a variety of logics and have measured them on raw performance on individual problems. This year we have two additional goals. First, we are focusing the competition on a subset of the logics that are the more relevant to applications. Some of the simpler logics are now routine for nearly all solvers and therefore not a good base for a competition. Others have received only light interest in the past. Some of the less expressive logics are subsumed into the more expressive logics for selecting competition benchmarks.

Second, we want to encourage support for additional capabilities, namely, determining unsatisfiable cores and generating proofs. Finding small unsatisfiable cores is important, for example, in finding contradictions within sets of assertions; compact unsatisfiable cores also produce more compact proofs. Finding a minimal unsatisfiable core is a hard problem with no known practical algorithm; thus, good heuristics that apply to problems of interest are valuable and worth a competition. So, the organizers added an unsat core track to the 2012 competition. The winner of that track will be the solver that, without producing any erroneous results, produces the smallest unsatisfiable cores on the benchmark set within the timeout period.

Similarly, constructing proofs of unsatisfiability is also useful, particularly if quantified assertions are included. Since there is as yet no standard method to express proofs and thus no easy way to check them, the organizers added a proof generation track solely in demonstration mode. We encourage submission of solvers with this capability and will highlight this capability in the results of the competition, but we will not attempt to measure the speed or accuracy of such solvers this year. We do hope that attention to proof generation will encourage standardization of proof format and of proof checkers.

The competition uses a subset of benchmarks from those available at www.smtlib.org. The full benchmark suite contains nearly 100,000 benchmarks. New benchmarks are continually being added — additional benchmarks were added to the main and application tracks for 2012. The unsat
core benchmarks are adapted from main track benchmarks that are unsatisfiable. The benchmarks are a collection of more or less relevant problems, rather than benchmarks that measure specific metrics. Some benchmarks are families of constructed problems of arbitrary size; these can test the scalability of a solver as the size of the benchmark instance is increased. Other benchmarks are formed from problems that arise in actual applications. For example, software verification of real programs produces many SMT problems that are suitable as benchmarks.

The full description of the 2012 SMT competition’s rules is found in the rules document (www.smtcomp.org/2012/rules12.pdf). The document describes the procedures for determining benchmark difficulties, selecting benchmarks for competition, and for judging the results.

Procedure. The competition’s traditional ‘main’ track tests a solver’s ability to determine the satisfiability or unsatisfiability of a single problem (perhaps with multiple assertions) within a given logic. A second track tests the performance of solvers that are multi-threaded on similar problems.

The ‘application’ or incremental track, introduced last year, tests a qualitatively different capability. Software verification tools often use SMT solvers as a back-end proof engine. These tools repeatedly invoke the solver with different, related satisfiability problems; the problems may have a substantially similar set of assertions, produced by the tool’s adjusting, correcting, adding, or retracting assertions interactively; in batch mode different properties may be checked using substantially the same set of assertions. The effect is that the solver must respond to a sequence of requests to assert or retract logical statements, check satisfiability, produce counterexamples, and so on. The application track tests a solver’s performance in responding to such a sequence of commands, as produced by actual application problems.

The benchmarks are each assigned a difficulty. The difficulty is based on how long it takes a group of solvers to produce a correct answer to the benchmark. For competition, benchmarks are selected, at random, from each difficulty category.

The winning solver in each category is the one that produces the most correct answers in the least time. An additional change this year is that incorrect answers are a disqualifier: the organizers considered that correct technology has progressed sufficiently in capability and importance that incorrect answers should not be tolerated (a solver can always produce an answer of ‘unknown’). Each solver is given a fixed timeout period in which to answer a benchmark. The winner is the solver that produces the most correct (non-unknown) answers; in the case of ties, the winner is the solver that took the least time to produce its correct answers. In the unsat-core track, it is the size reduction of the core that is measured, rather than the number of correct answers.

The competition infrastructure. The competition is actually executed on a cluster of machines at the University of Iowa, under the control of the SMT-EXEC software suite (cf. www.smtexec.org). This software suite has been used in past years as well. A new hardware and software infrastructure, Star-Exec (cf. www.starexec.org), is under development and is expected to be announced at IJCAR’12. Interested persons can experiment with Star-Exec at the Star-Exec workshop, a satellite event of the IJCAR conference.

The SMTLIB language. A competition based on benchmark problems needs a standard language in which to express those problems. For SMTCOMP, that language is the SMT-LIB language (cf. www.smtlib.org, [1] [3]). In 2010, a significantly reworked version of the language was agreed.

4 There is an anomaly in this scoring system. Solvers A and B may produce the same correct answers, with A taking slightly less time to do so than B, and thus being the winner. Answers of unknown do not count towards correct answers, but the time taken also does not penalize the total time used. It may be the case the A takes a long time to determine an answer of unknown on some benchmarks, where as B can do so quickly. Thus B may be overall preferable in an application, even though A is the competition winner.
This version 2 increased the flexibility and expressiveness of the language while also simplifying the syntax. It also includes a command language that improves the language’s usefulness for interactive applications. In particular, the standard specifies a typed (sorted), first-order logical language for terms and formulas, a language for specifying background logical theories and logics, and the command language. Some other tools that process SMT-LIBv2 are listed in the SMT-LIB web pages (cf. http://www.smtlib.org/utilities.html).

History. The number of solvers competing each year has consistently remained in the range of 9-13 entrants. Some solvers have competed for several consecutive years. Others are new entrants. The introduction of SMT-LIBv2 as the standard language for benchmarks was a significant event. The new language required solvers to revise their front-ends and to add new capabilities. As a result, some solvers did not continue participating, at least not immediately. However, the use of SMTLIBv2 also increased the expressiveness of benchmarks. Thus benchmarks representing the needs of industrial applications were able to be added; the application track of the competition was added to demonstrate this capability and the corresponding abilities of solvers.

Solvers. The competition registration includes information about each competing solver. In addition, some solver groups provided summaries of their solvers and their recent technical advances. The provided summaries are included in these proceedings as additional papers.

Acknowledgments. Particular thanks is due to Morgan Deters, who is running the computational details of SMT-COMP 2012 on SMT-Exec, when it be came clear that the competition would need to reuse SMT-Exec in 2012. The cost of executing the SMT competition is underwritten by the SMT Workshop. The SMT-Exec computational resources are hosted by the University of Iowa Computer Science Department and maintained by Aaron Stump and the university’s IT group. Funds for the SMT-Exec cluster were provided by the U.S. National Science Foundation, under grant CNS-0551697. David Cok is chairing the SMT-COMP organizing committee for 2012, with Alberto Griggio and Roberto Bruttomesso as co-organizers. Morgan Deters and Aaron Stump designed and implemented the SMT-Exec service.

References


---

5. Not all competing solvers submitted summaries for the proceedings; it may also be that a solver with a summary for some reason was not able to compete.
Introduction

SMTInterpol [CHN12] is a proof-producing and interpolating SMT-solver written in Java. It is available from http://ultimate.informatik.uni-freiburg.de/smtinterpol under the GNU Lesser General Public License (LGPL) version 3.0. The solver reads input in SMTLIB format. It includes a parser for version 1.2, and a parser for the current version. All required and some optional commands of the SMTLIB standard are supported. SMTInterpol supports the quantifier-free combination of uninterpreted functions and linear (real and integer) arithmetic, i.e., the SMTLIB logics QF_UF, QF_LIA, QF_LRA, QF_UFLIA, and QF_UFLRA. For all these logics, SMTInterpol supports the computation of inductive sequences of Craig interpolants, which are used by several interpolation-based model checkers [HHP09, HHP10, EHP12].

All formulas are stored in a central term repository. The repository type-checks the formulas. Asserted formulas are converted to CNF using Plaisted–Greenbaum encoding [PG86]. The core of the solver is a CDCL engine that is connected to multiple theories. The engine uses these theories during constraint propagation, backtracking, and consistency checking.

For uninterpreted functions and predicates, we use a theory solver based on the congruence closure algorithm. An extension to arrays and quantifiers via e-matching is under development. For linear arithmetic, we use a theory solver based on the Simplex algorithm [DdM06]. It always computes the strongest bounds that can be derived for a variable and uses them during satisfiability checks. If a conflict cannot be explained using known literals, the solver derives new literals and uses them in conflict explanation. Disequalities are used to strengthen bounds, or are delayed until final checks. The solver supports integer arithmetic using a variant of the cuts from proof technique [DDA09] together with a branch-and-bound engine.

SMTInterpol uses a variant of model-based theory combination [dMB08]. The linear arithmetic solver does not propagate equalities between shared variables but introduces them as decision points. The model mutation algorithm resolves disequalities and tries to create as many distinct equivalence classes as possible.

Interpolation

SMTInterpol produces inductive sequences of interpolants for the SMTLIB logics QF_UF, QF_LRA, QF_UFLRA, QF_LIA, and QF_UFLIA. Since the integer logics defined in the SMTLIB standard are not closed under interpolation, SMTInterpol extends these logics with the division and modulo operators with constant divisor.

The architecture of the interpolation engine roughly follows the DPLL(T) paradigm: A core interpolator produces partial interpolants for the resolution steps while theory specific interpolators produce partial interpolants for T-lemmas. In the presence of mixed literals, i.e., literals that use symbols from more than one block of the interpolation problem, an approach loosely based on the method of Yorsh et al. [YM05] is used. The basic idea of the approach used in SMTInterpol is to virtually purify each mixed literal using an auxiliary variable, to restrict the places where the variable may occur in partial
interpolants, and to use special resolution rules to eliminate the variable when the mixed literal is used as a pivot. In essence, for convex theories, this approach can be seen as a lazy version of the method of Yorsh et al. The approach also works for non-convex theories using disjunctions in the interpolants.

New Developments

Compared to the version that participated in the SMT competition in 2011, several performance improvements have been implemented. These include clause minimization techniques and a new pivot strategy in the linear arithmetic solver. Additionally, the assertion stack management has been reworked to be more stable.

For unsatisfiable formulas, SMTInterpol supports the optional SMTLIB command `get-unsat-core`. This command extracts an unsatisfiable core from a proof tree. This technique does not guarantee minimality of the returned core. The optional command `get-proof` was already supported in last year’s version. Additionally, the non-standard command `get-interpolants` can be used to compute an inductive sequence of Craig interpolants. The interpolation engine is complete for all logics supported by SMTInterpol.

For satisfiable formulas, SMTInterpol supports the optional SMTLIB command `get-value` and the non-standard command `get-model`. These commands can be used to inspect the model produced by SMTInterpol. For uninterpreted sorts, SMTInterpol generates a finite sort interpretation. The domain of this interpretation contains input terms instead of abstract values (see the SMTLIB standard).

References

SONOLAR SMT-Solver
Florian Lapschies, Jan Peleska, Elena Gorbachuk and Tatiana Mangels
University of Bremen, FB3/AGBS, 28359 Bremen, Germany
{florian,jp,elenav,tatiana}@informatik.uni-bremen.de

SONOLAR is an SMT-solver for the theory of bit-vectors and for bit-vectors
with arrays. It uses bit-blasting to translate problems to SAT. To this end, the
input formula is first represented as a directed acyclic formula graph on which
word-level rewrite rules, normalization and variable substitution are applied.
The result is then translated to an And-inverter-Graph (AIG) which allows to
apply bit-level rewrite rules. After translation to CNF the problem is finally fed
into MiniSat 2.2[2].

In order to reason about arrays we adopted the approach described in [1]
which over-approximates reads, writes and equalities on arrays and incremen-
tally adds lemmas on demand until the SAT solver produces a consistent model
or decides the problem as unsatisfiable.

In addition to the standard bit-vector operations SONOLAR also supports
IEEE-754 floating point operations, including all rounding modes, that are bit-
blasted to SAT as well.

In comparison with last years version the solver received the following im-
provements:

• Many new rewrite-rules have been added.
• The code for applying substitutions has been rewritten to apply all current
  substitutions in one rewrite pass.
• A new library for performing multiple precision modular arithmetic more
  efficiently by using word-level arithmetic has been developed.
• The AIG implementation has been rewritten to only submit nodes to the
  SAT solver that are relevant for deciding satisfiability. After solving, the
  model of the SAT solver is then extended as needed.
• The SMTLIB2 front-end now supports the get-value and get-assignment-
  commands, the :named-attribute and the definition of interpreted func-
  tions.
• A rich C++ interface has been added to use the solver as a library.

Moreover – this is currently work in progress and not activated for the com-
petition – SONOLAR performs constraint propagation on the formula graph
using two domains, bit vector intervals and symbolic bit vectors. In our sym-

dolic bit vector approach each individual bit of a vector is a member of a set of
symbolic bits, where they optionally may be inverted. For example, the set
\{\overline{b_0}, b_1, \overline{b_2}\} is interpreted as constraint \(b_0 = b_1 \land b_2 = \overline{b_1}\). While initially
every bit is contained in its own set, these sets successively get merged during
constraint propagation. Propagation is performed in a forward-backward fashion until a fix-point is reached or an inconsistency has been found. Finally this information is used to generate a more efficient SAT encoding.

The solver is currently used for automated test data generation in model based testing \cite{7,6,5} as well as for C/C++ programs \cite{3,3}.

The latest version of SONOLAR is available at \url{http://www.informatik.uni-bremen.de/~florian/sonolar/}. Binary releases are available for Linux and Windows and may be used for research and evaluation purposes in an academic environment.

References


\cite{5} Jan Peleska, Artur Honisch, Florian Lapschies, Helge Löding, Hermann Schmid, Peer Smuda, Elena Vorobev, and Cornelia Zahlten. Embedded systems testing benchmark, 2011. \url{http://www.mbt-benchmarks.org}.


STP2

Trevor Hansen, Peter Schachte and Harald Søndergaard
The University of Melbourne

Vijay Ganesh
Massachusetts Institute of Technology

June 2012

STP [Gan07] is an efficient open source solver for $\text{QF}_{\text{BV}}$ and arrays without extensionality. STP recursively simplifies bit-vector constraints, solves linear bit-vector equations, and then eagerly encodes them to CNF for solving. Array axioms are added as needed, during an abstraction-refinement phase.

The version of STP submitted to STMCOMP 2012 is based on revision 1659 from STP’s publicly available source code repository\(^1\).

For the parallel track STP2 converts $\text{QF}_{\text{BV}}$ problems into CNF then loads that CNF into ppFolio, a simple portfolio SAT Solver. ppFolio in turn calls: Cryptominisat, LingelLing, Clasp, March, hi and TNM.

Contributions to STP

STP 1 was developed by Vijay Ganesh under the supervision of Professor David Dill. STP 2 was developed by Trevor Hansen under the supervision of Peter Schachte and Harald Søndergaard. STP handles arbitrary precision integers using Steffen Beyers library. STP encodes into CNF via the and-inverter graph package ABC of Alan Mishchenko [BM10]. We found many defects using Robert Brummayer and Armin Biere’s fuzzing and delta debugging tools [BB09].

Thanks for recent bug reports and patches to: Xu Zhongxing (help with the C-API), Edward Schwartz (nice test cases), Spencer Whitman (build script), Tom Bergan (help with the C-API), Stephan Falke (build script), Khoo Yit Phang (important defects), Jianjun Huang (Bug report), and Jingyue Wu (infinite loop).

References


\(^1\)To obtain the C++ source code follow the link from STP’s website: http://sites.google.com/site/stpfastprover/
Tiffany de Wintermonte (TdW), is an experimental eager solver for the combined theory of bit-vectors and arrays. TdW shares some code with STP2, another bit-vector and array solver.

TdW handles arbitrary precision integers using Steffen Beyers library. TdW encodes into CNF via the and-inverter graph package ABC of Alan Mishchenko et al. [BM10]. TdW uses Glucose 2.0 [AS09] as its SAT solver. We found many defects using Robert Brummayer and Armin Biere's fuzzing and delta debugging tools [BB09].

References

