DRAFT DRAFT Satisfiability Modulo Theories Competition (SMT-COMP) 2012: Rules and Procedures

Roberto Bruttomesso
Computer Science
USI, Lugano (Switzerland)

David R. Cok
GrammaTech, Inc.
Ithaca, NY USA

Alberto Griggio
ES Division
FBK, Trento (Italy)

This version revised 2012-1-22

1 Introduction

The annual Satisfiability Modulo Theories Competition (SMT-COMP) is held to spur advances in SMT solver implementations 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 CASC and SAT competitions for first-order and propositional reasoning tools, respectively, have spurred significant innovation in their fields [7, 5]. More information on the history and motivation for SMT-COMP can be found at the SMT-COMP web site, `www.smtcomp.org`, and in reports on previous competitions ([3, 4, 2, 1]). SMT-COMP 2012 is affiliated with the SMT workshop (`http://smt2012.loria.fr/`) at the 6th International Joint Conference on Automated Reasoning (IJCAR) (`http://ijcar.cs.manchester.ac.uk/`).

The rest of this document, updated from the last year's version[1], describes the rules and competition procedures for SMT-COMP 2012. The principal changes from last year's version are the following:

- We are concentrating on just a few of the benchmark divisions this year. Some divisions, such as QF_UF, are subsumed into more expressive logics; others have received only light interest in past competitions. Our goal is to focus the competition on divisions of particular interest to applications. To encourage new entrants, we will also accept solvers against any benchmark class in an exhibition category: results will be displayed and publicly reported.
- We are retaining and accenting last year's "application track" and encourage submission of benchmarks and competition against these benchmarks.
- The penalty for producing an incorrect result is increased. Competitors producing all correct results are ranked above those producing incorrect results (a "soft" disqualification for unsound tools).

[1]Earlier versions of this document include contributions from Clark Barrett, Albert Oliveras, Aaron Stump, and Morgan Deters.

- We are adding an “unsat core” division of QF_BV.
- We will have an exhibition division of solvers that produce proofs.
- The competition will be run with a new StarExec service, replacing the previous SMT-Exec service. At this writing, the new service is still being completed and tested; it will be announced later. (The previous service will be maintained in reserve.)

Additionally, non-competitive divisions and tracks will not be exhibited as part of the competition: at least two competitors must be entered into a division for it to run as part of the competition. If necessary, and based on final submissions, the organizers may elect to combine competition divisions to make them competitive (such decisions will be made with input from the community).

As in SMT-COMP 2011, the 2012 version will incorporate a small number of random “fuzzer-generated” instances (see below) to help promote attention to solver correctness.

It is important for competitors to track discussions on the SMT-COMP mailing list, as clarifications and any updates to these rules will be posted there.

2 Entrants

Solver format. An entrant to SMT-COMP is an SMT solver submitted using the StarExec[2] service. StarExec enables members of the SMT research community to run solvers on jobs consisting of benchmarks from the SMT-LIB benchmark library. Jobs are run on a computer cluster purchased with funds from a National Science Foundation Computing Research Infrastructure grant. StarExec is provided free of charge, but it does require a minimal registration, which verifies an email address and prevents misuse of the service. Registered users may then upload their own solvers to run, or may run public solvers already uploaded to StarExec. StarExec provides a variety of tabular and graphical displays of results of solver executions, for comparison. For the main (sequential-solver) and application tracks of SMT-COMP 2012, StarExec will be configured so that jobs are run on a 64-bit, uniprocessor Linux kernel. For the parallel-solver track, StarExec will be configured so that jobs are run on a 64-bit, multiprocessor Linux kernel.

For participation in SMT-COMP, a solver must be uploaded as a “competition” solver via the StarExec upload mechanism, or, alternatively, a previously-uploaded solver may be marked as a “competition” solver. In either case, *uploads must be marked for competition before the deadline;* uploading a solver to StarExec is not sufficient for competition entry if it’s not marked as being a competition entrant. The md5 checksums of competition submissions will be public immediately after the deadline for competition entry has passed to ensure transparency, and the submissions themselves will be made public after the competition. Source code need not be provided. However, in order to encourage sharing of source code, extra recognition will be given to solvers providing source code distributions including recognition for the top such solver in each division. Instructions for uploading solvers and machine specifications will be posted as soon as they are available.

[2]StarExec is still under development as of 21 January 2012; updates on its progress and use will be posted on www.smtcomp.org and through the SMT-COMP mailing list. Statements in these rules may need correction once the UI for the service is completed.

System description. As part of their submission via StarExec, SMT-COMP entrants must also include a short (1–2 pages) description of the system. This should include a list of all authors of the system and their present institutional affiliations. The programming language(s) and basic SMT solving approach employed should be described (*e.g.*, lazy integration of a Nelson-Oppen combination with SAT, translation to SAT, etc.). System descriptions are encouraged to include a URL for a web site for the submitted tool, but this is optional. System descriptions must also include a 32-bit unsigned integer. These numbers, collected from all submissions, are used to seed the pseudo-random benchmark selection algorithm, as well as the benchmark scrambler.

The StarExec upload system will ask for the system description and the random seed when a solver is marked for competition, so their inclusion in the uploaded archive itself is optional.

Other systems. As in previous years, due to limitations on computational resources, the organizers reserve the right not to accept multiple versions of the same solver (defined as sharing 50% or more of its source code). The organizers reserve the right to submit their own systems, or other systems of interest, to the competition.

Wrapper tools. A *wrapper tool* is defined as any tool which calls an SMT solver not written by the author of the wrapper tool. The other solver is called the *wrapped tool*. There are several rules governing wrapper tools that wrap tools that have been or could be submitted as independent entrants.[3] For the purposes of these rules, multiple versions of a wrapped tool are considered different tools. The goal of these rules is to require wrapper tools to outperform the tools they wrap (since otherwise, there is no apparent quantitative way to argue that the wrapper tool improves upon the wrapped tool).

- The name of the wrapper tool must end with "+name", where name is the name of the wrapped tool (*e.g.* "Flash+CVC3" for a tool wrapping CVC3).
- If the wrapped tool is from last year's SMT-COMP or earlier, then for each division entered by the wrapper tool, if the wrapper tool does not place ahead, according to the scoring rules below, of last year's winner in that division, it will be disqualified from that division (but not necessarily from the whole competition).
- If the wrapped tool was released after last year's SMT-COMP, then the wrapper tool can be entered only if
  - Permission has been given by the author of the wrapped tool
  - The wrapped tool is submitted and entered in every division in which the wrapper tool is entered.

  For each division entered by the wrapper tool, if the wrapper tool does not place ahead of the wrapped tool in that division, it will be disqualified from that division.

Attendance. As with previous SMT-COMPs, submitters of an SMT-COMP entrant need not be physically present at the competition to participate or win.

[3]A wrapper for a wrapped tool that does not accept SMT-LIBv2 format is considered to simply be an SMT version of the underlying tool.

Deadlines

Main competition track. SMT-COMP entries must be submitted via StarExec by 7pm, Eastern U.S. time, June 15, 2012. At that time the StarExec service will be closed to the public to prepare for the competition, with the exception that resubmissions of existing entries will be accepted until 7pm, Eastern U.S. time, June 18, 2012. We strongly encourage participants to use this weekend grace period *only* for the purpose of fixing any bugs that may be discovered and not for adding new features as there will be no opportunity to do extensive testing using StarExec after the original deadline on June 15.

The versions that are present on StarExec at the conclusion of the grace period will be the ones used for the competition, and versions submitted after this time will not be used. The organizers reserve the right to start the competition itself at any time after the open of the New York Stock Exchange on the morning of June 22. See Section 6 below for a full timeline.

Application track. The same deadlines and procedure for submitting to the main track will be used also for the application track. Submissions to both the application track and the main competition are independent: participants must submit explicitly to both events to participate in both, and they may submit different (or differently configured) solvers to each. Benchmarks will be scrambled also for this division, using the same scrambler and seed as the main track. Entrants should still include a system description, as for the regular competition.

Parallel solver track. Solvers employing concurrency are invited to participate in a demonstration that will be run before the regular competition. Parallel solvers must be submitted by the same June 15th deadline, with the same grace period for resubmission as in the regular competition. Submissions to both the parallel solver track and the main competition are independent: participants must submit explicitly to both events to participate in both, and they may submit different (or differently configured) solvers to each. Benchmarks may or may not be scrambled for this division. Entrants should still include a system description, as for the regular competition.

Note that the SMT-COMP organizers may include other ("historical" or otherwise relevant) solvers in all the competition tracks for demonstration and comparison. The organizers reserve the right to include or exclude such solvers, and to make simple modifications to historical sequential solvers to (naïvely) take advantage of multiple processors or to connect them with a parser for the SMT-LIB version 2.0 format (*e.g.* by implementing a wrapper tool).

3 Execution of Solvers

Solvers will be publicly evaluated in the following tracks, listed here and described in detail below. In exhibition tracks, solvers are evaluated and results posted, but no awards are announced. The other tracks are competition tracks; if there are sufficient entrants, a winner will be announced.

- a main track: sequential execution evaluated separately on benchmarks from each of several different logics

- a parallel track: parallel execution evaluated on the same benchmarks and the same divisions as the main track
- application track: evaluation on command scripts [TODO: FOR WHICH LOGICS]
- exhibition track: solvers (particularly new entrants) may be submitted against any benchmark division; the solver will be evaluated and the results posted, though not included in competition
- unsat-core track: a competition among solvers capable of computing unsat cores
- proof-generation exhibition: an exhibition track for solvers capable of generating proofs

3.1 Logistics

Dates of competition. We anticipate that the bulk of the main track of the competition will take place during the course of IJCAR 2012, from June 26 to the 29th. Results will be announced in a special session of IJCAR, on the last day of the conference, as well as at the SMT workshop and on the SMT-COMP web site. Intermediate results will be regularly posted to the SMT-COMP website as the competition runs.

If there are enough competitors to run the parallel track, we intend to run it before or after the main competition (depending on the estimated duration of the track).

The organizers will prioritize the running of the competition tracks, and may shift the time period or order of the competition or exhibition tracks in order to complete SMT-COMP in the course of the IJCAR conference.

Input and Output. Participating solvers must read a single benchmark script (defined below, not part of the 2.0 standard), presented on its standard input channel. The script is in the concrete syntax of the SMT-LIB format, version 2.0. A benchmark script is essentially just the translation of a benchmark from the version 1.2 specification. In more detail, a **benchmark script** is just a script where:

1. The (single) **set-logic** command setting the benchmark's logic is the first command.
2. The **exit** command is the last command.
3. There is exactly one **check-sat** command, following possibly several **assert** commands.
4. There is at most one **set-info** command for `status`.
5. The formulas in the script belong to the benchmark's logic, with any free symbols declared in the script.
6. Extra symbols are declared exactly once before any use, using **declare-sort** or **declare-fun**. They must be part of the allowed signature expansion for the logic. Moreover, all sorts declared with a **declare-sort** command must have zero arity.

7. In the unsat-core competition, the **set-logic** command is preceded by a **set-option :produce-unsat-cores** command and the **check-sat** command is followed by a **get-unsat-core** command. Also, some or all of the **assert** commands will assert named formula (for example, **assert (! P :named F1)**). The **get-unsat-core** command must return a parenthesized list of formula names, as specified by the SMTLIBv2 standard.

8. In the proof generation competition, the **set-logic** command is preceded by a **set-option :produce-proofs** command and the **check-sat** command is followed by a **get-proof** command. The **get-proof** command must return a proof in a solver-dependent format as an S-expression.

9. No other commands besides the ones just mentioned may be used.

The SMT-LIB format specification is publicly available from the "Documents" section of the SMT-LIB website [8]. Solvers will be given formulas just from the Problem Divisions indicated during their submission to SMT-Exec. Example benchmark scripts for several Problem Divisions are reported in the Appendix. Note that they are provided for illustrative purposes only: please refer to the SMT-LIB format specification and the above definition of benchmark script for the official specification of the input format for SMT-COMP.

3.2 Main track and parallel track

The main track competition will consist of selected benchmarks in each of the logic divisions given below. Each benchmark script will be presented to the standard input of the solver. Each SMT-COMP entrant is then expected to attempt to report on its standard output channel whether the formula is satisfiable ("`sat`", in lowercase, without the quotation marks) or unsatisfiable ("`unsat`"). An entrant may also report "`unknown`" to indicate that it cannot determine satisfiability of the formula. For more detailed information on the output format, see the description on the StarExec "Upload a Solver" page.

Timeouts. Each SMT-COMP solver will be executed on an unloaded competition machine for each given formula, up to a fixed time limit. The time limit is yet to be determined, but it is anticipated to be 20 minutes, as it was in 2011. Solvers that take more than this time limit will be killed. Solvers are allowed to spawn other processes. These will be killed at approximately the same time as the first started process, using the TreeLimitedRun script, developed for the CASC competition and available on the SMT-COMP web page. A timeout scores the same as if the output is "`unknown`".

Aborts and unparsable output. Solvers which exit before the time limit without reporting a result (*i.e.* due to exhausting memory, crashing, or producing output other than `sat`, `unsat`, or `unknown`) will be considered to have aborted. An abort scores the same as if the output is "`unknown`". Also, as a further measure to prevent misjudgments of solvers, any "`success`" outputs will be ignored[4].

[4]Note that SMT-LIBv2 requires to produce a "`success`" answer after each **set-logic**, **declare-sort**, **declare-fun** and **assert** command (among others), unless the option **:print-success** is set to false; ignoring the `success` outputs therefore allows for submitting fully-compliant solvers without the need of a wrapper script, while still allowing entrants of previous competitions to run without changes.

Persistent state. Solvers are allowed to create and write to files and directories during the course of an execution, but they are not allowed to read such files back during later executions. Any files written should be put in the directory in which the tool is started, or in a subdirectory.

3.3 Application track

The application track evaluates SMT solvers when interacting with an external verification framework, *e.g.*, a model checker. This interaction, ideally, happens by means of an online communication between the model checker and the solver: the model checker repeatedly sends queries to the SMT solver, which in turn answers either `sat` or `unsat`. In this interaction an SMT-solver is required to accept queries incrementally via its standard input channel.

In order to facilitate the evaluation of the solvers in this track, we will set up a "simulation" of the aforementioned interaction, as was done in 2011. In particular each benchmark in the application track represents a realistic communication trace, containing multiple **check-sat** commands (possibly with corresponding **push 1/pop 1** commands), which is parsed by a *trace executor*. The trace executor serves the following purposes:

- it simulates the online interaction by sending single queries to the SMT solver (through stdin);
- it prevents "look-ahead" behaviours of SMT solvers;
- it records time and answers for each call, possibly aborting the execution in case of a wrong answer;
- it guarantees a fair execution for all solvers by abstracting from any possible crash, misbehaviour, etc. that may happen on the model checker side.

Input and Output. Participating solvers will be connected to a trace executor which will incrementally send commands to the standard input channel of the solver and read responses from the standard output channel of the solver. The commands will be taken from an **incremental benchmark script**, which is an SMT-LIB 2.0 script which satisfies points 1., 2., 4.–7. of the definition of benchmark script given in §3.2, plus the following:

3'. There are one or more **check-sat** commands, each preceded by one or more **assert** commands and zero or more **push 1** commands, and followed by zero or more **pop 1** commands.

Furthermore, the trace executor will send a single **set-option :print-success true** command to the solver before sending commands from the incremental benchmark script.

Solvers must respond immediately to the commands sent by the trace executor, with the answers defined in the SMT-LIB 2.0 format specification, that is, with a `success` answer for **set-option**, **declare-sort**, **declare-fun**, **assert**, **push 1**, and **pop 1** commands, and with a `sat`, `unsat`, or `unknown` for **check-sat** commands.

Timeouts. A time limit is set for the whole execution of each application benchmark (consisting of multiple queries). We anticipate the timeout to be around 30 minutes (as it was in 2011).

3.4 Exhibition track

To encourage new entrants in any logic, we will have an exhibition track. Any solver submitted for "competition" in a non-competitive track will be executed on benchmarks for the logic divisions chosen for that solver. The results will be made public and the solver acknowledged. Any logic divisions for which there are benchmarks may be specified.

3.5 Unsat-core and proof-generation tracks

Applications such as software verification and model checking are enhanced by having proofs and unsatisfiable cores available from SMT solvers. Accordingly we are encouraging solvers to add such capabilities by recognizing them in SMTCOMP 2012.

The unsat-core track will evaluate solvers' capability to generate unsatisfiable cores for problems that are known to be unsatisfiable. Solvers will be measured by the smallness of the unsat core they return. The SMT-LIBv2 language accommodates this functionality by providing two features: the ability to name top-level (asserted) formula and the ability to request an unsat-core after a check-sat command returns `unsat`. The unsat-core that is returned consists of a list of names of formula. In the competition we will check that the returned unsat-core is well-defined and is still unsatisfiable.

Similarly, the competition will feature an exhibition track of proof-generating solvers. This is simply an exhibition track because there is as yet no SMT-LIB standard way to express proofs. We hope that encouraging the capability in solvers will also encourage a common proof format. The exhibition will count the number of benchmarks for which a solver successfully generates a proof; the proofs themselves may be spot checked by hand or in a semi-automated fashion, depending on the format.

TODO: What benchmark logics will be used.

4 Benchmarks and Problem Divisions

We expect the problem divisions for SMT-COMP 2012 to include the following SMT-LIB *logics*. These logics are specified in SMT-LIB format on the SMT-LIB web page. Note that the "QF_" prefix means the division's formulas are quantifier-free. However, the organizers reserve the right to add (remove) divisions if (not) enough benchmarks and solvers exist for a particular division.

Note that this year we are reducing the number of competitive benchmark divisions. Our reasons are to concentrate the competition and to focus development and attention on benchmarks that are more challenging and relevant to applications.

- QF_BV: fixed-width bitvectors. This logic is important to "bit-blasting" model checking of software.

- QF_AUFBV: arrays, fixed-width bitvectors and uninterpreted functions. This logic is also key to software model-checking, but adds arrays (which can be used for memory models) and uninterpreted functions.

- QF_UFLIA: uninterpreted functions and linear integer arithmetic (benchmarks may also be taken from QF_LIA). This division evaluates reasoning about integers.
- QF_UFLRA: uninterpreted functions and linear real arithmetic (benchmarks may also be taken from QF_LRA). This division evaluates reasoning about real numbers.
- AUFLIA$+p$: (quantified) arrays, uninterpreted functions and linear integer arithmetic, patterns included. This and the subsequent divisions evaluate solvers' abilities to handle quantified formula.
- AUFLIA$-p$: (quantified) arrays, uninterpreted functions and linear integer arithmetic, patterns not included.
- AUFNIRA: (quantified) arrays, uninterpreted functions and mixed nonlinear integer and real arithmetic

4.1 Main and parallel tracks

Benchmark sources. Benchmark formulas for these divisions will be drawn from the SMT-LIB library. Any benchmarks added to SMT-LIB by the April 15th release (see the timeline in Section 6) will be considered eligible. SMT-COMP attempts to give preference to benchmarks that are "real-world," in the sense of coming from or having some intended application outside SMT.

Benchmark availability. Benchmarks will be made available by April 15, 2012. No additional benchmarks will be added after this date, but benchmarks may be modified or removed to fix possible bugs or other issues, or to adjust their difficulty score (see below). The final release that will be used for the competition will be posted on June 1. The set of selected benchmarks will be published when the competition begins.

Benchmark demographics. In SMT-LIB, benchmarks are organized according to *families*. A benchmark family contains problems that are similar in some significant way. Typically they come from the same source or application, or are all output by the same tool. *Each top-level subdirectory within a division represents a distinct family.*

Each benchmark in SMT-LIB also has a *category*. There are four possible categories:

- *check*. These benchmarks are hand-crafted to test whether solvers support specific features of each division. In particular, there are checks for integer completeness (*i.e.* benchmarks that are satisfiable under the reals but not under the integers) and big number support (*i.e.* benchmarks that are likely to fail if integers cannot be represented beyond some maximum value, such as $2^{31} - 1$).

  Using the same random seed as for benchmark selection and scrambling, 5 "check" benchmarks will be randomly generated using Robert Brummayer's SMT fuzzing tool [6]. The (exact version of this) tool will be publicly available from the SMT-COMP 2012 web site before the competition. Since these benchmarks are generated after the random seed is fixed, they cannot be known in advance to any competitors, including the organizers. The rationale for including these benchmarks is to try to take a step towards stronger certification that solvers are correct. In future years, we envision SMT-COMP requiring that solvers pass some kind of pre-qualifying round based on SMT fuzzing. The inclusion of these fuzzing benchmarks that are not known in advance is intended to encourage (without requiring) solver implementors to test their solvers using fuzzing or similar bug-discovery techniques.

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

Benchmark selection. Before the selection process, each benchmark will be assigned a *difficulty:* a number between 0.0 and 5.0 inclusive, calculated as in 2011. The difficulty for a particular benchmark will be assigned by running SMT solvers from previous competitions that finished in good standing and using the formula:

$$\text{difficulty} = \frac{5 \cdot \ln(1 + A^2)}{\ln(1 + 30^2)}$$

where A is the average time for the solvers to correctly solve the instance (in minutes). This computation of difficulties replaces a simpler formulation in earlier SMT-COMPs that didn't take into account the time solvers take. This calculation of difficulty recognizes that problems requiring more time by many solvers are more difficult problems. The logarithm is used to mark a larger change in difficulty (given a corresponding increase in solver average time) at smaller time scales than at higher ones (if $A = 1$, difficulty is 0.5; at $A = 1.7$, difficulty is 1.0; but a difficulty of 2.0 requires that $A = 4$); the square is used to flatten out this curve sightly at the low end.

When 5 or more solvers are used for the calculation, two times are dropped from the calculation of the average (one at the maximum and one at the minimum). Solvers giving an incorrect answer are not counted in the average; solvers crashing, timing out, or giving an unknown result are considered as taking 30 minutes (which, with the above formula, pulls the difficulty toward 5.0). If there are available solvers, but no average is defined under these rules, the difficulty shall be 5.0. For new divisions, where there are no available solvers to compute the difficulty, the difficulty will be computed using whatever means are available to the organizers for that purpose.

The following scheme will be used to choose competition benchmarks within each division. Unknown-status benchmarks from SMT-LIB are considered ineligible for competition and are not used. The selection is implemented by our benchmark selection tool, source for which will be available at `www.smtcomp.org`.

1. **Check benchmarks included.** All benchmarks in category *check* are included.
2. **Retire very easy benchmarks.** The most difficult 300 non-check non-unknown benchmarks in each division are always included, together with all benchmarks on which at least one 2011 solver required more than 5 seconds. *This is intended to have the effect of retiring "very easy" benchmarks that were solved by every 2011 solver in less than 5 seconds,* unless *doing so reduces the pool of benchmarks for the division to less than 300.*

3. **Retire inappropriate benchmarks.** The competition organizers will remove from the eligibility pool certain SMT-LIB benchmarks that are inappropriate or uninteresting for competition, or cut the size of certain benchmark families to avoid their over-representation.

4. **Division selection pools created.** For non-*check* benchmarks, selection pools are created. For benchmark families with ≤ 200 eligible, non-*check* benchmarks, all are added to this pool; otherwise, 200 such benchmarks are added to the pool with the following distribution:

   - 20 with solution **sat** and difficulty on $[0, 1]$
   - 20 with solution **sat** and difficulty on $(1, 2]$
   - 20 with solution **sat** and difficulty on $(2, 3]$
   - 20 with solution **sat** and difficulty on $(3, 4]$
   - 20 with solution **sat** and difficulty on $(4, 5]$
   - 20 with solution **unsat** and difficulty on $[0, 1]$
   - 20 with solution **unsat** and difficulty on $(1, 2]$
   - 20 with solution **unsat** and difficulty on $(2, 3]$
   - 20 with solution **unsat** and difficulty on $(3, 4]$
   - 20 with solution **unsat** and difficulty on $(4, 5]$

   If 20 are not available in one of these subdivisions, all that are available are added, and remaining slots are reallocated to the others. This process is iterated so that it is guaranteed that 200 benchmarks from the benchmark family are in the selection pool, in equal numbers from each subdivision, so far as possible. (In cases where *e.g.* there are only two available slots and they can be allocated to one of three subdivisions, they are allocated randomly but are guaranteed to be allocated to *distinct* subdivisions.)

5. **Category slot allocation.** Next, 200 slots are allocated for the division as follows:[5]

   - 170 from category *industrial*
   - 20 from category *crafted*
   - 10 from category *random*

   If there are fewer than 20 (respectively, 10) *crafted* or *random* benchmarks in the division pool, more *industrial* slots are allocated to make 200 total for the division. If there are too few *industrial* benchmarks in the division pool, more *crafted* slots are allocated to make 200 total for the division. (In no division are there not enough of both industrial and crafted benchmarks.)

6. **Category subdivision slot allocation.** For each category, given that it has n slots allocated to it, the slot allocation is further subdivided as follows:

[5]The number "200" is a guideline, and is expected to be used. However, the SMT-COMP organizers reserve the right to reduce this to an appropriate value to ensure a timely end to the competition, and may do so on a per-division basis. The category allotments, etc., will remain the same proportion of the total.

- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **sat** with difficulty on $[0, 1]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **sat** with difficulty on $(1, 2]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **sat** with difficulty on $(2, 3]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **sat** with difficulty on $(3, 4]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **sat** with difficulty on $(4, 5]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **unsat** with difficulty on $[0, 1]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **unsat** with difficulty on $(1, 2]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **unsat** with difficulty on $(2, 3]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **unsat** with difficulty on $(3, 4]$
- $\left\lfloor \frac{n}{10} \right\rfloor$ slots for solution **unsat** with difficulty on $(4, 5]$

Remaining slots are allocated randomly to distinct subdivisions. If there aren't enough benchmarks in the pool meeting one or more of the above subdivision requirements for the category, the subdivision allocation is reduced to the number available in the pool that meet the requirements. To make up the full category allotment, remaining slots are allocated equally to subdivisions with enough benchmarks in the pool meeting their requirements. This process ensures that all slots can be filled with benchmarks from the pool.

7. **Benchmark selection.** Benchmarks from the pool are assigned randomly to slots.

In the end, up to 200 non-*check* benchmarks per division are included in the competition, together with all the *check* benchmarks. Some divisions may have fewer than 200 non-*check* benchmarks, in which case all of them are included using this selection scheme.

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

Pseudo-random numbers will be generated using the standard C library function `random()`, seeded (using `srandom()`) with the sum, modulo 2^{30}, of the numbers provided in the system descriptions (see Section 2 above) by all SMT-COMP entrants other than the organizers. Additionally, the integer part of the opening value of the New York Stock Exchange Composite Index on June 25 will be added to the other seeding values. This helps provide transparency, by guaranteeing that the organizers cannot manipulate the seed in favor of or against any particular submitted solver. Benchmarks will also be slightly scrambled before the competition, using a simple benchmark scrambler seeded with the same seed as the benchmark selector. Both the scrambler and benchmark selector will be publicly available before the competition. Naturally, solvers must not rely on previously determined identifying syntactic characteristics of competition benchmarks in testing satisfiability (violation of this is considered cheating).

4.2 Application track

Benchmark sources. Benchmarks for the application track will be collected by the SMT-COMP organizers. Any benchmark available to the organizers by April 15th (see the timeline in Section 6) will be considered eligible.

Benchmark availability. A first release of the application track benchmarks will be made available March 15, 2012. More benchmarks can be collected, until April 15. No additional benchmarks will be added after this date, but benchmarks can be modified or removed to fix possible bugs or other issues. The final release that will be used for the competition will be posted on June 1.

Benchmark demographics and selection. All the available benchmarks will be used for the competition. As was the case in 2011, no difficulty will be assigned to the benchmarks for the application track. Benchmarks will be slightly scrambled before the competition, using the same scrambler and random seed as the main track. For this year, the selection algorithm for the application track will be chosen after the April 15th deadline, when the exact demographic of the benchmarks (collected with a usual "call for benchmarks") will be available.

TODO: Divisions for unsat core and proof tracks

5 Judging and Scoring

Main and application tracks The score for each benchmark is a triple $\langle e, n, m \rangle$, with e a non-negative number of erroneous results, $n \in [0, N]$ an integral number of points scored for the benchmark, where N is the number of **check-sat** commands in the benchmark, and $m \in [0, T]$ is the (real-valued) time is seconds, where T is the timeout. Recall that main track benchmarks will have just one **check-sat** command; application track benchmarks may have multiple **check-sat** commands. The score for the benchmark is initialized with $\langle 0, 0, 0 \rangle$ and then computed as follows.

- A correctly-reported `sat` or `unsat` answer after s seconds (counting from the beginning of this particular **check-sat**) contributes $\langle 0, 1, s \rangle$ to the running score.

- An answer of `unknown`, an unexpected answer, a crash, or a memory-out during execution of the query, or a benchmark timeout, aborts the execution of the benchmark and assigns the current value of the running score to the benchmark. (Recall that there is one timeout for the entire benchmark; there are no individual timeouts for queries.)

- An incorrect answer to a **check-sat** command has the effect of terminating the evaluation of that individual benchmark, and the returned score for the benchmark will be $\langle 1, 0, 0 \rangle$.

For example, if a benchmark has 5 **check-sat** commands, and a timeout of 100 seconds, and a solver solves the first four in 10 seconds each, then times out on the fifth, then the solver's score is $\langle 0, 4, 40 \rangle$. If another solver solves each of the first four in 10 seconds each, and the fifth in another 40 seconds, its score is $\langle 0, 5, 80 \rangle$. If a third solver solves the first four queries in 2 seconds each, but incorrectly answers the fifth, its score is $\langle 1, 0, 0 \rangle$.

As queries are only presented in order, this scoring system may mean that relatively "easier" queries are hidden behind more difficult ones located at the middle of the query sequence.

Benchmarks' scores are summed componentwise to form a solver's total score for the competition. Total scores are compared lexicographically—a score $\langle e, n, m\rangle$ is better than $\langle e', n', m'\rangle$ iff $e < e'$ or ($e = e'$ and $n > n'$) or ($e = e'$ and $n = n'$ and $m < m'$). That is, fewer errors takes precedence over more correct solutions, which takes precedence over less time taken.

Unsat-core track

The unsat-core track will be scored as follows. The score for each benchmark is a triple $\langle e, n, m\rangle$, with e a non-negative number of erroneous results, n the number of formula in the unsat core for the benchmark, and $m \in [0, T]$ is the (real-valued) time is seconds, where T is the timeout. The score for the benchmark is initialized with $\langle 0, 0, 0\rangle$ and then computed as follows.

- A correctly-reported unsat-core answer after s seconds (counting from the beginning of this particular **check-sat**) contributes $\langle 0, n, s\rangle$ to the running score, where n is the number of formula in the unsat core.

- An answer of `unknown`, an unexpected answer, a crash, or a memory-out during execution of the query, or a benchmark timeout, aborts the execution of the benchmark and assigns a score of $\langle 0, N, T\rangle$ for the benchmark, where N is the number of named asserted formula in the benchmark and T is the timeout value.

- The score for an incorrect answer to a **check-sat** or **get-unsat-core** command will be $\langle 1, N, T\rangle$, where N is the number of named asserted formula in the benchmark and T is the timeout value.

Scores are ordered as for the main track, except that lower n values are preferred, rather than higher values.

Number of competitors Winners in each Problem Division for which there are at least three entrants from distinct research groups competing will be taken to be those with the highest score and no erroneous results. In addition to recognizing the overall winner in each division, the top solver that provides its source code will also be recognized in each division. For Problem Divisions with fewer than three entrants, the results will be reported but no winner officially declared.

6 Timeline

March 15 First version of the benchmark library for the application track posted for comment. First version of the benchmark scrambler, benchmark selector and trace exector made available.

April 15 Final version of the benchmark scrambler, benchmark selector and trace exector made available.

April 15 No new benchmarks can be added after this date, but problems with existing benchmarks may be fixed.

June 1 Benchmark libraries are frozen.

June 15 (7pm EDT) Solvers due via StarExec (for all tracks), including system descriptions and magic numbers for benchmark scrambling.

June 18 (7pm EDT) Final versions due, fixing any last-minute bugs (this marks the end of the grace period for submissions).

June 22 Opening value of NYSE Composite Index used to complete random seed.

June 25–29 Anticipated dates for SMT-COMP 2012.

7 Mailing List

Interested parties should subscribe to the SMT-COMP mailing list, a link to which is found at `www.smtcomp.org`. Important late-breaking news and any necessary clarifications and edits to these rules will be announced there, and it is the primary way that such announcements will be communicated.

8 Disclaimer

- David Cok is the chief organizer of SMT-COMP 2012. He is responsible for all policy and procedure decisions, such as these rules. He is not associated with any group creating or submitting solvers. He has used solvers in industrial settings and is keenly interested to know which are the best.

- Alberto Griggio and Roberto Bruttomesso are co-organizers. They were also co-organizers in 2011. They are responsible for bringing recommendations from the previous year's experience. They will also be validating the competition setup, checking benchmarks, and operating the competition. They are associated with solver groups that will be submitting the solvers MathSat and OpenSMT, respectively.

References

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

[2] C. Barrett, L. de Moura, and A. Stump. Design and Results of the 2nd Annual Satisfiability Modulo Theories competition (SMT-COMP 2006). *Formal Methods in System Design*, 31(3):221–239, 2007.

[3] Clark Barrett, Morgan Deters, Albert Oliveras, and Aaron Stump. Design and results of the 4th annual satisfiability modulo theories competition (SMT-COMP 2008). In preparation.

[4] Clark Barrett, Morgan Deters, Albert Oliveras, and Aaron Stump. Design and results of the 3rd annual satisfiability modulo theories competition (SMT-COMP 2007). *International Journal on Artificial Intelligence Tools*, 17(4):569–606, 2008.

[5] D. Le Berre and L. Simon. The essentials of the SAT 2003 competition. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 452–467. Springer-Verlag, 2003.

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

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

[8] Silvio Ranise and Cesare Tinelli. The SMT-LIB web site, 2004. http://combination.cs.uiowa.edu/smtlib.

A Sample benchmark scripts for the main track

QF_UF

```
(set-logic QF_UF)
(set-info :status sat)
(declare-sort U 0)
(declare-fun f (U) U)
(declare-fun g (U) U)
(declare-fun A () Bool)
(declare-fun x () U)
(declare-fun y () U)
(assert
(let ((fx (f x))
      (cls1 (or A (= x y))))
  (and cls1 (distinct fx (g y)))))
(check-sat)
(exit)
```

QF_LRA

```
(set-logic QF_LRA)
(declare-fun x () Real)
(declare-fun y () Real)
(declare-fun A () Bool)
(assert
  (let ((i1 (ite A (<= (+ (* 2.0 x) (* (/ 1 3) y)) (- 4))
                   (= (* y 1.5) (- 2 x)))))
    (and
      i1
      (or (> x y) (= A (< (* 3 x) (+ (- 1) (* (/ 1 5) (+ x y)))))))))
(check-sat)
(exit)
```

QF_LIA

```
(set-logic QF_LIA)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun A () Bool)
(assert
  (let ((i1 (ite A (<= (+ (* 2 x) (* (- 1) y)) (- 4))
                   (= (* y 5) (- 2 x)))))
    (and
      i1
      (or (> x y) (= A (< (* 3 x) (+ (- 1) (* 1 (+ x y)))))))))
(check-sat)
(exit)
```

QF_BV

```
(set-logic QF_BV)
(declare-fun x () (_ BitVec 32))
(declare-fun y () (_ BitVec 16))
(declare-fun z () (_ BitVec 20))
(assert
  (let ((c1 (= x ((_ sign_extend 12) z))))
   (let ((c2 (= y ((_ extract 18 3) x))))
    (let ((c3
            (bvslt (concat z (_ bv5 12))
              (bvand (bvor (bvxor (bvnot x) ((_ zero_extend 28) #b1111))
                                   (concat #xAF02 y))
                     (concat (bvmul ((_ extract 31 16) x) y)
                             (bvashr (_ bv42 16) #x0001))))))
   (and c1 (xor c2 c3))))))
(check-sat)
(exit)
```

QF_AUFLIA

```
(set-logic QF_AUFLIA)
(declare-fun A () (Array Int Int))
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun P () Bool)
(declare-sort U 0)
(declare-fun f (U) (Array Int Int))
(declare-fun c () U)
(assert
  (let ((fc (f c)))
    (and
      (=> (= A (store fc x 5)) (> (+ (select fc y) (* 4 x)) 0))
      (= P (< (select A (+ 3 y)) (* (- 2) x))))))
(check-sat)
(exit)
```

QF_ABV

```
(set-logic QF_ABV)
(declare-fun x () (_ BitVec 32))
(declare-fun y () (_ BitVec 16))
(declare-fun z () (_ BitVec 20))
(declare-fun A () (Array (_ BitVec 16) (_ BitVec 32)))
(assert
  (let ((c1 (= ((_ sign_extend 12) z) (select A y)))
        (A2 (store A ((_ extract 15 0) x) x)))
   (let ((c2 (= A A2)))
    (let ((c3
            (bvslt (concat z (_ bv5 12))
              (bvand (bvor (bvxor (bvnot x)
                                  (select A2 ((_ zero_extend 12) #b1111)))
                                    (concat #xAF02 y))
                      (concat ((_ extract 15 0)
                                   (bvmul x (select (store A y x) #x35FB)))
                               (bvashr (_ bv42 16) #x0001))))))
   (and c1 (xor c2 c3))))))
(check-sat)
(exit)
```

B Sample benchmark scripts for the application track

```
(set-option :print-success true)
(set-logic QF_LRA)
(declare-fun c0 () Bool)
(declare-fun E0 () Bool)
(declare-fun f0 () Bool)
(declare-fun f1 () Bool)
(push 1)
(assert
  (let ((.def_10 (not f0)))
   (let ((.def_9 (not c0)))
    (let ((.def_11 (or .def_9 .def_10)))
     (let ((.def_7 (not f1)))
      (let ((.def_8 (or c0 .def_7)))
       (let ((.def_12 (and .def_8 .def_11)))
 .def_12
)))))))
(check-sat)
(pop 1)
(declare-fun f2 () Bool)
(declare-fun f3 () Bool)
(declare-fun f4 () Bool)
(declare-fun c1 () Bool)
(declare-fun E1 () Bool)
(assert
 (let ((.def_23 (not f2)))
  (let ((.def_20 (= c0 c1)))
   (let ((.def_22 (or E0 .def_20)))
    (let ((.def_24 (or .def_22 .def_23)))
     (let ((.def_18 (not f4)))
      (let ((.def_19 (or c1 .def_18)))
       (let ((.def_25 (and .def_19 .def_24)))
 .def_25
))))))))
(push 1)
(check-sat)
(assert (and f1 (not f1)))
(check-sat)
(pop 1)
(exit)
```