13th International Satisfiability Modulo Theories Competition (SMT-COMP 2018): Rules and Procedures

Matthias Heizmann
University of Freiburg
Germany
heizmann@informatik.uni-freiburg.de

Aina Niemetz
Stanford University
USA
niemetz@cs.stanford.edu

Giles Reger
University of Manchester
UK
giles.reger@manchester.ac.uk

Tjark Weber
Uppsala University
Sweden
tjark.weber@it.uu.se

This version revised 2018-4-17

Comments on this document should be emailed to the SMT-COMP mailing list (see below) or, if necessary, directly to the organizers.

1 Communication

Interested parties should subscribe to the SMT-COMP mailing list. 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.

- SMT-COMP mailing list: smt-comp@cs.nyu.edu
- Sign-up site for the mailing list: cs.nyu.edu/mailman/listinfo/smt-comp

Additional material will be made available at the competition web site, www.smtcomp.org or smtcomp.sourceforge.net/2018.

2 Important Dates

April 8 Deadline for new benchmark contributions.

May 15 Final versions of competition tools (e.g., benchmark scrambler) are made available. Benchmark libraries are frozen.

May 27 Deadline for first versions of solvers (for all tracks), including information about which tracks and divisions are being entered, and magic numbers for benchmark scrambling.

June 10 Deadline for final versions of solvers, including system descriptions.

June 11 Opening value of NYSE Composite Index used to complete random seed for benchmark scrambling.

July 12/13 SMT Workshop; end of competition, presentation of results.

3 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 [6, 13]. More information on the history and motivation for SMT-COMP can be found at the competition web site, `www.smtcomp.org`, and in reports on previous competitions ([2, 3, 4, 5, 8, 9, 10]).

SMT-COMP 2018 is part of the SMT Workshop 2018 (`http://smt-workshop.cs.uiowa.edu/2018/`), which is affiliated with IJCAR 2018 (`http://ijcar2018.org/`), part of FLoC 2018 (`http://www.floc2018.org/`). The SMT Workshop will include a block of time to present the results of the competition.

Accordingly, researchers are highly encouraged to submit both new benchmarks and new or improved solvers to raise the level of competition and advance the state-of-the-art in automated SMT problem solving.

SMT-COMP 2018 will have three tracks: the conventional main track, an application (i.e., incremental) track, and an unsat-core track.

Within each track there are multiple divisions, where each division uses benchmarks from a specific SMT-LIB logic (or group of logics). We will recognize winners as measured by number of benchmarks solved (taking into account the weighting detailed in Section 7); we will also recognize solvers based on additional criteria.

The rest of this document, revised from the previous version,[1] describes the rules and competition procedures for SMT-COMP 2018. The principal changes from the previous competition rules are the following:

[1] Earlier versions of this document include contributions from Clark Barrett, Roberto Bruttomesso, David Cok, Sylvain Conchon, David Déharbe, Morgan Deters, Alberto Griggio, Matthias Heizmann, Albert Oliveras, Giles Reger, Aaron Stump, and Tjark Weber.

- The organizers plan to recognize additional special contributions. These might include:
  - Best application track solver (awarded based on a competition-wide scoring for the application track)
  - Best unsat-core track solver (awarded based on a competition-wide scoring for the unsat-core track)
  - Best complementary solver (awarded to the solver that performs best when all benchmarks solved by the winning solver are removed)
  - Most unique solver (awarded to the solver that performs best on benchmarks solved by only one solver)
  - The Prophet (awarded to the solver that performs best on benchmarks with unknown status)
  - The 10 Second Wonder (awarded to the solver that would have won if the time limit was 10 seconds)

  The details are still being discussed and subject to change.

- In the unsat-core track, the unsatisfiability check for the returned core will be based on a simple majority vote. The case that a solver finds its own core satisfiable will no longer be treated specially. *Rationale:* This allows to simplify the post-processing of unsat-core track results considerably.

- In 2017, the competition featured experimental divisions for benchmarks that use algebraic datatypes. Provided the competition tools can be enhanced in time to support datatypes, these divisions may become regular (non-experimental) in 2018. *Rationale:* Algebraic datatypes are specified in the latest official release (Version 2.6) of the SMT-LIB standard.

- The competition may feature experimental divisions for benchmarks that use strings, and for benchmarks that use transcendental functions. *Rationale:* Corresponding theories and benchmarks are expected to be added to SMT-LIB in the near future.

4 Entrants

Solver submission. An entrant to SMT-COMP is an SMT solver submitted by its authors using the StarExec (`http://www.starexec.org`) service. The execution service 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 shared computer cluster. The execution service is provided free of charge, but it does require registration to create a login account. Registered users may then upload their own solvers to run, or may run public solvers already uploaded to the service. Information about how to configure and upload a solver is contained in the StarExec user guide, `https://wiki.uiowa.edu/display/stardev/User+Guide`.

For participation in SMT-COMP, a solver must be uploaded to StarExec and made publicly available. StarExec supports solver configurations; for clarity, *each submitted solver must have* *one configuration only*. Moreover, the organizers must be informed of the solver's presence *and the tracks and divisions in which it is participating* via the web form at

[TBA]

A submission must also include a 32-bit unsigned integer. These integer numbers, collected from all submissions, are used to seed the benchmark scrambler.

System description. As part of a submission, SMT-COMP entrants are strongly encouraged to provide a short (1–2 pages) description of the system. This should include a list of all authors of the system, their present institutional affiliations, and any appropriate acknowledgements. 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 should also include a URL for a web site for the submitted tool. System descriptions may be submitted after the solver deadline, but to be useful should be sent to the organizers before the competition ends. We intend to make system descriptions publicly available.

Multiple versions. The organizers' intent is to promote as wide a comparison among solvers and solver options as possible. However, if the number of solver submissions is too large for the computational resources available to the competition, the organizers reserve the right not to accept multiple versions of solvers from the same solver team.

Other solvers. The organizers reserve the right to include other solvers of interest (such as entrants in previous SMT competitions) in the competition, e.g., for comparison purposes.

Wrapper tools. A *wrapper tool* is defined as any solver which calls (or is in some other way based on) one or more SMT solvers not written by the author of the wrapper tool. The other solvers are called the *wrapped* solvers. A wrapper tool must explicitly acknowledge any solvers that it wraps. Its system description should make clear the technical innovations by which the wrapper tool expects to improve on the wrapped solvers.

Attendance. Submitters of an SMT-COMP entrant need not be physically present at the competition or the SMT Workshop to participate or win.

Deadlines

SMT-COMP entrants must be submitted via StarExec (solvers) *and* the above web form (accompanying information) until the end of **May 27, 2018** anywhere on earth. After this date no new entrants will be accepted. However, updates to existing entrants on StarExec will be accepted until the end of **June 10, 2018** anywhere on earth.

We strongly encourage participants to use this grace period *only* for the purpose of fixing any bugs that may be discovered, and not for adding new features, as there may be no opportunity to do extensive testing using StarExec after the initial deadline.

The solver versions that are present on StarExec at the conclusion of the grace period will be the ones used for the competition. 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 day after the final solver deadline.

These deadlines and procedures apply equally to all tracks of the competition.

5 Execution of Solvers

Solvers will be publicly evaluated in all tracks and divisions into which they have been entered. All results of the competition will be made public.

5.1 Logistics

Dates of competition. The bulk of the computation will take place during the weeks leading up to SMT 2018. Intermediate results will be regularly posted to the SMT-COMP website as the competition runs.

The organizers reserve the right to prioritize certain competition tracks or divisions to ensure their timely completion, and in exceptional circumstances to complete divisions after the SMT Workshop.

Input and output. In the main and unsat-core track, a participating solver must read a single benchmark script, whose filename is presented as the solver's first command-line argument. In the application track, a trace executor will send commands from a benchmark script to the solver's standard input channel.

The benchmark script is in the concrete syntax of the SMT-LIB format, version 2.6, though with a restricted set of commands. A benchmark script is a text file containing a sequence of SMT-LIB commands that satisfies the following requirements:

1. (a) In the main and unsat-core track, there may be a single **(set-option :print-success ...)** command. Note that `success` outputs are ignored by the post-processor used by the competition.[2]

   (b) In the application track, the **:print-success** option must not be disabled. The trace executor will send an initial **(set-option :print-success true)** command to the solver.

2. In the unsat-core track, there is a single **(set-option :produce-unsat-cores true)** command.

3. The (single) **set-logic** command setting the benchmark's logic is the first command after any **set-option** commands.

4. The script may contain any number of **set-info** commands.

5. The script may contain any number of **declare-sort** and **define-sort** commands. All sorts declared or defined with these commands must have zero arity.

6. The script may contain any number of **declare-fun** and **define-fun** commands.

7. If the benchmark's logic features algebraic datatypes, the script may contain any number of **declare-datatype(s)** commands.

[2] SMT-LIB 2.6 requires solvers 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 allows for submitting fully SMT-LIB 2.6 compliant solvers without the need for a wrapper script, while still allowing entrants of previous competitions to run without changes.

8. The script may contain any number of **assert** commands. All formulas in the script belong to the benchmark's logic, with any free symbols declared in the script.

9. (a) In the main and application track, named formulas are not used.

   (b) In the unsat core track, top-level assertions may be named.

10. (a) In the main and unsat-core track, there is exactly one **check-sat** command.

    (b) In the application track, there are one or more **check-sat** commands. There may also be zero or more **push 1** commands, and zero or more **pop 1** commands, consistent with the use of those commands in the SMT-LIB standard.

11. In the unsat-core track, the **check-sat** command (which is always issued in an unsatisfiable context) is followed by a single **get-unsat-core** command.

12. The script may optionally contain an **exit** command as its last command. In the application track, this command must not be omitted.

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

The SMT-LIB format specification is available from the "Standard" section of the SMT-LIB website [14]. Solvers will be given formulas only from the divisions into which they have been entered.

Time and memory limits. Each SMT-COMP solver will be executed on a dedicated processor of a competition machine, for each given benchmark, up to a fixed wall-clock time limit T. Each processor has 4 cores. Detailed machine specifications are available on the competition web site.

The time limit T is yet to be determined, but it is anticipated to be at most 20 minutes of wall-clock time per solver/benchmark pair.[3] 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.

The StarExec service also limits the memory consumption of the solver processes. We expect the memory limit per solver/benchmark pair to be on the order of 60 GB. The values of both the time limit and the memory limit are available to a solver process through environment variables. See the StarExec user guide for more information.

5.2 Main track

The main track competition will consist of selected benchmarks in each of the logic divisions. Each benchmark script will be presented to the solver as its first command-line argument. The solver is then expected to attempt to report on its standard output channel whether the formula is satisfiable (`sat`, in lowercase) or unsatisfiable (`unsat`). A solver may also report `unknown` to indicate that it cannot determine satisfiability of the formula.

The main track competition uses a StarExec post-processor (named "SMT-COMP 2018") to accumulate the results.

[3] The time limit may be adjusted once we know the number of competition entrants and eligible benchmarks.

Aborts and unparsable output. Any `success` outputs will be ignored. Solvers that exit before the time limit without reporting a result (e.g., due to exhausting memory or crashing) *and* do not produce output that includes `sat`, `unsat` or `unknown` will be considered to have aborted.

Persistent state. Solvers may create and write to files and directories during the course of an execution, but they must not read such files back during later executions. Each solver is executed with a temporary directory as its current working directory. Any generated files should be produced there (and not, say, in the system's `/tmp` directory). The StarExec system sets a limit on the amount of disk storage permitted—typically 20 GB. See the StarExec user guide for more information. The temporary directory is deleted after the job is complete. Solvers must not attempt to communicate with other machines, e.g., over the network.

5.3 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 framework and the solver: the framework 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 solvers in this track, we will set up a "simulation" of the aforementioned interaction. Each benchmark in the application track represents a realistic communication trace, containing multiple **check-sat** commands (possibly with corresponding **push 1** and **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" behaviors of SMT solvers;
- it records time and answers for each command;
- it guarantees a fair execution for all solvers by abstracting from any possible crash, misbehavior, etc. that might happen in the verification framework.

The trace executor terminates processing the benchmark script upon receiving an incorrect response from the solver.

The disk space and memory limits for the application track are the same as for the main track (see Section 5.2).

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 SMT-LIB benchmark script that satisfies the requirements for application track scripts given in Section 5.1.

Solvers must respond to each command sent by the trace executor with the answers defined in the SMT-LIB format specification, that is, with an answer of `sat`, `unsat`, or `unknown` for **check-sat** commands, and with a `success` answer for other commands.

5.4 Unsat-core track

The unsat-core track will evaluate the capability of solvers to generate unsatisfiable cores (for problems that are known to be unsatisfiable). Solvers will be measured by the smallness of the unsatisfiable core they return.

The SMT-LIB language accommodates this functionality by providing two features: the ability to name top-level (asserted) formulas, and the ability to request an unsatisfiable core after a **check-sat** command returns `unsat`. The unsatisfiable core that is returned must consist of a list of names of formulas, in the format prescribed by the SMT-LIB standard.

The result of a solver is considered erroneous if the response to the **check-sat** command is `sat`, or if the returned unsatisfiable core is not well-formed (e.g., contains names of formulas that have not been asserted before), or if the returned unsatisfiable core is not, in fact, unsatisfiable.

In order to perform this unsatisfiability check, the organizers will use a selection of SMT solvers that participate in the main track of this competition. For each division, the organizers will use only solvers that have been sound (i.e., not produced any erroneous result) in the main track for this division. The unsatisfiability of a produced unsatisfiable core is refuted if the number of checking solvers whose result is `sat` exceeds the number of checking solvers whose result is `unsat`. The time limit for checking unsatisfiable cores is yet to be determined, but is anticipated to be around 5 minutes of wall-clock time per solver.

Solvers must respond to each command in the benchmark script with the answers defined in the SMT-LIB format specification. In particular, solvers that respond `unknown` to the **check-sat** command must respond with an error to the following **get-unsat-core** command.

6 Benchmarks and Problem Divisions

Benchmark sources. Benchmarks for each division will be drawn from the SMT-LIB benchmark library. The main track will use a subset of all *non-incremental* benchmarks; the application track will use a subset of all *incremental* benchmarks. The unsat-core track will use unsatisfiable main track benchmarks, modified to use named top-level assertions.

New benchmarks. The deadline for submission of new benchmarks is **April 8, 2018**. The organizers, in collaboration with the SMT-LIB maintainers, will be checking and curating these until **May 15, 2018**. The SMT-LIB maintainers intend to make a new release of the benchmark library publicly available on or close to this date.

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.

Benchmark selection. The competition will use a large subset of SMT-LIB benchmarks. The benchmark pool is culled as follows:

1. *Remove inappropriate benchmarks.* The competition organizers may remove benchmarks that are deemed inappropriate or uninteresting for competition, or cut the size of certain

benchmark families to avoid their over-representation. 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.

2. *Remove incremental benchmarks whose first **check-sat** command has unknown status.* Incremental benchmarks may contain multiple **check-sat** commands, each with its own status. If an incremental benchmark contains a **check-sat** command whose status is unknown, only the prefix of the benchmark up to the preceding **check-sat** command is eligible for the application track of the competition; if the benchmark's first **check-sat** command has unknown status, the entire benchmark is ineligible.[4]

All remaining benchmarks are used for the competition. There will be no further selection of benchmarks, e.g., based on benchmark difficulty or benchmark category.

The set of benchmarks selected for the competition will be published when the competition begins.

Heats. Since the organizers at this point are unsure how long the set of benchmarks may take (which will depend also on the number of solvers submitted), the competition may be run in *heats*. For each track and division, the selected benchmarks may be randomly divided into a number of (possibly unequal-sized) heats. Heats will be run in order. If the organizers determine that there is adequate time, all heats will be used for the competition. Otherwise, incomplete heats will be ignored.

Benchmark scrambling. Benchmarks will be slightly scrambled before the competition, using a simple benchmark scrambler. The benchmark scrambler will be made 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 rule is considered cheating.

Pseudo-random numbers. Pseudo-random numbers used, e.g., for the creation of heats or the scrambling of benchmarks, will be generated using the standard C library function `random()`, seeded (using `srandom()`) with the sum, modulo 2^{30}, of the integer numbers provided in the system descriptions (see Section 4) 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 the first day the exchange is open on or after the date specified in the timeline (Section 2) 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.

7 Scoring

7.1 Competitive divisions

Scores will be computed for all solvers and divisions. However, winners will be declared only for *competitive* divisions. A division in a track is competitive if at least two substantially different

[4] It might be desirable to also include **check-sat** commands with unknown status in the application track (for the same reasons they are now included in the main track), but this would require substantial changes to the trace executor and scoring tools.

solvers (i.e., solvers from two different teams) were submitted. Although the organizers may enter other solvers for comparison purposes, only solvers that are explicitly submitted by their authors determine whether a division is competitive, and are eligible to be designated as winners.

7.2 Benchmark scoring

A solver's *raw score* for each benchmark is a quadruple $\langle e, n, w, c\rangle$, with $e \in \{0, 1\}$ the number of erroneous results (usually $e = 0$), $0 \leq n \leq N$ the number of correct results,[5] $w \in [0, T]$ the (real-valued) wall-clock time in seconds, and $c \in [0, 4T]$ the (real-valued) CPU time in seconds, measured across all cores and sub-processes, until the solver process terminates.

Main track. More specifically, for the main track, we have

- $e = 0$, $n = 0$ if the solver aborts without a response, or the result of the **check-sat** command is unknown,
- $e = 0$, $n = 1$ if the result of the **check-sat** command is sat or unsat, and the result either agrees with the benchmark status or the benchmark status is unknown,[6]
- $e = 1$, $n = 0$ if the result of the **check-sat** command is incorrect.

Note that a (correct or incorrect) response is taken into consideration even when the solver process terminates abnormally, or does not terminate within the time limit. Solvers should take care not to accidentally produce output that contains sat or unsat.

Application track. An application benchmark may contain multiple **check-sat** commands. Solvers may partially solve the benchmark before timing out. The benchmark is run by the trace executor, measuring the total time (summed over all individual commands) taken by the solver to respond to commands.[7] Most time will likely be spent in response to **check-sat** commands, but **assert**, **push** or **pop** commands might also entail a reasonable amount of processing. For the application track, we have

- $e = 1$, $n = 0$ if the solver returns an incorrect result for any **check-sat** command within the time limit,
- otherwise, $e = 0$, and n is the number of correct results for **check-sat** commands returned by the solver before the time limit is reached.

Unsat-core track. For the unsat-core track, we instead have $0 \leq n \leq A$, where A is the number of named top-level assertions in the benchmark, and

- $e = 0$, $n = 0$ if the solver aborts without a response, or the result of the **check-sat** command is unknown,

[5] Here, N is the number of **check-sat** commands in the benchmark. Recall that main track benchmarks have just one **check-sat** command; application track benchmarks may have multiple **check-sat** commands.

[6] If the benchmark status is unknown, we thus treat the solver's answer as correct. Disagreements between different solvers on benchmarks with unknown status are governed in Section 7.4.

[7] Times measured by StarExec may include time spent in the trace executor. We expect that this time will likely be insignificant compared to time spent in the solver, and nearly constant across solvers.

- $e = 1, n = 0$ if the result is erroneous according to Section 5.4,
- otherwise, $e = 0$, and n is the reduction in the number of formulas, i.e., $n = A$ minus the number of formula names in the reported unsatisfiable core.

7.3 Sequential performance (main track)

SMT-COMP has traditionally emphasized sequential performance (i.e., CPU time) over parallel performance (i.e., wall-clock time). StarExec measures both, and we intend to recognize both best sequential and best parallel solvers in all competitive main track divisions.

The raw score, as defined in Section 7.2, favors parallel solvers, which may utilize all available processor cores. To evaluate sequential performance, we derive a *sequential score* by imposing a (virtual) CPU time limit equal to the wall-clock time limit T. A solver result is taken into consideration for the sequential score only if the solver process terminates within this CPU time limit. More specifically, for a given raw score $\langle e, n, w, c\rangle$, the corresponding sequential score is defined as $\langle e_S, n_S, c_S\rangle$, where

- $e_S = 0$ and $n_S = 0$ if $c > T$, $e_S = e$ and $n_S = n$ otherwise,
- $c_S = \min\{c, T\}$.[8]

7.4 Division scoring

Main track: removal of disagreements. Before division scores are computed for the main track, benchmarks with unknown status are removed from the competition results if two (or more) solvers that are sound on benchmarks with known status disagree on their result. More specifically, a solver (including a solver that was entered into the competition by the organizers for comparison purposes) is *sound on benchmarks with known status* for a division if its raw score (Section 7.2) is of the form $\langle 0, n, w, c\rangle$ for each benchmark in the division, i.e., if it did not produce any erroneous results. Two solvers *disagree* on a benchmark if one of them reported `sat` and the other reported `unsat`. Only the remaining benchmarks are used in the following computation of division scores (but the organizers may report disagreements for informational purposes).

To compute a solver's score for a division, the solver's individual benchmark scores for all benchmarks in the division are first multiplied by a scalar weight that depends on the benchmark's family, and then summed component-wise.

For a given competition benchmark b, let $F_b \geq 1$ be the total number of benchmarks in b's benchmark family that were used in the competition track to which the division belongs (and not removed because of disagreements). We define the weight for benchmark b as $\alpha_b = (1 + \log_e F_b)/F_b$.[9] We define the *normalized weight* for benchmark b as $\alpha'_b = \alpha_b/(\sum_{b'} \alpha_{b'})$, where the sum is over all benchmarks in the division. Let N be the total number of benchmarks in the division.

[8] *Rationale:* Under this score, a solver should not benefit from using multiple processor cores. Conceptually, the sequential score should be (nearly) unchanged if the solver was run on a single-core processor, up to a time limit of T.

[9] See Section 7.5 for a motivating discussion of log scaling.

For main track and unsat-core track divisions, we will separately compute the weighted sum of all raw scores (Section 7.2)

$$\sum_b \alpha'_b \cdot \langle e_b \cdot N, n_b \cdot N, w_b, c_b \rangle$$

where the sum is over all benchmarks in the division to assess parallel performance, and the weighted sum of all sequential scores (Section 7.3) to assess sequential performance. For application track divisions, division scores will be based on raw scores only.[10]

Division scores are compared lexicographically:

- A weighted sum of raw scores $\langle e, n, w, c \rangle$ is better than $\langle e', n', w', c' \rangle$ iff $e < e'$ or ($e = e'$ and $n > n'$) or ($e = e'$ and $n = n'$ and $w < w'$) or ($e = e'$ and $n = n'$ and $w = w'$ and $c < c'$). That is, fewer errors takes precedence over more correct solutions, which takes precedence over less wall-clock time taken, which takes precedence over less CPU time taken.

- A weighted sum of sequential scores $\langle e_S, n_S, c_S \rangle$ is better than $\langle e'_S, n'_S, c'_S \rangle$ iff $e_S < e'_S$ or ($e_S = e'_S$ and $n_S > n'_S$) or ($e_S = e'_S$ and $n_S = n'_S$ and $c_S < c'_S$). That is, fewer errors takes precedence over more correct solutions, which takes precedence over less CPU time taken.

We will not make any comparisons between raw scores and sequential scores, as these are intended to measure fundamentally different performance characteristics.

7.5 Competition-wide scoring (main track)

We define a competition-wide metric for the main track, separately for parallel and sequential performance, as follows. Let N_i be the total number of benchmarks in division i that were used in the competition (and not removed because of disagreements), and let $\langle e_i, n_i, w_i, c_i \rangle$ be a solver's raw score for this division (Section 7.4). The solver's competition-wide raw score is

$$\sum_i (e_i == 0\, ?\, (n_i/N_i)^2 : -4) \log_e N_i$$

where the sum is over all competitive divisions into which the solver was entered.[11] The solver's competition-wide sequential score is computed from its sequential division scores (Section 7.4)

[10] Since application track benchmarks may be partially solved, defining a useful sequential score for the application track would require information not provided by the raw score, e.g., detailed timing information for each result.

[11] *Rationale:* This metric purposely emphasizes breadth of solver participation—a solver participating in many logics need not be the best in any one of them. The use of the square in the metric limits this somewhat—a solver still needs to do reasonably well compared to winners to be able to catch up by breadth of participation. The non-linear metric also gives added weight to completing close to all benchmarks in a division.

The constant penalty for errors reflects the fact that any error (in a particular division) renders a solver untrustworthy for that division. The value 4 balances the community's strong interest in correct (thoroughly tested) solvers against the risk of stifling innovation: entering a (possibly buggy) solver that can solve all benchmarks into a division has a positive expected value if the probability of a soundness bug is below $\frac{1}{1+4} = 20\,\%$.

The log scaling is a (somewhat arbitrary) means to adjust the scores for the wide variety of numbers of benchmarks in different divisions. It seems a reasonable compromise between linearly combining numbers of benchmarks, which would overweigh large divisions, and simply summing the fraction of benchmarks solved, which would overweigh small divisions.

The metric is also quite simple, and the metric for a solver is independent of the performance of other solvers. Time is omitted from the metric because it is only of third importance in the regular competition metric, and is difficult to compare across divisions.

according to the same formula. We will recognize the best three solvers according to these metrics.

7.6 Other recognitions

The organizers will also recognize the following contributions:

- *Best new entrant*. The best performing entrant from a new solver implementation team, as measured by the competition-wide metric.
- *Benchmarks*. Contributors of new benchmarks.

The organizers reserve the right to recognize other outstanding contributions that become apparent in the competition results.

8 Judging

The organizers reserve the right, with careful deliberation, to remove a benchmark from the competition results if it is determined that the benchmark is faulty (e.g., syntactically invalid in a way that affects some solvers but not others); and to clarify ambiguities in these rules that are discovered in the course of the competition. Authors of solver entrants may appeal to the organizers to request such decisions. Organizers that are affiliated with solver entrants will be recused from these decisions. The organizers' decisions are final.

9 Acknowledgments

SMT-COMP 2018 is organized under the direction of the SMT Steering Committee. The organizing team is

- Matthias Heizmann – University of Freiburg, Germany (co-organizer)
- Aina Niemetz – Stanford University, USA (co-organizer)
- Giles Reger – University of Manchester, UK (co-organizer)
- Tjark Weber – Uppsala University, Sweden (chair)

Tjark Weber is responsible for policy and procedure decisions, such as these rules, with input from the co-organizers. He is not associated with any group creating or submitting solvers.

Many others have contributed benchmarks, effort, and feedback. Clark Barrett and Pascal Fontaine are maintaining the SMT-LIB benchmark library. The competition uses the StarExec service, which is hosted at the University of Iowa. Aaron Stump is providing essential StarExec support.

Disclosure. Aina Niemetz is part of the developing teams of the SMT solvers Boolector [12] and CVC4 [1]. Matthias Heizmann is associated with the group producing the SMTInterpol [7] solver. Giles Reger is associated with the group producing the Vampire system [11].

References

[1] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

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

[3] Clark Barrett, Leonardo de Moura, and Aaron 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.

[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] Clark Barrett, Morgan Deters, Albert Oliveras, and Aaron Stump. Design and Results of the 4th Annual Satisfiability Modulo Theories Competition (SMT-COMP 2008). Technical Report TR2010-931, New York University, 2010.

[6] Daniel Le Berre and Laurent 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, 2003.

[7] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. SMTInterpol: An interpolating SMT solver. In Alastair Donaldson and David Parker, editors, *Model Checking Software*, pages 248–254, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[8] David R. Cok, David Déharbe, and Tjark Weber. The 2014 SMT Competition. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:207–242, 2014.

[9] David R. Cok, Alberto Griggio, Roberto Bruttomesso, and Morgan Deters. The 2012 SMT Competition. Available online at `http://smtcomp.sourceforge.net/2012/reports/SMTCOMP2012.pdf`.

[10] David R. Cok, Aaron Stump, and Tjark Weber. The 2013 Evaluation of SMT-COMP and SMT-LIB. *Journal of Automated Reasoning*, 55(1):61–90, 2015.

[11] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35, 2013. `https://vprover.github.io/`.

[12] Aina Niemetz, Mathias Preiner, and Armin Biere. Boolector 2.0 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:53–58, 2015.

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

[14] Silvio Ranise and Cesare Tinelli. The SMT-LIB web site. `http://www.smtlib.org`.