14th International Satisfiability Modulo Theories Competition (SMT-COMP 2019): Rules and Procedures

Liana Hadarean
Amazon
USA
hadarean@amazon.com

Antti Hyvarinen
Universita della Svizzera italiana
Switzerland
antti.hyvaerinen@usi.ch

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

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

This version revised 2019-5-13

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: http://cs.nyu.edu/mailman/listinfo/smt-comp

Additional material will be made available at the competition web site, http://www.smtcomp.org.

2 Important Dates

March 1 Deadline for new benchmark contributions.

May 1 Final versions of competition tools (e.g., benchmark scrambler) are made available. Benchmark libraries are frozen.

May 19 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 2 Deadline for final versions of solvers, including system descriptions.

June 3 Opening value of NYSE Composite Index used to compute random seed for competition tools.

July 7/8 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, http://www.smtcomp.org, and in reports on previous competitions ([2, 3, 4, 5, 7, 8, 9]).

SMT-COMP 2019 is part of the SMT Workshop 2019 (http://smt2019.galois.com/), which is affiliated with SAT 2019 (http://sat2019.tecnico.ulisboa.pt/). 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 2019 will have five tracks: the Single Query Track (previously: Main Track), the Incremental Track (previously: Application Track), the Unsat-Core Track, the (new) Industry-Challenge Track, and the (new) Model-Validation 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 2019. 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, Aina Niemetz, Albert Oliveras, Giles Reger, Aaron Stump, and Tjark Weber.

- **Mandatory System Descriptions.** SMT-COMP entrants are now required to provide a short (1–2 pages) description of the system. *Rationale:* The main incentive for this change is twofold. First, we want to improve transparency when submitted solvers are wrapper or derived tools according to the rules of the competition. Second, we want to encourage documenting technical improvements that lead to the current results.

- **Naming Convention for Derived Tools.** This year, we require that a derived tool (a tool based on and extending another SMT solver) should follow the *naming convention* [name of base solver]-[my solver name]. *Rationale:* Give credit where credit is due.

- **Renaming of Tracks.** This year, the track previously known as 'Main Track' will be renamed to *Single Query Track*, and the previous 'Application Track' will be renamed to *Incremental Track*. *Rationale:* We believe the current names are misleading, as the previous 'Main Track' also contains problems coming from applications. Additionally, having it called 'Main' de-emphasizes the importance of the other tracks and use cases of SMT.

- **Incremental Track.** In previous years, benchmarks were not eligible for the Incremental Track if their first **check-sat** command had unknown status. Similarly, the trace executor used to send commands from the benchmark to the solver's standard input channel stopped execution at the first **check-sat** command with unknown status. This year, benchmarks whose first **check-sat** command has unknown status are eligible for the Incremental Track and trace execution is only stopped before the solver completes a benchmark if the execution runs into the time limit. *Rationale:* The previous limitations were imposed by the trace executor, which now has been extended to support executing solvers on benchmarks beyond **check-sat** commands with unknown status.

- **New Model-Validation Track for QF_BV.** This year, we introduce a new experimental Model-Validation Track for the QF_BV logic. The benchmarks for this track will include all eligible *non-incremental* QF_BV benchmarks with known satisfiable status in SMT-LIB. Participating solvers are required to support the **get-model** command. *Rationale:* In many SMT applications, model generation is an essential feature. Previously, none of the SMT-COMP tracks required model generation. One of the challenges is that the model format is not consistent across different solvers. While imposing a standard over all logics is challenging, there are several logics (e.g., QF_BV) where it is straightforward. In the future we hope to expand this track to other logics as a way of pushing for model standardization. Since the QF_BV division was the one with the largest number of participants in previous years, we hope for a high number of participants in this track.

- **New Industry-Challenge Track.** This year, we introduce a new Industry-Challenge Track. This track will contain new challenging SMT-LIB benchmarks (with an emphasis on industrial applications) that are either unsolved or unsolved within some reasonable time limit, as indicated by the benchmark submitters. We will also include benchmarks nominated by the community as challenging and of interest. In this track, solvers will run with a significantly longer time limit. *Rationale:* Different application domains require different time limits. For example software verification traditionally requires much lower time limits, compared to hardware verification. To reward solvers optimized for different use cases, we propose

this new track and a new score for benchmarks solved within a very low time limit (see item "New Scores in the Single Query Track"). See Section 5.4 for the logics included in the Industry-Challenge Track for this year.

- **Time Limit.** The time limit per solver/benchmark pair is anticipated to be at most 40 minutes in the Single Query Track, Incremental Track, Unsat-Core Track and Model-Validation Track. For the Industry-Challenge Track, it is anticipated to be at most 720 (12 hours) minutes. *Rationale:* In 2017 and 2018, the time limit for the Main Track was reduced to 20 minutes (down from 40 minutes in earlier years) to cope with the inclusion of a large number of benchmarks with unknown status. This year, the reduced number of benchmarks selected per division allows us to increase the time limit back to 40 minutes for the Single Query Track. For the Industry-Challenge Track, 12 hours seem to be a reasonable compromise for the expected number of participants and benchmarks submitted to this track.

- **Benchmark Selection.** Since 2015, the competition evaluated all solvers on all eligible benchmarks in SMT-LIB. This year, we will use an alternative benchmark selection scheme that selects a subset of the eligible benchmarks. The benchmark selection will be random, but will guarantee inclusion of newly submitted benchmarks. The upper bound for the size of each division is not fixed, but depends on the size of the corresponding logic. Prior to this selection, we will remove all benchmarks in a division that were solved by all solvers (including non-competing solvers) in this division in under one second in last year's competition. *Rationale:* Evaluating solvers on all eligible benchmarks in SMT-LIB makes results more predictable and seems to be more of an evaluation than a competition. In the Main track last year, 78% of the 258,741 benchmarks were solved by all supported solvers within the time limit (71% within 1 second). In 7 (out of 46) logics, over 99% of benchmarks were solved by all solvers. Removing 'easy' (or at least 'unsurprising') benchmarks attempts to shift the focus towards challenging benchmarks, in particular since we now use an alternative benchmark selection scheme. It can be argued that the size of a logic in SMT-LIB can be seen as an indicator of its relevance. We thus do not use a fixed upper bound for the size of a division in order to reflect the suggested importance of a division.

- **Scoring Scheme.** This year, we will abandon the weighted scoring scheme that was introduced for the Main Track and Unsat-Core Track in 2016. We will fall back to the scoring scheme based on the number of solved instances that was last used in 2015. *Rationale:* Since 2016, the competition has used a scoring scheme based on benchmark weights in order to de-emphasize large benchmark families. The scoring scheme is fairly complicated and not necessarily intuitive. It further makes comparing results in papers with results from the competition more difficult. A recent analysis of competition data from 2015-2018 for a report on the competition of these years (currently under submission to JSAT) suggests that benchmark families do not have a significant impact on the (weighted) scores When determining winners for each division using the scoring scheme of 2015, the winners for only 7 divisions (out of 139) over all three years would have changed. While producing these results we further noticed that the description of what constitutes a benchmark family (as stated in the rules documents) had been incorrectly interpreted by the scoring scripts in 2016-2018. After fixing this misinterpretation, only one single division winner changes - in

2017 AUFNIRA should have been won by CVC4 and not Vampire.

- **New Scores in the Single Query Track.** This year, additionally to the separate scores given for sequential and parallel performance, we will reward three new scores in the Single Query Track. The *24-second score* will reward solving performance within a time limit of 24 seconds (wall clock time), the *sat score* will reward performance on satisfiable instances, and the *unsat score* will reward performance on unsatisfiable instances. *Rationale:* Different application domains of SMT typically impose a wide range of time limits (from hours to seconds). Previous time limits used in the competition were the same for all benchmarks and thus agnostic to the application domain of a benchmark. The new Industry-Challenge Track and the new 24-second score try to address use cases on both extreme ends of the spectrum. Further, in many cases an SMT application produces either mainly satisfiable or mainly unsatisfiable queries to the SMT solver. The new sat and unsat scores are intended to reward solvers that implement specialized techniques and optimizations for either case.

- **Do Not Run Non-Competitive Divisions.** This year, we will not run non-competitive divisions. *Rationale:* Evaluating solvers in non-competitive divisions is more in the spirit of an evaluation than a competition.

- **Experimental Strings Division.** The competition will feature experimental divisions for benchmarks that use strings. Participating solvers must implement the semantics of the current SMT-LIB draft for the theory of unicode strings (http://smtlib.cs.uiowa.edu/theories-UnicodeStrings.shtml). *Rationale:* Corresponding theories and benchmarks are expected to be added to SMT-LIB in the near future.

- **Competition-Wide Recognitions.** This year, we will replace the previous notion of competition-wide scoring as introduced for the Main Track in 2014 (and improved over the years) with recognitions for all tracks that do not directly compare divisions. *Rationale:* Competition-wide scoring was introduced as a metric to compare solver performance across all divisions. By definition, this metric is biased towards solvers that enter a large number of divisions.

4 Entrants

SMT Solver. A Satisfiability Modulo Theories (SMT) solver that can enter SMT-COMP is a tool that can determine the (un)satisfiability of benchmarks from the SMT-LIB benchmark library (http://www.smt-lib.org/benchmarks.shtml).

Wrapper Tool. A *wrapper tool* is defined as any solver that calls one or more other SMT solvers (the *wrapped solvers*). Its system description **must** explicitly acknowledge and state the **exact** version of any solvers that it wraps. It *should* further make clear technical innovations by which the wrapper tool expects to improve on the wrapped solvers.

Derived Tool. A *derived tool* is defined as any solver that is *based on and extends* another SMT solver (the *base solver*). Its system description **must** explicitly acknowledge the solver it is based on and extends. It *should* further make clear technical innovations by which the derived tool

expects to improve on the original solver. A derived tool should follow the *naming convention* [name of base solver]-[my solver name].

SMT Solver Submission. An entrant to SMT-COMP is a solver submitted by its authors using the StarExec (http://www.starexec.org) service.

Solver execution. The StarExec 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 requires registration to create a login account. Registered users may then upload 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.

Participation in the Competition. 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

https://forms.gle/y8xB4C8TB2WkdKPx9

A submission **must** also include a *system description* (see below) and a *32-bit unsigned integer*. These integer numbers, collected from all submissions, are used to seed competition tools.

System description. As part of the submission, SMT-COMP entrants are **required** to provide a short (1-2 pages) description of the system, which **must** explicitly acknowledge any solver it wraps or is based on in case of a *wrapper* or *derived* tool (see above). In case of a *wrapper* tool, it **must** also explicitly state the exact version of each wrapped solver. A system description *should* further include the following information (unless there is a good reason otherwise):

- a list of all authors of the system and their present institutional affiliations,
- the basic SMT solving approach employed,
- details of any non-standard algorithmic techniques as well as references to relevant literature (by the authors or others),
- in case of a *wrapper* or *derived tool*: details of technical innovations by which a wrapper or derived tool expects to improve on the wrapped solvers or base solver
- appropriate acknowledgement of tools other than SMT solvers called by the system (e.g., SAT solvers) that are not written by the authors of the submitted solver, and
- a link to a website for the submitted tool.

System descriptions **must** be submitted **until the final solver deadline**, and will be made publicly available on the competition website. Organizers will check that they contain sufficient information and may withdraw a system if its description is not sufficiently updated upon request

Multiple versions. The intent of the organizers 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.

Attendance. Submitters of an SMT-COMP entrant are not required (but encouraged) to 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 19, 2019** 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 2, 2019** 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. Solvers will be made publicly available after the competition and it is a minimum licence requirement that (i) solvers can be distributed in this way, and (ii) all submitted solvers may be freely used for academic evaluation purposes.

5.1 Logistics

Dates of Competition. The bulk of the computation will take place during the weeks leading up to SMT 2019. 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.

Competition Website. The competition website (www.smtcomp.org) will be used as the main form of communication for the competition. The website will be used to post updates, link to these rules and other relevant information (e.g. the benchmarks), and to announce the results. We also use the website to archive previous competitions. Starting from 2019 we will include the submitted solvers in this archive to allow reproduction of the competition results in the future.

Tools. The competition uses a number of tools/scripts to run the competition. In the following, we briefly describe these tools. Unless stated otherwise, these tools are found at https://github.com/SMT-COMP/smt-comp/tree/master/tools.

- **Benchmark Selection.** We use a script to implement the benchmark selection policy described on page 14. It takes a seed for the random benchmark selection. The same seed is used for all tools requiring randomisation.

- **Scrambler.** This tool is used to scramble benchmarks during the competition to ensure that tools do not rely on syntactic features to identify benchmarks. The scrambler can be found at https://github.com/SMT-COMP/scrambler.

- **Trace Executor.** This tool is used in the Incremental Track to emulate an on-line interaction between an SMT solver and a client application and is available at https://github.com/SMT-COMP/trace-executor

- **Post-Processors.** These are used by StarExec to translate the output of tools to the format required for scoring. All post-processors (per track) are available at https://github.com/SMT-COMP/postprocessors.

- **Scoring.** We use a script to implement the scoring computation described on in Section 7. It also includes the scoring computations used in previous competitions (since 2015).

Input and Output. In the *Incremental Track*, the *trace executor* will send commands from an (incremental) benchmark file to the standard input channel of the solver. In *all other tracks*, a participating solver must read a *single* benchmark file, whose filename is presented as the first command-line argument of the solver.

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

- **(set-logic ...)**
  A (single) **set-logic** command is the *first* command after any **set-option** commands.

- **(set-info ...)**
  A benchmark file may contain any number of **set-info** commands.

- **(set-option :print-success ...)**
  (a) In the *Single Query Track*, *Industry-Challenge Track*, *Model-Validation Track* and *Unsat-Core Track*, there may be a single **set-option** command. Note that `success` outputs are ignored by the post-processors used by the competition.[2]
  (b) In the *Incremental 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.
  (c) In the *Model-Validation Track*, a benchmark file contains a single **(set-option :produce-models true)** command.

[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.

(d) In the *Unsat-Core Track*, a benchmark file contains a single **(set-option :produce-unsat-cores true)** command.

- **(declare-sort ...)**
  A benchmark file may contain any number of **declare-sort** and **define-sort** commands. All sorts declared or defined with these commands must have zero arity.

- **(declare-fun ...)** and **(define-fun ...)**
  A benchmark file may contain any number of **declare-fun** and **define-fun** commands.

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

- **(assert ...)**
  A benchmark file may contain any number of **assert** commands. All formulas in the file belong in the declared logic, with any free symbols declared in the file.

- **:named**
  (a) In *all* tracks *except* the Unsat-Core Track, named terms (i.e., terms with the **:named** attribute) are *not* used.
  (b) In the *Unsat-Core Track*, top-level assertions may be named.

- **(check-sat)**
  (a) In *all* tracks *except* the Incremental Track, there is *exactly one* **check-sat** command.
  (b) In the *Incremental 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.

- **(get-unsat-core)**
  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.

- **(get-model)**
  In the *Model-Validation Track*, the **check-sat** command (which is always issued in a satisfiable context) is followed by a single **get-model** command.

- **(exit)**
  It may *optionally* contain an **exit** command as its last command. In the *Incremental Track*, this command must not be omitted.

- **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. The individual track descriptions on pages 10-12 specify the time limit for each track. Each processor has 4 cores. Detailed machine specifications are available on the competition web site.

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.

Aborts and Unparsable Output. In all tracks except the Incremental Track, 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`, `unknown` or other track specific output as specified in the individual track sections e.g. unsat cores or models, 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.

5.2 Single Query Track (Previously: Main Track)

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

Benchmark Selection. See page 14.

Time Limit. This track will use a wall-clock time limit of 40 minutes per solver/benchmark pair.

Post-Processor. This track will use https://github.com/SMT-COMP/postprocessors/tree/master/single-query-track/process as a post-processor to validate and accumulate the results.

5.3 Incremental Track (Previously: Application Track)

The incremental 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 represents a realistic communication trace, contain-

ing multiple **check-sat** commands (possibly with corresponding **push 1** and **pop 1** commands). It is parsed by a (publicly available) *trace executor*, which serves the following purposes:

- simulating online interaction by sending single queries to the SMT solver (through stdin),
- preventing "look-ahead" behaviors of SMT solvers,
- recording time and answers for each command,
- guaranteeing a fair execution for all solvers by abstracting from any possible crash, misbehavior, etc. that might happen in the verification framework.

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 incremental 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.

Benchmark Selection. See page 14.

Time Limit. This track will use a wall-clock time limit of 40 minutes per solver/benchmark pair.

Trace Executor. This track will use the trace executor to execute a solver on an incremental benchmark file.

Post-Processor. This track will use https://github.com/SMT-COMP/postprocessors/tree/master/incremental-track/process as a post-processor to validate and accumulate the results.

5.4 Industry-Challenge Track

The Industry-Challenge Track will include both non-incremental and incremental benchmarks. It will follow the same rules as the Single Query Track and Incremental Track, respectively, with two exceptions: benchmark selection and the time limit.

Benchmark Selection. This track will run on challenging industrial benchmarks provided by the community. This year, the Industry-Challenge Track will include the complete set of provided benchmarks dedicated to this track, which consist of both incremental and single query benchmarks in the logics QF_BV, QF_ABV and QF_AUFBV. The complete list of benchmarks along with instructions on how to access them will be provided on the SMT-COMP website as soon as the benchmark library is released.

Time Limit. This track will use a wall-clock time limit of 12 hours per solver/benchmark pair.

Post-Processor. This track will use the post-processors from the Single Query Track (for non-incremental benchmarks) and the Incremental Track (for incremental benchmarks) to accumulate the results.

5.5 Unsat-Core Track

The Unsat-Core Track will evaluate the capability of solvers to generate unsatisfiable cores. Performance of solvers will be measured by correctness and size of the unsatisfiable core they provide.

Benchmark Selection. This track will run on a selection of non-incremental benchmarks with status `unsat` (as described on page 14), modified to use named top-level assertions of the form **(assert (! t :named f))**.

Input/Output. The SMT-LIB language provides a command **(get-unsat-core)**, which asks a solver to identify an unsatisfiable core after a **check-sat** command returns `unsat`. This unsat core must consist of a list of all named top-level assertions in the format prescribed by the SMT-LIB standard. 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.

Result. 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.

Validation. The organizers will use a selection of SMT solvers (the *validation solvers*) that participate in the Single Query Track of this competition in order to validate if a given unsat core is indeed unsatisfiable. For each division, the organizers will use only solvers that have been sound (i.e., they did not produce any erroneous result) in the Single Query Track for this division. The unsatisfiability of an unsat core is refuted if the number of validation solvers whose result is `sat` exceeds the number of checking solvers whose result is `unsat`.

Time Limit. This track will use a wall-clock time limit of 40 minutes per solver/benchmark pair. 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.

Post-Processor. This track will use https://github.com/SMT-COMP/postprocessors/tree/master/unsat-core-track/process as a post-processor to validate and accumulate the results.

5.6 Model-Validation Track (experimental)

The Model-Validation Track will evaluate the capability of solvers to produce models for satisfiable problems. Performance of solvers will be measured by correctness and well-formedness of the model they provide.

Benchmark Selection. This experimental track only has one division, QF_BV. It will run on all selection of non-incremental benchmarks with status `sat` from logic QF_BV (as described on page 14).

Input/Output. The SMT-LIB language provides a command **(get-model)** to request a satisfying model after a **check-sat** command returns `sat`. This model must consist of definitions specifying all and only the current user-declared function symbols, in the format prescribed by the SMT-LIB standard.

Result. The result of a solver is considered erroneous if the response to the **checks-sat** command is `unsat`, if the returned model is not well-formed (e.g. does not provide a definition for all the user-declared function symbols), or if the returned model does not satisfy the benchmark.

Validation. In order to check that the model satisfies the benchmark, the organizers will use the model validating tool available at https://github.com/SMT-COMP/postprocessors/tree/master/model-validation-track. The tool will output VALID for full satisfying models, and INVALID for partial models, malformed models, models that do not satisfy the benchmark or if the solver returns `unsat`. If the solver provides no output or returns `unknown` the model validating tool will output UNKNOWN.

Time Limit. This track will use a wall-clock time limit of 40 minutes per solver/benchmark pair. The time limit for checking the satisfying assignment is yet to be determined, but is anticipated to be around 5 minutes of wall-clock time per solver.

Post-Processor. This track will use https://github.com/SMT-COMP/postprocessors/tree/master/model-validation-track/process as a post-processor to validate and accumulate the results.

6 Benchmarks and Problem Divisions

Divisions. Within each track there are multiple divisions, and each division selects benchmarks from a specific SMT-LIB logic in the SMT-LIB benchmark library.

Competitive Divisions. A division in a track is competitive if at least two substantially different 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. We will **not** run *non-competitive* divisions.

Benchmark sources. Benchmarks for each division will be drawn from the SMT-LIB benchmark library. The Single Query Track will use a subset of all *non-incremental* benchmarks and the Incremental Track will use a subset of all *incremental* benchmarks. The Industry-Challenge Track will use all incremental and non-incremental benchmarks dedicated to this track by their submitters. The Unsat-Core Track will use a selection of non-incremental benchmarks with status `unsat` and more than one top-level assertion, modified to use named top-level assertions. The Model-Validation Track will use a selection of non-incremental benchmarks with status `sat` from logic QF_BV.

New benchmarks. The deadline for submission of new benchmarks is **March 1, 2019**. The organizers, in collaboration with the SMT-LIB maintainers, will be checking and curating these until **May 1, 2019**. The SMT-LIB maintainers intend to make a new release of the benchmark library publicly available on or close to this date.

Benchmark demographics. The set of all SMT-LIB benchmarks in a given division can be naturally partitioned to sets containing benchmarks that are similar from the user community perspective. Such benchmarks could all come from the same application domain, be generated by the same tool, or have some other obvious common identity. The organizers try to identify a meaningful partitioning based on the directory hierarchy in SMT-LIB. In many cases the hierarchy consists of the

top-level directories each corresponding to a submitter, who has further imposed a hierarchy on the benchmarks. The organizers believe that the submitters have the best information on the common identity of their benchmarks and therefore partition each division based on the bottom-level directory imposed by each submitter. These partitions are referred to as *families*.

Benchmark selection. The competition will use a large subset of SMT-LIB benchmarks, with some guarantees on including new benchmarks. In **all** tracks **except** the Industry-Challenge Track, the following selection process will be used.

1. *Remove inappropriate benchmarks.* The 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 easy benchmarks.* The organizers will remove all benchmarks that were solved by all solvers (including non-competitive solvers) in less than one second in 2018.

3. *Unsat-Core Track.* In addition, for the Unsat-Core Track, all benchmarks with a single assertion will be removed.

4. *Cap the number of instances in a division.* The organizers will limit the number of benchmarks in a division based on the size of the corresponding logic in SMT-LIB as follows:
   (a) If a logic contains less than 300 instances, all instances will selected.
   (b) If a logic contains between 300 and 600 instances, a subset of 300 instances from the set will be selected.
   (c) If a logic contains more than 600 instances, 50% of the benchmarks will be selected.

The selection process in cases 4b and 4c above will guarantee the inclusion of new benchmarks by first picking randomly one benchmark from each new benchmark family. The rest of the benchmarks will be chosen randomly from the remaining benchmarks using a uniform distribution. The benchmark selection script will be publicly available at https://github.com/SMT-COMP/smt-comp/tree/master/tools and will use the same random seed as the rest of the competition. 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 available at https://github.com/SMT-COMP/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 Benchmark scoring

The **parallel benchmark score** of a solver is a quadruple $\langle e, n, w, c \rangle$, with

- $e \in \{0, 1\}$ number of erroneous results (usually $e = 0$)
- $0 \leq n \leq N$ number of correct results (resp. *reduction* for the Unsat-Core Track)
- $w \in [0, T]$ wall-clock time in seconds (real-valued)
- $c \in [0, mT]$ CPU time in seconds (real-valued)

Error Score (e). For the Single Query Track, Incremental Track and Industry-Challenge Track, e is the number of returned statuses that disagree with the given expected status (as described above, disagreements on benchmarks with unknown status lead to the benchmark being disregarded). For the Unsat-Core Track, e includes, in addition, the number of returned unsat cores that are ill-formed or are not, in fact, unsatisfiable (as validated by a selection of other solvers selected by organizers). For the Model-Validation Track, e includes, in addition, the number of returned models that are ill-formed or not full satisfiable models.

Correctly Solved Score (n). For the Single Query Track, Incremental Track, Industry-Challenge Track and Model-Validation Track, N is defined as the number of **check-sat** commands, and n is defined as the number of correct results. For the Unsat-Core Track, N is defined as the number of named top-level assertions, and n is defined as the *reduction*, i.e., the difference between N and the size of the unsat core.

Wall-Clock Time Score (w). The (real-valued) wall-clock time in seconds, until time limit T or the solver process terminates.

CPU Time Score (c). The (real-valued) CPU time in seconds, measured across all m cores until time limit mT is reached or the solver process terminates.

7.1.1 Sequential Benchmark Score

The parallel score as defined above 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 parallel performance $\langle e, n, w, c\rangle$, the corresponding sequential performance is defined as $\langle e_S, n_S, c_S\rangle$, where

- $e_S = 0$ and $n_S = 0$ if $c > T$, and $e_S = e$ and $n_S = n$ otherwise,
- $c_S = \min\{c, T\}$.[3]

7.1.2 Single Query Track and Industry-Challenge Track

For the Single Query Track and Industry-Challenge Track, the error score e and the correctly solved score n are defined as

- $e = 0$ and $n = 0$ if the solver
  - aborts without a response, or
  - the result of the **check-sat** command is `unknown`,
- $e = 0$ and $n = 1$ if the result of the **check-sat** command is `sat` or `unsat` and either
  - agrees with the benchmark status,
  - or the benchmark status is unknown,[4]
- $e = 1$ and $n = 0$ 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`.

7.1.3 Incremental 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.[5] 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 Incremental Track, we have

- $e = 1$ and $n = 0$ 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.

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

[4] 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 7.2.

[5] 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.

7.1.4 Unsat-Core Track

For the Unsat-Core Track, the error score e and the correctly solved score n are defined as

- $e = 0$ and $n = 0$ if the solver
  - aborts without a response, or
  - the result of the **check-sat** command is `unknown`,
- $e = 1$and $n = 0$ if the result is erroneous according to Section 5.5,
- otherwise, $e = 0$ and n is the *reduction* in the number of formulas, i.e., $n = N$ minus the number of formula names in the reported unsatisfiable core.

7.1.5 Model-Validation Track

For the Model-Validation Track, the error score e and the correctly solved score n are defined as

- $e = 0$ and $n = 0$ if the solver
  - aborts without a response, or
  - the result of the **check-sat** command is `unknown`,
- $e = 1$ and $n = 0$ if the result is invalid according to the output of the model validating tool described in Section 5.6,
- otherwise, $e = 0$ and $n = 1$.

7.2 Division scoring

For each track and division, we compute a division score based on the parallel performance of a solver (the *parallel division score*). For the Single Query Track, Industry-Challenge Track, Unsat-Core Track and Model-Validation Track we also compute a division score based on the sequential performance of a solver (the *sequential division score*). Additionally, for the Single Query Track, we further determine three additional scores based on parallel performance: The *24-second score* will reward solving performance within a time limit of 24 seconds (wall clock time), the *sat score* will reward (parallel) performance on satisfiable instances, and the *unsat score* will reward (parallel) performance on unsatisfiable instances.

Sound Solver. A solver is *sound* on benchmarks with *known status* for a division if its parallel performance (Section 7.1) 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.

Disagreeing Solvers. Two solvers *disagree* on a benchmark if one of them reported `sat` and the other reported `unsat`.

Removal of Disagreements. Before division scores are computed for the Single Query Track and Industry-Challenge 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. Only the remaining benchmarks are used in the following computation of division scores (but the organizers *will report disagreements* for informational purposes).

7.2.1 Parallel Score

The parallel score for a division is computed for *all* tracks. It is defined for a participating solver in a division with M benchmarks as the sum of all the individual parallel benchmark scores:

$$\sum_{b \in M} \langle e_b, n_b, w_b, c_b \rangle$$

.

A parallel division score $\langle e, n, w, c \rangle$ is better than a parallel division score $\langle e', n', w', c' \rangle$ 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.

7.2.2 Sequential Score

The sequential score for a division is computed for *all* tracks *except* the Incremental Track and incremental benchmarks in the Industry-Challenge Track[6]. It is defined for a participating solver in a division with M benchmarks as the sum of all the individual sequential benchmark scores:

$$\sum_{b \in M} \langle e_{S_b}, n_{S_b}, w_{S_b}, c_{S_b} \rangle$$

.

A sequential division score $\langle e_S, n_S, c_S \rangle$ is better than a sequential division score $\langle e'_S, n'_S, c'_S \rangle$ 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 parallel and sequential performances, as these are intended to measure fundamentally different performance characteristics.

7.2.3 24-Seconds Score (Single Query Track)

The 24-seconds score for a division is computed for the Single Query Track as the parallel division score with a wall-clock time limit T of 24 seconds.

7.2.4 Sat Score (Single Query Track)

The sat score for a division is computed for the Single Query Track as the parallel division score when only satisfiable instances are considered.

7.2.5 Unsat Score (Single Query Track)

The unsat score for a division is computed for the Single Query Track as the parallel division score when only unsatisfiable instances are considered.

[6]Since incremental track benchmarks may be partially solved, defining a useful sequential performance for the incremental track would require information not provided by the parallel performance, e.g., detailed timing information for each result.

7.3 Competition-Wide Recognitions

In 2014 the SMT competition introduced a competition-wide scoring to allow it to award medals in the FLoC Olympic Games and has been awarded each year since. This scoring purposefully emphasized the breadth of solver participation by summing up a score for each (competitive) division a solver competed in. Whilst this rationale is reasonable, we observed that this score had become dictated by the number of divisions being entered by a solver.

This year we will replace the competition-wide score with two new *rankings* that select one solver per division and then rank those solvers. The rationale here is to take the focus away from the number of divisions entered and focus on measures that make sense to use to compare different divisions.

7.3.1 Biggest Lead Ranking

This ranking aims to select the solver that *won by the most* in some competitive division. The winners of each division are ranked by the distance between them and the next competitive solver in that division.

Let n_i^D be the correctness score of the ith solver (for a given scoring system e.g. number of correct results or reduction) in division D. The rank of division D is given as

$$\frac{n_1^D + 1}{n_2^D + 1}$$

The *biggest lead winner* is the winner of the division with the highest (largest) rank. This can be computed per scoring system.

7.3.2 Largest Contribution Ranking

This ranking aims to select the solver that *uniquely contributed* the most in some division, or to put another way, the solver that would be most missed. This is achieved by computing a solver's contribution to the *virtual best solver* for a division.

Let $\langle e_b^s, n_b^s, w_b^s, c_b^s \rangle$ be the parallel benchmark score for benchmark b and solver s (for a given scoring system e.g. n is either number of correct results or reduction). The virtual best solver score for a division D using solvers S is given as

$$vbss(D, S) = \sum_{b \in D} \mathsf{min}\{n_b^s \times c_b^s \mid s \in S \ and \ n_b^s > 0\}$$

where the minimum of an empty set is 0 (e.g. no contribution if a benchmark is unsolved). In other words, for the single query track, $vbss(D, S)$ is the smallest amount of time taken to solve all benchmarks solved in division D using solvers in S.

Let S be the set of competitive solvers competing in division D. The rank of solver $s \in S$ in division D is then

$$1 - \frac{vbss(D, S - s)}{vbss(D, S)}$$

e.g. the difference in virtual best solver score when removing s from the computation. This will be a number between 0 and 1 with 0 indicating that s made no impact on the *vbss* and 1 indicating

that s is the only solver that solved anything in the division. The *largest contribution winner* is the solver across all divisions with the highest (largest) rank. Again, this can be computed per scoring system.

7.4 Other Recognitions

The organizers will also recognize the following contributions:

- *New entrants*. All new entrants (to be interpreted by the organisers, but broadly a significantly new tool that has not competed in the competition before) that beat an existing solver in some division will be awarded special commendations.
- *Benchmarks*. Contributors of new benchmarks used in the competition will receive a special mention.

These recognitions will be announced at the SMT workshop and published on the competition website. 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 2019 is organized under the direction of the SMT Steering Committee. The organizing team is

- Liana Hadarean – Amazon, USA (co-organizer)
- Antti Hyvarinen – Universita della Svizzera italiana, Switzerland (co-organizer)
- Aina Niemetz – Stanford University, USA (co-chair)
- Giles Reger – University of Manchester, UK (co-chair)

The competition chairs are responsible for policy and procedure decisions, such as these rules, with input from the co-organizers.

Many others have contributed benchmarks, effort, and feedback. Clark Barrett, Pascal Fontaine, Aina Niemetz and Mathias Preiner 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. Liana Hadarean was part of the developing team of the SMT solver CVC4 [1] and is currently not associated with any group creating or submitting solvers. Antti Hyvarinen is part of the developing team of the SMT solver OpenSMT [10]. Aina Niemetz is part of the developing teams of the SMT solvers Boolector [12] and CVC4 [1]. 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] David R. Cok, David Déharbe, and Tjark Weber. The 2014 SMT Competition. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:207–242, 2014.

[8] 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.

[9] 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.

[10] Antti E. J. Hyvärinen, Matteo Marescotti, Leonardo Alt, and Natasha Sharygina. Opensmt2: An SMT solver for multi-core and cloud computing. In Nadia Creignou and Daniel Le Berre, editors, *SAT 2016*, volume 9710 of *Lecture Notes in Computer Science*, pages 547–553, 2016.

[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.