

19th International Satisfiability Modulo Theories Competition (SMT-COMP 2024): Rules and Procedures

François Bobot
CEA List
France

<https://github.com/bobot>

Martin Bromberger
Max Planck Institute for Informatics
Saarland Informatics Campus
Germany

mbromber@mpi-inf.mpg.de

Martin Jonáš
Masaryk University
Czechia

martin.jonas@mail.muni.cz

This version revised 2024-5-10

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

April 13 Deadline for new benchmark contributions.

May 11 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 8 Deadline for final versions of solvers, including system descriptions.

June 10 Opening value of NYSE Composite Index used to compute random seed for competition tools.

July 22-23 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 [5, 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 ([1, 2, 3, 4, 9, 10, 11]).

SMT-COMP 2024 is part of the SMT Workshop 2024 (<http://smt-workshop.cs.uiowa.edu/2024/>), which is affiliated with CAV 2024 (<https://i-cav.org/2024/>). 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 2024 will have seven tracks: the Single Query Track (before 2019: Main Track), the Incremental Track (before 2019: Application Track), the Unsat-Core Track, the Model-Validation Track, and the Parallel Track and the Cloud Track (both sponsored by Amazon Web Services). Within each track there are multiple divisions, where each division uses benchmarks from a specific group of SMT-LIB logics. We will recognize winners in all tracks. They will be determined by the 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 2024.

As in previous years, we have revised the rules slightly. Most changes are due to our move from StarExec to BenchExec. The principal changes from the previous competition rules are the following:

¹Earlier versions of this document include contributions from Clark Barrett, Roberto Bruttomesso, David Cok, Sylvain Conchon, David Déharbe, Morgan Deters, Alberto Griggio, Liana Hadarean, Matthias Heizmann, Antti Hyvarinen, Jochen Hoenicke, Aina Niemetz, Albert Oliveras, Giles Reger, Aaron Stump, and Tjark Weber.

- **Move from StarExec to BenchExec.** The single query, incremental, unsat-core, and model-validation will no longer run on the StarExec platform hosted by the University of Iowa. Starting this year, we will run these tracks on the BenchExec cluster (<https://github.com/sosy-lab/benchexec>) owned by LMU’s Software and Computational Systems Lab (SoSy-Lab <https://www.sosy-lab.org/>). *Rationale:* The StarExec cluster is in the process of being decommissioned. We picked BenchExec as the new platform because it is host to similar competitions and the experience collected there is a big help in our migration process. Moreover, it is in academic hands, so less likely to change conditions on short notice.
- **Submission process.** We changed the submission process from a Google form to a push request to the SMT-COMP GitHub repository (<https://github.com/SMT-COMP/smt-comp.github.io/tree/master/submissions>). Detailed instructions can be found in the README file there. Moreover, participants are now required to upload their final solver versions as a Zenodo (<https://zenodo.org/>) submission that must contain a docker file that compiles the solver. *Rationale:* The change from Google form to GitHub push request makes it easier for us to automatically create scripts for BenchExec. The submission of the final solver versions to Zenodo makes it more likely that the competition results stay replicable.
- **Time limits.** The time limits for the cloud and parallel tracks are reduced to 2 minutes per problem. *Rationale:* The analysis of last years results showed that most problems in these tracks are solved quickly or not at all.
- **Memory limits.** The memory limits for the single query, incremental, unsat-core, and model-validation tracks are reduced to 30 GB. *Rationale:* The reasons are hardware limits of the new execution platform.
- **Proof track.** The previously experimental evaluation proof track will be discontinued. *Rationale:* There are three reasons that led to this decision. First, we were unable to find a way of turning the proof track into a competition that would not lead to an unmanageable amount of overhead on the competition organizers or the participants. Second, we have collected vast amounts of data in the last two years and thoroughly evaluating it takes time. Third, this year we would need to restrict the allowed proof size due to hardware limits of the new execution platform.
- **Tools.** All the relevant scripts in <https://github.com/SMT-COMP/smt-comp> have been combined into one tool called **smtcomp** that is available at <https://github.com/SMT-COMP/smt-comp.github.io>. *Rationale:* The reason for this new tool is that the sheer number of old scripts and their various locations caused a lot of confusion to many new and several long-time participants. The tool also has better run time compared to some of the old scripts because it uses parallelization where possible. Moreover, it provides a nicer UI compared to the old scripts.

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 (<https://smtlib.cs.uiowa.edu/benchmarks.shtml>).

Portfolio Solver. A *portfolio solver* is a solver using a combination of two or more sub-solvers, developed by different groups of authors, on the same component or abstraction of the input problem. For example, a solver using one subsolver to solve the ground abstraction of a quantified problem is allowed, while a solver using two or more subsolvers from different groups of authors is not. Portfolio solver are in general **allowed only in the Parallel Track and Cloud Track**. If you are unsure if your tool is a portfolio solver according to this definition and you feel that it should be allowed contact the organizers of the SMT-COMP for clarification.

Wrapper Tool. A *wrapper tool* is defined as any solver that calls one or more other SMT solvers (the *wrapped solvers*) from a different group of authors to solve distinct subtasks. Examples of such subtasks are preprocessing steps or solving a subproblem that belongs to a strict sublogic of the input logic. For example, a solver using one subsolver to solve the ground abstraction of a quantified problem would be a wrapper tool. However, a wrapper tool solving a benchmark of logic A is not allowed to call an SMT solver to solve a problem for logic A; even if the problem is just a subproblem. The system description of a wrapper tool **must** explicitly acknowledge and state the **exact** version of any solvers that it wraps. It **must** 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*) from a different group of authors. In contrast to a wrapper tool, a derived tool solving a benchmark of logic A is allowed to call an SMT solver to solve a problem for logic A. However, it is not allowed to call more than one different SMT solver to solve a problem over the input logic. Examples of derived tools are tools that change the strategy selection for the base solver, tools that add new preprocessing steps on top of the base solver, tools that add new heuristics or incomplete procedures to a base solver. The system description of a derived tool **must** explicitly acknowledge the solver it is based on and extends. It **must** further make clear technical innovations by which the derived tool expects to improve on the original solver. A derived tool **must** 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 via a push request to the SMT-COMP GitHub repository (<https://github.com/SMT-COMP/smt-comp.github.io/tree/master/submissions>), or, for Parallel Track and Cloud Track, otherwise communicated to the organisers. The final solver version needs to be uploaded as a Zenodo (<https://zenodo.org/>; <https://zenodo.org/i>) submission that must contain a docker file that compiles the solver.

Solver execution. BenchExec (<https://github.com/sosy-lab/benchexec>) is a framework for reliable benchmarking and resource measurement developed by LMU's Software and Computational Systems Lab (SoSy-Lab <https://www.sosy-lab.org/>). The framework itself can be downloaded and used locally by anyone. The competition itself will be executed on a BenchExec cluster owned by SoSy-Lab, who are kind enough to support our competition with their computing power. To be more precise, the competition will be run on the 168 apollon nodes of the SoSy-Lab BenchExec

cluster (for more details see <https://vcloud.sosy-lab.org/cpachecker/webclient/master/info>). It is also possible to locally emulate and test the computing environment on the competition machines using the following instructions: <https://gitlab.com/sosy-lab/benchmarking/competition-scripts/#computing-environment-on-competition-machines>.

Participation in the Competition. For participation in SMT-COMP, the organizers must be informed of the solver's presence *and the tracks and divisions which it enters* via a push request to the SMT-COMP GitHub repository

<https://github.com/SMT-COMP/smt-comp.github.io/tree/master/submissions>

The easiest way to create this push request is via the web-interface available under the above URL. The actual solver must be submitted via a link to a publicly available docker file that compiles the solver. For the Cloud Track and Parallel Track, it must be communicated separately to the organisers by email to aws-smtcomp-2024@googlegroups.com. Note that independent of the tracks, the final solver version must be uploaded as a Zenodo (<https://zenodo.org/> | <https://zenodo.org/>) submission. Moreover, for each track, the submission must specify the logics which are supported by the solver. The solver will enter all divisions where it supports at least one logic. It is possible to submit specific configurations of command line options per track/logic/division as well as a default configuration of command line options for all tracks/logics/divisions without their own configuration. A submission **must** also include a link to the *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, excluding references) 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 **must** further include the following information:

- a list of all (current) authors of the system and their present institutional affiliations,
- the basic SMT solving approach employed,
- 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 acknowledgment 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 system description *should* further include the following information (unless there is a good reason otherwise):

- details of any non-standard algorithmic techniques as well as references to relevant literature (by the authors or others), and
- a link to a website for the submitted tool.

System descriptions **must** be submitted **by the preliminary solver submission 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. The updates must happen **by the final solver submission deadline**.

Multiple versions. The intent of the organizers is to promote as wide a comparison among solvers and solver options as possible. However, to keep the number of solver submissions low, each team should only provide multiple solvers if they are substantially different. A justification must be provided for the difference. We strongly encourage the teams to keep the number of solvers per team per category at at most two. By allowing up to two submissions we want to encourage the development of new, experimental techniques via an “alternative solver” while keeping the competition manageable.

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.

Deadlines

SMT-COMP entrants must be submitted via a push request as explained above and, in the case of the Parallel Track and Cloud Track, communicated separately to the organisers via email until the end of **May 27, 2024** anywhere on earth. After this date *no new entrants* will be accepted. However, updates to existing entrants via push requests or, in the case of the Parallel Track and Cloud Track, via email, will be accepted until the end of **June 8, 2024** 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 the execution service or other means after the initial deadline.

The last solver links pushed to the repository or communicated via email to the organisers for Parallel Track and Cloud Track 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. A solver enters a division in a track if it supports at least one logic in this division. A solver not supporting all logics in a division will not be run on the benchmarks from the unsupported logics and will be scored as if it returned the result `unknown` within zero time. All results of the competition will be made public. Solvers will be made publicly available after the competition and it is a minimum license 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 2024. 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.

- **smtcomp.** This tool combines the functionality of most of the scripts used in previous years and more. It is available at <https://github.com/SMT-COMP/smt-comp.github.io>. Some of its functionalities include:
 - **Benchmark Selection.** The tool has a command that executes 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.
 - **Scoring.** The tool has a command that executes the scoring computation described on in Section 7. It also includes the scoring computations used in previous competitions (since 2015).

For a full list of the capabilities of the `smtcomp` tool and an explanation of all important commands see <https://github.com/SMT-COMP/smt-comp.github.io/blob/master/README.md>

- **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 the execution service 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>.

Input. 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-option ...)** The input contains the following **set-option** commands.
 - (a) 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.
 - (b) In all other tracks, the scrambler will add an initial **(set-option :print-success false)** command to the solver.
 - (c) In the *Model-Validation Track*, a benchmark file contains a single **(set-option :produce-models true)** command as the second command.
 - (d) In the *Unsat-Core Track*, a benchmark file contains a single **(set-option :produce-unsat-cores true)** command as the second command.

- **(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. During the competition all **set-info** commands are removed from the benchmark by the scrambler.

- **(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.

Output. In all tracks except the Incremental Track, any `success` outputs will be ignored². 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. Note that there is no distinction between output and error channel and tools should not write any message to the error channel because it could be misinterpreted as a wrong result.

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 10-11 specify the time limit for each track. Each processor has 4 cores. Detailed machine specifications are available on the competition web site.

The execution 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 30 GB. The values of both the time limit and the memory limit are available to a solver process through environment variables. See the user guide for the execution service for more information.

The limits for Parallel Track and Cloud Track are available at <https://smt-comp.github.io/2024/parallel-and-cloud-tracks.html>.

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. This is ensured by BenchExec by executing each solver with the whole filesystem mounted as read-only with an overlay writable layer that is mounted as a RAM disk. Any generated files will be therefore written to the RAM disk. The used storage is counted into the memory limit. The temporary overlay layer is deleted after the job is complete. Solvers must not attempt to communicate with other machines, e.g., over the network.

²SMT-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.

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 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 20 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, containing 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 should include a script `smtcomp_run_incremental` (if this is not present the script for the default configuration will be used). This script will be called without arguments and will be connected to a trace executor, which will incrementally send commands to the standard input channel of the solver and read responses from both 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. Solvers must not write anything to the standard error channel.

Benchmark Selection. See page 14.

Time Limit. This track will use a wall-clock time limit of 20 minutes per solver/benchmark pair.

Trace Executor. This track will use the trace executor from <https://github.com/SMT-COMP/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 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 *unsat* benchmarks (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 (i) the response to the **check-sat** command is *sat*, (ii) the returned unsatisfiable core is not, in fact, unsatisfiable. If the solver replies *unsat* to **check-sat** but gives no response to **get-unsat-core**, this is considered as no reduction, i.e., as if the solver would have returned the entire benchmark as an *unsat* core.

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 20 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.

5.5 Model-Validation Track

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 track has the divisions QF_Bitvec, QF_DataTypes, QF_Equality, QF_Equality+Bitvec, QF_Equality+(Non)LinearArith, QF_(Non)LinearIntArith, and QF_(Non)LinearRealArith. This year all divisions with non-linear arithmetic, arrays, and datatypes are experimental divisions. The track will run on a selection of non-incremental *sat* benchmarks from these logics (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 **check-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>. It expects as model input a file with the answer to the **check-sat** command followed by the solver response to the **get-model** command. The model validator tool will output

1. VALID for a *sat* solver response followed by a full satisfying model;
2. INVALID for
 - an *unsat* solver response to **check-sat** or
 - models that do not satisfy the input problem.
3. UNKNOWN for
 - no solver output (no response to either both commands or **get-model**),
 - an *unknown* response to **check-sat**, or
 - malformed models, e.g., partial models.

The new experimental divisions require additional syntax to represent their models. We propose the new syntax on the SMT-COMP website <https://smt-comp.github.io/2024/model.html>.

Time Limit. This track will use a wall-clock time limit of 20 minutes per solver/benchmark pair. The time limit for checking the satisfying assignment is yet to be determined, but is anticipated to be around 15 minutes of wall-clock time.

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.

5.6 Parallel Track

The Parallel Track will evaluate the capability of solvers to determine the satisfiability of problems in a shared-memory parallel computing environment. The track will be experimental.

Benchmark Selection. We will select non-incremental benchmarks from the *smt-lib* divisions based on the participating solvers. In total 400 instances will be chosen such that their run times are sufficiently high based on our estimation.

Time Limit. This track will use a wall-clock time limit of 2 minutes per solver/benchmark pair.

5.7 Cloud Track

The Cloud Track will evaluate the capability of solvers to determine the satisfiability of problems in a distributed computing environment. The track will be experimental.

Benchmark Selection. We will select non-incremental benchmarks from the smt-lib divisions based on the participating solvers. In total 400 instances will be chosen such that their run times are sufficiently high based on our estimation.

Time Limit. This track will use a wall-clock time limit of 2 minutes per solver/benchmark pair.

6 Benchmarks and Problem Divisions

Divisions. Within each track there are multiple divisions, and each division selects benchmarks from a specific group of SMT-LIB logics 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, Parallel Track and Cloud Track will use a subset of all *non-incremental* benchmarks and the Incremental Track will use a subset of all *incremental* benchmarks. The Unsat-Core Track will use a selection of non-incremental *unsat* benchmarks 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 logics QF_BV, QF_IDL, QF_RDL, QF_LIA, QF_LRA, QF_LIRA, QF_UF, QF_UFBV, QF_UFIDL, QF_UFLIA, QF_UFLRA. To determine whether a benchmark is `sat` or `unsat`, a combination of the benchmark's status and the result of the Single Query Track will be used.

New benchmarks. The deadline for submission of new benchmarks was **April 13, 2024**. The organizers, in collaboration with the SMT-LIB maintainers, will be checking and curating these until **May 11, 2024**.

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 the logics of 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 logic in a 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 Parallel Track and Cloud 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 / uninteresting benchmarks.* For the following tracks, all benchmarks that can be considered as easy or uninteresting based on the following criteria will be removed.
 - *Single Query Track.* All benchmarks that were solved by all solvers (including non-competitive solvers) in less than one second in the corresponding track in 2018–2023.
 - *Unsat-Core Track.* All benchmarks with only a single assertion.
3. For the Unsat-Core Track, all benchmarks with status `sat` are removed. We further remove benchmarks with status `unknown` for which no sound solver reported them to be `unsat`. For the Model-Validation Track, all benchmarks with status `unsat` are removed, as well as benchmarks with status `unknown` for which no sound solver reported them to be `sat`.

In case of a dispute (some solver marks a benchmark as `sat` and some other solver as `unsat`), the benchmark may be retained in the selection.

4. *Cap the number of instances in a division.* The number of benchmarks in a division based on the size of the corresponding logics in SMT-LIB will be limited as follows. Let n be the number of benchmarks in an SMT-LIB logic, then the number of chosen benchmarks is $\min(n, \max(300, 50n/100))$:
 - (a) if $n \leq 300$, all instances will be selected;
 - (b) if $300 < n \leq 600$, a subset of 300 instances from the logic will be selected;
 - (c) and if $n > 600$, 50% of the benchmarks of the logic 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 one hundred times 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)
- $aw \in [0, T]$ actual wall-clock time in seconds (real-valued)
- $w \in [0, T]$ wall-clock time score in seconds (real-valued)
- $ac \in [0, mT]$ actual CPU time in seconds (real-valued)
- $c \in [0, mT]$ CPU time score in seconds (real-valued)

Error Score (e). For the Single Query Track, Incremental Track, Parallel Track, and Cloud 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 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 not full satisfiable models.

Correctly Solved Score (n). For the Single Query Track, Incremental Track, Model-Validation Track, Parallel Track, and Cloud 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.

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

Wall-Clock Time Score (w). For the Single Query Track, Unsat-Core Track, Model-Validation Track, Parallel Track, and Cloud Track, the wall-clock time score w is the same as the actual (real-valued) wall-clock time aw , except that it is zero if the benchmark was not correctly solved within the time limit T , i.e., $w = 0$ if $e = 1$, the process did not terminate within the time limit T , or it did return unknown or an unknown result. For the Incremental Track, the wall-clock time score w is the (real-valued) wall-clock time in seconds until the process returned the last time sat/unsat within the time limit; this means especially that $w = 0$ if the process never returned sat/unsat within the time limit.

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

CPU Time Score (c). For the Single Query Track, Unsat-Core Track, Model-Validation Track, Parallel Track, and Cloud Track, the CPU time score c is the same as the actual (real-valued) CPU time ac , except that it is zero if the benchmark was not correctly solved within the time limit mT , i.e., $c = 0$ if $e = 1$, the process did not terminate within the time limit mT , or it did return unknown or an unknown result. For the Incremental Track, the CPU time score c is the (real-valued) CPU time in seconds until the process returned the last time sat/unsat within the time limit; this means especially that $c = 0$ if the process never returned sat/unsat within the time limit.

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 $\langle e, n, aw, w, ac, c \rangle$, the corresponding sequential performance is defined as $\langle e_S, n_S, c_S \rangle$, where

- $e_S = 0, n_S = 0$, and $c_S = 0$ if $c > T$;
- $e_S = e, n_S = n$, and $c_S = c$ otherwise.³

7.1.2 Single Query Track, Parallel Track, and Cloud Track

For the Single Query Track, Parallel Track, and Cloud 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,

³Under 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 .

- 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.

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 to **check-sat**, or
 - the result of the **check-sat** command is `unknown`,
 - the result of the **get-unsat-core** command is not wellformed.
- $e = 1$ and $n = 0$ 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., $n = N$ 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 result is UNKNOWN according to the output of the model validating tool described in Section 5.5,

⁴If 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.

⁵Times measured by the execution service 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.

- $e = 1$ and $n = 0$ if the result is INVALID according to the output of the model validating tool described in Section 5.5,
- 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, 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. Finally, in divisions composed by more than one logic, all the above scores will be presented not only for the overall division but also for each logic composing the division.

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 $\langle 0, n, aw, w, ac, c \rangle$ 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, 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:

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

A parallel division score $\langle e, n, aw, w, ac, c \rangle$ is better than a parallel division score $\langle e', n', aw', w', ac', c' \rangle$ iff $e < e'$ or $(e = e' \text{ and } n > n')$ or $(e = e' \text{ and } n = n' \text{ and } w < w')$ or $(e = e' \text{ and } n = n' \text{ and } w = w' \text{ and } c < c')$. 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, Parallel Track, and Cloud 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:

$$\sum_{b \in M} \langle e_b^s, n_b^s, aw_b^s, w_b^s, ac_b^s, c_b^s \rangle$$

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 $(e^s = e^{s'} \text{ and } n^s > n^{s'})$ or $(e^s = e^{s'} \text{ and } n^s = n^{s'} \text{ and } c^s < c^{s'})$. 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.

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 score has been replaced the competition-wide score with two *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.

⁶Since 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. Due to the nature of Parallel Track and Cloud Track we will not consider the sequential scores

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 i th solver (for a given scoring system e.g. number of correct results or reduction) in division D . The *correctness rank* of division D is given as

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

Let c_i^D be the CPU time score of the i th solver in division D . The *CPU time rank* of division D is given as

$$\frac{c_2^D + 1}{c_1^D + 1}$$

Let w_i^D be the wall-clock time score of the i th solver in division D . The *wall-clock time rank* of division D is given as

$$\frac{w_2^D + 1}{w_1^D + 1}$$

The *biggest lead winner* is the winner of the division with the highest (largest) correctness rank. In case of a tie, the winner is determined as the solver with the higher corresponding CPU (resp. wall-clock) time rank for sequential (resp. parallel) scoring. 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 $\langle e^s, n^s, aw^s, w^s, ac^s, c^s \rangle$ be the parallel division score for solver s (for a given scoring system, i.e., n is either number of correct results or reduction). If the division error score $e^s > 0$, then solver s is considered unsound and excluded from the ranking. If the number of sound competitive solvers S in a division D is $|S| \leq 2$, the division is excluded from the ranking.

Let $\langle e_b^s, n_b^s, aw_b^s, w_b^s, ac_b^s, c_b^s \rangle$ be the parallel benchmark score for benchmark b and solver s (for a given scoring system). The virtual best solver *correctness score* for a division D with competitive sound solvers S is given as

$$vbss_n(D, S) = \sum_{b \in D} \max\{n_b^s \mid s \in S \text{ and } n_b^s > 0\}$$

where the maximum of an empty set is 0 (i.e., no contribution if a benchmark is unsolved).

The virtual best solver *CPU time score* $vbss_c$ and the virtual best solver *wall-clock time score* $vbss_w$ for a division D with competitive sound solvers S is given as

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

$$vbss_w(D, S) = \sum_{b \in D} \min\{w_b^s \mid s \in S \text{ and } n_b^s > 0\}$$

where the minimum of an empty set is 1200 seconds (no solver was able to solve the benchmark).

In other words, for the single query track, $vbss_c(D, S)$ and $vbss_w(D, S)$ is the smallest amount of CPU time and wall-clock time taken to solve all benchmarks solved in division D using all sound competitive solvers in S .

Let S be the set of competitive solvers competing in division D . The *correctness rank* $vbss_n$, the *CPU time rank* $vbss_c$ and the *wall-clock time rank* $vbss_w$ of solver $s \in S$ in division D are then defined as

$$1 - \frac{vbss_n(D, S - s)}{vbss_n(D, S)} \quad 1 - \frac{vbss_c(D, S)}{vbss_c(D, S - s)} \quad 1 - \frac{vbss_w(D, S)}{vbss_w(D, S - s)}$$

i.e., the difference in virtual best solver score when removing s from the computation.

These ranks will be numbers 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 ranks for a division D in a given track will be normalized by multiplying with $\frac{n_D}{N}$, where n_D corresponds to the number of competitive solver/benchmark pairs in division D and N being the overall number of competitive solver/benchmark pairs of this track.

The *largest contribution winner* is the solver across all divisions with the highest (largest) normalized correctness rank. Again, this can be computed per scoring system. In case of a tie, the winner is determined as the solver with the higher corresponding normalized CPU (resp. wall-clock) time rank for sequential (resp. parallel) scoring.

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 2024 is organized under the direction of the SMT Steering Committee. The organizing team is

- François Bobot – CEA List, France (chair)
- Martin Bromberger – MPI-INF, Germany
- Jochen Hoenicke – Certora, Germany

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 a BenchExec cluster owned by LMU’s Software and Computational Systems Lab (SoSy-Lab <https://www.sosy-lab.org/> ; <https://www.sosy-lab.org/i>), who are kind enough to support our competition with their computing power. Dirk Beyer is providing essential BenchExec support.

Disclosure. François Bobot is part of the developing team of the SMT solver COLIBRI [6]. Martin Bromberger is part of the developing team of the SMT solver SPASS-SATT [7]. Martin Jonáš is part of the developing team of the SMT solver Q3B [12] and was a member of the developing team of the SMT solver MathSAT5 [8].

References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] François Bobot, Zakaria Chihani, and Bruno Marre. Real behavior of floating point. In Martin Brain and Liana Hadarean, editors, *Proceedings of the 15th International Workshop on Satisfiability Modulo Theories affiliated with the International Conference on Computer-Aided Verification (CAV 2017), Heidelberg, Germany, July 22 - 23, 2017*, volume 1889 of *CEUR Workshop Proceedings*, pages 50–62. CEUR-WS.org, 2017.
- [7] Martin Bromberger, Mathias Fleury, Simon Schwarz, and Christoph Weidenbach. SPASS-SATT - A CDCL(LA) solver. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2019.
- [8] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 SMT solver. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.
- [9] David R. Cok, David Déharbe, and Tjark Weber. The 2014 SMT Competition. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:207–242, 2014.
- [10] 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>.
- [11] 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.
- [12] Martin Jonáš and Jan Strejček. Q3B: an efficient bdd-based SMT solver for quantified bit-vectors. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st Interna-*

tional Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II, volume 11562 of *Lecture Notes in Computer Science*, pages 64–73. Springer, 2019.

- [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>.