A Catalog of SMT-LIB Benchmarks

Hans-Jörg Schurr
University of Iowa, Iowa, USA

Mathias Preiner
Stanford University, Stanford, USA

Aina Niemetz
Stanford University, Stanford, USA

Clark Barrett
Stanford University, Stanford, USA

Pascal Fontaine
Université de Liège, Liege, BE

Cesare Tinelli
University of Iowa, Iowa, USA

Abstract

The SMT-LIB benchmark library is a large set of benchmarks for SMT solvers. It is used by the annual SMT competition to evaluate SMT solvers, and by researchers to study novel solving techniques. Effective use of the benchmark library often requires access to benchmark metadata, such as as the number of user defined symbols. We present a comprehensive metadata catalog for the SMT-LIB benchmark library. It combines features extracted from the SMT-LIB benchmarks with the results of all past SMT-COMP competitions since 2005. The catalog is implemented as a SQLite database. This allows users to use standard industry tools to perform queries, and the database to be distributed as a single file. Since SQLite libraries are available for all major programming languages, it is also easy to integrate the catalog with existing benchmarking tools. In the future, we will distribute the catalog with each annual release of the SMT-LIB library.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Information systems → Information integration

Keywords and phrases SMT, benchmarks, data integration, SQLite, database, automated reasoning

Category Draft

Acknowledgements We thank the many contributors of SMT-LIB benchmarks, and the organizers of the SMT competition. Geoff Sutcliffe provided valuable insights into benchmark difficulty ratings.

1 Introduction

The SMT-LIB [3] initiative is an ongoing international initiative created in 2003 whose goal is to facilitate research and development in Satisfiability Modulo Theories (SMT). Part of this initiative is the development of the SMT-LIB language [4] for specifying SMT problems in text format and the collection and maintenance of a large library of benchmarks problems written in that format. The benchmark library, currently maintained by the authors of this paper, is curated and continually extended with contributions from the SMT community and is published online as yearly releases. The library is used by developers of SMT solvers as well as by the annual SMT solver competition SMT-COMP [9].

The 2024 release of the library, the latest release to date, contains a total of 482,961 benchmarks divided into two categories: *non-incremental* benchmarks (440,874), which contain single satisfiability queries problems, and *incremental* benchmarks (42,087), which involve multiple satisfiable queries. Benchmarks are usually contributed to the library by users and developers of SMT solvers. During the submission process, benchmarks are checked for compliance with the SMT-LIB language and adherence to simple formatting rules.

While benchmarks are categorized by logic, essentially labeling the family of symbols used in the file, in the past there used to be no further categorization or curation. However,

SMT solver users and developers often rely on more advanced metadata. For instance, when evaluating the performance of a procedure or solver configuration that targets a specific fragment of a theory, it may be desirable to only include benchmarks that contain only symbols of that fragment. Other examples are identifying benchmarks that are uniquely solved by a solver configuration, or not solved by any known configuration—both corner cases that may serve as important starting point for developing solving procedures. Metadata can also be used for solving itself, e.g., to guide automatic selection of solving strategies [12].

It is common practice to collect such metadata on demand, via ad-hoc scripts. This is not only error-prone but also inconvenient. We present a benchmark catalog that integrates benchmark metadata, results from all past SMT competitions and hand-curated data. The metadata provides context to select and understand benchmarks, and the competition results track the historical performance of SMT solvers. We intend to release an updated catalog file as part of the yearly benchmark library release.

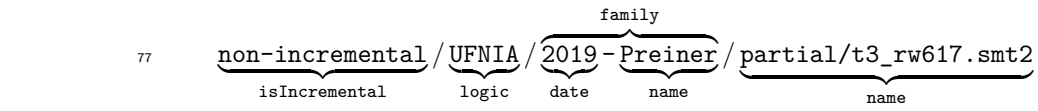
The catalog is published as an SQLite database, which allows for easy integration with existing benchmarking and testing systems. SQLite databases are stored as single files, making them easy to share. Moreover, libraries for interacting with SQLite databases exist for all major programming languages. For ease of maintenance and expansion of the catalog, we implemented the data integration pipeline as flexible Python scripts. For benchmark metadata extraction, we developed an optimized standalone tool called *Klammerhammer*. We further provide a simple web frontend for navigating benchmark metadata.

Section 2 describes how individual benchmarks are represented in the database and gives examples for using the catalog. In Section 3, we describe our data integration pipeline. There are two sources of data: the benchmark metadata (Section 3.1), and the outcome of SMT evaluations (Section 3.2). Finally, we discuss future directions for the catalog, and how it compares to related work in Section 4.

2 The Structure of the Catalog

The database schema of the catalog is given in Figure 1. The database tables of the schema fall into three categories, distinguished visually. The three tables highlighted with ▣ form the core of the database and list benchmarks and the queries contained within the benchmarks; the tables marked by □ store static metadata, such as symbol frequencies; and the tables shown as ⬚ boxes store the results of large scale evaluations.

Every row of the **Benchmarks** table represents one SMT-LIB benchmark, and each benchmark belongs to exactly one *family* stored in the **Families** table. This classification follows the folder structure of the benchmark library. There, each benchmark is uniquely identified by its file path, for example:



The family of the benchmark is the third component of the file path. A benchmark family usually shares common properties. Benchmarks in a family often originate from the same application, or are generated by the same tool. Note that a family may contain benchmarks from several logics, and may contain both incremental and non-incremental benchmarks. The **Families** table models this accurately. The `date` (either a full date, or just a year) is chosen by the submitter, and is usually the date on which the benchmark family was generated. Some benchmark families are not associated with a date for historical reasons.

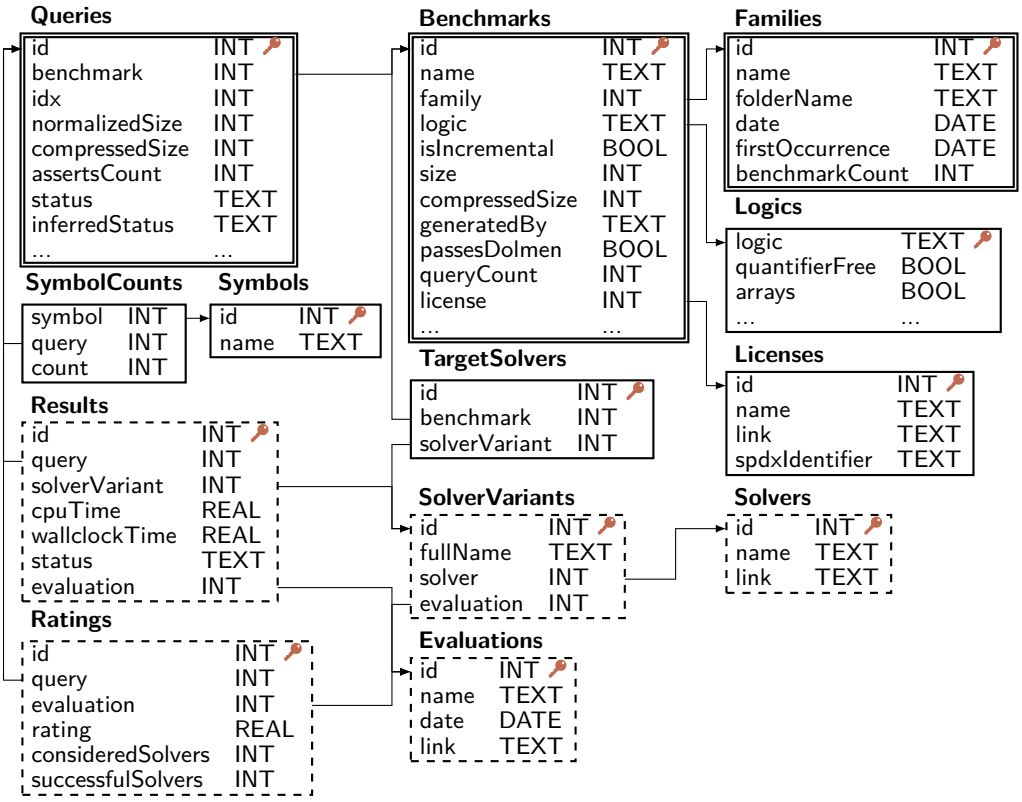


Figure 1 Database schema of the catalog with some fields omitted.

Field firstOccurence records the date of the first competition that used a benchmark from the family. Overall, there are currently 273 families.

The name of the benchmark corresponds to the tail of the file path after the family. The associated logic is a string that indicates the SMT theories referred to by the benchmark.

Finally, the topmost folder indicates whether the benchmark is incremental or not. Incremental benchmarks contain more than one satisfiability query expressed with a `check-sat` command. The command instructs the solver to determine the satisfiability of a set of formulas previously asserted with one or more `assert` commands. Asserted formulas are stored on a stack, which can be manipulated using the `push` and `pop` commands.

Each `check-sat` command corresponds to a row in the **Queries** table. The `idx` field of the **Queries** table is the index of the query in the benchmark. For example, if `idx` is 3, the query corresponds to the third `check-sat` call. Overall, there are 34,507,767 queries. Some benchmarks individually contain thousands of queries. The benchmark with the highest number of queries has 2,630,828 of them.

Perusing the Catalog

The SMT-LIB benchmark library is released on the open-access repository Zenodo [10, 11]. Starting 2025, the catalog will be an additional Zenodo artifact consisting of a compressed archive with the SQLite database and a number of helper files. Since this archive is large (currently, around 1.5 GiB, 5.4 GiB uncompressed), we expect users to download the database and perform queries locally. To reduce the file size, the database has no query indexes. However, the archive contains a script that generates default indexes.

The simplest way to use the database is to use the SQLite command line tool to perform queries. Alternatively, language bindings, such as Python's `sqlite3` module, and graphical tools, such as the DB Browser for SQLite, can be used. For example, the following query returns the number of non-incremental benchmarks containing at least 100 `bvxor` calls: 7130.

```
SELECT COUNT(Benchmarks.id) FROM Benchmarks
  JOIN Queries      ON Queries.benchmark = Benchmarks.id
  JOIN SymbolCounts ON SymbolCounts.query = Queries.id
  JOIN Symbols      ON Symbols.id = SymbolCounts.symbol
 WHERE isIncremental = False AND Symbols.name = 'bvxor'
   AND SymbolCounts.count > 100;
```

The following query returns 6,525, which is the number of benchmarks solved by the SONOLAR solver, but not by the Abziz solver at SMT-COMP 2014 (with id 10).

```
WITH Eval AS (
  SELECT Queries.id, Solvers.name AS sn FROM Queries
  JOIN Results        ON Results.query = Queries.id
  JOIN SolverVariants ON SolverVariants.id = Results.solverVariant
  JOIN Solvers        ON Solvers.id = SolverVariants.solver
  WHERE Results.evaluation = 10 AND Results.status != 'unknown' )
SELECT COUNT(DISTINCT ev.id) FROM Eval AS ev
  WHERE (sn == 'SONOLAR') AND
  NOT EXISTS (SELECT * FROM Eval WHERE sn == "Abziz" AND Ev.id == id)
```

We also provide a simple demonstration webapp that can be used to quickly inspect the data associated with a benchmark. The webapp is part of the release and can be started locally using Docker. `smtlib.schurr.io` (username: `smtlib`, password: `notzenodo`) for reviewing. The webapp allows users to search for benchmarks by logic, name, or family. Once a benchmark is found, it shows the benchmark and metadata. It also lists the solvers that attempted to solve the benchmark at a competition, the years of the competition such an attempt was made, and the solvers that succeeded. Every benchmark is associated with a static URL in the webapp based on its id. For example, the benchmark in Section 3.1 is available at `http://localhost:8000/benchmark/106394`.

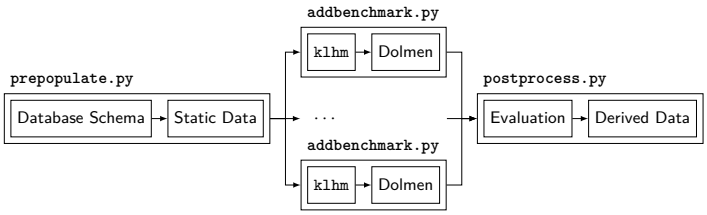


Figure 2 The data integration pipeline.

3 Data Integration

The catalog data is collected from two main sources: the individual benchmark files and the SMT competition data. The data collection from these sources and its integration into the catalog is implemented as a modular and easy-to-extend pipeline, written in Python (available at `github.com/SMT-LIB/SMT-LIB-db`). Figure 2 depicts the workflow of the pipeline. It is divided into three stages, each implemented as a Python script. In the first stage, script

`prepopulate.py` creates the database scheme, and initializes the database with static data, such as the list of licenses used in SMT-LIB benchmarks, SMT-LIB logics, and the names of SMT solver that participated in SMT-COMP. In the second stage, script `addbenchmark.py` is used to parse the individual benchmark files to extract the benchmark metadata. Since this stage is the most time consuming, it can be run in parallel to speed-up the data collection. As final step, script `postprocess.py` integrates the SMT-COMP results data into the database and computes and stores additional data derived from these results.

3.1 Integrating Benchmark Metadata

The metadata tables (☐) store data associated with benchmarks or queries. Each benchmark includes a header section that stores metadata (Section 3.1). The header uses the `set-info` command to declare metadata fields. This command allows to specify a `:source` field, which is populated with information about the source of the benchmark, containing fields as a structured string. Most of these fields relate to the entire benchmark. Hence, most metadata fields are mapped directly to a corresponding field in the **Benchmarks** table. The entry for the example benchmark would store `Mathias Preiner` in the generatedBy field. The `:status` field relates to a specific *query*. It indicates whether the next query is known to be satisfiable or unsatisfiable. This is stored in the status field in the **Queries** table.

Field license associates one license to the benchmark, currently among a list of eleven, in the manually curated **Licenses** table. The license is usually identified by a short code, such as `GPL`, or by a link. However, some benchmarks (e.g., in the *CPAchecker_kInduction-SoSy_Lab* family) contain the entire license text. We shorten this to a license code ("CMU SoSy Lab" in this case). The `Target solver` field lists the solvers targeted by the benchmark creator. We store this information in the **TargetSolvers** table that maps solver variants to benchmarks. Solver variants are also used for SMT competition results (Section 3.2).

A key data point about an SMT query is which theory symbols and SMT-LIB features are used, and how often they occur in a benchmark. While these counts are not directly available in the benchmark header, they can be computed by scanning the benchmark. We distinguish two different categories of counters: one for SMT-LIB commands, such as `assert` and `define-const` which assert formulas and define constants, and one for predefined theory symbols that appear in formulas. The first category is small and fixed, while the second category is large and grows as new theories are added to SMT-LIB. Counts from the first category are therefore fields of the **Queries** table, e.g., assertsCount gives the number of `assert` commands used by a query. For the second category, we use the **Symbols** and **SymbolCounts** tables. The former lists all theory symbols we consider. We extracted this

```
(set-info :smt-lib-version 2.6) (set-logic UFNIA)
(set-info :source |
  Generated by: Mathias Preiner
  Generated on: 2019-03-22
  Application: Verifying bit-vector rewrite rule candidates.
  Target solver: CVC4, Z3, Vampire |)
(set-info :license "https://creativecommons.org/licenses/by/4.0/")
(set-info :category "crafted")
(assert ...) ...
(set-info :status unknown) (check-sat) (exit)
```

Listing 1 Abridged content of the SMT-LIB file from Section 2.

table from the SMT-LIB parser of the *cvc5* SMT solver [1]. The **SymbolCounts** has one entry for each symbol that appears at least once in a query. To simplify the scanner, we do not distinguish theory symbols from user declared symbols. Hence, the **SymbolCounts** table may contain entries for symbols that are not part of the benchmark logic.

The normalizedSize field of the **Queries** table is the a *size* of a query in bytes. A query is identified with each `check-sat` command and encompasses all the commands that assert in the stack information relevant to that `check-sat` command. Its size is computed by correctly tracking the `push` and `pop` commands in the benchmark. The compressedSize field is the size of the query after compression with the *zstd* algorithm. This field measures the problem size and is independent of syntactic factors such as the length of symbol names. The size and compressedSize fields of the **Benchmarks** table are the sizes of the entire benchmark.

Finally, the passesDolmen field of the **Benchmark** table records whether Dolmen, the reference parser and type checker for SMT-LIB [5], reports no error for the benchmark.

Klammerhammer. To extract the metadata quickly we use *Klammerhammer*, a standalone tool we developed, that performs a simple scan of the benchmark. It stores symbol counts on a stack. SMT-LIB `push` commands push a copy of the counts onto the stack, while `pop` commands remove the topmost entry. Whenever a `check-sat` command is encountered, the tool prints the current counts as JSON data. After scanning the entire benchmark, the tool prints the metadata fields for the entries in the **Benchmarks** table. To compute the query size we also store the byte offset of `push` and `pop` calls on the stack. Klammerhammer is implemented in the low level programming language Zig, and uses the zstd library to compute the compressed sizes.

3.2 Integrating SMT-COMP Results

The catalog not only stores metadata of individual benchmarks in the SMT-LIB library, but also combines it with the historical data of all SMT competitions. This allows users of the catalog to get answers for questions like "Can solver X solve the benchmark?" or "How difficult is this benchmark?". The yearly organized SMT competition [9] publishes the raw competition data each year. To answer these questions one must evaluate SMT solvers on the benchmarks. Instead of performing our own evaluations, we use the results of the yearly SMT competition [9]. At that competition, solvers can compete in multiple tracks. For instance *single query* (resp. *incremental*) track tasks solvers with solving non-incremental (resp. *incremental*) benchmarks. To record historical developments, we also integrate past SMT competitions. Integrating multiple competition years also allows us to cover more benchmarks, since recent competitions use only a random subset of the benchmarks from the SMT-LIB library because of its large size. Unfortunately, competitions before 2024 only archived summary results of the incremental track, and not a solver's answers for each individual query.

Each competition year is a row in the **Evaluations** table. This row stores some basic data about the competition, such as a link to its website. The solvers that participate in an evaluation are collected in the **Solvers** and **SolverVariants** tables. A solver variant is a concrete version of a solver that participated in an evaluation (or is mentioned as a target solver in a benchmark). Since solvers have different versioning schemes, we do not attempt to record solver *versions*. Both the **Solvers** and the **SolverVariants** table are manually curated. Overall, we record 82 solvers and 484 variants.

The **Results** table connects queries and solvers for each evaluation. The status field is `sat`, `unsat`, or `unknown`, depending on the answer of the solver. Furthermore, we record

		Results			Benchmarks		
Year	Format	Missing	Total	Percent	Missing	Total	Percent
2005	HTML	10	3,299	0.30%	1	355	0.28%
2006	"	158	7,067	2.24%	58	1,127	5.15%
2007	SQL	684	12,370	5.53%	149	2,297	6.49%
2008	"	933	16,110	5.79%	253	2,993	8.45%
2009	"	471	14,948	3.15%	232	3,711	6.25%
2010	"	684	12,898	5.30%	208	3,731	5.57%
2011	"	380	18,588	2.04%	88	3,779	2.33%
2012	"	496	8,020	6.18%	90	1,557	5.78%
2013	CSV_1	1,336	1,663,478	0.08%	85	95,491	0.09%
2014	CSV_2	38,149	347,147	10.99%	9 096	67,426	13.49%
2015	"	68,670	980,235	7.01%	9 254	154,238	6.00%
2016	"	68,752	1,003,075	6.85%	9 273	154,424	6.00%
2017	"	416	1,186,056	0.04%	114	238,758	0.05%
2018	JSON	29,772	1,388,191	2.14%	29 472	333,241	8.84%
2019	"	91	730,685	0.01%	13	64,154	0.02%
2020	"	878	563,052	0.16%	175	89,910	0.19%
2021	"	0	772,681	0.00%	0	99,254	0.00%
2022	"	0	658,873	0.00%	0	93,791	0.00%
2023	"	0	740,591	0.00%	0	111,285	0.00%
2024	"	0	491,221	0.00%	0	123,486	0.00%

Table 1 Competition results that could not be assigned to benchmarks and data formats used.

both the wallclock time and the CPU time. However, not all competitions recorded both. A major challenge is that the folder structure of the benchmark library changed over time. For example, the logic of misclassified benchmarks was changed. Benchmarks were also removed if they were found not to comply with the SMT-LIB standard. To address this we search benchmarks heuristically in multiple steps. First we only use the name field, since this seldom changes. If this returns an unique benchmark, we use that benchmark. Otherwise, we add the family to the search, and finally the logic. If we are unable to uniquely determine the benchmark using this method, we discard the result. Our goal is not to record the entire evaluation, but collect the results related to benchmarks in the current release. Often the discarded results correspond to cleanup of the SMT-LIB benchmark library. For example, from 2014 to 2016 the *APro VE* family contained duplicate benchmarks, and in 2018 the missing benchmark are in the `QF_SLIA` logic that was experimental that year.

Table 1 lists the competition years and the missing results. The second column shows the file format used. The first two competitions are available as HTML websites. From 2007 until 2012 the competition used on the SMTExec platform [2]. Since this platform is no longer online, the results for these years are not publicly available. We used an archived backup of the SMTExec database to integrate those years and we are currently working on restoring the public results. The SMT Evaluation in 2013 [6] and the SMT competitions between 2014 to 2017 use a very similar CSV format with slightly different column names. Since 2018 the competition provides the data as a JSON file.

We compute derived fields from the evaluation results. The firstOccurence field of the **Families** table is the date of the first evaluation where any benchmark of the family was used. This is useful to for benchmarks without metadata header or date in the file path. The inferredStatus field is a status (`sat` or `unsat`), if at a single evaluation two distinct solvers agreed on that status, and there was no disagreement by a third solver. Hence, this field allows users to know the likely status of a query if no status is given in the benchmark.

Finally, we compute a difficulty *rating* for each query at each evaluation. This rating is the fraction of solvers that solved a benchmark over the solvers that attempted it:

successfulSolvers/consideredSolvers. We *consider* a solver if any of its variants responded to any benchmark in the same logic. This excludes solvers that do not support the benchmark logic. A solver is *successful* if any of its variants gave a `sat`/`unsat` answer that did not contradict the status or inferredStatus. This rating is inspired by TPTP [14], a benchmark library for automated theorem provers. Our calculation, however, is slightly different. TPTP removes from the computation the solvers that solve only a strict subset of queries solved by another solver. We decided to keep such solvers, because a superseded solver represents a serious research effort, and its success or failure to solve a query is evidence of the difficulty of that query. Furthermore, one motivation to remove superseded solvers from the count is that the weaker solver often is a specialized variant of a stronger solver. Instead, our computation combines the variants into a single virtual solver.

4 Conclusion

Benchmark libraries are common in the automated reasoning and theorem proving communities. Among those, the TPTP library [13] is close to SMT-LIB, but it targets theorem provers instead of SMT solvers. TPTP problems store metadata in their header, including syntactic features similar to our **SymbolCounts** table. The metadata header also contains a difficulty rating that inspired our rating. Instead of a standalone database, TPTP provides tools to search the library for benchmarks with specific characteristics. This is possible, because TPTP contains fewer benchmarks (25,775 in the 9.0.0 release) due to careful curation.

The Global Benchmark Database [7] is a database of SAT benchmarks and their metatdata. In contrast to our work, this database does not use a standard SQLite database, but a custom data model and language. Benchmarks can appear in different contexts with different metadata fields. For example, the `cnf` context represents benchmark in conjunctive normal form, and contains fields such as the number of clauses.

SMTQuery [8] is a SMT benchmark analysis tool focused on the theory of strings. It uses a custom, SQL-like, language. The focus on strings and the custom implementation allows SMTQuery to search for metadata that goes beyond symbol frequency. For example, it is possible to search for benchmarks with word equations that have a specific shape. Symbol frequency is also used for machine learning-based SMT solver selection [12].

As future work, we will release an updated database file together with the annual benchmark library release. These releases will also provide an opportunity to add metadata requested by the community. A large change will be the transition to the upcoming Version 3 of the SMT-LIB language. Since this is a major change to the language, it will require updates to the data integration pipeline, and possible changes to the metadata fields, notably for the logic identifier, that will evolve to a more flexible notion.

We are also considering using the database to improve the consistency of the benchmark library. For example, using the inferredStatus field to update the status in the benchmark would make it easier for solver developers to detect errors. Furthermore, we can also fix errors, such as the flipped moth and day in the *20172804-Barrett* family, and replace the license text with a link in the benchmarks that reproduce the entire text.

References

1 Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu,

editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022. doi:10.1007/978-3-030-99524-9_24.

2 Clark Barrett, Morgan Deters, Leonardo de Moura, Albert Oliveras, and Aaron Stump. 6 years of smt-comp. volume 50, pages 243–277, Mar 2013. doi:10.1007/s10817-012-9246-5.

3 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

4 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.7. Technical report, Department of Computer Science, The University of Iowa, 2025. Available at www.SMT-LIB.org.

5 Guillaume Bury. Dolmen: A validator for SMT-LIB and much more. In Alexander Nadel and Aina Niemetz, editors, *Proceedings of the 19th International Workshop on Satisfiability Modulo Theories*, volume 2908 of *CEUR Workshop Proceedings*, pages 32–39. CEUR-WS.org, 2021.

6 David R. Cok, Aaron Stump, and Tjark Weber. The 2013 evaluation of SMT-COMP and SMT-LIB. *J. Autom. Reason.*, 55(1):61–90, 2015. doi:10.1007/S10817-015-9328-2.

7 Markus Iser and Christoph Jabs. Global Benchmark Database. In Supratik Chakraborty and Jie-Hong Roland Jiang, editors, *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:10, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAT.2024.18.

8 Mitja Kulczynski, Kevin Lotz, Florin Manea, Danny Bøgsted Poulsen, and Paul Sarnighausen-Cahn. Smtquery: Analysing smt-lib string benchmarks. In Sidney C. Nogueira and Ciprian Teodorov, editors, *Formal Methods: Foundations and Applications*, pages 22–34, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-78116-2_2.

9 SMT-COMP Organizers. The SMT competition. https://smt-comp.github.io, 2025.

10 Mathias Preiner, Hans-Jörg Schurr, Clark Barrett, Pascal Fontaine, Aina Niemetz, and Cesare Tinelli. Smt-lib release 2024 (incremental benchmarks), May 2024. doi:10.5281/zenodo.11186591.

11 Mathias Preiner, Hans-Jörg Schurr, Clark Barrett, Pascal Fontaine, Aina Niemetz, and Cesare Tinelli. Smt-lib release 2024 (non-incremental benchmarks), April 2024. doi:10.5281/zenodo.11061097.

12 Joseph Scott, Aina Niemetz, Mathias Preiner, Saeed Nejati, and Vijay Ganesh. Algorithm selection for SMT. *Int. J. Softw. Tools Technol. Transf.*, 25(2):219–239, 2023. doi:10.1007/S10009-023-00696-0.

13 Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *J. Autom. Reason.*, 59(4):483–502, 2017. doi:10.1007/s10817-017-9407-7.

14 Geoff Sutcliffe. Stepping stones in the tptp world. In Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt, editors, *Proceedings of the 12th International Joint Conference on Automated Reasoning*, number 14739 in Lecture Notes in Artificial Intelligence, pages 30–50, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-63498-7_3.

A Database Schema

This database schema is also included in the README file of the catalog release.

```
-- One row for each benchmark file.
CREATE TABLE Benchmarks(
        id INTEGER PRIMARY KEY,
        -- File path after the family (not unique)
        name TEXT NOT NULL,
        -- Reference to the family of the benchmark
        family INT,
        logic NVARCHAR(100) NOT NULL, -- Logic string
         -- True if benchmark is in incremental folder
        isIncremental BOOL,
        -- Size of the benchmark file in bytes
        size INT,
        -- Size in bytes after compression with zstd
        compressedSize INT,
        -- Reference to license of the benchmark
        license INT,
        -- 'Generated on' field of the :source header.
        generatedOn DATETTIME,
        -- 'Generated by' field of the :source header.
        generatedBy TEXT,
        -- 'Generator' field of the :source header.
        generator TEXT,
        -- 'Application' field of the :source header.
        application TEXT,
        -- Text of the :source header after standard fields.
        description TEXT,
         -- Either 'industrial', 'crafted', or 'random'.
        category TEXT,
        -- The Dolmen checker reports no error.
        passesDolmen BOOL,
        -- Dolmen with '--strict=true' reports no error.
        passesDolmenStrict BOOL,
        -- Number of (check-sat) calls in the benchmark.
        queryCount INT NOT NULL,
        FOREIGN KEY(family) REFERENCES Families(id)
        FOREIGN KEY(license) REFERENCES Licenses(id)
        FOREIGN KEY(logic) REFERENCES Logics(logic)
    );
-- One row for each (check-sat) call in a benchmark.
CREATE TABLE Queries(
        id INTEGER PRIMARY KEY,
        -- Reference to the benchmark this query belongs to.
        benchmark INT,
        -- Index of the query in the benchmark.  Counted from 1.
        idx INT,
        normalizedSize INT, -- Size in bytes of the query.
        -- Size in bytes of the query compressed with zstd.
        compressedSize INT,
        assertsCount INT, -- Number of asserts in the query.
        -- Number of `declare-fun` commands that declare function
        -- with at least one argument.  Otherwise, these
```

```
393         -- `declare-fun` commands are counted as constants.
394         declareFunCount INT,
395         -- Number of `declare-const` and 0-ary `declare-fun`.
396         declareConstCount INT,
397         declareSortCount INT, -- Num. of `declare-sort` commands.
398         -- Number of `define-fun` commands that expect at least one
399         -- argument. Otherwise, these are counted as
400         -- `constantFunCount`.
401         defineFunCount INT,
402         -- Number of recursive functions.  That is, functions
403         -- introduced by `define-fun-rec` or `define-funs-rec`. Each
404         -- function in `define-funs-rec` is counted individually.
405         defineFunRecCount INT,
406         -- Num. of 0-ary `define-fun` (i.e., constants).
407         constantFunCount INT,
408         defineSortCount INT, -- Num. of `define-sort` commands.
409         -- Number of datatypes.  That is, datatypes introduced by
410         -- `declare-datatype` or `declare-datatypes`.  Each datatype
411         -- in `declare-datatypes` is counted individually.
412         declareDatatypeCount INT,
413         -- Maximum of "open parenthesis" of any term in this query.
414         -- For example, `(a (b (c d) (e (f g))))` has a term depth of
415         -- 4.  See the description of `symbolCounts` for the lists of
416         -- terms considere.
417         maxTermDepth INT,
418         -- Status of the query as declared in the benchmark.
419         status TEXT,
420         -- Status derived from evaluation results.
421         inferredStatus TEXT,
422         FOREIGN KEY(benchmark) REFERENCES Benchmarks(id)
423     );
424 -- Represents a family of benchmarks.  Usually, all benchmarks in a
425 -- family are submitted together.  A family can contain benchmarks
426 -- from different logics, and even incremental and
427 -- non-incremental benchmarks.
428 CREATE TABLE Families(
429         id INTEGER PRIMARY KEY,
430         name NVARCHAR(100) NOT NULL, -- Name of the family.
431         -- Full name of the folder, including the date.
432         folderName TEXT NOT NULL,
433         -- Family date according to folder name.  If only a year is
434         -- given, the date is the first of January of that year.
435         date DATE,
436         -- Date of the first evaluation where any benchmark of this
437         -- family was used.
438         firstOccurrence DATE,
439         -- Number of benchmarks in the family.
440         benchmarkCount INT NOT NULL,
441         UNIQUE(folderName)
442     );
443 -- A solver listed as a target solver in the bechnmark header.
444 CREATE TABLE TargetSolvers(
445         id INTEGER PRIMARY KEY,
446         -- Benchmark with this solver as a target.
447         benchmark INTEGER NOT NULL,
```

```
448         -- Solver variant given by the benchmark.
449         solverVariant INT NOT NULL,
450         FOREIGN KEY(benchmark) REFERENCES Benchmarks(id),
451         FOREIGN KEY(solverVariant) REFERENCES SolverVariants(id)
452     );
453 CREATE TABLE Licenses(
454         id INTEGER PRIMARY KEY,
455         name TEXT, -- Name used for the license in the benchmarks
456         link TEXT, -- Link to webpage of the license
457         -- License identifier see https://spdx.org/licenses/
458         spdxIdentifier TEXT
459     );
460 -- One entry for each logic string currently in use.
461 CREATE TABLE Logics(
462         logic TEXT PRIMARY KEY,  -- Logic string
463         -- Theories and features activated by the logic.
464         quantifierFree BOOL,
465         arrays BOOL,
466         uninterpretedFunctions BOOL,
467         bitvectors BOOL,
468         floatingPoint BOOL,
469         dataTypes BOOL,
470         strings BOOL,
471         -- If false, only linear arithmetic is allowed.
472         nonLinear BOOL,
473         -- If true, only difference logic is allowed.
474         difference BOOL,
475         reals BOOL,
476         integers BOOL
477     );
478 -- This tables list symbols that we count.  Most of them are
479 -- predefined operators, but we also count quantifiers (eg. 'forall').
480 CREATE TABLE Symbols(
481         id INT PRIMARY KEY,
482         name TEXT
483     );
484 -- The number of occurences of that symbol.
485 -- We count occurences in: assert, define-fun, define-fun-rec,
486 -- define-funs-rec, and declare-datatype.
487 CREATE TABLE SymbolCounts(
488         symbol INT,
489         query INT,
490         count INT NOT NULL,
491         FOREIGN KEY(symbol) REFERENCES Symbols(id)
492         FOREIGN KEY(query) REFERENCES Queries (id)
493     );
494 -- List of solvers that participated in the competition or are
495 -- mentioned as target solver.  Solvers based on other solvers (such
496 -- as the Z3-based string solvers are listed as their own entries.
497 CREATE TABLE Solvers(
498         id INTEGER PRIMARY KEY,
499         name TEXT,
500         -- Link to solver webpage or publication.
501         link TEXT
502     );
```

```
-- Since solvers use different versioning schemes, there is
-- no proper version table.  Instead there is only one tables
-- that can be used both for versions, and multiple variants
-- submited to the same competition.
CREATE TABLE SolverVariants(
        id INTEGER PRIMARY KEY,
        -- Full string that was used to refer to the variant.
        fullName TEXT,
        solver INT,
        -- The evaluation that used that variant.  NULL for variants
        -- that are target solvers of benchmarks.
        evaluation INT,
        FOREIGN KEY(solver) REFERENCES Solvers(id)
        FOREIGN KEY(evaluation) REFERENCES Evaluations(id)
    );
-- This table lists evaluations. These are usually, but not necessary,
-- SMT competitions.
CREATE TABLE Evaluations(
        id INTEGER PRIMARY KEY,
        name TEXT,
        -- Date when results were published (at the SMT workshop).
        date DATE,
        link TEXT
    );
-- This table maps queries to solver variants and results.
-- Both cpu time and wallclock time can be NULL if they are not known.
-- Time is in seconds.
CREATE TABLE Results(
        id INTEGER PRIMARY KEY,
        evaluation INTEGER,
        query INT,
        solverVariant INT,
        cpuTime REAL,
        wallclockTime REAL,
        -- sat, unsat, or unknown.  Might disagree with known status.
        status TEXT,
        FOREIGN KEY(evaluation) REFERENCES Evaluations(id)
        FOREIGN KEY(query) REFERENCES Queries(id)
        FOREIGN KEY(solverVariant) REFERENCES SolverVaraiants(id)
   );
-- Dificulty ratings (see below)
CREATE TABLE Ratings(
        id INTEGER PRIMARY KEY,
        query INT,
        evaluation INT,
        rating REAL, -- 1 - m/n
        consideredSolvers INT, -- n
        successfulSolvers INT, -- m
        FOREIGN KEY(query) REFERENCES Queries(id)
        FOREIGN KEY(evaluation) REFERENCES Evaluations(id)
   );
```