Broadening the Scope of SMT-COMP: the Application Track

Roberto Bruttomesso[1] and Alberto Griggio[2]*

[1] Atrenta France
[2] Fondazione Bruno Kessler

Abstract. During the last decade, SMT solvers have seen impressive improvements in performance, features and popularity, and nowadays they are routinely applied as reasoning engines in several domains. The annual SMT solvers competition, SMT-COMP, has been one of the main factors contributing to this success since its first edition in 2005. In order to maintain its significance and positive impact, SMT-COMP needs to evolve to capture the many different novel requirements which applications pose to SMT solvers, which often go beyond a single yes/no answer to a satisfiability check. In this paper, we present a first step in this direction: the "Application" track introduced in SMT-COMP 2011. We present its design and implementation, report and discuss the results of its first edition, and highlight its features and current limitations.

1 Introduction and Motivation

During the last decade, SMT solvers have seen impressive improvements in performance, features and popularity, and nowadays they are routinely applied as reasoning engines in several domains, from verification to planning and scheduling. Part of this success is the result of a standardization process initiated by the introduction of the SMT-LIB 1.2 standard [13] and continued with the currently adopted SMT-LIB 2.0 standard [3].

Before SMT-LIB was introduced, every solver had his own proprietary input language to specify satisfiability queries: the rich variety of SMT approaches in the last decade resulted in the proliferation of different syntaxes, with the effect of complicating experimental comparison between the solvers. It was not uncommon for research groups to maintain, beyond the solver itself, a set of tools for translating between the various languages.

The process of adopting the SMT-LIB standard, however, was not a completely automatic process. At the time of proposing SMT-LIB 1.2 some solvers were already mature tools, usually specialized for a particular task and with a substantial amount of benchmark database in their own language. Certainly switching to a new language at that point was not much in the interest of SMT

* Supported by Provincia Autonoma di Trento and the European Community's FP7/2007-2013 under grant agreement Marie Curie FP7 - PCOFUND-GA-2008-226070 "progetto Trentino", project ADAPTATION.

research groups: changing language implies writing a new parser, maybe new data structures, translate all the benchmark suites, and perform extensive testing. All this effort "without glory" was probably superior to that of maintaining the proprietary language.

The SMT-COMP was a major driving force towards the adoption of the SMT-LIB standard, as it gave the necessary motivation to standardize the input languages. The effort of implementing the new solver infrastructure was justified by the enthusiasm of participating to a friendly race and by the possibility of acquiring visibility in the formal verification and automated deduction communities.

Another big contribution of the SMT-COMP is that of favoring the collection of application-specific benchmarks. Several users of SMT usually submit their benchmarks to the competition initiative as they know that the participating solvers will try their best optimizations to solve them. As a result the submitted benchmarks will be solved in less amount of time and therefore the original application automatically receive a boost in performance.

The Application track. The Application track (as opposed to the traditional Main track) was conceived to stimulate the development of *incremental* SMT solvers: nowadays, SMT solvers are often tightly integrated with other higher-level environments, such as, e.g., a model-checker, from which they receive many successive satisfiability queries to be answered. Although these queries could be solved by just restarting the search from scratch everytime, the solving process is much more efficient if the SMT solver is instructed to cope with them incrementally, by retaining some information from the result of the previous queries: thus, the solver will perform only the new bit of search that is strictly necessary.

Expressing this behavior in a benchmark requires the specification of multiple satisfiability queries, as well as the ability to set and restore backtrack-points in the SMT solver: the SMT-LIB 2.0 standard allows to cope with this requirement by means of the commands `push` and `pop`, which allow to dynamically control the stack of assertions to be solved. As a side note, from the point of view of the SMT-LIB 2.0 the Main track benchmarks can be seen as a restriction of the Application track ones to contain only one query.

2 Design and Implementation

In designing the Application track of the competition, we had to face several different requirements. The first, and perhaps most important, is that we wanted to be able to mimic the interaction between an SMT solver and the higher level application using it as faithfully as possible. In principle, we could have achieved this by selecting some open-source "reference" application from some important domains (e.g. model checking or program analysis), and requiring competitors in the Application track to link their SMT solvers with the selected applications.

Although this solution was initially considered, it was however quickly dropped, for the following reasons. First, this solution puts a lot of burden on the shoul-

```
(declare-fun c0 () Int)
(declare-fun E0 () Bool)
(declare-fun f0 () Bool)
(declare-fun f1 () Bool)
(push 1) ;; push one checkpoint
(assert (and (or (<= c0 (- 3)) (not f1)) (or (not (= c0 0)) (not f0))))
(check-sat)
(pop 1) ;; discard all the formulas asserted after the most recent checkpoint
(declare-fun f2 () Bool)
(declare-fun f3 () Bool)
(declare-fun f4 () Bool)
(declare-fun c1 () Int)
(declare-fun E1 () Bool)
(assert (and (or (>= (+ c1 (* 3 c0)) 0) (not f4)) (or E0 (= c0 c1) (not f2))))
(push 1)
(check-sat)
(assert (and f1 (not f2)))
(check-sat)
(pop 1)
(exit)
```

Fig. 1. Example of a simple trace for the Application track.

ders of potential competitors, thus contributing to the general perception that the barrier for participating in the SMT competition is too high. Second, it makes the competition heavily depend on some specific (versions of) applications, which could result in unwanted bias and/or difficulty in reproducing the results. Finally, this solution is in strong contrast with one of the main goals that SMT-COMP has pursued since its first edition in 2005, which is the promotion of the SMT-LIB standard input format and library of benchmarks for SMT solvers.

A solution that addresses the above points is to generate a *trace* of the interaction between an application and its back-end SMT solver, by exploiting the capabilities of the new SMT-LIB 2.0 language [3] of expressing incremental problems and complex interactions with the SMT solver. This decouples the competition from the applications that provide the benchmarks, it eases the task of reproducing the results, and it allows for collecting, storing and managing benchmarks using the same formats, tools and infrastructure as the main SMT-COMP track. Moreover, it helps in promoting the adoption of the features of the SMT-LIB 2.0 language for specifying incremental SMT problems. In particular, the Application track makes use of the features of the SMT-LIB 2.0 language that allow for specifying a dynamic stack of formulas (by using the `push`, `pop` and `assert` commands) and performing multiple satisfiability checks (via the `check-sat` command) on it. A simple example trace is shown in Figure 1.

A drawback of the latter solution, however, is that solvers can see the whole sequence of queries that occur in the trace before actually solving them. This is in contrast with the "real world" scenario in which applications query the solvers in an interactive, "online" manner, and do not generate the next query until the solver has produced an answer for the current one. In principle, knowing all the queries in advance might allow some solvers to apply some techniques that would not be available in an online setting. To prevent this possibility, we have developed a *trace executor*, a tool which is designed to mimic the interaction between an application and an SMT solver used as a back-end reasoning engine. More specifically, the trace executor serves the following purposes: (i) it simulates the online interaction by sending single queries to the SMT solver (through their standard input); (ii) it prevents "look-ahead" behaviors of SMT solvers; (iii) it records time and answers for each call, possibly aborting the execution in case of a wrong answer; (iv) it guarantees a fair execution for all solvers by abstracting from any possible crash, misbehavior, etc. that may happen on the application side. The trace executor tool is open source, and it is available from [14]. Its concept and functionalities are similar to those used for the evaluation of different BDD packages for model checking described in [17].

Scoring mechanism. For the first edition of the Application track, the following scoring procedure was implemented. The score for each benchmark is a pair $\langle n, m \rangle$, with $n \in [0, N]$ an integral number of points scored for the benchmark, where N is the number of satisfiability queries in the benchmark. $m \in [0, T]$ is the (real-valued) time in seconds, where T is the timeout. The score of a solver is obtained by summing component-wise the scores for the individual benchmarks. Scores of solvers are compared lexicographically: a solver with a higher n-value wins, with the cumulative time only used to break ties.

The score for a single benchmark is initialized with $\langle 0, 0 \rangle$, and then computed as follows. (i) A correctly-reported `sat` or `unsat` answer after s seconds (counting from the previous answer) contributes $\langle 1, s \rangle$ to the score. (ii) An answer of `unknown`, an unexpected answer, a crash, or a memory-out during execution of the query, or a benchmark timeout, aborts the execution of the benchmark and assigns the current value of the score to the benchmark.[3] (iii) The first incorrect answer has the effect of terminating the trace executor, and the returned score for the overall benchmark is $\langle 0, 0 \rangle$, effectively canceling the score for the current benchmark. As queries are only presented in order, this scoring system may mean that relatively "easier" queries are hidden behind more difficult ones located at the middle of the query sequence.

Example 1. For example, suppose that there are 3 solvers, $S1$, $S2$ and $S3$, competing on 2 benchmark traces, $T1$ and $T2$, containing respectively 5 and 3 queries, with a timeout of 100 seconds. Suppose that the solvers behave as follows:

– $S1$ solves each of the first four queries of $T1$ in 10 seconds each and the fifth in another 40 seconds. Then, it solves the first 2 queries of $T2$ in 2 seconds each, timing out on the third;
– $S2$ solves the first four queries of $T1$ in 10 seconds each, timing out on the fifth, and then all the 3 queries of $T2$ in 5 seconds each;
– $S3$ solves the first four queries of $T1$ in 2 seconds each, but it incorrectly answers the fifth. It then solves all the 3 queries of $T2$ in 1 second each.

[3] The timeout is set globally for the entire benchmark; there are no individual timeouts for queries.

Then, the scores for the solvers are as follows:

– $S1$ obtains a score of $\langle 5, 80\rangle$ on $T1$, and a score of $\langle 2, 4\rangle$ on $T2$. Its total score is therefore $\langle 7, 82\rangle$;
– $S2$ obtains a score of $\langle 4, 40\rangle$ on $T1$, and a score of $\langle 3, 15\rangle$ on $T2$, resulting in a total of $\langle 7, 55\rangle$;
– $S3$ obtains a score of $\langle 0, 0\rangle$ on $T1$, due to the wrong answer on the last query, and a score of $\langle 3, 3\rangle$ on $T2$. Its total score is therefore $\langle 3, 3\rangle$.

The final ranking of the solvers is therefore $S2$, $S1$, $S3$. ◇

During the discussions following the end of the competition and the preparation for the next edition, some issues were raised concerning the potential bias of the above scoring mechanism towards certain kinds of benchmarks/applications. Benchmarks collected from different applications vary a lot in the number and the difficulty of their individual satisfiability queries. For instance, benchmarks from hardware bounded model checking typically consist of relatively few (in the order of hundreds) incremental SMT calls of increasing size, in which each query might be exponentially more difficult to solve than its predecessor in the sequence. In contrast, benchmarks taken e.g. from software verification based on predicate abstraction consist of hundreds of thousands of satisfiability checks of much more uniform size and complexity. These differences are not properly reflected in the above score, in which each correct answer adds one point to the result, independently of the context/benchmark in which it occurs, and the main ranking criterion is the total number of correct answers, with the execution time used only for breaking ties. As a consequence, solvers that are optimized for benchmarks with many easy queries have a potential advantage over those designed for bounded model checking.

Different solutions for fixing the above problem are currently being evaluated for the 2012 edition of the Application track. In particular, the current candidate proposal is that of giving different weights to queries depending on the benchmark in which they occur. This could be achieved for instance by incrementing the score of $1/N_i$ points for each solved query, where N_i is the total number of queries of the current trace. In this way, solving one more problem in a BMC trace of bound 100 would count much more than solving one more query in a predicate abstraction trace with 100000 trivial satisfiability checks.

3 Benchmarks

The availability of good quality benchmarks is a crucial factor for the success of solver competitions, and the Application track is no exception. In order to ensure the widest possible range of sources and application domains, a public call for benchmarks was issued several months before the competition dates. No restrictions were put on the nature of the benchmarks or the used theories, as long as they conformed to the SMT-LIB 2.0 specification [3] and they represented realistic sequences of incremental calls to an SMT solver issued by a higher level application. For the first edition of the Application track, more than 6400 benchmarks were submitted, for a total of more than 4800000 satisfiability queries. The following gives brief descriptions of the benchmarks.

BMC and k-Induction queries from the NuSMV Model Checker [7]. These are verification problems on Linear Hybrid Automata and Lustre designs, using linear rational (QF_LRA) and integer (QF_LIA) arithmetic.

BMC and k-Induction queries from the Kratos Software Model Checker [8]. These are verification problems on SystemC designs. Each benchmark comes in two versions: the first using linear rational arithmetic (QF_LRA), and the second using bit-vector arithmetic (QF_BV).

Predicate abstraction queries from the Blast Software Model Checker [4]. The benchmarks have been generated by logging the calls to the Simplify theorem prover made by Blast for computing predicate abstractions of some Windows device driver C programs. They use the combined theory of linear integer arithmetic and uninterpreted functions (QF_UFLIA).

k-Induction queries from the Kind Model Checker [11]. These benchmarks are invariant verification problems on Lustre programs, using the combined theory of linear integer arithmetic and uninterpreted functions (QF_UFLIA).

Access control policy benchmarks from the ASASP project [1]. ASASP implements a symbolic reachability procedure for the analysis of administrative access control policies. The benchmarks use quantifiers, arrays, uninterpreted functions and linear integer arithmetic (AUFLIA).

Selection of Competition Benchmarks. In the main track of SMT-COMP, the subset of the SMT-LIB benchmarks to be used in the competition are selected with an algorithm that ensures a good balance of difficulties, status (`sat` vs `unsat`) and origin of the instances [2]. For the first edition of the Application track, however, in most of the divisions there was no need to perform a selection of benchmarks, given that the number of instances available was sufficiently small that all of them could be included in the competition. The only exceptions were the QF_UFLIA and the AUFLIA divisions, in which the number of benchmarks was too high for including all of them in the competition. In these two cases, we simply picked a subset of the instances with the largest number of incremental queries. As the Application track matures, and more incremental benchmarks become available, we expect to design more sophisticated selection criteria, inspired by those used in the main track.

4 Results

The first edition of the Application track was held during SMT-COMP 2011, as part of the CAV conference. The track was run on the SMT-Exec service [15], using the same infrastructure as the main SMT-COMP.

Participating solvers. Five different solvers took part to the first edition of the Application track. The following gives brief descriptions of the participants.

Boolector 1.4.1 (with SMT-LIB 2.0 parser) [5]. The original BOOLECTOR 1.4.1 was developed by Armin Biere and Robert Brummaryer at the Johannes Kepler University of Linz. BOOLECTOR is one of the most efficient SMT solvers for bit-vectors (QF_BV) and arrays (QF_AUFBV). The participating version was connected with a generic parser for SMT-LIB 2.0 [16], and submitted by the competition organizers as a solver of interest for the SMT community. BOOLECTOR competed in the QF_BV division.

MathSAT 5 [9]. MATHSAT5 is developed by Alberto Griggio, Bas Schaafsma, Alessandro Cimatti and Roberto Sebastiani of Fondazione Bruno Kessler and University of Trento. It is the latest incarnation of a series of solvers with the same name (but independent implementations) that have been developed in Trento as research platforms for SMT since 2002. MATHSAT5 competed in the QF_BV, QF_UFLIA, QF_LIA and QF_LRA divisions.

OpenSMT [6]. OPENSMT is an open-source SMT solver developed by Roberto Bruttomesso, Ondrej Sery, Natasha Sharygina and Aliaksei Tsitovich of Università della Svizzera Italiana, Lugano, with the objective of being an open platform for research, development and detailed documentation on modern SMT techniques. OPENSMT competed in the QF_LRA division.

SMTInterpol [10] 2.0pre. SMTINTERPOL is a solver developed by Jürgen Christ and Jochen Hoenicke of University of Freiburg, with particular focus on proof generation and interpolation. SMTINTERPOL competed in the QF_UFLIA, QF_LRA and QF_LIA divisions.

Z3 3.0 [12]. Z3 3.0 is the latest version of a very popular and efficient SMT solver developed by Leonardo de Moura, Nikolaj Bjørner and Cristoph Wintersteiger at Microsoft Research. Z3 competed in all divisions.

Results. A summary of the results of the 2011 Application track competition is shown in Figure 2. For each division, the table reports the participating solvers, their score expressed as a ratio between the number of solved queries and the total queries (and computed with the procedure described in §2), and the total execution time (not considering the timeouts). No results are reported for the AUFLIA division, since only one solver supporting it (Z3) was submitted.

Discussion. In order to assess the usefulness and significance of the Application track, it is interesting to compare its results with those of the main track of SMT-COMP 2011. Figure 3 summarizes the results of the main track for the solvers participating also in the Application track.[4] For each solver, besides the

[4] For BOOLECTOR, the version used in the main track is newer than those of the Application track.

QF_BV		
Solver	**Score**	**Time**
MathSAT5	1883/2277	14854
Z3	1413/2277	18836
Boolector (+SMT-LIB 2.0)	863/2277	16989

QF_UFLIA		
Solver	**Score**	**Time**
Z3	1238660/1249524	10015
MathSAT5	1237186/1249524	50464
SMTInterpol	1235238/1249524	25440

QF_LRA		
Solver	**Score**	**Time**
MathSAT5	795/1060	3596
Z3	656/1060	10073
SMTInterpol	465/1060	10333
OpenSMT	375/1060	6950

QF_LIA		
Solver	**Score**	**Time**
MathSAT5	12608/13941	40065
Z3	12262/13941	62512
SMTInterpol	9108/13941	66763

Fig. 2. Results of the Application track at SMT-COMP 2011.

score and total execution time, also their absolute ranking in the main track is reported.[5] By comparing the two groups of tables we can see that there are significant differences between the main and the Application tracks in all the divisions. In no single division the rankings are the same across the two tracks, and in three cases out of four the winners are different. There are many possible explanations for such differences, including differences in the nature and the domains of the benchmarks and in the features of the solvers that are put under stress by the two tracks (for example, some solvers apply aggressive and very effective preprocessing techniques, which might not be compatible with incremental problems). Moreover, some divisions in the Application track contain only benchmarks coming from a single source, which might increase the chances of bias towards a particular solver; this was difficult to avoid for the first edition, and it will become less and less evident as the library of incremental benchmarks increases in size. Nevertheless, the fact that there are such visible differences, on benchmarks coming from applications in important domains, is already a sufficient reason to justify the interest in the Application track.

5 Conclusions

In this report we have discussed the motivations, design, implementation and results of the Application track of the SMT-COMP 2011. Our hope is that this new track will contribute in advancing the state-of-the-art of the SMT solvers with respect to their incremental behavior. Such infrastructure is, in fact, very important in recent applications where a tight communication of similar satisfiability queries is demanded. The benchmarks and our implementation of the track were specifically designed to reflect as much as possible this kind of communication.

The results of the first edition of the Application track compared with those of the Main track, shows that the handling incrementality requires different

[5] This is the ranking considering also the other competitors of the main track that did not participate in the Application track, and which are therefore not shown here.

QF_BV			
Solver	**Ranking**	**Score**	**Time**
Z3	1st	188/210	7634
Boolector (v1.5.33)	3rd	183/210	5049
MathSAT5	4th	180/210	6214

QF_UFLIA			
Solver	**Ranking**	**Score**	**Time**
Z3	1st	207/207	205
SMTInterpol	2nd	207/207	2265
MathSAT5	3rd	206/207	2859

QF_LRA			
Solver	**Ranking**	**Score**	**Time**
Z3	1st	195/207	6088
MathSAT5	2nd	193/207	8963
OpenSMT	4th	178/207	18436
SMTInterpol	5th	169/207	16975

QF_LIA			
Solver	**Ranking**	**Score**	**Time**
Z3	1st	203/210	5276
MathSAT5	2nd	190/210	2608
SMTInterpol	3rd	116/210	2917

Fig. 3. Summary of results of the main track of SMT-COMP 2011 for the competitors of the Application track.

optimizations from those used in single-query benchmarks. This motivates us to collect more incremental benchmarks and to increase the visibility of Application track for its extremely practical orientation.

6 Acknowledgements

Morgan Deters was the third organizer of the SMT-COMP 2011. We wish to acknowledge him for having contributed to the definition of rules and scoring mechanisms and for his indispensable work in setting up the computational infrastructure that allowed the competition to be run.

References

1. Alberti, F., Armando, A., Ranise, S.: Efficient symbolic automated analysis of administrative attribute-based RBAC-policies. In: Cheung, B.S.N., Hui, L.C.K., Sandhu, R.S., Wong, D.S. (eds.) Proceedings of ASIACCS. pp. 165–175. ACM (2011)
2. Barrett, C., Deters, M., de Moura, L., Oliveras, A., Stump, A.: 6 Years of SMT-COMP. Journal of Automated Reasoning pp. 1–35 (2012)
3. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0 (December 2010), `http://goedel.cs.uiowa.edu/smtlib/papers/smt-lib-reference-v2.0-r10.12.21.pdf`
4. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker Blast. STTT 9(5-6), 505–525 (2007)
5. Brummayer, R., Biere, A.: Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: Kowalewski, S., Philippou, A. (eds.) Proceedings of TACAS. LNCS, vol. 5505, pp. 174–177. Springer (2009)
6. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT Solver. In: Esparza, J., Majumdar, R. (eds.) Proceedings of TACAS. LNCS, vol. 6015, pp. 150–153. Springer (2010)

7. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) Proceedings of CAV. LNCS, vol. 2404, pp. 359–364. Springer (2002)
8. Cimatti, A., Griggio, A., Micheli, A., Narasamdya, I., Roveri, M.: Kratos - A Software Model Checker for SystemC. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proceedings of CAV. LNCS, vol. 6806, pp. 310–316. Springer (2011)
9. Griggio, A.: A Practical Approach to Satisability Modulo Linear Integer Arithmetic. JSAT 8(1/2), 1–27 (2012)
10. Hoenicke, J., Christ, J.: SMTInterpol, `http://ultimate.informatik.uni-freiburg.de/smtinterpol/index.html`
11. Kahsai, T., Tinelli, C.: PKind: A parallel k-induction based model checker. In: Barnat, J., Heljanko, K. (eds.) Proceedings of PDMC. EPTCS, vol. 72, pp. 55–62 (2011)
12. de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Proceedings of TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
13. Ranise, S., Tinelli, C.: The SMT-LIB Standard: Version 1.2 (2006), `http://goedel.cs.uiowa.edu/smtlib/papers/format-v1.2-r06.08.30.pdf`
14. The SMT-COMP'11 trace executor, `https://es.fbk.eu/people/griggio/smtcomp11/smtcomp11_trace_executor.tar.gz`
15. SMT-Exec, `https://smtexec.org`
16. SMT-LIBv2 parser, `https://es.fbk.eu/people/griggio/misc/smtlib2parser.html`
17. Yang, B., Bryant, R.E., O'Hallaron, D.R., Biere, A., Coudert, O., Janssen, G., Ranjan, R.K., Somenzi, F.: A Performance Study of BDD-Based Model Checking. In: Gopalakrishnan, G., Windley, P.J. (eds.) Proc. of FMCAD. LNCS, vol. 1522, pp. 255–289. Springer (1998)