StarExec: a Cross-Community Infrastructure for Logic Solving*

Aaron Stump[1], Geoff Sutcliffe[2], and Cesare Tinelli[1]

[1] Department of Computer Science, The University of Iowa
[2] Department of Computer Science, University of Miami

Abstract. We introduce StarExec, a public web-based service built to facilitate the experimental evaluation of *logic solvers*, broadly understood as automated tools based on formal reasoning. Examples of such tools include theorem provers, SAT and SMT solvers, constraint solvers, model checkers, and software verifiers. The service, running on a compute cluster with 380 processors and 23 terabytes of disk space, is designed to provide a single piece of storage and computing infrastructure to logic solving communities and their members. It aims at reducing duplication of effort and resources as well as enabling individual researchers or groups with no access to comparable infrastructure. StarExec allows community organizers to store, manage and make available benchmark libraries; competition organizers to run logic solver competitions; and community members to do comparative evaluations of logic solvers on public or private benchmark problems.

1 Introduction

Ongoing breakthroughs in a number of fields depend on continuing advances in the development of high-performance automated reasoning tools such as SAT solvers, SMT solvers, theorem provers, model finders, constraint solvers, rewrite systems, model checkers, and so on, which we generically refer to here as *(logic) solvers*. Typically, application problems are translated into possibly large and complex formal descriptions (logical formulas, rewrite rules, transition systems, ...) for these tools to reason about. Different tradeoffs between linguistic expressiveness and the difficulty of the original problems have led to the adoption of a variety of reasoning approaches and logical formalisms to encode those problems. Distinct research communities have developed their own research infrastructure to spur innovation and ease the adoption of their solver technology. This includes standard input/output formats for solvers, e.g., [3, 14]; libraries of benchmark problems, e.g., [2, 6, 11, 14]; solver execution services, e.g., [12, 13, 15]; and solver competitions, e.g., [1, 4, 5, 7, 8, 9, 10]. By and large, so far these different infrastructures have been developed separately in the various logic communities, at significant and largely duplicated cost in development effort, equipment and support.

StarExec is a solver execution and benchmark library service aimed at facilitating the experimental evaluation of automated reasoning tools. It is designed to provide a single piece of storage and computing infrastructure to all logic solving communities, with the twofold goal of reducing the effort and resources duplication while

* Work made possible in large part by the support of the National Science Foundation through grants 0957438, 1058748, and 1058925.

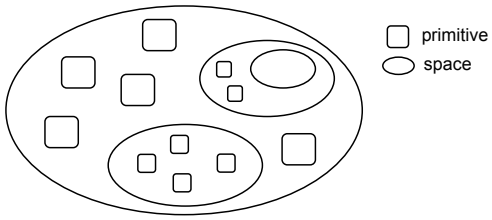

Fig. 1. The StarExec space hierarchy.

also enabling communities and individuals that could not afford developing their own infrastructure. The service allows community organizers to store, manage and make available benchmark libraries, competition organizers to run competitions, and individual community members to run comparative evaluations of logic solvers on benchmark problems. These capabilities are accessible through a web browser interface at the StarExec web site which provides facilities to upload and organize benchmarks and solvers, browse and query the stored benchmark libraries, run user-selected solvers on user-selected benchmarks, and view and analyze execution results and statistics.

This paper gives an overview of StarExec, briefly describing its main design, components, functionality, and usage, and discusses its current status and further development plans. For more details on the service and its usage, we refer the reader to the online documentation on the StarExec web site: `http://www.starexec.org` .

2 Main Concepts

StarExec is built as a service for logic solving *communities*, groups of logic solver developers and users such as the SAT, SMT, theorem proving, confluence, model checking, and software verification communities. In StarExec, a community is a user group administered by one or more designated *community leaders*, users with special privileges and responsibilities such as accepting new users or uploading and managing benchmarks and solvers. The system is built on top of a few basic concepts: *spaces*, *users*, *solvers*, *benchmarks*, *job pairs*, *jobs*, and *processors*, which are described below.

Spaces. A space is a collection of solvers, benchmarks, jobs, and users—collectively referred to as *primitives*—as well as other spaces—also referred to as *subspaces*. Spaces are similar to folders in a file system and are the means by which StarExec primitives are organized (see Figure 1). The subspace hierarchy has a tree structure: each space other than the root space is contained in exactly one superspace. In contrast, each primitive can be contained in more than one space. The direct subspaces of the root space are *community spaces*. Each logic community has one community space which contains, directly or within subspaces, all benchmarks, solvers, users and jobs in that community. Spaces can be *public* or *private*, *locked* or *unlocked*. A public space is visible to all StarExec users; a private one is visible only to the users contained in it. If a space is locked no one can copy items *out* of it. This is useful to store copy-restricted benchmarks and solvers while still allowing other users to include them in job pairs.

Users. A user is a registered and verified person who belongs to at least one community and is therefore endorsed within the system by a community leader. Every user has visibility over a set of (sub)spaces and has a set of *permissions* for each of these spaces controlling the kind of allowed operations. Users also have *leadership* over some spaces. A leader of a space has the full set of permissions over that space. A community leader is simply the leader of a community space.

Solvers. Solvers are programs that solve logic problems. Specifically, they are Linux programs that take a text file on standard input and may produce text on standard output. Solvers can consist of groups of files and folders—as opposed to a single executable. Each solver is associated with one or more *configurations*, executable scripts that invoke the solver with specific input flags and settings. Configurations and processor scripts (see below) can be written in any of the scripting languages (Bash, Perl, Python, and so on) available by default in Linux distributions. A configuration takes as input a benchmark and feeds it to its solver.

Benchmarks. A benchmark is a single text file. Currently there is no support for multi-file benchmarks; however, StarExec supports benchmark dependencies, the referencing of other benchmark files (e.g., TPTP axioms files [14]) within a benchmark. Each benchmark has a *type* which consists of a (system-wide) unique name and an associated benchmark processor.

Jobs and Job Pairs. A job pair consists of a solver configuration and a benchmark to be run by that configuration. Job pairs are atomic execution units. After execution they are associated to various pieces of information such as results, CPU time, wall clock time, etc. A job is simply a collection of job pairs executed or to be executed.

Processors. Processors are executables that take textual input and produce textual output at certain stages in the job execution pipeline. They are currently of two kinds: benchmark processors and post-processors. The former are run on benchmarks being uploaded to a space. They can be used to perform various checks and validations on a benchmark and to extract meta-data to be stored in a central database. Post-processors are run on solver output produced by executing a job pair. They can be used to extract specific data from this output which too are stored in a database for later access.

3 Functionality and Usage

StarExec was designed as a web service accessible through a web browser interface. Most of the web service functionality is also available through a downloadable client application, called StarExecCommand. That program provides a command line shell and a large number of commands mirroring the operations that can be performed via a web browser. In the following, we focus on the web browser interface.

User Accounts Using StarExec requires first obtaining a user account. Since every registered user belongs to at least one of the StarExec communities, every new account request requires joining one of them (more communities can be joined later). Such requests are sent to the leaders of the chosen community each of whom has the power to approve them or not. Communities themselves and community leader users can be created only by the StarExec administrators. Registered users must log in to StarExec

before being able to use it. External observers can view all public information on the StarExec website by logging in as the special user *guest*.

Spaces and Communities Using an interface similar to that of file explorers in computer desktop GUIs, users can browse the StarExec space hierarchy or, rather, the subset of it visible to them; view the contents of any spaces they belong to; and inspect details of users, benchmarks, solvers, and jobs in those spaces. As an example, details about benchmarks include name, description, owner, upload date, file size, type, and any user-defined attributes automatically extracted from the benchmark when it was uploaded. Users can also perform a number of stateful operations on a space depending on the set of permissions they have on it. These operations may include creating and deleting subspaces; uploading, copying, linking and deleting benchmarks and solvers; creating, removing jobs; copying, removing users; making other users leaders; changing the space's default permissions; and locking or unlocking the space.

In general, users can copy any visible primitive from an unlocked space to any space where they have permission to add that kind of primitive. While copy operations are done uniformly by a drag-and-drop action, their semantics depends on the copied item. Copying a user U from a space A to a space B amounts to giving U certain permissions on B. Copying a job J from A to B only creates a link to J in B. Copying a benchmark or a solver, on the other hand, can be done either way: as an actual copy or just as a link. All users are by default *leaders* of the spaces they create. The leaders of a space have full permissions on that space, which consist in the ability to add/remove primitives and subspaces as well as make other users space leaders. A space leader can also decide which set of permissions from the above to grant to regular (i.e., non-leader) users in the space. For security and integrity reasons, only StarExec administrators can remove space leaders from a space or demote them to regular users.

Since communities are spaces, users can operate on them as discussed above. However, users can also browse the communities visible to them, request to join or leave a community, and download a community's benchmark processors. Community leaders (i.e., leaders of a community space) can, in addition, set several community-wide defaults such as timeouts or post-processors for job pairs. They can also upload post-processors, define benchmark types, and upload their associated processor.

Solvers StarExec users can add a solver to a space (in which they have the permission to do so) either by copying/linking the solver from another space or by uploading it. Solvers must be uploaded as tar archives or compressed tar or zip archives. The archives can consist of any collection of folders and files but must contain at least one configuration script—used to establish proper settings and input flags for the solver and then launch the solver. All configuration scripts must reside in a designated top-level directory (`bin`) in the archive and have a name starting with a designated prefix (`starexec_run_`) so that they can be easily identified by the StarExec system.

During a job execution, StarExec will run a configuration script in an environment with a number of predefined environment variables. Those variables contain such information as the absolute path to the benchmark input file, the absolute path to a designated output directory for the script, and limits on wall clock time, CPU time, memory, and disk space. Any files written to the output directory during the execution of the script are

saved by StarExec for later download by the user. Files written elsewhere are removed after each job run. If a job pair exceeds any of its time or space limits it is terminated.

As with any space item, users can inspect details of any solver visible to them. In particular, they can view the list and the content of each configuration script. If permitted by the owner of the containing space, they can also download the entire solver and its configurations as a compressed archive.

Benchmarks Users can add benchmarks to a space either by copying/linking them from another space or by uploading an archive file containing benchmarks all of the same type. The user has the option of (i) recreating the archive's directory structure as a space structure in the destination space and place each benchmark in the (sub)space corresponding to the benchmark's source (sub)directory, or (ii) simply placing all the files contained in the archive directly in the destination space. With the first option the new subspaces take the name of the corresponding subdirectory and all get the same set of permissions, specified by the user before uploading the archive. All uploaded files are run through the benchmark processor associated with their type, and are added to their destination space only if accepted by the processor. An upload status page summarizes the results of the upload as it progresses and lists all the discarded files, if any.

Benchmark types are global across all communities. They can be created only by community leaders who must also provide an associated processor. Every benchmark processor is expected to print for its input benchmark a sequence of lines of the form key=$value$ where key is an attribute name and $value$ its value. These attribute-value pairs are stored in StarExec's database and will be shown later when inspecting the benchmarks. Most attribute names are user-defined. A predefined attribute, and the only mandatory one, is `starexec-valid`. The processor is required to print a pair with that key and the value `true` or `false` depending on whether the benchmark was accepted by the processor or not. The system will discard the benchmark if the value of this attribute is `false`. Other predefined but optional attributes allow the processor to specify dependencies on other benchmarks or an expected result for the benchmark.

Compute Cluster StarExec compute nodes are partitioned into a number of execution queues. Each job is submitted to one of these queues. There is a general queue available to all users. The other queues serve special functions, such as solver competitions, or are reserved for exclusive use by a community. Only community leaders can request the creation and the reservation of a queue. A dedicated page on the web server allows all users to inspect any queue to see pending job pairs for that queue, and inspect any node in a queue to see currently executing job pairs on that node. To assure result reproducibility, each job pair is run in isolation on one compute node processor, i.e., no two job pairs share the same processor at the same time. To assure a basic level of security a job pair's solver is run as a *sandboxed* user with very limited permissions.

Jobs Users can create and immediately execute a job in a space. At creation time, they set execution parameters specifying timeouts, post-processors, and execution queues. They can choose to have job pairs executed in depth-first or in round-robin fashion. In the first case, StarExec will execute all job pairs in one subspace before moving on to the next; in the second, all job pairs in all subspaces will make progress in the execution concurrently. Users have different ways to generate a job pairs starting from the

root space, the space in which the job is created. For instance, they can instruct the system to find all subspaces containing solvers and benchmarks, and execute all possible combinations of those benchmarks and available configurations for those solvers. Alternatively, users can manually select which benchmarks and solvers to execute. In that case, they also have different options for how to pair solvers with benchmarks.

After a job has been set up, its job pairs are created automatically and sent to the specified execution queue. A running job can be monitored by looking at its details web page, which gets updated in real time with information on how many pairs have been completed, have been solved, have failed and so on. Specific job pair information includes its status, final runtimes, and user-defined results. After the job completes, its page will also provide various statistics in graphical form, such as scatter and cactus plots. All job data can be also downloaded in real time in CSV format (with one line per job pair) for off-line analysis, process and visualization. The job's web page is assigned a unique, persistent URL that can be used as a reference in publications and the like.

4 Infrastructure and Technologies

The StarExec software infrastructure relies on common web standards and several freely available software applications and libraries. Almost all of the software developed in-house for this service is written in Java, with the rest consisting mostly of shell scripts. We plan to open source the entirety of this software in the near future.

The StarExec hardware infrastructure is located in a dedicated state-of-the-art server hosting facility at the University of Iowa. The service runs on a Red Hat Fedora compute cluster consisting of 3 head nodes and 190 rack-mounted compute nodes. Each node has two 4-core 2.4GHz Intel processors with 256GB of RAM and a 1TB hard drive. Local disk space is used only for caching purposes during job execution. Solvers, benchmarks and all other persistent data and meta-data are stored centrally in a dual NetApp network-attached storage system with a capacity of 23TB.

5 Current Status and Future Development

At the time of this writing, StarExec has been used by two public events: the Confluence Competition (CoCo) 2013, which ran in June, 2013; and the SMT Evaluation (SMT-EVAL) 2013, which ran over several months in 2013. Each event had an execution queue giving it exclusive access to a subset of the compute nodes. Otherwise, they had quite different requirements for StarExec. CoCo 2013 ran during a meeting of the Confluence Workshop, and thus it was very important that the CoCo organizers could initiate the job and monitor its results live. They used StarExecCommand for this. On the other hand, their workload was small, just 509 job pairs. In contrast, SMT-EVAL 2013 had a workload of 1,663,478 job pairs, split over four jobs. This meant that operating at scale was an important criterion for success for SMT-EVAL.

For the FLoC Olympic Games of Summer 2014, many additional events have expressed interest or made plans to run on StarExec. The latter include the CASC, QBF, and SMT-COMP competitions. With its basic functionality now in place, we anticipate that future developments of the StarExec system and service will be driven by the

needs of such events or of whole communities. The challenge in adding new features to meet these needs will be to make them general enough to be potentially useful to other communities. For example, in some tracks of the Termination Competition, termination proofs produced by termination checkers for rewriting systems are fed on the fly to a proof checker. To support this we plan to add to StarExec a general facility for pipelining tools, something that we expect will be useful to other competitions as well.

Acknowledgements We would like to thank several people who have contributed in various capacities to the StarExec project so far. The following people were involved in the development of the software infrastructure at various stages of the project: E. Burns, T. Elvers, T. Jensen, W. Kaiser, B. McCune, M. Nassar, CJ Palmer, V. Sardeshmukh, S. Stark, and R. Zhang. Computer system support and assistance in designing and building the hardware infrastructure was provided by H. Brown, D. Holstad, J. Tisdale, and JJ Ulrich. Several people, from user communities and from the StarExec Advisor Board, provided useful feedback and input. A full list can be found on StarExec website.

References

[1] C. Barrett, M. Deters, L. de Moura, A. Oliveras, and A. Stump. 6 Years of SMT-COMP. *Journal of Automated Reasoning*, 50(3):243–277, 2012.

[2] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). http://www.SMT-LIB.org, 2010.

[3] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories*, 2010.

[4] D. Beyer. Competition on software verification. In *Proceedings of TACAS*, LNCS, pages 504–524. Springer, 2012.

[5] A. Biere and K. Claessen. Hardware model checking competition. In *Hardware Verification Workshop*, 2010.

[6] H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In *Proceedings of the 3rd Workshop on the Satisfiability Problem*, 2001. http://www.satlib.org/.

[7] D. Le Berre and L. Simon, editors. *Special Issue on the SAT 2005 Competitions and Evaluations*, volume 2. JSAT, 2006.

[8] C. Marché and H. Zantema. The termination competition. In *Proceedings of RTA 2007*, pages 303–313. Springer, 2007.

[9] R. Nieuwenhuis. Special Issue: The CADE ATP System Competition. *AI Communications*, 15(2-3), 2002.

[10] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The seventh QBF solvers evaluation (QBFEVAL'10). In *Proceedings of SAT 2010*, LNCS, pages 237–250. Springer, 2010.

[11] T. Raths, J. Otten, and C. Kreitz. The ILTP Problem Library for Intuitionistic Logic - Release v1.1. *Journal of Automated Reasoning*, 38(1-2):261–271, 2007.

[12] L. Simon and P. Chatalic. SatEx: A Web-based Framework for SAT Experimentation. In *Proceedings of SAT 2001*, volume 9 of *ENDM*, pages 129–149, 2001.

[13] A. Stump and M. Deters. SMT-Exec. http://www.smtexec.org.

[14] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[15] G. Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In *Proceedings of LPAR 2010*, volume 6355 of *LNAI*, pages 1–12. Springer, 2010.