SAT Heritage: A Community-Driven Effort for Archiving, Building and Running More Than Thousand SAT Solvers

Gilles Audemard[1(✉)], Loïc Paulevé[2], and Laurent Simon[2]

[1] CRIL, Artois University, Lens, France
audemard@cril.fr
[2] Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, 33400 Talence, France
{loic.pauleve,lsimon}@labri.fr

Abstract. SAT research has a long history of source code and binary releases, thanks to competitions organized every year. However, since every cycle of competitions has its own set of rules and an adhoc way of publishing source code and binaries, compiling or even running any solver may be harder than what it seems. Moreover, there has been more than a thousand solvers published so far, some of them released in the early 90's. If the SAT community wants to archive and be able to keep track of all the solvers that made its history, it urgently needs to deploy an important effort.

We propose to initiate a community-driven effort to archive and to allow easy compilation and running of all SAT solvers that have been released so far. We rely on the best tools for archiving and building binaries (thanks to Docker, GitHub and Zenodo) and provide a consistent and easy way for this. Thanks to our tool, building (or running) a solver from its source (or from its binary) can be done in one line.

1 Introduction

As Donald Knuth wrote in [11], "The story of satisfiability is the tale of a triumph of software engineering". In this success story of computer science, the availability of SAT solvers source code have been crucial. Archiving and maintaining this important amount of knowledge may be as important as archiving the scientific papers that made this domain. The release of the source code of `MiniSat` [6] had, for instance, a dramatic impact on the field. However, nothing has yet been done to ensure that source code and recipes to build SAT solvers will be archived in the best possible way. This is a recent but important concern in the more broadly field of computer science. The Software Heritage [3] initiative is, for instance, a recent and strong initiative to handle this. In the domain of SAT solvers, however, collecting and archiving may not be sufficient: we must embed the recipe to build the code and to run it in the most efficient way. As input format for SAT solvers remains the same since more than 25 years [4],

L. Pulina and M. Seidl (Eds.): SAT 2020, LNCS 12178, pp. 107–113, 2020.
https://doi.org/10.1007/978-3-030-51825-7_8

it is always possible to compare the performances of all existing solvers, given a suitable way of compiling and running them. At that time, some code was using EGCS, a fork of GCC 2.8 including more features. Facebook and Google didn't exist and Linux machines were running with kernels 1.X. Solvers were distributed with source code to be compiled on Intel or SPARC computers. Fortunately enough, binaries for Intel 386 machines distributed at that time are still executable on recent computers, given the availability of compatible libraries.

Collecting and distributing SAT solvers source code is, luckily, not new. SAT competitions, organized since the beginning of the 21st century, have almost always forced the publication of the source code of submitted solvers. If source code was not distributed, binaries were often available. However, since the first competitions, the landscape of computer science has changed a lot. New technologies like `Docker` [5] are now available, changing the way tools are distributed.

We propose in this work to structure and bootstrap a collective effort to maintain a comprehensive and user-friendly library of all the solvers that shaped the SAT world. We build our tool, called `SAT Heritage`, on top of other recent tools, typically developed for archiving and distributing source code and applications, like `Docker` [5], `GitHub` [8], `Guix` [9], `Zenodo` [22]. The community is invited to contribute by archiving, from now on, all the solvers used in competitions (and papers). We also expect authors of previous solvers to contribute by adding informations about their solvers or special command lines not especially used during competitive events. Our tool allows, for instance, to add a DOI (thanks to `Zenodo`) to the exact version of any solver used in a paper, allowing simple but powerful references to be used.

In summary, the goals of our open-source tool are to:

- Collect and archive all SAT solvers, binaries and sources,
- Easily retrieve a `Docker` image with the binary of any solver, directly from the `Docker` Hub, or, when source code is available, by locally building the image from the source code of the solver,
- Allow to easily run any SAT solver that have ever been available (typically in the last 30 years), by a one line call (consistent over all solvers),
- Open an convenient solution for reproducibility (binaries, source code and receipt to build binaries are archived in a consistent way), thanks to strong connection with tools like `Guix` and `Zenodo`.

2 History of SAT Solvers Releases and Publications

The first SAT competitions happened in the 90's [1,2]. Their goals were multiple: collect and compare SAT solvers performances in the fairest possible way, collect and distribute benchmarks, and also take a snapshot of the performances reached so far. Table 1 reports the number of SAT solvers that took part in the different competitions. We counted more than a thousand solvers, but even counting them was not an easy task: one source code can hide a number of subversions (with distinct parameters) and distinct tracks, and some information were only partially available.

Table 1. Number of solvers to the different competitions. Note that some solvers may be counted twice or more (some solvers did not change from year to the next or have been included in a competition as reference). (*) binaries and sources are available, but by navigating individually to each solver result. Different numbers indicate different organizers and different way of distributing results, source code (s) and binaries (b).

Date	#Solvers	Collection	Type	Date	#Solvers	Collection	Type
≤2000	24	Satex	s/b	2011	104	Contest (2)	s/b
2002	27	Contest (1)	b	2012	65	Challenge	-
2003	33	Contest (1)	b	2013	140	Contest (3)	s(*)/b(*)
2004	63	Contest (1)	b	2014	150	Contest (3)	s(*)/b(*)
2005	47	Contest (1)	b	2015	31	Race (2)	-
2006	16	Race (1)	-	2016	32	Contest (4)	s/b
2007	31	Contest (2)	s/b	2017	71	Contest (4)	s/b
2008	19	Race (1)	-	2018	66	Contest (4)	s/b
2009	64	Contest (2)	s/b	2019	55	Race (3)	s/b
2010	20	Race (1)	-	Total	1058		

Following the ideas of these first competitions organized in the 90's, and thanks to the development of the web, the `satex` [17] website published solvers and benchmarks gathered by the website maintainer. `satex` was running SAT solvers on only one personal computer. Some solvers were modified to comply with the input/output of the `satex` framework (like a normalized exit code value). It was a personal initiative, made possible by the relatively few solvers available (all solvers of the initial `satex` are available in our tool).

During the first cycle of competitions (numbered 1 in Table 1) [16], submitters had to compile a static binary of their solver (to prevent library dependencies) via remote access to the same computer. To ensure the deployment of their solver, this computer had the exact same Linux version as the one deployed on the cluster used to run the contest. Some solvers were coming from industry, which explains why no open source code was mandatory: the priority was to draw the most accurate picture of solvers performances. However, it was quickly decided (competitions numbered 2 in the above table) that it was even more important to require submitters to open their code. Binaries were then allowed to enter the competition, but only in the demonstration category (no prizes). More recently, thanks to the `starexec` environment [19], compilation of solvers was somehow normalized (an image of a virtual Linux machine on which the code would be built and run was distributed). With each cycle of competition or race, came its own set of rules with an *ad hoc* way of publishing source code and binaries, with a non uniform way of providing details on which parameters to use. For example, since 2016, solvers must provide a certificate for unsatisfiable instances [10,21]. One has thus to go through all the solvers to find the correct parameters for running them without proof logging.

Thus, despite the increasing importance of software archiving [3], the way SAT solvers are distributed had not really changed in the last 25 years. It is still mainly done via personal websites, or SAT competitions and races websites, each cycle of events defining its own rules for this. As a result, it is often unclear how to recover any SAT solver (same code, same arguments) used in many papers, old or recent. It is even more questionable whether, despite the importance of SAT solvers source code, we are able to correctly archive and maintain them.

3 SAT Heritage `Docker` Images

The `SAT Heritage` project provides a centralized repository of instructions to build and execute the SAT solvers involved in competitions since the early ages of SAT. To that aim, it relies on `Docker` images which are self-contained Linux-based environments to execute binaries. `Docker` allows to explicitly mention all the packages needed to compile the source code and to build a temporary image (the "builder") for compiling the solver. Then, the compiled solver is embedded in another, lighter, image which contains only the libraries required to execute it. So, each version of each collected solver is made available in a dedicated `Docker` image. Thanks to the layer structure of images, all solvers sharing the same environment will share the major part of the image content, thus substantially saving disk space. At the end, the `Docker` image will not be much heavier than the binary of the solver.

`Docker` images can be executed on usual operating systems. On Linux, `Docker` offers the same performance as native binaries: only filesystem and network operations have a slight overhead due to the isolation [7], which is not of concern for SAT solvers. On other systems, the images are executed within a virtual machine, adding a noticeable performance overhead, although considerably reduced on recent hardware [7].

3.1 Architecture

The instructions to build and run the collected solvers are hosted publicly on `GitHub` [13], on which the community is invited to contribute.

The solvers are typically grouped by year of competition. Images are then named as `satex/<solver-name>:<year>`.

The images are built by compiling solver sources whenever available. The compiling environment matches with a Linux distribution of the time of the competition. We selected the Debian GNU/Linux distribution which provides `Docker` images for each of its version since 2000. For instance, the solvers from the 2000 competition are built using the Debian "Potato" as it was back at that time. In principle, each solver can have its own recipe and environment for building and execution. Nevertheless, we managed to devise `Docker` recipes compatible with several generations of competitions. The architecture of the repository also allows custom sets of solvers. For example, the `SAT Heritage` collection includes the different Knuth's solvers or solvers with Java or Python.

The image building `Docker` recipes indicate where to download the sources or the binaries whenever the former are not available. At the time of the writing of this article, most recipes use URL from the website of the SAT competitions. In order to provide as most as persistent locations as possible, we are regularly moving more resources on `Zenodo` services to host sources and binaries in a near future [15] (currently, only the binaries of the original `satex` and the 2002's competition are hosted on it).

The images can be built locally from the git repository, and are also available for download from the main public `Docker` repository [14], that distributes "official" binaries of solvers. This allows to directly run any collected (or compiled) solver very quickly.

3.2 Running Solvers

We provide a Python script, called `satex`, which eases the execution and management of available `Docker` images, although images can be directly run without it. The script can be installed using `pip` utility: `pip3 install -U satex`.

The list of available solvers can be fetched using the command `satex list`.

We provide a generic wrapper in each image giving a unified mean to invoke the solver: a DIMACS file (possibly gzipped) as first argument, and optionally an output file for the proof:

```
# run a solver on a cnf file
satex run cadical:2019 file.cnf
# run and produce a proof
satex run glucose:2019 file.cnf proof
```

The `satex info` command gives, together with general information on the solver and the image environment, the specific options used for the run. Alternatively, custom options can be used with the `satex run-raw` command. If the image has not been built locally, it will attempt to fetch it from the online `Docker` repository. See the `satex -h` for other available commands, such as extracting binaries out of `Docker` images and invoking shells within a given image.

3.3 Building and Adding New Solvers

The building of images, which involve the compilation of the solvers when possible, also relies on `Docker` images, and thus only requires `Docker` and Python for the `satex` command. The following command, executed at the root of the `sat-heritage/docker-images` repository, will build the matching solvers with their adequate recipe:

```
satex build '*:2000' # build all 2000 solvers
```

Sets of solvers are added by specifying which `Docker` recipes to use for building the images and how to invoke the individual solvers. Managing sets of solvers allows sharing common configurations (such as linux distributions, compilers and so on) for docker images. A complete and up-to-date documentation can be found in the README file of the repository.

4 Ensuring Reproducibility

Reproducibility is a corner stone of science. In computer science, it recently appealed for significant efforts by researchers, institutions and companies to devise good practices and provide adequate infrastructures. Among the numerous initiatives, Software Heritage [3,18] and `Zenodo` [12,22] are probably the most important efforts for archiving source code, repositories, datasets, and binaries, for which they provide persistent storage, URLs, and references (DOI). Another example is the `GitHub` Archive Program, a repository on a 500-years lifespan storage preserved in the Artic World Archive [20]. Created more recently, the `Guix` [9] initiative aims at keeping the details of any Linux machine configuration, thanks to a declarative system configuration. External URL used for building any image are also archived. Our tool produces `Docker` images that can be easily frozen thanks to `Guix`, by building `Guix` images from the Dockerfile recipe. It is also worth mentioning that `Guix` has strong connections with Software Heritage and `GitHub`.

If we look at reproducibility of SAT solvers experiments on a longer time scale, we can expect that, some day, current binaries (for i386) will not genuinely run on computers any more. We can expect, however, that there will be i386 emulators. Once such an emulator is set up, we can also expect `Docker` to be available on it, and then all the images we built will be handled natively. If not, as `Docker` recipes are plain text, it will be easy to convert them to another framework.

Therefore, facilitating the accessibility of software in time now boils down to simple habits, such as using source versioning platforms, taking advantage of services like `Zenodo` or Software Heritage to freeze packages dependencies, source code, binaries, and benchmarks, and provide `Docker` images to give both environments and recipes to build and run your software.

5 Conclusion

We presented a tool for easily archiving and running all SAT solvers produced so far. Such a tool is needed because of (1) source code and experiments are crucial for the SAT community and (2) there are already too many SAT solvers produced so far, with many different ways of publishing sources.

In order to complete our tool we think at further improvements, like including `Docker` images for compiling SAT solvers for other architectures than i386 (ARM for instance), but also initiating another important effort for the community: including `Docker` images for benchmarks generations and maintenance. Many benchmarks are combinatoric ones, typically generated by short programs. These generators are generally not distributed by the different competitive events and may contain a lot of information on the structure of the generated problems. We also think that our tool could be very interesting for SAT solvers configurations and easy cloud-deployment in a portfolio way. We also expect our work to give the community the best possible habits for state of the art archiving and reproducibility practices.

References

1. Buro, M., Buning, H.K.: Report on a SAT competition. Technical report (1992)
2. Crawford, J.: International competition and symposium on satisfiability testing (1996)
3. Di Cosmo, R., Zacchiroli, S.: Software heritage: why and how to preserve software source code. In: International Conference on Digital Preservation, pp. 1–10 (2017)
4. Second challenge on satisfiability testing organized by the center for discrete mathematics and computer science of Rutgers University (1993)
5. https://www.docker.com
6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
7. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 171–172 (2015)
8. https://www.github.com
9. https://guix.gnu.org
10. Heule, M., Hunt Jr., W.A., Wetzler, N.: Trimming while checking clausal proofs. In: Formal Methods in Computer-Aided Design, FMCAD, pp. 181–188 (2013)
11. Knuth, D.E.: The art of computer programming, vol. 4, p. iv, Fascicle 6 (2015)
12. Peters, I., Kraker, P., Lex, E., Gumpenberger, C., Gorraiz, J.I.: Zenodo in the spotlight of traditional and new metrics. Front. Res. Metrics Anal. **2**, 13 (2017)
13. https://github.com/sat-heritage/docker-images
14. https://hub.docker.com/u/satex
15. https://zenodo.org/communities/satex
16. Simon, L., Le Berre, D., Hirsch, E.A.: The SAT2002 competition report. Ann. Math. Artif. Intell. **43**(1), 207–342 (2005)
17. Simon, L., Chatalic, P.: SATEx: a web-based framework for SAT experimentation. Electron. Notes Discret. Math. **9**, 129–149 (2001)
18. https://www.softwareheritage.org
19. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: a cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. LNCS (LNAI), vol. 8562, pp. 367–373. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08587-6_28
20. Thorkildsen, M., Sjøvik, J.-F., Bryde, B.: Preserving irreplaceable national digital cultural heritage in the arctic world archive. In: Archiving Conference, vol. 2019, pp. 39–41. Society for Imaging Science and Technology (2019)
21. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 422–429. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_31
22. https://zenodo.org