OpenSAT: An Open Source SAT Software Project

Gilles Audemard[1], Daniel Le Berre[1], Olivier Roussel[1]⋆, Inês Lynce[2], and João Marques-Silva[2] ⋆⋆

[1] CRIL/CNRS, Université d'Artois, Lens, France
{audemard,leberre,roussel}@cril.univ-artois.fr
[2] IST/INESC-ID/CEL, Technical University of Lisbon, Portugal
{ines,jpms}@sat.inesc.pt

Abstract. This paper presents the OpenSAT project, an open source framework of SAT algorithms in Java. The objective of this framework is twofold: first, to provide some specifications about SAT solver services for a better interoperability between SAT solvers and SAT powered applications, and second, to give a reference implementation of different specifications, such that people can plug into their applications precisely what they need. Moreover, the OpenSAT framework is designed in a way that allows several levels of compliance with the framework. This paper also includes results for the OpenSAT reference implementation, as well as for *JQuest2*, the first OpenSAT-compliant state-of-the-art SAT solver.

1 Introduction

SAT solvers have seen dramatic improvements in recent years [1, 3, 5, 7, 8, 10], being currently capable of tackling large and hard real-world problem instances, and being actively applied in industrial settings. Nevertheless, the improvements in SAT solvers also motivated users of SAT technology to devise more complex applications, that yield problem instances of ever increasing size and hardness. As a result, applied research in SAT solver technology is currently a very active field, with new ideas and promises being proposed at a very fast pace.

This scenario poses a few difficulties when evaluating the actual merits of new SAT solvers. Besides differing data structures and coding styles, each existing SAT solver implements its own set of search techniques, strategies and heuristics. Hence, to experimentally evaluate the different algorithms, in a controlled experiment that ensures that only specific differences are evaluated, a dedicated SAT solving framework is needed. Because of the size of the current instances (tens of thousands of variables, and hundreds of thousands of clauses[9]), a less careful

⋆ This work has been supported in part by the IUT de Lens, the CNRS and the Region Nord/Pas-de-Calais under the TACT Programme.

⋆⋆ The OpenSAT project is a joined project between SAT@INESC and CRIL partly supported by an "action intégrée luso-française de la Conférence des Présidents d'Université".

implementation (e.g., a $O(n^2)$ algorithm instead of a $O(n \log(n))$ algorithm) can make a dramatic difference in running time. So building a SAT solver competitive to current state-of-the-art SAT solvers is getting more and more difficult for a newcomer, as it is difficult for each SAT researcher to keep up-to-date his own research prototype. We believe that providing core data structures and algorithms needed for building a SAT solver on an open source basis will provide to newcomers facilities to develop their ideas on a competitive implementation. We also believe that sharing the maintenance effort of building code blocks for SAT solvers on an open source basis will allow research prototypes to stay up-to-date with industrial SAT solvers. With this purpose, we developed OpenSAT, a Java SAT framework that can be used to conduct unbiased experimental evaluations of SAT algorithms and techniques. But OpenSAT is more than a SAT framework. OpenSAT is a software development project that aims the development of open source SAT algorithms and their dissemination to the SAT community. With OpenSAT, one can obtain reference implementations of SAT algorithms and techniques. Moreover, OpenSAT enables to developing competitive state-of-the-art open source SAT solvers. A similar proposal was made in [2] but it seems that there was no follow-up. Furthermore, we hope that providing this framework for SAT solvers will promote interoperability of SAT solvers in SAT based applications. In addition, people willing to can make their own implementation of the OpenSAT specifications in C or C++.

2 Code Organization

OpenSAT interfaces establish the signature of the methods for implementing OpenSAT-compliant solvers. They define the two OpenSAT compliance specifications:

1. the user level, which defines basic SAT services, usually sufficient for SAT powered applications.
2. the developper level, which allows a more detailled control of the data structures and algorithms.

Figure 1 illustrates how easy it is to create a SAT-powered Java application using the OpenSAT framework first level of specification.

Those interfaces are quite stable right now. They can be safely used for a light OpenSAT compliance. Moreover, additional specialized interfaces can be added depending on each SAT solver functionalities.

The OpenSAT reference implementation (`RefImpl`) is a proof of concept designed using the following rationale: the code should respect as much as possible object oriented design guidelines, while being reasonably efficient. Building a new SAT solver must be straigthforward by reusing or extending existing codebase. We guarantee an implementation that provides the services defined by the interfaces. We do not guarantee it is the most efficient one.

The current design is not stable yet. However, we expect to have a better version with the first release of the framework.

```
// org.OpenSAT .Default contains default implementations
// for an easy start up with the framework.
// default simple data structure for representing a CNF
IFormula formula = Default.cnf();
// create a parser for the Dimacs format.
IParser parser = Default.parser();
// get a solver
ISolver solver = Default.solver();
// read the instance
parse.parseInstance("filename.cnf",formula);
try {
  if (solver.solve(formula)) {
    // instance is SAT
  } else {
    // instance is UNSAT
  }
} catch (TIMEOUTException e) {
  // TIMEOUT
}
```

Fig. 1. SAT powered application using OpenSAT .

3 OpenSAT Project

From the very beginning, the project was planned to be open source, that is, people will be allowed to get it, modify it and distribute it freely. In order to allow people from industry to use it, we decided to use the more permissive GNU Lesser GPL license: people can use the OpenSAT framework in a project without having to disclose the source of that project (but improvements of the OpenSAT framework for that particular project must be released if the project is distributed).

The project inherits from former projects from each research group, namely JQuest, a Java based solver that can be viewed as framework of SAT algorithms, that implements a significant number of the most well-known SAT techniques, and JSAT [3], an open source implementation of a few SAT solvers in Java. Those two projects were complementary: the first one is using sophisticated data structures and advanced techniques, but suffered from a high *C+* (C programming with C++ data structures) inheritance which make its code difficult to reuse. On the other hand, the JSAT project proposed a better design in terms of reusability but was using very naive data structures.

3.1 Downloading OpenSAT

The first rule when working on an open source project is to announce the project only when people have something to play with. That was our primary goal. The

[3] http://cafe.newcastle.edu.au/daniel/JSAT

framework contains a few simple algorithms and heuristics to play with. The distribution is composed of 2 jar files: one for the OpenSAT project itself, and another one for the command line utility framework from the Jakarta project [4].

The simplest way to try it is to download the jar files from `http://www.opensat.org` and to save them in the same directory. Then use a recent Java virtual machine (Sun j2se 1.4.1 for instance): `java -jar opensat.jar <instance name>`. The default time out is 30s, you can change it from the command line: `java -jar opensat.jar -t 120 <instance name>`. The other command line options are described using the `-h` option.

3.2 Using OpenSAT

The first level is to use the openSAT framework within your own project. If you are working on a Java program that needs a SAT solver, OpenSAT is well-suited for you. You simply need to write a few lines of code to build the formula to be solved from a DIMACS formatted file and to launch the solver. The main advantage of that approach is that you can easily build a SAT powered application, directly benefit from any improvements in the OpenSAT framework, and your are free to build your own SAT solver afterwards, if the current code is not efficient enough. In Figure 1 is shown how it can be done. That code is quite simple and allow anybody to add SAT solving capabilities to its own application in a few minutes.

While using openSAT, you may encounter bugs or *strange features*. Then a good idea would be to send a bug report using OpenSAT bugzilla [5] server at `http://www.opensat.org/bugzilla`.

Afterward, the second level is to use a closer integration with the framework.

1. One can think about writing his own formula builder (e.g. for SAT based planning, a parser that takes a STRIPS description and creates a CNF from it).
2. One can think about writing his own data structure: a lightweight way to do it is just to implement the basic interfaces defined in the `org.opensat` package. There is almost no constraints on the code this way, but then there is almost no code to reuse. The other way around is to implement the interfaces available in the `org.opensat.data` package defining a simple data structure for SAT (where literals, clauses and CNF are objects). Then the data structure can be used with all of the available algorithms.
3. The third and certainly most interesting thing for a researcher is to build its own SAT solver. It can be done using the existing formula data structure, in which case it will probably be straightforward (but not necessarily efficient).

It may (or will) happen that you need some extra features in OpenSAT to be able to fully reach your goal. Then filling a feature request in bugzilla is the right way to have your wish done.

---

[4] `http://jakarta.apache.org`

[5] `http://www.bugzilla.org`

Another way to participate in the OpenSAT project it to take a look at the project design and implementation in detail. We designed openSAT using our former experience on JQUEST and JSAT, and with an extreme programming approach: the design is based only on the functionality we need right now, not the ones we may need later. It means every now and then we have to refactor heavily some parts of the code, but it is straightforward with a good programming environment (we use Eclipse [6], which has built-in refactoring support).

One of the major issues in this project is to build a framework able to match various kind of techniques: local search algorithms, DPLL-like algorithms, with or without learning capabilities, with or without randomization, etc. The main difficulty is to find the right abstraction for a family of related solvers, and to make a good tradeoff between genericity and efficiency.

3.3 Contributing to OpenSAT

One of the main goals of the current implementation was to have something that works, something that people can begin to play with. We adopted a whole object oriented point of view (everything is object: literals, clauses, formulas). On the one hand, it clarifies the code. On the other hand, object creation is costly since memory allocation must be done at run time (a small overhead). So a tradeof must be found between object oriented design and efficiency. We are beginning to investigate some solutions (we have some non object oriented implementations providing us a goal to reach in terms of efficiency).

We also welcome people having a strong experience in Java programming. Our first goal is to provide a truly object oriented SAT framework in a pure Java mindset. Our second goal is to provide a Java framework competitive with state of the art C/C++ frameworks.The key ideas to achieve it: a clean design and good data structures.

One of the educational aspects of OpenSAT is to propose a Java implementation of state-of-the-arts SAT techniques very similar to the pseudo algorithms published. So ideally each algorithm should contain references to papers, and even better a little description of the algorithm itself.

There are mainly two ways to manage contributions to an open source project:

– Everybody can commit its changes to the current code repository.
– A group of selected people are allowed to commit their changes directly to the code repository. Other people need to send their code to one member of this group.

For the moment, we decided to choose the second approach. This strategy will assure the correctness of the added code.

[6] `http://www.eclipse.org`

4 Current Status

The latest stable revision of OpenSAT is `0.44`, and we are quickly progressing toward revision `0.5` (beta version). As mentioned earlier, we built OpenSAT using an extreme programming approach: the design is adapted while we are adding new functionality to the framework. For this reason, the basic services are quite stable now, even though the more specific services are subject to change.

The next release of OpenSAT – `release 1.0` – should also include local search algorithms[4], tuned data structures and good documentation. We will appreciate any help from the community to help us reaching the 1.0 status by summer.

OpenSAT is currently distributed with a reference implementation. The reference implementation is using watched literals, backjumping, learning of small clauses (having strictly less than 5 literals) with no clause deletion, a simple implementation of the VSIDS heuristic, and search restarts (beginning at 100 conflicts and being updated with a factor of 1.5). Those choices were made to study the effect of the VSIDS heuristic[8] with limited learning, which is not possible with GRASP/Chaff/JQuest2 since in those solvers, learning is used for simulating backtracking.

In addition, OpenSAT Java interfaces allow to develop *compliant* solvers. For example, JQuest2 is a new backtrack search SAT solver, which can be run as an OpenSAT-compliant solver. JQuest2 implements new lazy data structures [6], backjumping with clause recording and opportunistic deletion of large clauses, the VSIDS heuristic and search restarts.

Table 1. Results on benchmark examples.

Class of Instances	#	OpenSAT				`Chaff`		`Ratio`	
		`RefImpl`		`JQuest2`					
		Time	X	Time	X	Time	X	`RefImpl`	`JQuest2`
sss-sat-1.0	100	72015.77	45	929.73	0	301.09	0	293.18	3.09
vliw-sat-1.0	100	115678.68	76	14050.59	0	6271.24	0	18.44	2.24
vliw-sat-1.1	100	116394.43	76	12648.07	0	6588.11	0	17.67	1.42
fvp-unsat-1.0	4	4508.79	3	1071.16	0	731.12	0	6.17	1.47
bounded model checking	34	18417.69	10	8954.27	0	3383.24	0	5.44	2.64
equivalence checking	25	23176.40	14	4953.91	2	3636.46	2	6.37	1.36
beijing	16	3788.14	2	3453.96	2	3150.05	2	1.20	1.10

Table 1 compares the results obtained for OpenSAT `Reference Implementation` (version `0.44`), `JQuest2` (OpenSAT-compliant solver) and `Chaff` [8]. as well as the ratios between `ReferenceImplementation/Chaff` and `JQuest2/Chaff` is provided. This comparison was performed on several hard, real-world classes of instances [7]. For the results shown a P-IV@1.7 GHz Linux machine with 1 GByte of physical memory was used. The Java Virtual Machine used was SUN's HotSpot JVM for JDK1.4. The CPU time was limited to 1500 seconds. For each aborted instance (X), 1500s were added to the total time.

From the above table of results, which include CPU time for solving some of the most difficult classes of instances, one can conclude that we have already a competitive framework. Observe that `JQuest2` tends to be less than a factor of 2 slower than `Chaff`. Hence, `JQuest2` performance is very very close to Chaff's. Consequently, *it is possible to build a Java SAT solver competitive with state-of-the-art C/C++ SAT solvers.* Note that `JQuest2` and `Chaff` share a lot of concepts, while `RefImpl` doesn't use most of the learning and memory management techniques that are known to be sucessful on that type of instances. OpenSAT is already in our opinion a useful platform for prototyping new data structures and new algorithms.

Finally, observe that we have ignored instances that can be easily solved by most of the existing SAT solvers. We simply wanted to prove that our framework is already competitive for solving hard real-world instances. Moreover, we believe that contributions to the OpenSAT project will help us to achieve more robust and competitive SAT solvers.

5 Conclusions

Recent solvers based on the satisfiability test for propositional formulas use more and more elaborated data structures. This paper has presented a framework named OpenSAT which aims at providing the community of scientists and the community of users with the most recent data structures and algorithms for SAT. To allow anybody to reuse and modify these tools, the OpenSAT framework is built on an object-oriented langage, widely used and portable : Java.

OpenSAT has two goals: the first one is to promote the use of SAT solvers and the second one is to promote the development of new algorithms by providing the most recent techniques such as watched literals, intelligent backtracking, clauses learning... Besides, OpenSAT is a common framework for SAT solvers which allows a fair comparison of the various algorithms.

This goals are already almost reached. The web site `http://www.opensat.org` allows anyone to download the latest version of the project, to keep informed of the developments around the project, to submit bug reports and requests for enhancement. Two SAT solvers linked to OpenSAT have been submitted to the SAT 2003 competition : JQUEST2 [6] and the reference implementation.

Anybody may benefit from the common effort of developpement around OpenSAT and may share with the community the best of his techniques by joining the open source development used for this project.

[7] All the instances are available from `http://www.satlib.org/` (SATLIB web site), with the exception of the *formal verification* instances.

This framework is open and doesn't restrict itself to the development of SAT solvers. It has already been used to develop a solver for QBF formulas (OpenQBF) and is currently extended to support BDD representations.

The OpenSAT framework is only a few months old. Its growth is likely to require a few changes in its conception, but undoubtedly, the support of the community will give birth to a fully mature project in a short time. *You* can help !

Project home page:
`http://www.opensat.org`
To be kept up-to-date, use the OpenSAT mailing lists:
`http://www.opensat.org/mailman/listinfo`
To submit a bug or a feature request, use our bugzilla server:
`http://www.opensat.org/bugzilla`

References

1. R. Bayardo Jr. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the National Conference on Artificial Intelligence*, pages 203–208, July 1997.
2. A. Tacchella E. Giunchiglia, M. Narizzano and M. Vardi. Towards an efficient library for sat: a manifesto. In Henry Kautz and Bart Selman, editors, *Electronic Notes in Discrete Mathematics*, volume 9. Elsevier Science Publishers, 2001.
3. E. Goldberg and Y. Novikov. BerkMin: a fast and robust sat-solver. In *Proceedings of the Design and Test in Europe Conference*, pages 142–149, March 2002.
4. H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.
5. C. M. Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 341–355, October 1997.
6. I. Lynce and J. Marques-Silva. On implementing more efficient data structures. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, May 2003.
7. J. P. Marques-Silva and K. A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design*, pages 220–227, November 1996.
8. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, pages 530–535, June 2001.
9. L. Simon, D. Le Berre, and E. Hirsch. The SAT2002 Competition. Technical report, August 2002. http://www.satlive.org/SATCompetition/2002/onlinereport.
10. H. Zhang. SATO: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction*, pages 272–275, July 1997.