

SBT: SatBox with Theories

Hantao Zhang*, Haiou Shen, John Wheeler

Computer Science Department

The University of Iowa

Iowa City, IA 52242

{hzhang, hshen, wheeer}@cs.uiowa.edu

1 Introduction

Recent advances in SAT solvers based on CNF have focused, in particular, on innovative techniques for Boolean constraint propagation (BCP), variable selection, and back-jumping with conflict-driven learning. These techniques have proved to be critical for the high performance of SAT solvers. A great effort by the SAT community has been made to present these SAT techniques clearly and neatly so that it will be easier for people to implement them in other systems. However, it is still a great challenge to implement these SAT techniques as efficient as the top SAT solvers, as it involves a lot of complicated details and needs a lot of fine-tuning.

Recently, we have found a new way of the reuse of the advanced SAT techniques for other applications. The basic idea is to create a library of functions for the interactive use of SAT solvers. We call this library of functions **SatBox**, which hides the details of SAT techniques. To show the power of **SatBox**, we have implemented decision procedures based on the framework of $\text{DPLL}(\mathcal{T})$ [4, 3]. The implemented system is called **SBT**, standing for **SatBox** with Theories. Written in C, **SBT** uses some code of Oliveras' $\text{DPLL}(\text{T})$ [1] and we are very grateful for his generosity. The current version of **SBT** supports the theories of EUF (equality with uninterpreted functions) and LIA (linear integer arithmetic). We are still in the process of debugging and adding more theories into **SBT**. We are also looking for serious applications of **SBT**.

2 SatBox: Hiding Details of SAT Solvers

The main motivation for the development of **SatBox** [6] is to easily intertwine the processes of partial model generation in a SAT solver and validation in the theory \mathcal{T} in the framework of $\text{DPLL}(\mathcal{T})$. **SatBox** does everything, such unit propagation and conflict-directed learning, as in the standard DPLL method, except the splitting rule. As a result, **SatBox** is not a complete search tool. **SatBox** has a standard interface which is defined as a set of library functions. All the state-of-the-arts SAT solvers can be modified easily to implement these library functions and support such an interface, so that any progress made in the development of SAT solvers can be brought into **SatBox**.

Currently, **SatBox** provides seven basic functions and requires two outside functions. We already have a working version of **SatBox** and its code is publicly available [6]. In addition to improve the performance of **SatBox** we plan to add more functions into **SatBox**, to meet the need of $\text{DPLL}(\mathcal{T})$ for other theories. The current implementation of **SatBox** is based on the code of **SATO** [5]. We plan to modify the publicly available code of high-performance SAT solvers, such as **Chaff**, **Limmat**, and **Jerusat**, to obtain different implementations of **SatBox**.

*Partially supported by the National Science Foundation under Grant CCR-0098093.

3 SBT: SatBox with Theories

The first version of SBT supports the theory of EUF (equality with uninterpreted functions). SBT has used the congruence closure algorithm of Nieuwenhuis and Oliveras [3] to support EUF.

In [1], Ganzinger et al. reported that DPLL(T) can be a very efficient tool for circuit verification. Table 1 shows the computing times of BerkMin [2], one of the fast SAT solvers, Ganzinger et al.'s DPLL(T) [1], and SBT0.1, our implementation of DPLL(EUF), on several benchmarks generated from the circuit verification problem. It is clear from the table that SBT0.1 outperforms both BerkMin and DPLL(T) on all the benchmarks except one case. Note also that the performance of SBT0.1 on pure SAT problems is comparable to the best SAT solvers such as Chaff and BerkMin [2] on most problems.

To participate the SMT competition, we have added recently the theory of linear integer arithmetic (LIA) into SBT0.2. As SBT becomes mature, more modules will be added for handling commonly used data structures, including finite bit vectors, finite arrays and finite sets. We will select decision procedures one by one and use them in the framework of DPLL(\mathcal{T}).

SBT0.2 will participate in the following areas: QF_UF, QF_LIA, QF_UFLIA, QF_IDL, QF_UFIDL.

Table 1: Experimental results on benchmarks for verification (in seconds).

Benchmark family	BerkMin	DPLL(T)	SBT0.1
Buggy Cache	2.4	6.7	1.71
Code Validation Suite	44.9	3.7	0.82
DLX Processor	10.2	1.2	0.3
Elf Processor	5882	575	99
Out of order proc.(rf)	18211	6385	2692
Out of order proc.(tag)	247	1979	665
Load-store processor	51.4	30.3	25.9
Cache coherence prot.	4151	3601	975
Two queues	407	73.6	31.2

References

- [1] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(\mathcal{T}): Fast decision procedures. Proceedings of CAV'04.
- [2] E. Goldberg, and Y. Novikov: BerkMin: A fast and robust SAT solver. In *Design, Automation, and Test in Europe (DATE'02)*, 2002.
- [3] R. Nieuwenhuis and A. Oliveras. Congruence closure with integer offsets, *10th Intl. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, LNAI 2850, 78-90, 2003.
- [4] C. Tinelli. A DPLL-based calculus for ground satisfiability modulo theories, *8th European Conf. on Logics in Artificial Intelligence*, LNAI 2424, 308–319, 2002.
- [5] H. Zhang, SATO: An efficient propositional prover, Proc. *International Conference on Automated Deduction (CADE-97)*, 308–312, LNAI 1104, Springer-Verlag, 1994.
- [6] H. Zhang. The SatBox Library <http://www.cs.uiowa.edu/~hzhang/satbox/>, 2005