LEARNSAT: A SAT Solver for Education

Moti Ben-Ari

http://www.weizmann.ac.il/sci-tea/benari/

© 2017 by Moti Ben-Ari.

1 Introduction

A SAT solver is a program that searches for an assignment of truth values to the atoms of a propositional formula in conjunctive normal form (CNF) that satisfies the formula [1, Chapter 6]. If no such assignment exists, the SAT solver will report that the formula is unsatisfiable. Since all $\mathcal{NP}$-complete problems can be encoded in a CNF formula and the encoding is relatively simple to perform, an efficient implementation of SAT solving can be applied to find solutions to many practical problems. For this reason, Donald Knuth calls SAT solvers a "killer app" [3].

The literature on SAT solving is extensive: the *Handbook of Satisfiability* [2] of almost one thousand pages covers theory, algorithms and applications, and Knuth's 300-page Section 7.2.2.2 contains over 500 exercises [3]. Since SAT solvers are widely used, it is essential that quality learning materials be available for students, even those who do not intend to become researchers, for example, undergraduate students taking a course in mathematical logic. Such learning materials will also be helpful for people using SAT solvers in applications. Instructors should be enabled to create learning materials that demonstrate the central algorithms in detail.

LEARNSAT is a SAT solver designed for educational use according to the following criteria:

- A detailed trace of the algorithm's execution is displayed. Its content can be set by the user.

- Implication graphs and assignment trees are generated automatically.

- The implementation is concise, easy to understand and well documented.

- The software is easy to install and use.

- The input to the program is a formula in clausal form written in a readable symbolic form. Programs to convert to and from DIMACS format are provided.

- SAT solving is explained in a detailed tutorial and many example programs are provided.

2 The LEARNSAT SAT solver

LEARNSAT implements the core algorithms of many modern SAT solvers: DPLL with conflict-driven clause learning (CDCL) and non-chronological backtracking (NCB). LEARNSAT can be run in three modes—plain DPLL, DPLL with CDCL, and DPLL with both CDCL and NCB—so that the student can examine the improvements obtained by each refinement. CDCL is implemented by backwards resolution from a conflict clause to a unique implication point (UIP). It is also possible to compute dominators in the implication graph although this computation is just displayed and not used. The user can specify the order in which literals are assigned. In addition, two heuristics for lookahead can be selected.

3 The output of LEARNSAT

The key to learning sophisticated algorithms like SAT solving is a trace of the step-by-step execution of the algorithm. The user of LEARNSAT can choose *any* subset of 24 display options in order to tailor the output to a specific learning context. The display options include elementary steps like decision assignments, unit propagations and identifying conflict clauses, as well as the advanced steps of CDCL: the resolution steps used to obtained a learned clause and the search for UIPs. The Appendix shows the (default) output for the example in [5] run in NCB mode.

LEARNSAT can generate two types of graphs that are rendered using the DOT tool: trees showing the search through the assignments (Figure 1) and implication graphs that display the process for learning clauses from conflicts (Figure 2). These graphs incrementally.

4 Examples

The LEARNSAT archive includes the examples used in [4, 5, 6] to help students read these articles. The archive includes encodings of the following problems:

- Graphical combinatorics: graph coloring, Tseitin graphs.

- Games: Sudoku, 4-queens, colored queens.

- Number theory: Ramsey theory, Schur triples, Langford's and van der Waerden's problems.

- Constrained arrangements: pebbling formulas, seating guests at a table.
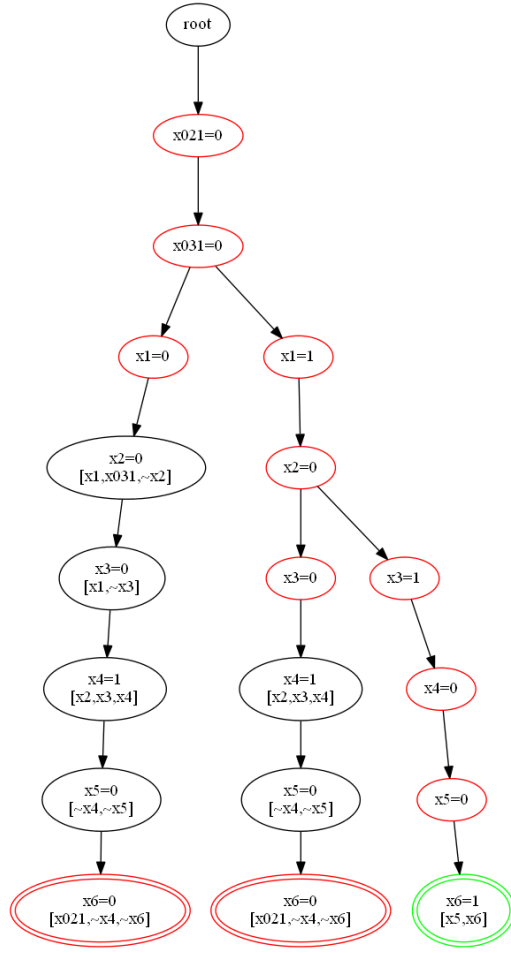
- Bounded model checking.
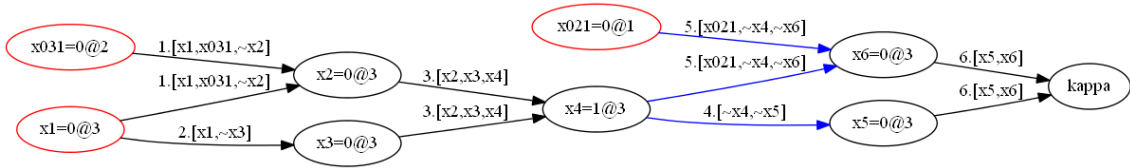
Figure 1: Assignment tree for DPLL mode

Figure 2: Implication graph for CDCL mode

3

The list of clauses for the 2-hole pigeonhole problem is:

```
hole2 :- dpll([
  [p11, p12],   [p21, p22],   [p31, p32],    % Each pigeon in hole 1 or 2
  [~p11, ~p21], [~p11, ~p31], [~p21, ~p31],  % No pair is in hole 1
  [~p12, ~p22], [~p12, ~p32], [~p22, ~p32],  % No pair is in hole 2
  ], _).
```

5   Implementation

LEARNSAT is implemented in PROLOG which was chosen because PROLOG programs are extremely concise: the core algorithms take only about 250 lines. The SWI-PROLOG compiler was used: it is available for the systems used by students, such as Windows and Mac. It is easy to install and has a comprehensive reference manual.

6   Documents

LATEX and PDF files are provided for the LEARNSAT User's Guide and Software Documentation and the LEARNSAT Tutorial.

References

[1] M. Ben-Ari. *Mathematical Logic for Computer Science (Third Edition)*. Springer, 2012. ISBN 978-1-4471-4129-7.

[2] A. Biere, M. Heule, H. Van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009. ISBN 978-1-60750-376-7.

[3] D.E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Pearson, 2015. ISBN 978-0-1343-9760-3.

[4] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, 2009. http://dx.doi.org/10.1145/1536616.1536637.

[5] J. P. Marques-Silva, I. Lynce, and S. Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. In Biere et al. [2], 2009. http://dx.doi.org/10.3233/978-1-58603-929-5-131.

[6] J. P. Marques-Silva and K. A. Sakallah. Grasp: A search algorithm for propositional satisfiability. volume 48, pages 506–521, 1999. http://dx.doi.org/10.1109/12.769433.

A Output for the example in [5]

```
LearnSAT (version 2.0)
Decision assignment: x021=0@1
Decision assignment: x031=0@2
Decision assignment: x1=0@3
Propagate unit: ~x2 derived from: 1. [x1,x031,~x2]
Propagate unit: ~x3 derived from: 2. [x1,~x3]
Propagate unit: x4 derived from: 3. [x2,x3,x4]
Propagate unit: ~x5 derived from: 4. [~x4,~x5]
Propagate unit: ~x6 derived from: 5. [x021,~x4,~x6]
Conflict clause: 6. [x5,x6]
Not a UIP: two literals are assigned at level: 3
Clause: [x5,x6] unsatisfied
Complement of: x5 assigned true in the unit clause: [~x4,~x5]
Resolvent of the two clauses: [x6,~x4] is also unsatisfiable
Not a UIP: two literals are assigned at level: 3
Clause: [x6,~x4] unsatisfied
Complement of: x6 assigned true in the unit clause: [x021,~x4,~x6]
Resolvent of the two clauses: [x021,~x4] is also unsatisfiable
UIP: one literal is assigned at level: 3
Learned clause: [x021,~x4]
Non-chronological backtracking to level: 1
Skip decision assignment: x1=1@3
Skip decision assignment: x031=1@2
Decision assignment: x021=1@1
Decision assignment: x031=0@2
Decision assignment: x1=0@3
Propagate unit: ~x2 derived from: 1. [x1,x031,~x2]
Propagate unit: ~x3 derived from: 2. [x1,~x3]
Propagate unit: x4 derived from: 3. [x2,x3,x4]
Propagate unit: ~x5 derived from: 4. [~x4,~x5]
Propagate unit: x6 derived from: 6. [x5,x6]
Satisfying assignments:
[x021=1@1,x031=0@2,x1=0@3,x2=0@3, x3=0@3,x4=1@3,x5=0@3,x6=1@3]
Statistics: clauses=6,variables=8,units=10,decisions=6,conflicts=1
```