LearnSAT: A SAT Solver for Education

Mordechai (Moti) Ben-Ari

Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel
http://www.weizmann.ac.il/sci-tea/benari/

Abstract. The extensive research on SAT solving and the development of software for applications have not been matched by the development of educational materials for introducing students to this field. LEARNSAT is a SAT solver designed for educational purposes. It implements the DPLL algorithm with CDCL and NCB. LEARNSAT produces detailed output of the execution of the algorithms. It generates assignment trees and the implication graphs of CDCL which are rendered by DOT. LEARNSAT is written in PROLOG so that the algorithms are concise and easy to read.

Keywords: education, CDCL SAT solver, Prolog.

1 Introduction

The literature on SAT solving is extensive: the *Handbook* [2] of almost one thousand pages covers theory, algorithms and applications. 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. Furthermore, these demonstrations should use real problems in place of the artificial examples that appear in research papers.

LEARNSAT is a SAT solver designed for educational use. Design considerations include: (a) the student and instructor should be provided with maximum flexibility in specifying the trace output when running the SAT solver; (b) it should be simple to install and run on the vanilla computers used by students (Windows and Mac); (c) given the wide variety of programming languages taught to undergraduates, the program should be usable with only a superficial knowledge of a particular language; (c) the source code should be concise and very well documented so that advanced students can easily understand it.

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).

M. Järvisalo and A. Van Gelder (Eds.): SAT 2013, LNCS 7962, pp. 403–407, 2013.

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. The user can specify the order in which literals are assigned. 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.

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 25 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 [4] run in NCB mode.

LEARNSAT can generate two types of graphs that are rendered using the DOT tool (Figure 1): a tree showing the search through the assignments and the implication graphs that display the process for learning clauses from conflicts.[1] It is also possible to generate these graphs incrementally after each step in the algorithm that modifies the graphs.

4 Examples

The LEARNSAT archive includes the examples used in [3,4,5] to help advanced students read these articles. The archive also includes encodings of the following problems: (i) 4-queens,[2] (ii) Tseitin clauses associated with the graphs $K_{2,2}$ and $K_{3,3}$,[3] (iii) two and three-hole pigeonhole problems, and (iv) two- and three-level grid pebbling. With experience we will learn which of these problems is best for educational purposes.

The input to the program is a formula in clausal form written in a readable symbolic form; the 2-hole pigeonhole problem is:

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

[1] In the Figure, the default color decoration for decision and conflict nodes has been changed to bold for black-and-white printing.

[2] The encoding and its solution by DPLL are explained in detail in [1, Section 6.4].

[3] See [1, Section 6.5].

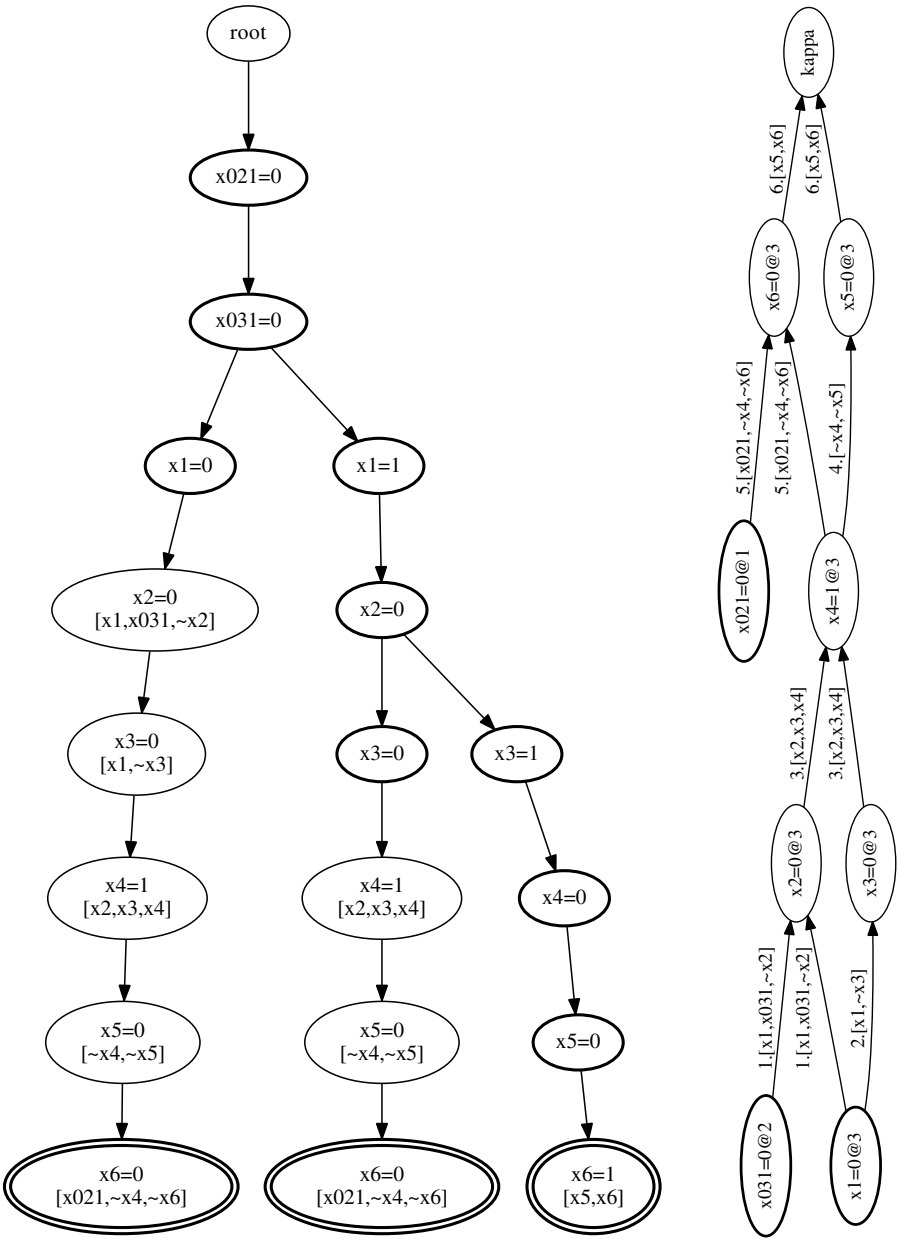

Fig. 1. Assignment tree for DPLL mode (left) and implication graph for NCB mode (right)

A program is provided to convert from DIMACS format to this symbolic form (and conversely) in order to faciliate the student's transition to more advanced SAT solvers.

5 Implementation

LEARNSAT is implemented in PROLOG which was chosen because PROLOG programs are extremely concise: the core algorithms take only 150 lines. The source code itself reads almost like pseudo-code (although making extensive modifications to the code would require mastery of the language). Furthermore, students are likely to know some PROLOG since it is often taught in a course on logic. Finally, the widely used high-quality SWI-PROLOG compiler is distributed with installers for Windows and Mac. Its default interface is minimal and easy to use.

The source code is extensively commented and the documentation in the archive includes: a user's guide, a tutorial using the example from [4], and documentation of the software.

6 Future Plans

LEARNSAT is not meant as a research tool nor even to train graduate students in the latest implementation techniques (MinSAT and Sat4j are more appropriate for this). Instead, the focus of future development will be on improving the pedagogical aspects of the tool. This will include expanding the user interface and the graphical features, and—perhaps more important—developing extensive tutorials. The tutorials will be based on standard puzzles like the 4-queens and hopefully also on actual applications.

7 Availability

LEARNSAT is open source and is available at `http://code.google.com/p/mlcs/`.

Acknowledgements. I would like to thank the anonymous referees for their comments and suggestions concerning LEARNSAT.

References

1. Ben-Ari, M.: Mathematical Logic for Computer Science, 3rd edn. Springer (2012)
2. Biere, A., Heule, M., Van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press (2009)
3. Malik, S., Zhang, L.: Boolean satisfiability from theoretical hardness to practical success. Commun. ACM 52(8), 76–82 (2009)
4. Marques-Silva, J.P., Lynce, I., Malik, S.: Conflict-Driven Clause Learning SAT Solvers. In: Biere, et al. (eds.) [2], ch. 4, pp. 131–153 (2009)
5. Marques-Silva, J.P., Sakallah, K.A.: GRASP—a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, pp. 220–227 (1996)

A Output for the Example in [4]

```
LearnSAT v1.3.2. Copyright 2012-13 by Moti Ben-Ari. GNU GPL.
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
```