

SAT Solving with Conflict Driven Clause Learning

William Schultz

CS 7240 Final Project

May 1, 2022

Overview and Project Goals

- Satisfiability is the canonical NP-complete problem.
- Much work has been devoted to building efficient SAT solvers over last decades.
- **Project Goal:** Implement a basic SAT solver based on *conflict driven clause learning* (CDCL), the dominant core technique used in modern solvers.
 - ▶ Gain a deeper understanding of the DPLL and CDCL based algorithms for SAT solving
 - ▶ Use as a platform for potentially exploring new SAT solving techniques
 - ▶ E.g. learning heuristics using a data-driven approach, extending methods of *CrystalBall* [SKM19]

Review: The SAT Problem

The SAT problem:

Given a boolean formula in conjunctive normal form (CNF), determine whether there exists an assignment to the variables of the formula that makes the overall formula true.

Review: The SAT Problem

The SAT problem:

Given a boolean formula in conjunctive normal form (CNF), determine whether there exists an assignment to the variables of the formula that makes the overall formula true.

e.g.

$$(x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1)$$

Review: The SAT Problem

The SAT problem:

Given a boolean formula in conjunctive normal form (CNF), determine whether there exists an assignment to the variables of the formula that makes the overall formula true.

e.g.

$$(x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1)$$

SAT, with $\{x_1 = 1, x_2 = 0, x_3 = 0\}$.

Review: The SAT Problem

The SAT problem:

Given a boolean formula in conjunctive normal form (CNF), determine whether there exists an assignment to the variables of the formula that makes the overall formula true.

e.g.

$$(x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1)$$

SAT, with $\{x_1 = 1, x_2 = 0, x_3 = 0\}$.

CNF notation:

$$\{\{x_1, x_2\}, \{\neg x_3, \neg x_1\}\}$$

DPLL: SAT as Search

- A basic approach to solving SAT is to view it as a search problem over possible assignments.

DPLL: SAT as Search

- A basic approach to solving SAT is to view it as a search problem over possible assignments.
- This is the basis of the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [DLL62]

DPLL: SAT as Search

- A basic approach to solving SAT is to view it as a search problem over possible assignments.
- This is the basis of the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [DLL62]
- Basic idea of DPLL is to do a depth first, brute force search with backtracking along with some basic formula simplification as you go.

DPLL: SAT as Search

- A basic approach to solving SAT is to view it as a search problem over possible assignments.
- This is the basis of the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [DLL62]
- Basic idea of DPLL is to do a depth first, brute force search with backtracking along with some basic formula simplification as you go.
 - ▶ Also employs the *unit propagation rule*

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg d\}\}$$

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg d\}\}$$

$$\{\{\neg d\}\}$$

Unit Propagation

- Core simplification rule employed in DPLL, and also in CDCL as we will see later.
- A *unit clause* is a clause that contains exactly one literal.
- If a CNF formula contains a unit clause then we can apply unit propagation i.e. set that literal to the appropriate truth value to satisfy its clause e.g.

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{b\}, \{\neg b, \neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

$$\{\{\neg c\}, \{c, \neg d\}\}$$

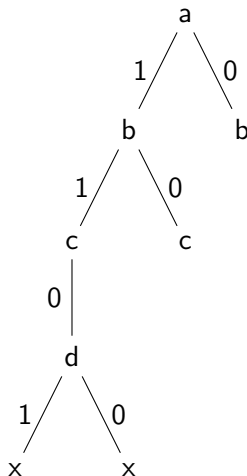
$$\{\{\neg d\}\}$$

$$\{\{\neg d\}\}$$

$$\{\} \quad (\text{SAT})$$

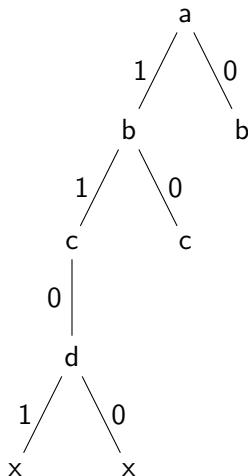
DPLL: Example

$\{\neg a, b\}$
 $\{\neg b, \neg c\}$
 $\{c, \neg d\}$



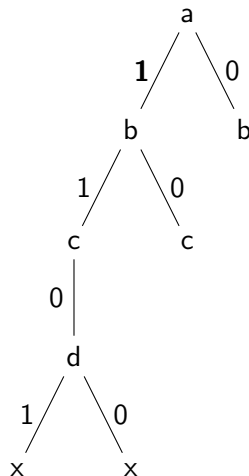
DPLL: Example

$\{\neg a, b\}$
 $\{\neg b, \neg c\}$
 $\{c, \neg d\}$



DPLL: Example

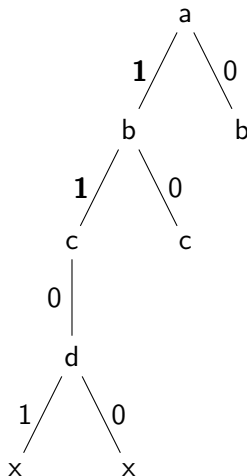
$\{\neg a, b\}$
 $\{\neg b, \neg c\}$
 $\{c, \neg d\}$



DPLL: Example

$\{\neg a, b\}$
 $\{\neg b, \neg c\}$
 $\{c, \neg d\}$

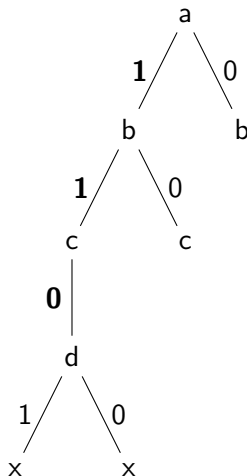
unit propagate b



DPLL: Example

$\{\neg a, b\}$
 $\{\neg b, \neg c\}$
 $\{c, \neg d\}$

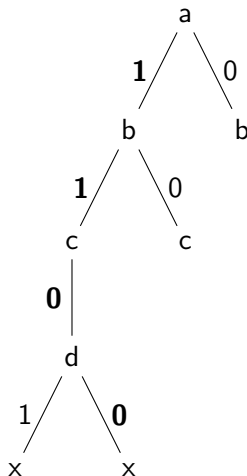
unit propagate $\neg c$



DPLL: Example

$\{\neg a, b\}$
 $\{\neg b, \neg c\}$
 $\{c, \neg d\}$

unit propagate $\neg d$



Beyond DPLL: Learning from Conflicts

- DPLL is a relatively naive algorithm

Beyond DPLL: Learning from Conflicts

- DPLL is a relatively naive algorithm
- An extension to this basic framework is to *learn from conflicts*

Beyond DPLL: Learning from Conflicts

- DPLL is a relatively naive algorithm
- An extension to this basic framework is to *learn from conflicts*
- When you encounter a conflict in the search tree, *learn* a clause that prevents you from making the similar mistakes again

Beyond DPLL: Learning from Conflicts

- DPLL is a relatively naive algorithm
- An extension to this basic framework is to *learn from conflicts*
- When you encounter a conflict in the search tree, *learn* a clause that prevents you from making the similar mistakes again
- This fundamental approach is known as *conflict-driven clause learning* (CDCL) and started being employed in SAT solvers around the late 90s and early 2000s.

Beyond DPLL: Learning from Conflicts

- DPLL is a relatively naive algorithm
- An extension to this basic framework is to *learn from conflicts*
- When you encounter a conflict in the search tree, *learn* a clause that prevents you from making the similar mistakes again
- This fundamental approach is known as *conflict-driven clause learning* (CDCL) and started being employed in SAT solvers around the late 90s and early 2000s.
- In addition, employ *non-chronological backtracking*

CDCL Example

- $c_1 \quad \{a, b\}$
- $c_2 \quad \{b, c\}$
- $c_3 \quad \{\neg a, \neg x, y\}$
- $c_4 \quad \{\neg a, x, z\}$
- $c_5 \quad \{\neg a, \neg y, z\}$
- $c_6 \quad \{\neg a, x, \neg z\}$
- $c_7 \quad \{\neg a, \neg y, \neg z\}$

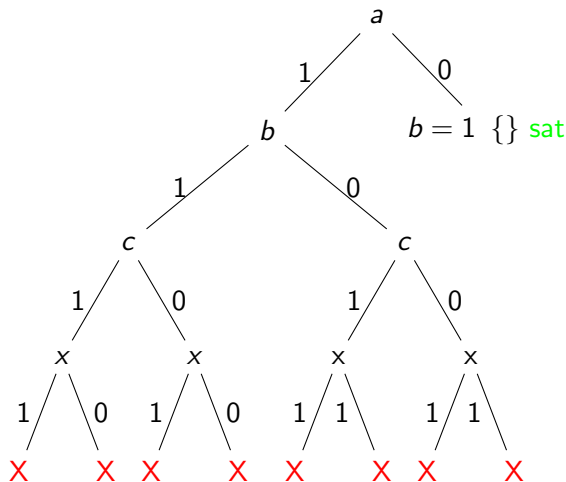


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

- $c_1 \quad \{a, b\}$
- $c_2 \quad \{b, c\}$
- $c_3 \quad \{\neg a, \neg x, y\}$
- $c_4 \quad \{\neg a, x, z\}$
- $c_5 \quad \{\neg a, \neg y, z\}$
- $c_6 \quad \{\neg a, x, \neg z\}$
- $c_7 \quad \{\neg a, \neg y, \neg z\}$

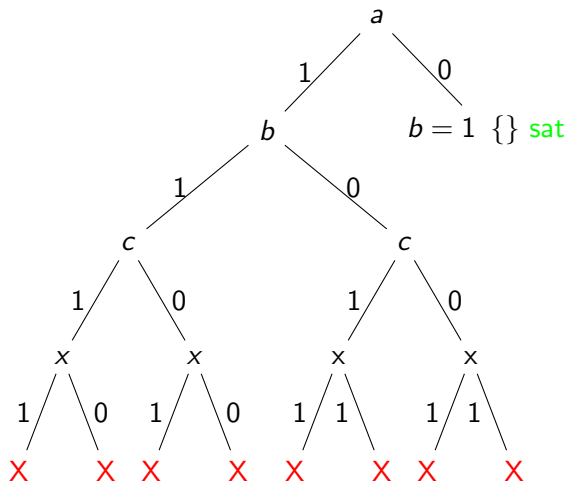


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

- $c_1 \quad \{a, b\}$
- $c_2 \quad \{b, c\}$
- $c_3 \quad \{\neg a, \neg x, y\}$
- $c_4 \quad \{\neg a, x, z\}$
- $c_5 \quad \{\neg a, \neg y, z\}$
- $c_6 \quad \{\neg a, x, \neg z\}$
- $c_7 \quad \{\neg a, \neg y, \neg z\}$

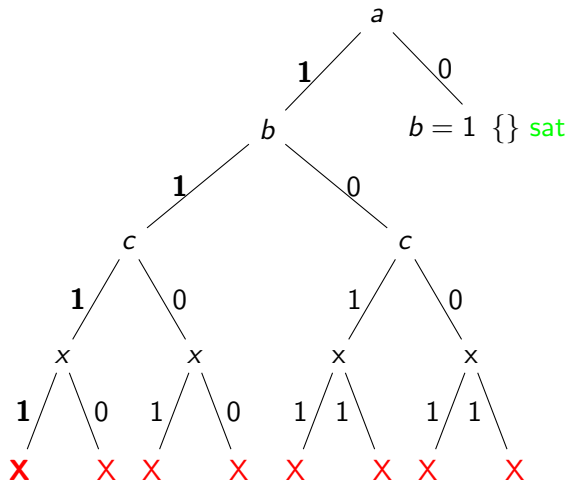


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

- $c_1 \quad \{a, b\}$
- $c_2 \quad \{b, c\}$
- $c_3 \quad \{\neg a, \neg x, y\}$
- $c_4 \quad \{\neg a, x, z\}$
- $c_5 \quad \{\neg a, \neg y, z\}$
- $c_6 \quad \{\neg a, x, \neg z\}$
- $c_7 \quad \{\neg a, \neg y, \neg z\}$

unit propagate y

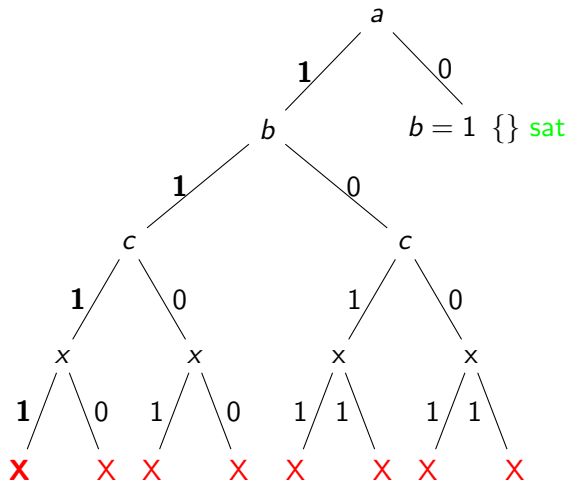


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

- $c_1 \quad \{a, b\}$
- $c_2 \quad \{b, c\}$
- $c_3 \quad \{\neg a, \neg x, y\}$
- $c_4 \quad \{\neg a, x, z\}$
- $c_5 \quad \{\neg a, \neg y, z\}$
- $c_6 \quad \{\neg a, x, \neg z\}$
- $c_7 \quad \{\neg a, \neg y, \neg z\}$

unit propagate z

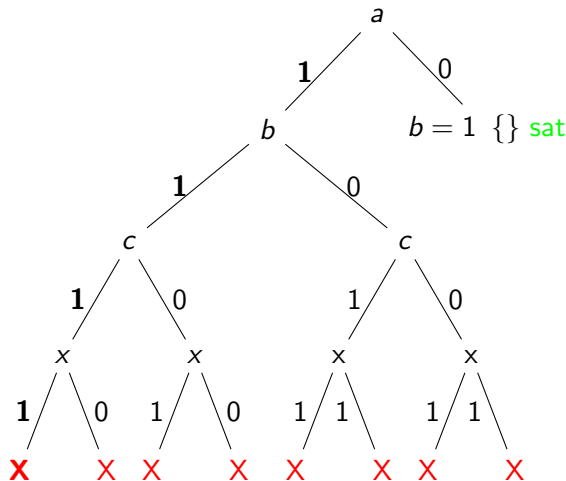


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

- c_1 $\{a, b\}$
- c_2 $\{b, c\}$
- c_3 $\{\neg a, \neg x, y\}$
- c_4 $\{\neg a, x, z\}$
- c_5 $\{\neg a, \neg y, z\}$
- c_6 $\{\neg a, x, \neg z\}$
- c_7 $\{\neg a, \neg y, \neg z\}$

Note that b and c are irrelevant to the c_7 conflict. $(a \wedge x)$ or $(a \wedge y)$ are sufficient.

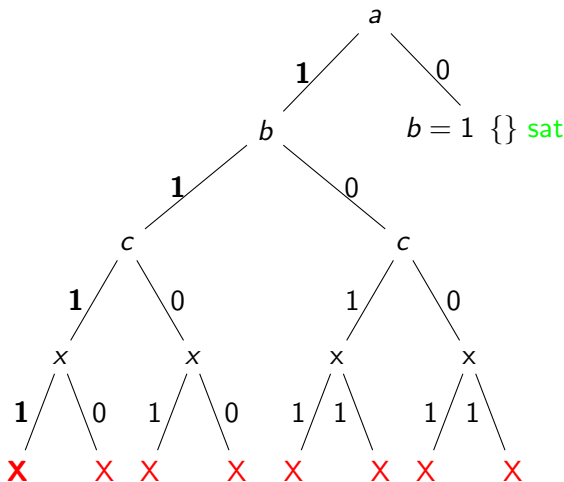


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

- $c_1 \quad \{a, b\}$
- $c_2 \quad \{b, c\}$
- $c_3 \quad \{\neg a, \neg x, y\}$
- $c_4 \quad \{\neg a, x, z\}$
- $c_5 \quad \{\neg a, \neg y, z\}$
- $c_6 \quad \{\neg a, x, \neg z\}$
- $c_7 \quad \{\neg a, \neg y, \neg z\}$

So, we can learn
 $\neg(a \wedge x) = (\neg a \vee \neg x)$
as a new constraint
i.e. a *learned clause*.

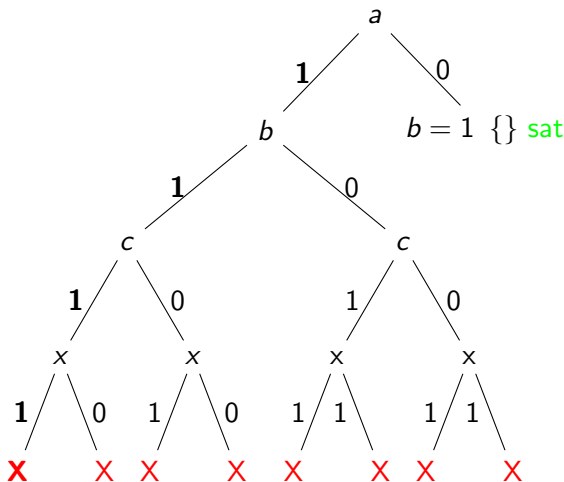


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

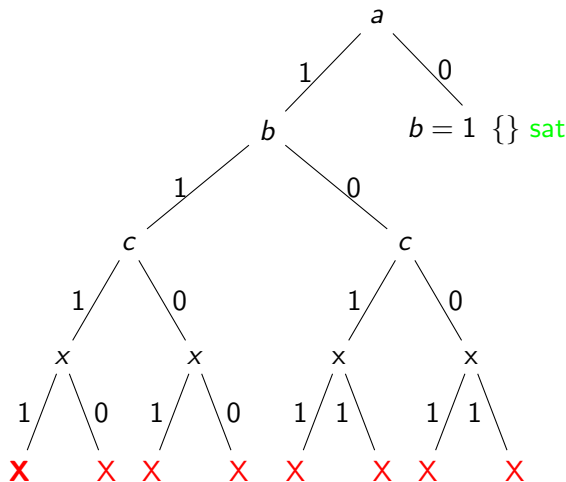


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

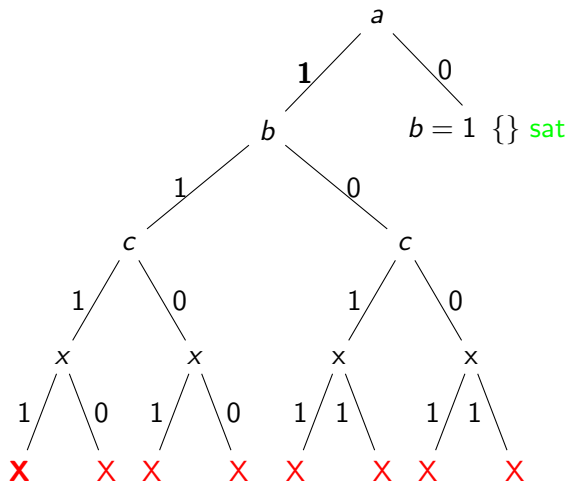


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

With the learned clause, we come to the conflict quickly.

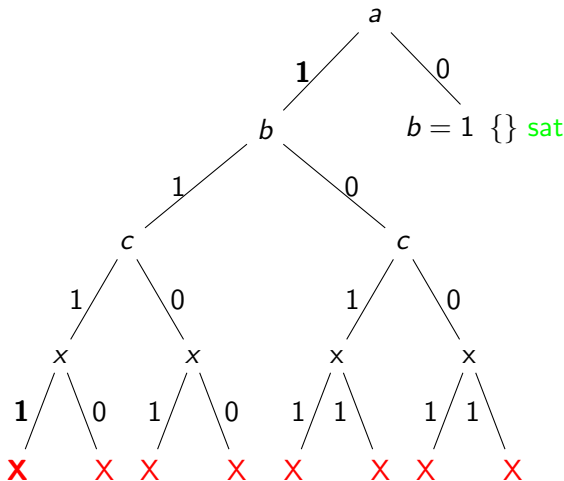


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

With the learned clause, we come to the conflict quickly.

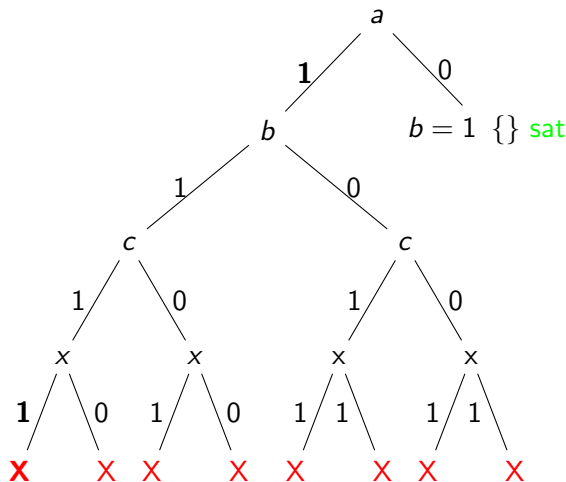


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

This time, a is sufficient to cause the conflict, so we learn $\neg a$.

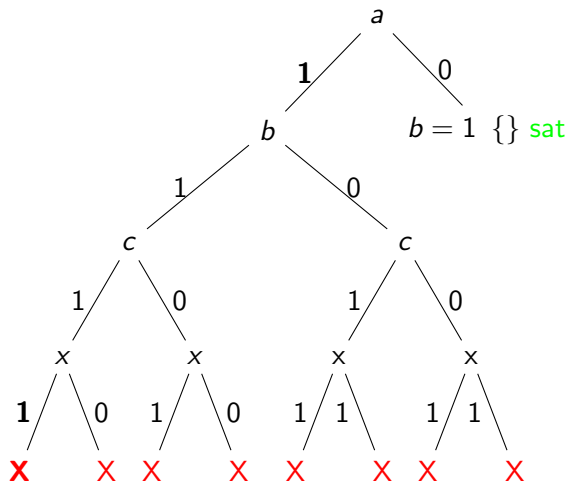


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_2	$\{\neg a\}$
l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

With $l_2 = \neg a$, we now get out of the unfruitful search space region.

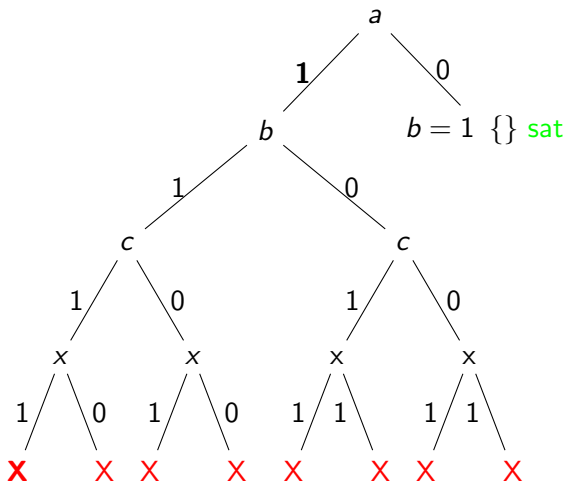


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_2	$\{\neg a\}$
l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

With $l_2 = \neg a$, we now get out of the unfruitful search space region.

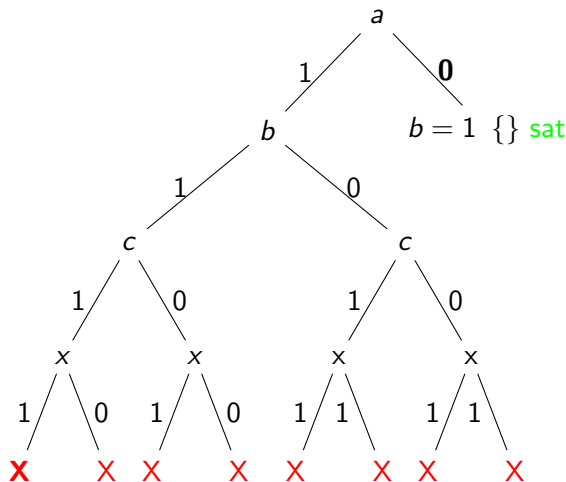


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL Example

l_2	$\{\neg a\}$
l_1	$\{\neg a, \neg x\}$
<hr/>	
c_1	$\{a, b\}$
c_2	$\{b, c\}$
c_3	$\{\neg a, \neg x, y\}$
c_4	$\{\neg a, x, z\}$
c_5	$\{\neg a, \neg y, z\}$
c_6	$\{\neg a, x, \neg z\}$
c_7	$\{\neg a, \neg y, \neg z\}$

With $l_2 = \neg a$, we now get out of the unfruitful search space region.

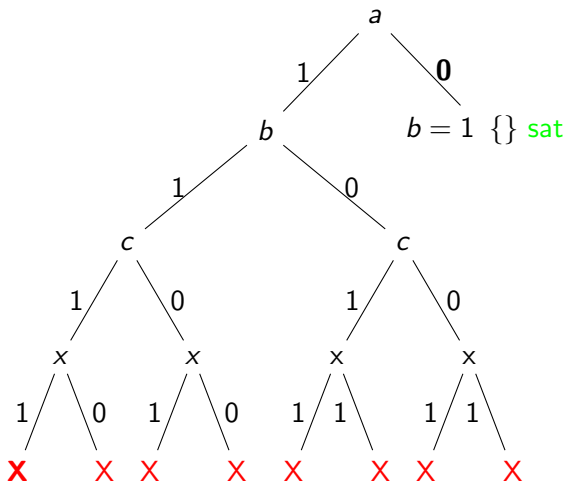


Figure: Basic DPLL termination tree. Explores large portion of left search tree.

CDCL: Implication Graph

- In general, can represent the propagation of variable assignments in an *implication graph*

CDCL: Implication Graph

- In general, can represent the propagation of variable assignments in an *implication graph*
 - ▶ Nodes of this graph represent variable assignments in the current search path. Edges to dependencies between these assignments.
 - ▶ Cuts in the graph correspond to a conflict set and, by negating it, a potential clause to learn

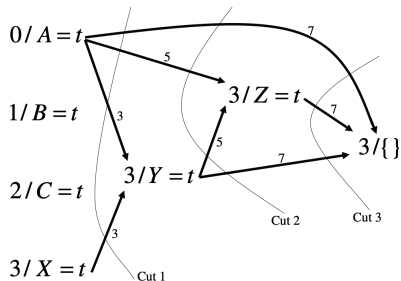


Figure 3.8. Three cuts in an implication graph, leading to three conflict sets.

SAT Solver Implementation

- Implementing my own CDCL SAT solver as a framework for exploring future potential SAT enhancements

SAT Solver Implementation

- Implementing my own CDCL SAT solver as a framework for exploring future potential SAT enhancements
- Around 1500 lines of C++, tested on a variety of easy to medium SAT benchmark problems

SAT Solver Implementation

- Implementing my own CDCL SAT solver as a framework for exploring future potential SAT enhancements
- Around 1500 lines of C++, tested on a variety of easy to medium SAT benchmark problems
 - ▶ <https://github.com/will62794/mysat>

SAT Solver Implementation

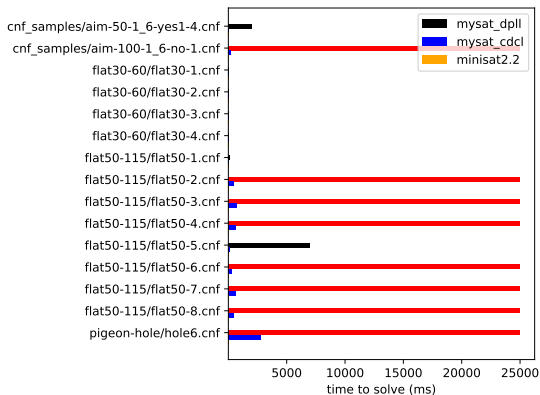
- Implementing my own CDCL SAT solver as a framework for exploring future potential SAT enhancements
- Around 1500 lines of C++, tested on a variety of easy to medium SAT benchmark problems
 - ▶ <https://github.com/will62794/mysat>
- Still order of magnitude slower than modern solvers (e.g. MiniSAT)

SAT Solver Implementation

- Still order of magnitude slower than modern solvers (e.g. MiniSAT [ES04]), but is improvement on basic DPLL

SAT Solver Implementation

- Still order of magnitude slower than modern solvers (e.g. MiniSAT [ES04]), but is improvement on basic DPLL
- e.g. runtime on some benchmarks with ≈ 50 -200 variables, time budget of 25 seconds (red bar indicates timeout)



Future Extensions and Learning Heuristics

- Modern CDCL based SAT solvers employ many heuristics
 - ▶ Variable ordering
 - ▶ Learned clause deletion policies
- Often these are “expertly tuned” heuristics

CrystalBall

- *CrystalBall* [SKM19]: Possible to learn better heuristics from data on SAT solver executions?

CrystalBall

- *CrystalBall* [SKM19]: Possible to learn better heuristics from data on SAT solver executions?
- Learned clause deletion is an important heuristic for CDCL based SAT solvers

CrystalBall

- *CrystalBall* [SKM19]: Possible to learn better heuristics from data on SAT solver executions?
- Learned clause deletion is an important heuristic for CDCL based SAT solvers
- Use data from SAT runs to train a model

CrystalBall

- *CrystalBall* [SKM19]: Possible to learn better heuristics from data on SAT solver executions?
- Learned clause deletion is an important heuristic for CDCL based SAT solvers
- Use data from SAT runs to train a model
- DRAT resolution proofs serve as a good source of data

Future Goals

- Implement support for resolution proof output in UNSAT cases
- Capture more statistics from solving runs as a basis for learning new heuristics

Questions?



Martin Davis, George Logemann, and Donald Loveland.

A machine program for theorem-proving.

Commun. ACM, 5(7):394–397, jul 1962.



Niklas Eén and Niklas Sörensson.

An extensible sat-solver.

In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.



Mate Soos, Raghav Kulkarni, and Kuldeep S. Meel.

CrystalBall: Gazing in the Black Box of SAT Solving.

In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019*, pages 371–387, Cham, 2019.

Springer International Publishing.