CS228 : Logic for Computer Science

Module 2: The Problem of Satisfiability

Lecture 10: Clause Learning and the CDCL Algorithm

Instructor: S. Akshay

IIT Bombay, India

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and Semantics
   - Questions of interest: Satisfiability, Validity

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and Semantics
   - Questions of interest: Satisfiability, Validity
2. Formal proof systems
   - Natural Deduction
   - Soundness and Completeness

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and Semantics
   - Questions of interest: Satisfiability, Validity
2. Formal proof systems
   - Natural Deduction
   - Soundness and Completeness
3. Normal Forms
   - Negation, Conjunctive and Disjunctive Normal Forms (NNF, CNF, DNF)
   - Tseitin's Encoding

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and Semantics
   - Questions of interest: Satisfiability, Validity
2. Formal proof systems
   - Natural Deduction
   - Soundness and Completeness
3. Normal Forms
   - Negation, Conjunctive and Disjunctive Normal Forms (NNF, CNF, DNF)
   - Tseitin's Encoding

Module 2: The Problem of Satisfiability

1. Propositional Resolution - soundness, termination and completeness

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and Semantics
   - Questions of interest: Satisfiability, Validity
2. Formal proof systems
   - Natural Deduction
   - Soundness and Completeness
3. Normal Forms
   - Negation, Conjunctive and Disjunctive Normal Forms (NNF, CNF, DNF)
   - Tseitin's Encoding

Module 2: The Problem of Satisfiability

1. Propositional Resolution - soundness, termination and completeness
2. The DPLL algorithm

Topics Covered till date

Module 1: Propositional Logic

1. Syntax and Semantics
   - Questions of interest: Satisfiability, Validity
2. Formal proof systems
   - Natural Deduction
   - Soundness and Completeness
3. Normal Forms
   - Negation, Conjunctive and Disjunctive Normal Forms (NNF, CNF, DNF)
   - Tseitin's Encoding

Module 2: The Problem of Satisfiability

1. Propositional Resolution - soundness, termination and completeness
2. The DPLL algorithm
3. Today: clause learning, the CDCL algorithm, and modern SAT solvers

A Quick Recap: DPLL

A DPLL run makes a sequence of

- decisions: choose an unassigned variable p and a bit b, set $\pi(p) = b$

A Quick Recap: DPLL

A DPLL run makes a sequence of

- decisions: choose an unassigned variable p and a bit b, set $\pi(p) = b$
- unit propagations: if clause $C \in F$ has exactly one unassigned literal ℓ and all its other literals are false under π, set $\pi(\ell) = 1$

A Quick Recap: DPLL

A DPLL run makes a sequence of

- decisions: choose an unassigned variable p and a bit b, set $\pi(p) = b$
- unit propagations: if clause $C \in F$ has exactly one unassigned literal ℓ and all its other literals are false under π, set $\pi(\ell) = 1$
- backtracking: on conflict (some clause of F false under π), undo the last decision and try the other bit

A Quick Recap: DPLL

A DPLL run makes a sequence of

- decisions: choose an unassigned variable p and a bit b, set $\pi(p) = b$
- unit propagations: if clause $C \in F$ has exactly one unassigned literal ℓ and all its other literals are false under π, set $\pi(\ell) = 1$
- backtracking: on conflict (some clause of F false under π), undo the last decision and try the other bit

Today

We open up DPLL's backtracking step: instead of just undoing the last decision, learn from a conflict which decisions actually caused it and jump back further – this is clause learning, and DPLL + clause learning is CDCL.

Implementation and Optimizations

In fact, in DPLL in practice

- The "pure" literal check is not done (too expensive!)
- Branching literal is not chosen randomly!
- Algo is implemented iteratively; with backtrack stack maintained explicitly! (may help to backtrack more than one level!)

Implementation and Optimizations

In fact, in DPLL in practice

- The "pure" literal check is not done (too expensive!)
- Branching literal is not chosen randomly!
- Algo is implemented iteratively; with backtrack stack maintained explicitly! (may help to backtrack more than one level!)

Further, DPLL allows many optimizations and improvements.

Implementation and Optimizations

In fact, in DPLL in practice

- The "pure" literal check is not done (too expensive!)
- Branching literal is not chosen randomly!
- Algo is implemented iteratively; with backtrack stack maintained explicitly! (may help to backtrack more than one level!)

Further, DPLL allows many optimizations and improvements.

- **clause learning**
- 2-watched literals
- ...

Implementation and Optimizations

In fact, in DPLL in practice

- The "pure" literal check is not done (too expensive!)
- Branching literal is not chosen randomly!
- Algo is implemented iteratively; with backtrack stack maintained explicitly! (may help to backtrack more than one level!)

Further, DPLL allows many optimizations and improvements.

- **clause learning**
- 2-watched literals
- ...

We will look at one such optimization – clause learning – in detail, next.

Topic 10.1

Clause learning

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

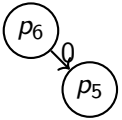
Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1



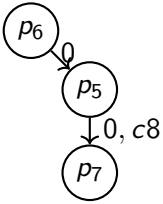
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

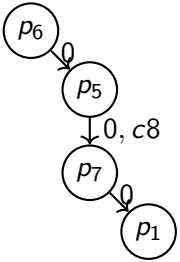


Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

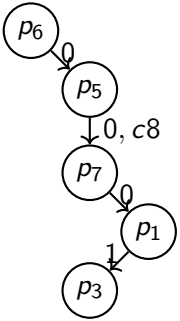


Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

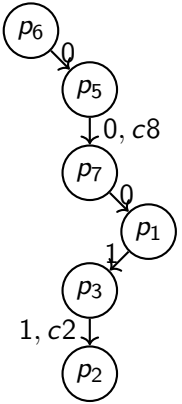


Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

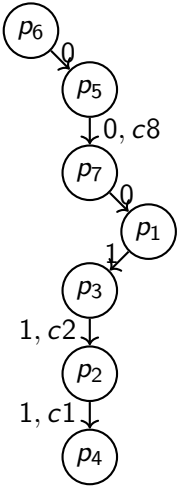


Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



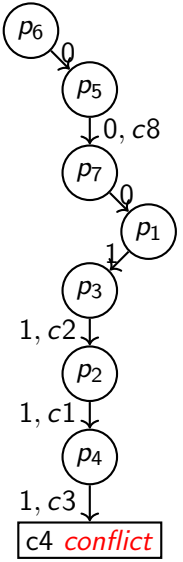
Blue : causing unit propagation
Green/Blue : true clause

Revisiting DPLL

Exercise 10.1: run DPLL on this set of clauses, deciding p_6, p_7, p_1 in that order: p_6=0, p_7=0, p_1=1

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

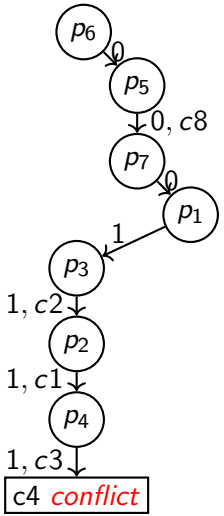
Blue : causing unit propagation
Green/Blue : true clause



Backtrack: both values of p_1 fail

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

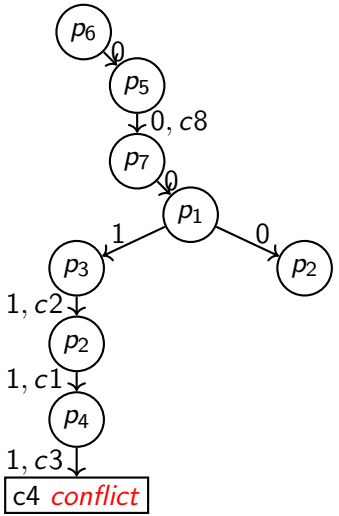
Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



Backtrack: both values of p_1 fail

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

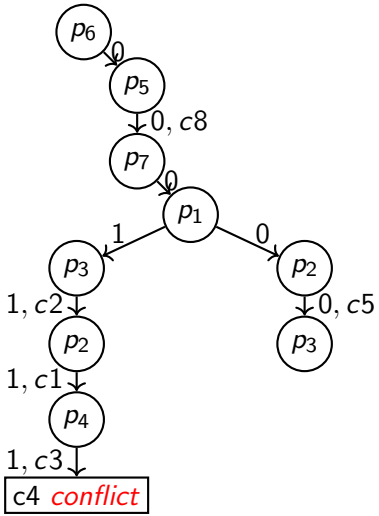
Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



Backtrack: both values of p_1 fail

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

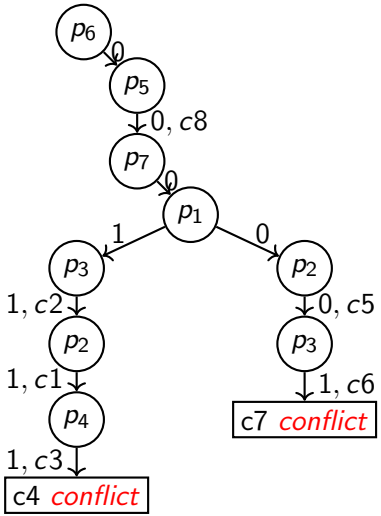
Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



Backtrack: both values of p_1 fail

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

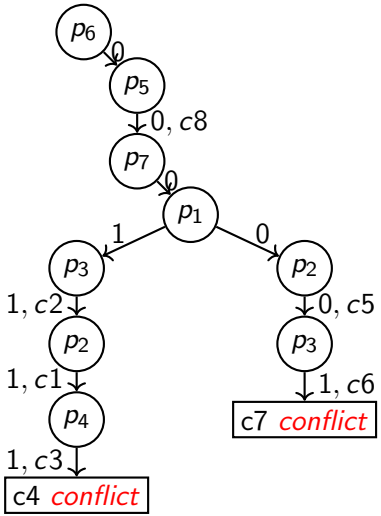
Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



Backtrack: both values of p_1 fail

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

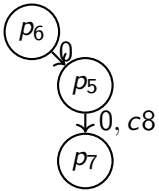
Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



Both values of p_1 fail – backtrack further, to p_7

The complete search tree

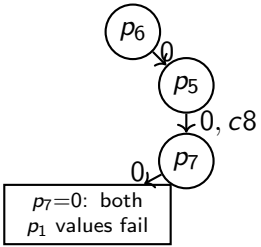
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



Green : true clause
Red : conflict clause

The complete search tree

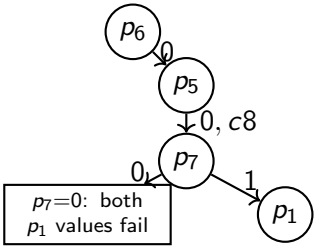
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



Green : true clause
Red : conflict clause

The complete search tree

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

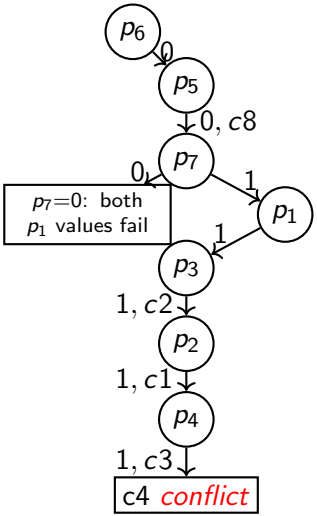


Green : true clause
Red : conflict clause

The complete search tree

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

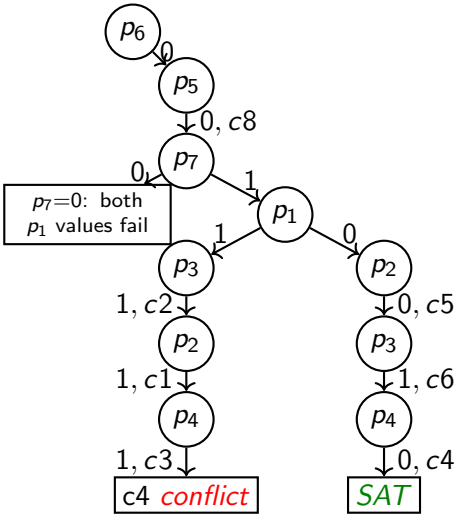
Green : true clause
Red : conflict clause



The complete search tree

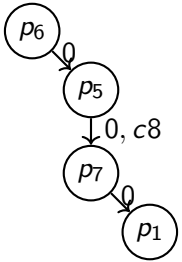
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Green : true clause
Red : conflict clause



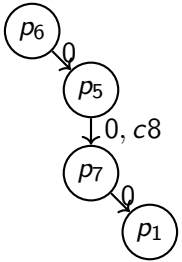
Decision level

Consider just the partial run so far, up to p_1.



Decision level

Consider just the partial run so far, up to p_1.

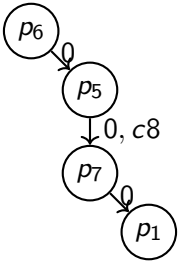


Definition

The decision level of a true literal is the number of decisions after which the literal was made true.

Decision level

Consider just the partial run so far, up to p_1.



Definition

The decision level of a true literal is the number of decisions after which the literal was made true.

Given the above run, we write $\neg p_6@1$ and $\neg p_5@1$ (p_5 was propagated while only one decision had been made), $\neg p_7@2$.

Example: implication graph

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause

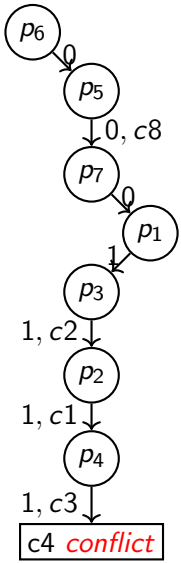
Example: implication graph

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation

Green/Blue : true clause

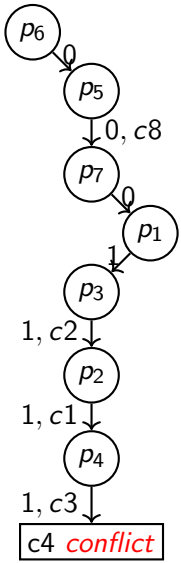
Red : conflict clause



Example: implication graph

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



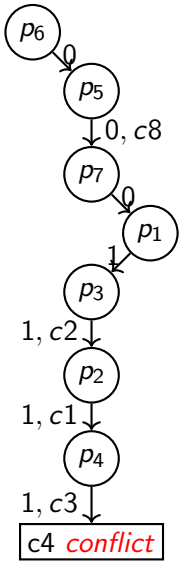
Implication graph

$\neg p_6$@1

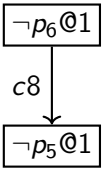
Example: implication graph

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



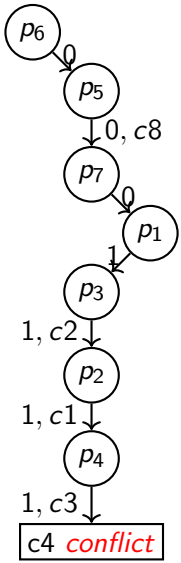
Implication graph



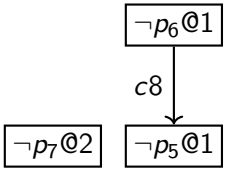
Example: implication graph

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

Blue : causing unit propagation
Green/Blue : true clause
Red : conflict clause



Implication graph



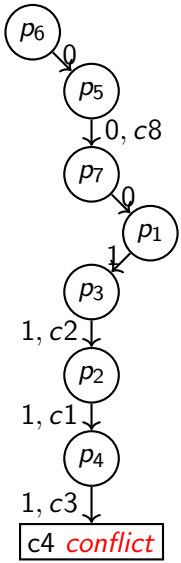
Example: implication graph

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$

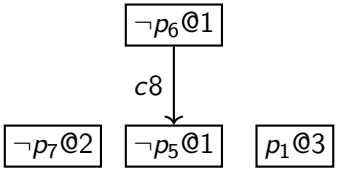
Blue : causing unit propagation

Green/Blue : true clause

Red : conflict clause



Implication graph



Example: implication graph

$c_1 = (\neg p_1 \vee p_2)$

$c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_3 = (\neg p_2 \vee p_4)$

$c_4 = (\neg p_3 \vee \neg p_4)$

$c_5 = (p_1 \vee p_5 \vee \neg p_2)$

$c_6 = (p_2 \vee p_3)$

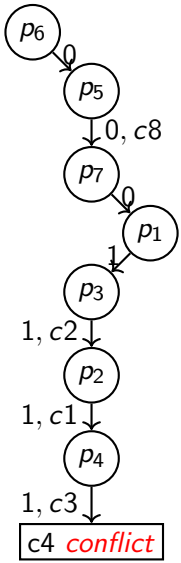
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$

$c_8 = (p_6 \vee \neg p_5)$

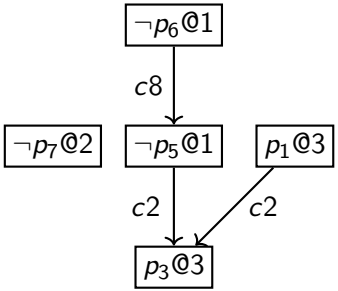
Blue : causing unit propagation

Green/Blue : true clause

Red : conflict clause



Implication graph



Example: implication graph

$c_1 = (\neg p_1 \vee p_2)$

$c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_3 = (\neg p_2 \vee p_4)$

$c_4 = (\neg p_3 \vee \neg p_4)$

$c_5 = (p_1 \vee p_5 \vee \neg p_2)$

$c_6 = (p_2 \vee p_3)$

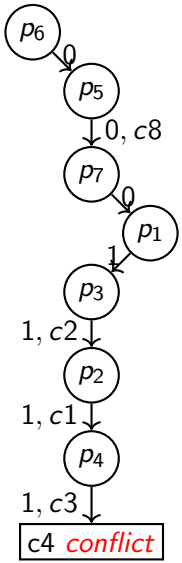
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$

$c_8 = (p_6 \vee \neg p_5)$

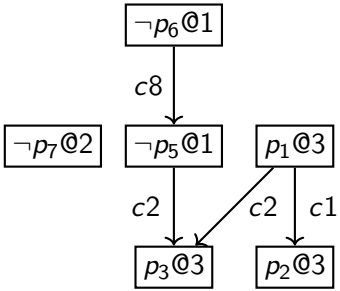
Blue : causing unit propagation

Green/Blue : true clause

Red : conflict clause



Implication graph



Example: implication graph

$c_1 = (\neg p_1 \vee p_2)$

$c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_3 = (\neg p_2 \vee p_4)$

$c_4 = (\neg p_3 \vee \neg p_4)$

$c_5 = (p_1 \vee p_5 \vee \neg p_2)$

$c_6 = (p_2 \vee p_3)$

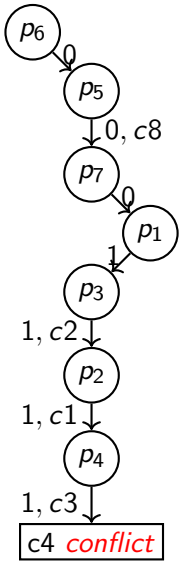
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$

$c_8 = (p_6 \vee \neg p_5)$

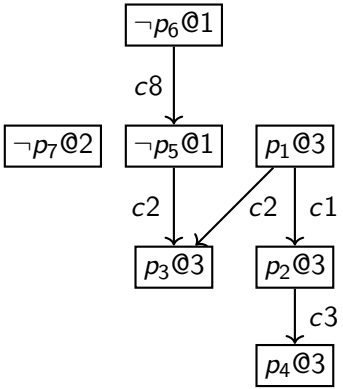
Blue : causing unit propagation

Green/Blue : true clause

Red : conflict clause



Implication graph



Example: implication graph

$c_1 = (\neg p_1 \vee p_2)$

$c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_3 = (\neg p_2 \vee p_4)$

$c_4 = (\neg p_3 \vee \neg p_4)$

$c_5 = (p_1 \vee p_5 \vee \neg p_2)$

$c_6 = (p_2 \vee p_3)$

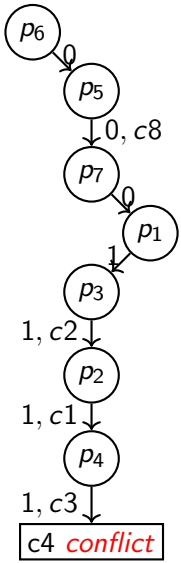
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$

$c_8 = (p_6 \vee \neg p_5)$

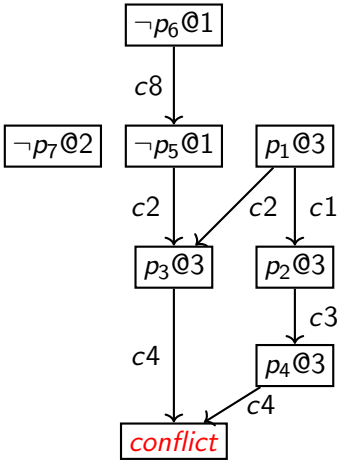
Blue : causing unit propagation

Green/Blue : true clause

Red : conflict clause



Implication graph



Implication graph

During the DPLL run, we maintain the following data structure.

Under a partial assignment π, the implication graph is a labeled DAG (V, E),

Implication graph

During the DPLL run, we maintain the following data structure.

Under a partial assignment π, the implication graph is a labeled DAG (V, E), where

- V is the set of true literals under π and a conflict node

Implication graph

During the DPLL run, we maintain the following data structure.

Under a partial assignment π, the implication graph is a labeled DAG (V, E), where

- V is the set of true literals under π and a conflict node
- $E = \{(\ell_1, C, \ell_2) | \ell_2$ was produced by unit propagation at C, with $\ell_2, \neg\ell_1 \in C\}$.

Implication graph

During the DPLL run, we maintain the following data structure.

Under a partial assignment π, the implication graph is a labeled DAG (V, E), where

- V is the set of true literals under π and a conflict node
- $E = \{(\ell_1, C, \ell_2) | \ell_2$ was produced by unit propagation at C, with $\ell_2, \neg\ell_1 \in C\}$.

We also annotate each node with its decision level. The decision level of a true literal is the number of decisions after which the literal was made true.

Implication graph

During the DPLL run, we maintain the following data structure.

Under a partial assignment π, the implication graph is a labeled DAG (V, E), where

- V is the set of true literals under π and a conflict node
- $E = \{(\ell_1, C, \ell_2) | \ell_2$ was produced by unit propagation at C, with $\ell_2, \neg\ell_1 \in C\}$.

We also annotate each node with its decision level. The decision level of a true literal is the number of decisions after which the literal was made true.

So what is the use of drawing this implication graph?

Conflict clause

We traverse the implication graph backwards to find the set of decisions that caused the conflict.

Conflict clause

We traverse the implication graph backwards to find the set of decisions that caused the conflict.

Definition

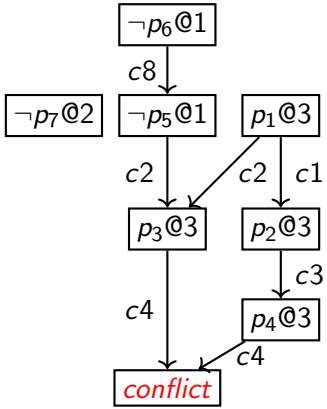
The clause of the negations of the causing decisions is called conflict clause.

Conflict clause

We traverse the implication graph backwards to find the set of decisions that caused the conflict.

Definition

The clause of the negations of the causing decisions is called conflict clause.



Conflict clause : $p_6 \vee \neg p_1$

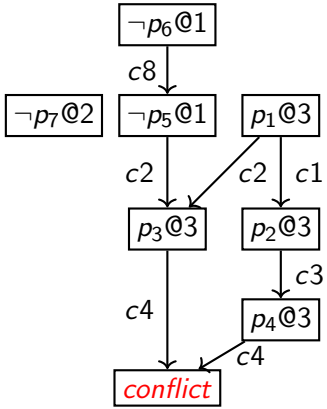
So what do we do with this learned conflict clause?

Conflict clause

We traverse the implication graph backwards to find the set of decisions that caused the conflict.

Definition

The clause of the negations of the causing decisions is called conflict clause.



Conflict clause : $p_6 \vee \neg p_1$

So what do we do with this learned conflict clause?
Add it to our formula!

What happens when we add the learned clause?

$$
\begin{aligned}
c_1 &= (\neg p_1 \vee p_2) \\
c_2 &= (\neg p_1 \vee p_3 \vee p_5) \\
c_3 &= (\neg p_2 \vee p_4) \\
c_4 &= (\neg p_3 \vee \neg p_4) \\
c_5 &= (p_1 \vee p_5 \vee \neg p_2) \\
c_6 &= (p_2 \vee p_3) \\
c_7 &= (p_2 \vee \neg p_3 \vee p_7) \\
c_8 &= (p_6 \vee \neg p_5) \\
c_9 &= (p_6 \vee \neg p_1)
\end{aligned}
$$

Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

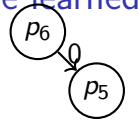
What happens when we add the learned clause?

p_6

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$

Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

What happens when we add the learned clause?

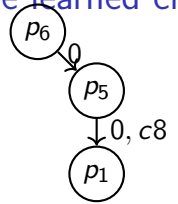


$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$

Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

What happens when we add the learned clause?

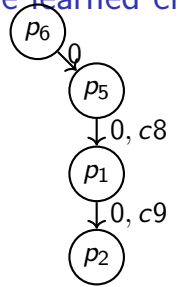
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$



Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

What happens when we add the learned clause?

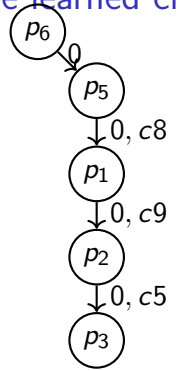
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$



Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

What happens when we add the learned clause?

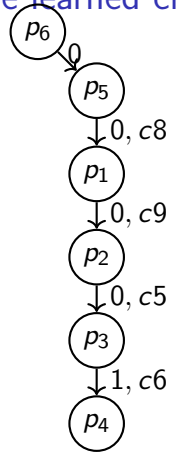
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$



Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

What happens when we add the learned clause?

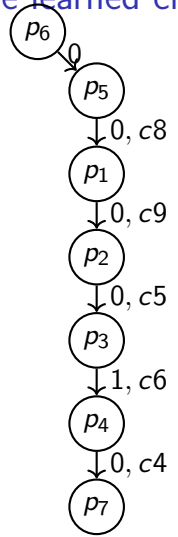
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$



Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

What happens when we add the learned clause?

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$

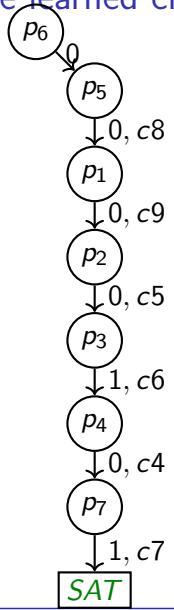


Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause

What happens when we add the learned clause?

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$
$$c_9 = (p_6 \vee \neg p_1)$$

Blue : causing unit propagation
Green/Blue : true clause
Violet : the learned clause



p_6 is the only decision – everything else is forced!

Clause learning

Clause learning heuristics

- add conflict clause in the input clauses and
- backtrack to just after the second last conflicting decision, and proceed like DPLL

Clause learning

Clause learning heuristics

- add conflict clause in the input clauses and
- backtrack to just after the second last conflicting decision, and proceed like DPLL

Lemma

Let C be the conflict clause. Then

1. $F \models C$ (so adding C does not change the set of satisfying assignments)

Clause learning

Clause learning heuristics

- add conflict clause in the input clauses and
- backtrack to just after the second last conflicting decision, and proceed like DPLL

Lemma

Let C be the conflict clause. Then

1. $F \models C$ (so adding C does not change the set of satisfying assignments)
2. the conflicting partial assignment will never be tried again

Clause learning

Clause learning heuristics

- add conflict clause in the input clauses and
- backtrack to just after the second last conflicting decision, and proceed like DPLL

Lemma

Let C be the conflict clause. Then

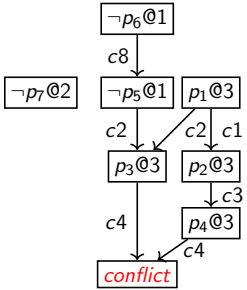
1. $F \models C$ (so adding C does not change the set of satisfying assignments)
2. the conflicting partial assignment will never be tried again

Exercise 10.2

1. Prove the above Lemma. (Hint: for part 1, show $F \wedge \neg C$ is unsat.)
2. Can you prove $F \vdash C$ using resolution?

Implication graphs and resolution proofs!

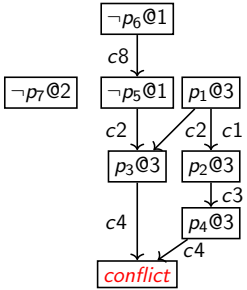
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



We show $F \wedge \neg C$ is unsat, i.e. derive $\bot$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.

Implication graphs and resolution proofs!

$c_1 = (\neg p_1 \vee p_2)$
$c_2 = (\neg p_1 \vee p_3 \vee p_5)$
$c_3 = (\neg p_2 \vee p_4)$
$c_4 = (\neg p_3 \vee \neg p_4)$
$c_5 = (p_1 \vee p_5 \vee \neg p_2)$
$c_6 = (p_2 \vee p_3)$
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$
$c_8 = (p_6 \vee \neg p_5)$

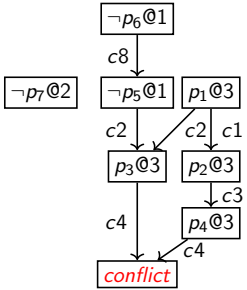


We show $F \wedge \neg C$ is unsat, i.e. derive $\perp$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.

$\neg p_1 \vee p_3 \vee p_5$ p_1
(c_2)
$p_3 \vee p_5$

Implication graphs and resolution proofs!

$c_1 = (\neg p_1 \vee p_2)$
$c_2 = (\neg p_1 \vee p_3 \vee p_5)$
$c_3 = (\neg p_2 \vee p_4)$
$c_4 = (\neg p_3 \vee \neg p_4)$
$c_5 = (p_1 \vee p_5 \vee \neg p_2)$
$c_6 = (p_2 \vee p_3)$
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$
$c_8 = (p_6 \vee \neg p_5)$

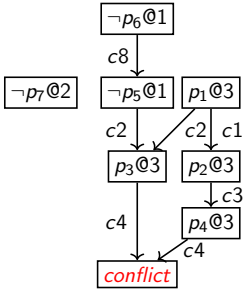


We show $F \wedge \neg C$ is unsat, i.e. derive $\perp$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.

$$\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5}{(c_2)} \quad p_1}{p_3 \vee p_5} \qquad \dfrac{\dfrac{p_6 \vee \neg p_5}{(c_8)} \quad \neg p_6}{\neg p_5}$$

Implication graphs and resolution proofs!

$c_1 = (\neg p_1 \vee p_2)$
$c_2 = (\neg p_1 \vee p_3 \vee p_5)$
$c_3 = (\neg p_2 \vee p_4)$
$c_4 = (\neg p_3 \vee \neg p_4)$
$c_5 = (p_1 \vee p_5 \vee \neg p_2)$
$c_6 = (p_2 \vee p_3)$
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$
$c_8 = (p_6 \vee \neg p_5)$



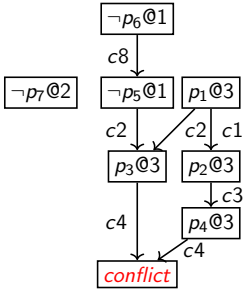
We show $F \wedge \neg C$ is unsat, i.e. derive $\perp$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.

$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5}{(c_2)} \quad p_1}{p_3 \vee p_5} \qquad \dfrac{\dfrac{p_6 \vee \neg p_5}{(c_8)} \quad \neg p_6}{\neg p_5}}{p_3} \qquad \dfrac{\qquad}{\qquad}$$

$$\dfrac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \qquad \dfrac{\qquad\qquad\qquad\qquad}{\qquad\qquad\qquad\qquad}$$

Implication graphs and resolution proofs!

$c_1 = (\neg p_1 \vee p_2)$
$c_2 = (\neg p_1 \vee p_3 \vee p_5)$
$c_3 = (\neg p_2 \vee p_4)$
$c_4 = (\neg p_3 \vee \neg p_4)$
$c_5 = (p_1 \vee p_5 \vee \neg p_2)$
$c_6 = (p_2 \vee p_3)$
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$
$c_8 = (p_6 \vee \neg p_5)$



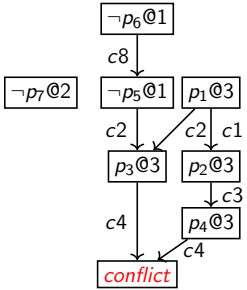
We show $F \wedge \neg C$ is unsat, i.e. derive $\perp$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.

$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5}{(c_2)} \quad p_1}{p_3 \vee p_5} \qquad \dfrac{p_6 \vee \neg p_5}{(c_8)} \quad \neg p_6}{\neg p_5} \qquad \dfrac{\dfrac{\neg p_1 \vee p_2}{(c_1)} \quad p_1}{p_2}$$

$$\dfrac{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$

Implication graphs and resolution proofs!

$c_1 = (\neg p_1 \vee p_2)$
$c_2 = (\neg p_1 \vee p_3 \vee p_5)$
$c_3 = (\neg p_2 \vee p_4)$
$c_4 = (\neg p_3 \vee \neg p_4)$
$c_5 = (p_1 \vee p_5 \vee \neg p_2)$
$c_6 = (p_2 \vee p_3)$
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$
$c_8 = (p_6 \vee \neg p_5)$

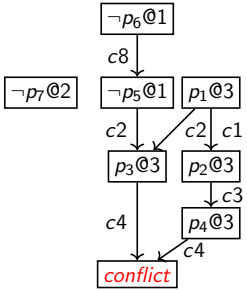


We show $F \wedge \neg C$ is unsat, i.e. derive $\perp$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.

$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \quad (c_2) \qquad p_1}{p_3 \vee p_5} \qquad \dfrac{p_6 \vee \neg p_5 \quad (c_8) \qquad \neg p_6}{\neg p_5}}{p_3} \qquad \dfrac{\dfrac{\neg p_1 \vee p_2 \quad (c_1) \qquad p_1}{p_2} \qquad \neg p_2 \vee p_4 \quad (c_3)}{p_4}}{}$$

Implication graphs and resolution proofs!

$c_1 = (\neg p_1 \vee p_2)$
$c_2 = (\neg p_1 \vee p_3 \vee p_5)$
$c_3 = (\neg p_2 \vee p_4)$
$c_4 = (\neg p_3 \vee \neg p_4)$
$c_5 = (p_1 \vee p_5 \vee \neg p_2)$
$c_6 = (p_2 \vee p_3)$
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$
$c_8 = (p_6 \vee \neg p_5)$

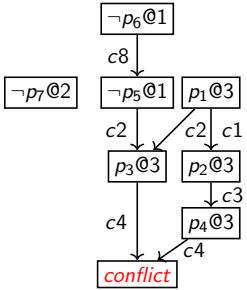


We show $F \wedge \neg C$ is unsat, i.e. derive $\perp$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.

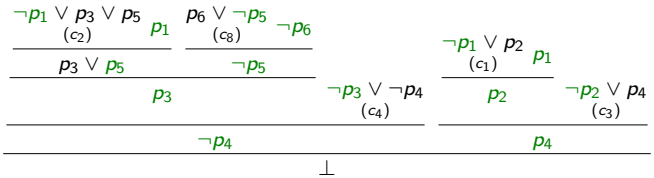
$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \quad (c_2) \qquad p_1}{p_3 \vee p_5} \qquad \dfrac{p_6 \vee \neg p_5 \quad (c_8) \qquad \neg p_6}{\neg p_5}}{p_3} \qquad \dfrac{\neg p_3 \vee \neg p_4 \quad (c_4)}{\neg p_4} \qquad \dfrac{\dfrac{\neg p_1 \vee p_2 \quad (c_1) \qquad p_1}{p_2} \qquad \dfrac{\neg p_2 \vee p_4 \quad (c_3)}{p_4}}{\neg p_4}}{}$$

Implication graphs and resolution proofs!

$c_1 = (\neg p_1 \vee p_2)$
$c_2 = (\neg p_1 \vee p_3 \vee p_5)$
$c_3 = (\neg p_2 \vee p_4)$
$c_4 = (\neg p_3 \vee \neg p_4)$
$c_5 = (p_1 \vee p_5 \vee \neg p_2)$
$c_6 = (p_2 \vee p_3)$
$c_7 = (p_2 \vee \neg p_3 \vee p_7)$
$c_8 = (p_6 \vee \neg p_5)$



We show $F \wedge \neg C$ is unsat, i.e. derive $\bot$ from F assuming $\neg p_6$ and p_1 – retracing the graph above, edge by edge.



Benefit of adding conflict clauses

1. Prunes away search space
2. Records past work of the SAT solver

Benefit of adding conflict clauses

1. Prunes away search space
2. Records past work of the SAT solver

For example,

In the previous example, we made decisions : $m(p_6) = 0$, $m(p_7) = 0$, and $m(p_1) = 1$

We learned a conflict clause : $p_6 \vee \neg p_1$

Benefit of adding conflict clauses

1. Prunes away search space
2. Records past work of the SAT solver

For example,

In the previous example, we made decisions : $m(p_6) = 0$, $m(p_7) = 0$, and $m(p_1) = 1$

We learned a conflict clause : $p_6 \vee \neg p_1$

Adding this clause to the input clauses results in

- $m(p_6) = 0$, $m(p_7) = 1$, and $m(p_1) = 1$ will never be tried

Benefit of adding conflict clauses

1. Prunes away search space
2. Records past work of the SAT solver

For example,

In the previous example, we made decisions : $m(p_6) = 0$, $m(p_7) = 0$, and $m(p_1) = 1$

We learned a conflict clause : $p_6 \vee \neg p_1$

Adding this clause to the input clauses results in

- $m(p_6) = 0$, $m(p_7) = 1$, and $m(p_1) = 1$ will never be tried
- In fact, $m(p_6) = 0$ and $m(p_1) = 1$ will never occur simultaneously.

Benefit of adding conflict clauses

1. Prunes away search space
2. Records past work of the SAT solver

For example,

In the previous example, we made decisions : $m(p_6) = 0$, $m(p_7) = 0$, and $m(p_1) = 1$

We learned a conflict clause : $p_6 \vee \neg p_1$

Adding this clause to the input clauses results in

- $m(p_6) = 0$, $m(p_7) = 1$, and $m(p_1) = 1$ will never be tried
- In fact, $m(p_6) = 0$ and $m(p_1) = 1$ will never occur simultaneously.

- There are many other clever choices for conflict clauses.
- DPLL+conflicts = CDCL (Conflict-driven clause learning!)

Algorithm 10.1: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

UnitPropagation(m, F);

do

while ;

- UnitPropagation(m, F) - applies unit propagation and extends m

Algorithm 10.1: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

UnitPropagation(m, F);

do

dl stands for decision level

while ;

- UnitPropagation(m, F) - applies unit propagation and extends m

Algorithm 10.1: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

UnitPropagation(m, F);

do

  // backtracking

  // Boolean decision

while ;

dl stands for decision level

- UnitPropagation(m, F) - applies unit propagation and extends m

Algorithm 10.1: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

UnitPropagation(m, F);

do

  // backtracking

  // Boolean decision

  if *m is partial* **then**

    $dstack(dl) := m.size()$;

    $dl := dl + 1$; Decide(m, F); UnitPropagation(m, F) ;

while ;

dl stands for decision level

- UnitPropagation(m, F) - applies unit propagation and extends m
- Decide(m, F) - chooses an unassigned variable in m and assigns a Boolean value

Algorithm 10.1: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

UnitPropagation(m, F);

do

  // backtracking

  // Boolean decision

  if *m is partial* **then**

    $dstack(dl) := m.size()$;

    $dl := dl + 1$; Decide(m, F) ;

while ;

dl stands for decision level

dstack records history for backtracking

- UnitPropagation(m, F) - applies unit propagation and extends m
- Decide(m, F) - chooses an unassigned variable in m and assigns a Boolean value

Algorithm 10.1: CDCL

Input: CNF F
$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;
UnitPropagation(m, F);
do
  // backtracking
  while $m \not\models F$ **do**
    if $dl = 0$ **then return** *unsat*;
    $(C, dl) :=$ AnalyzeConflict(m, F);
    $m.resize(dstack(dl))$; $F := F \cup \{C\}$; m := UnitPropagation(m, F);
  // Boolean decision
  if m *is partial* **then**
    $dstack(dl) := m.size()$;
    $dl := dl + 1$; Decide(m, F);
while ;

dl stands for decision level

dstack records history for backtracking

- UnitPropagation(m, F) - applies unit propagation and extends m
- Decide(m, F) - chooses an unassigned variable in m and assigns a Boolean value
- AnalyzeConflict(m, F) - returns a conflict clause learned using implication graph and a decision level

Algorithm 10.1: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

UnitPropagation(m, F);

do

  // backtracking

  while $m \not\models F$ **do**

    if $dl = 0$ **then return** *unsat*;

    $(C, dl) :=$ AnalyzeConflict(m, F);

    $m.resize(dstack(dl))$; $F := F \cup \{C\}$; m := UnitPropagation(m, F);

  // Boolean decision

  if *m is partial* **then**

    $dstack(dl) := m.size()$;

    $dl := dl + 1$; Decide(m, F);

while *m is partial or* $m \not\models F$;

dl stands for decision level

dstack records history for backtracking

- UnitPropagation(m, F) - applies unit propagation and extends m
- Decide(m, F) - chooses an unassigned variable in m and assigns a Boolean value
- AnalyzeConflict(m, F) - returns a conflict clause learned using implication graph and a decision level

Algorithm 10.1: CDCL

Input: CNF F

$m := \emptyset$; $dl := 0$; $dstack := \lambda x.0$;

UnitPropagation(m, F);

do

  // backtracking

  while $m \not\models F$ **do**

    if $dl = 0$ **then return** *unsat*;

    $(C, dl) :=$ AnalyzeConflict(m, F);

    $m.resize(dstack(dl))$; $F := F \cup \{C\}$; m := UnitPropagation(m, F);

  // Boolean decision

  if m *is partial* **then**

    $dstack(dl) := m.size()$;

    $dl := dl + 1$; Decide(m, F);

while m *is partial or* $m \not\models F$;

return *sat*

dl stands for decision level

dstack records history for backtracking

- UnitPropagation(m, F) - applies unit propagation and extends m
- Decide(m, F) - chooses an unassigned variable in m and assigns a Boolean value
- AnalyzeConflict(m, F) - returns a conflict clause learned using implication graph and a decision level

Other improvements

Decide

Use involved heuristics to decide which unassigned variable to branch from.

- VSIDS: score each variable by how often it appears in recent conflicts (score decays over time); branch on the highest-scoring variable.

Exercise 10.3: for Exercise 10.1's clauses, show that deciding p_1 first reaches SAT with no conflict at all – decision order matters!

Other improvements

Decide

Use involved heuristics to decide which unassigned variable to branch from.

- VSIDS: score each variable by how often it appears in recent conflicts (score decays over time); branch on the highest-scoring variable.

Exercise 10.3: for Exercise 10.1's clauses, show that deciding p_1 first reaches SAT with no conflict at all – decision order matters!

First UIP

Instead of tracing back to every causing decision, stop at the first node – through which all paths from the current decision to the conflict must pass.

Other improvements

Decide

Use involved heuristics to decide which unassigned variable to branch from.

- VSIDS: score each variable by how often it appears in recent conflicts (score decays over time); branch on the highest-scoring variable.

Exercise 10.3: for Exercise 10.1's clauses, show that deciding p_1 first reaches SAT with no conflict at all – decision order matters!

First UIP

Instead of tracing back to every causing decision, stop at the first node – through which all paths from the current decision to the conflict must pass.

Implementation-level improvements

- Clause deletion: learned clauses pile up fast – periodically discard the less useful ones to bound memory and keep propagation fast.

Other improvements

Decide

Use involved heuristics to decide which unassigned variable to branch from.

- VSIDS: score each variable by how often it appears in recent conflicts (score decays over time); branch on the highest-scoring variable.

Exercise 10.3: for Exercise 10.1's clauses, show that deciding p_1 first reaches SAT with no conflict at all – decision order matters!

First UIP

Instead of tracing back to every causing decision, stop at the first node – through which all paths from the current decision to the conflict must pass.

Implementation-level improvements

- Clause deletion: learned clauses pile up fast – periodically discard the less useful ones to bound memory and keep propagation fast.

Resets

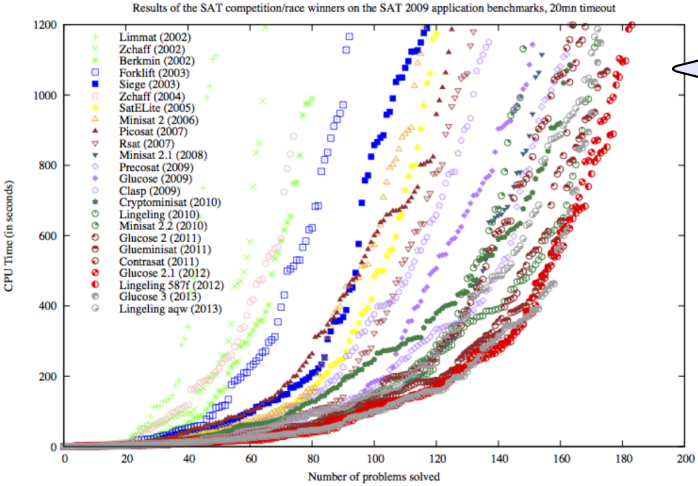
At some point give up and re-start!

Topic 10.2

SAT technology and its impact

Efficiency of SAT solvers over the years

(chart as of 2014 – solvers have improved substantially further since!)



Every solver here, from 2002 on, is a Chaff-lineage CDCL solver – this whole plot is watching CDCL refine itself for two decades!

Cactus plot:

X-axis: # problems solved

Y-axis: CPU time taken

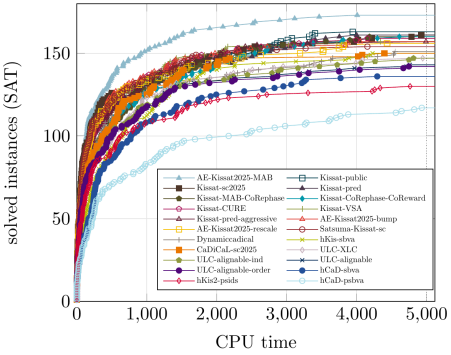
each curve = one solver

further right & lower = faster

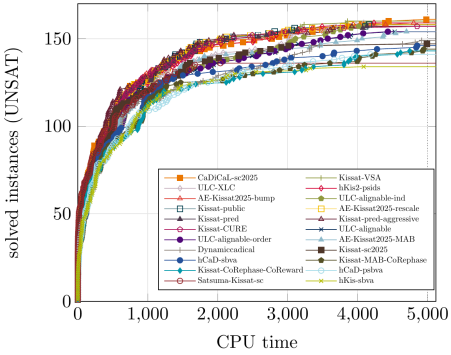
Source: `http://satsmt2014.forsyte.at/files/2014/07/SAT-introduction.pdf`

SAT solvers today: Competition 2025

Main (Sequential) Track SAT Plot



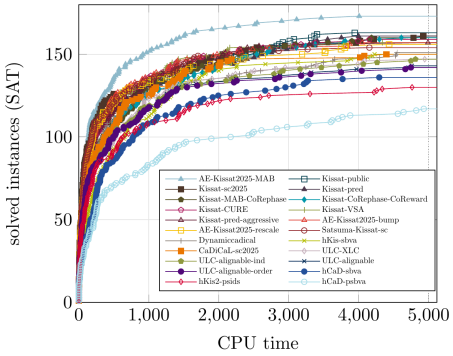
Main (Sequential) Track UNSAT Plot



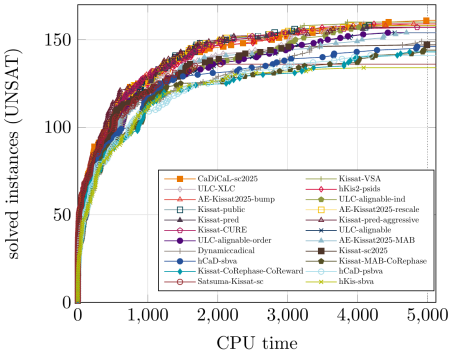
Source: SAT Competition 2025 results, `https://satcompetition.github.io/2025/satcomp25slides.pdf`

SAT solvers today: Competition 2025

Main (Sequential) Track SAT Plot



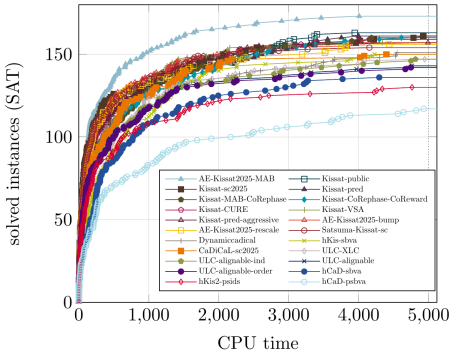
Main (Sequential) Track UNSAT Plot



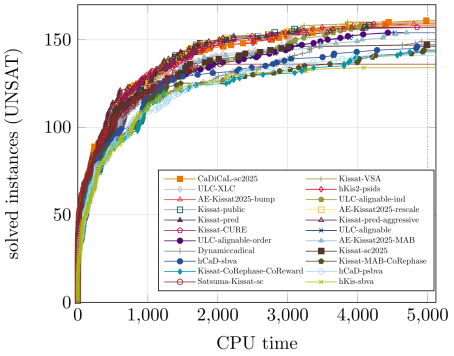
Source: SAT Competition 2025 results, `https://satcompetition.github.io/2025/satcomp25slides.pdf` Today's solvers routinely handle industrial instances with millions of variables – up from just hundreds in the pre-CDCL era; the largest published instances now reach into the billions of clauses.

SAT solvers today: Competition 2025

Main (Sequential) Track SAT Plot



Main (Sequential) Track UNSAT Plot



Source: SAT Competition 2025 results, `https://satcompetition.github.io/2025/satcomp25slides.pdf` Today's solvers routinely handle industrial instances with millions of variables – up from just hundreds in the pre-CDCL era; the largest published instances now reach into the billions of clauses. (e.g. A. Nadel, *Solving Huge Instances with Intel SAT Solver*, SAT 2023.)

Impact of SAT technology

Impact is enormous.

Impact of SAT technology

Impact is enormous.

A few are listed below

- Hardware verification and design assistance
  Almost all hardware/EDA companies have their own SAT solver
- AI/Planning: many resource allocation problems are convertible to SAT
- Security: analysis of crypto algorithms
- Solving hard problems, *e. g.*, travelling salesman problem

Impact of SAT technology

Impact is enormous.

A few are listed below

- Hardware verification and design assistance
  Almost all hardware/EDA companies have their own SAT solver
- AI/Planning: many resource allocation problems are convertible to SAT
- Security: analysis of crypto algorithms
- Solving hard problems, *e. g.*, travelling salesman problem

Next class

We will see a SAT solver in action! Come with your laptop, with a solver installed – instructions will be given on Piazza.