Logic and Mechanized Reasoning
Conflict-Driven Clause-Learning Solving

Marijn J.H. Heule

Carnegie Mellon University

First Midterm Exam

Monday February 19 at 12:30pm in NSH 1305 and GHC 4301

Material covered in the exam:

- All lectures up to (and including) February 7
- All homework through Assignment 4
- Textbook chapters 1-7, excluding Sections 6.3, 6.5, 7.4

Practice exam and solutions on course website

No new homework assigned this week

Extra office hours:

- Josh: Saturday from 5-6pm
- Tika: Sunday from 2-3pm
- Alex: Sunday from 6-7pm
- Joseph: Monday from 10:30-11:30am

The Satisfiability (SAT) problem

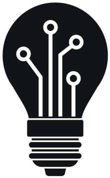
$$
\begin{array}{l}
(p_5 \lor p_8 \lor \neg p_2) \land (p_2 \lor \neg p_1 \lor \neg p_3) \land (\neg p_8 \lor \neg p_3 \lor \neg p_7) \land \\
(\neg p_5 \lor p_3 \lor p_8) \land (\neg p_6 \lor \neg p_1 \lor \neg p_5) \land (p_8 \lor \neg p_9 \lor p_3) \land \\
(p_2 \lor p_1 \lor p_3) \land (\neg p_1 \lor p_8 \lor p_4) \land (\neg p_9 \lor \neg p_6 \lor p_8) \land \\
(p_8 \lor p_3 \lor \neg p_9) \land (p_9 \lor \neg p_3 \lor p_8) \land (p_6 \lor \neg p_9 \lor p_5) \land \\
(p_2 \lor \neg p_3 \lor \neg p_8) \land (p_8 \lor \neg p_6 \lor \neg p_3) \land (p_8 \lor \neg p_3 \lor \neg p_1) \land \\
(\neg p_8 \lor p_6 \lor \neg p_2) \land (p_7 \lor p_9 \lor \neg p_2) \land (p_8 \lor \neg p_9 \lor p_2) \land \\
(\neg p_1 \lor \neg p_9 \lor p_4) \land (p_8 \lor p_1 \lor \neg p_2) \land (p_3 \lor \neg p_4 \lor \neg p_6) \land \\
(\neg p_1 \lor \neg p_7 \lor p_5) \land (\neg p_7 \lor p_1 \lor p_6) \land (\neg p_5 \lor p_4 \lor \neg p_6) \land \\
(\neg p_4 \lor p_9 \lor \neg p_8) \land (p_2 \lor p_9 \lor p_1) \land (p_5 \lor \neg p_7 \lor p_1) \land \\
(\neg p_7 \lor \neg p_9 \lor \neg p_6) \land (p_2 \lor p_5 \lor p_4) \land (p_8 \lor \neg p_4 \lor p_5) \land \\
(p_5 \lor p_9 \lor p_3) \land (\neg p_5 \lor \neg p_7 \lor p_9) \land (p_2 \lor \neg p_8 \lor p_1) \land \\
(\neg p_7 \lor p_1 \lor p_5) \land (p_1 \lor p_4 \lor p_3) \land (p_1 \lor \neg p_9 \lor \neg p_4) \land \\
(p_3 \lor p_5 \lor p_6) \land (\neg p_6 \lor p_3 \lor \neg p_9) \land (\neg p_7 \lor p_5 \lor p_9) \land \\
(p_7 \lor \neg p_5 \lor \neg p_2) \land (p_4 \lor p_7 \lor p_3) \land (p_4 \lor \neg p_9 \lor \neg p_7) \land \\
(p_5 \lor \neg p_1 \lor p_7) \land (p_5 \lor \neg p_1 \lor p_7) \land (p_6 \lor p_7 \lor \neg p_3) \land \\
(\neg p_8 \lor \neg p_6 \lor \neg p_7) \land (p_6 \lor p_2 \lor p_3) \land (\neg p_8 \lor p_2 \lor p_5)
\end{array}
$$

Does there exist an assignment satisfying all clauses?

Search for a satisfying assignment (or proof none exists)

$$
\begin{array}{l}
(p_5 \vee p_8 \vee \neg p_2) \wedge (p_2 \vee \neg p_1 \vee \neg p_3) \wedge (\neg p_8 \vee \neg p_3 \vee \neg p_7) \wedge \\
(\neg p_5 \vee p_3 \vee p_8) \wedge (\neg p_6 \vee \neg p_1 \vee \neg p_5) \wedge (p_8 \vee \neg p_9 \vee p_3) \wedge \\
(p_2 \vee p_1 \vee p_3) \wedge (\neg p_1 \vee p_8 \vee p_4) \wedge (\neg p_9 \vee \neg p_6 \vee p_8) \wedge \\
(p_8 \vee p_3 \vee \neg p_9) \wedge (p_9 \vee \neg p_3 \vee p_8) \wedge (p_6 \vee \neg p_9 \vee p_5) \wedge \\
(p_2 \vee \neg p_3 \vee \neg p_8) \wedge (p_8 \vee \neg p_6 \vee \neg p_3) \wedge (p_8 \vee \neg p_3 \vee \neg p_1) \wedge \\
(\neg p_8 \vee p_6 \vee \neg p_2) \wedge (p_7 \vee p_9 \vee \neg p_2) \wedge (p_8 \vee \neg p_9 \vee p_2) \wedge \\
(\neg p_1 \vee \neg p_9 \vee p_4) \wedge (p_8 \vee p_1 \vee \neg p_2) \wedge (p_3 \vee \neg p_4 \vee \neg p_6) \wedge \\
(\neg p_1 \vee \neg p_7 \vee p_5) \wedge (\neg p_7 \vee p_1 \vee p_6) \wedge (\neg p_5 \vee p_4 \vee \neg p_6) \wedge \\
(\neg p_4 \vee p_9 \vee \neg p_8) \wedge (p_2 \vee p_9 \vee p_1) \wedge (p_5 \vee \neg p_7 \vee p_1) \wedge \\
(\neg p_7 \vee \neg p_9 \vee \neg p_6) \wedge (p_2 \vee p_5 \vee p_4) \wedge (p_8 \vee \neg p_4 \vee p_5) \wedge \\
(p_5 \vee p_9 \vee p_3) \wedge (\neg p_5 \vee \neg p_7 \vee p_9) \wedge (p_2 \vee \neg p_8 \vee p_1) \wedge \\
(\neg p_7 \vee p_1 \vee p_5) \wedge (p_1 \vee p_4 \vee p_3) \wedge (p_1 \vee \neg p_9 \vee \neg p_4) \wedge \\
(p_3 \vee p_5 \vee p_6) \wedge (\neg p_6 \vee p_3 \vee \neg p_9) \wedge (\neg p_7 \vee p_5 \vee p_9) \wedge \\
(p_7 \vee \neg p_5 \vee \neg p_2) \wedge (p_4 \vee p_7 \vee p_3) \wedge (p_4 \vee \neg p_9 \vee \neg p_7) \wedge \\
(p_5 \vee \neg p_1 \vee p_7) \wedge (p_5 \vee \neg p_1 \vee p_7) \wedge (p_6 \vee p_7 \vee \neg p_3) \wedge \\
(\neg p_8 \vee \neg p_6 \vee \neg p_7) \wedge (p_6 \vee p_2 \vee p_3) \wedge (\neg p_8 \vee p_2 \vee p_5)
\end{array}
$$

SAT Solver Paradigms Overview



DPLL: Aims at finding a small search-tree by selecting effective splitting variables (e.g. via looking ahead).

Strength: Effective on small, hard formulas.

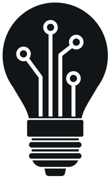
Weakness: Expensive.

SAT Solver Paradigms Overview

DPLL: Aims at finding a small search-tree by selecting effective splitting variables (e.g. via looking ahead).

Strength: Effective on small, hard formulas.

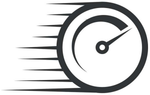
Weakness: Expensive.



Local search: Given a full assignment for a formula Γ, flip the truth values of variables until satisfying Γ.

Strength: Can quickly find solutions for hard formulas.

Weakness: Cannot prove unsatisfiability.

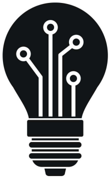


SAT Solver Paradigms Overview

DPLL: Aims at finding a small search-tree by selecting effective splitting variables (e.g. via looking ahead).

Strength: Effective on small, hard formulas.

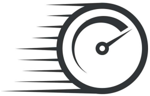
Weakness: Expensive.



Local search: Given a full assignment for a formula Γ, flip the truth values of variables until satisfying Γ.

Strength: Can quickly find solutions for hard formulas.

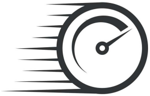
Weakness: Cannot prove unsatisfiability.



Conflict-driven clause learning (CDCL): Makes fast decisions and converts conflicts into learned clauses.

Strength: Effective on large, "easy" formulas.

Weakness: Hard to parallelize.



Conflict-driven Clause Learning Highlights

- Most successful architecture

Conflict-driven Clause Learning Highlights

- Most successful architecture
- Superior on industrial benchmarks

Conflict-driven Clause Learning Highlights

- Most successful architecture
- Superior on industrial benchmarks
- Brute-force?
  - Addition conflict clauses
  - Fast unit propagation

Conflict-driven Clause Learning Highlights

- Most successful architecture
- Superior on industrial benchmarks
- Brute-force?
  - Addition conflict clauses
  - Fast unit propagation
- Complete local search (for a refutation)?

Conflict-driven Clause Learning Highlights

- Most successful architecture
- Superior on industrial benchmarks
- Brute-force?
  - Addition conflict clauses
  - Fast unit propagation
- Complete local search (for a refutation)?
- State-of-the-art (sequential) CDCL solvers: CaDiCaL, Glucose, CryptoMiniSAT

Clause Learning

Data-structures

Heuristics

Proofs of Unsatisfiability

Clause Learning

Data-structures

Heuristics

Proofs of Unsatisfiability

Conflict-driven SAT solvers: Search and Analysis

$(p_1 \vee p_4) \wedge$
$(p_3 \vee \neg p_4 \vee p_5) \wedge$
$(\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge$
Γ_{extra}

0

Conflict-driven SAT solvers: Search and Analysis

$$
\begin{array}{l}
(p_1 \lor p_4) \land \\
(p_3 \lor \neg p_4 \lor p_5) \land \\
(\neg p_2 \lor \neg p_3 \lor \neg p_4) \land \\
\Gamma_{\mathrm{extra}}
\end{array}
$$

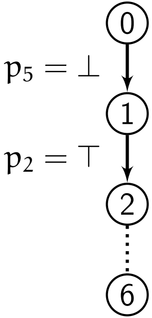
$$
\begin{tikzcd}
0 \arrow[d, "p_5 = \perp"'] \\
1
\end{tikzcd}
$$

Conflict-driven SAT solvers: Search and Analysis

$(p_1 \vee p_4) \wedge$
$(p_3 \vee \neg p_4 \vee p_5) \wedge$
$(\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge$
Γ_{extra}

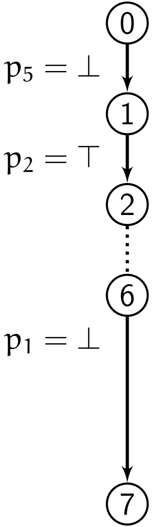
$$\begin{tikzcd} \textcircled{0} \arrow[d, "p_5 = \bot"'] \\ \textcircled{1} \arrow[d, "p_2 = \top"'] \\ \textcircled{2} \end{tikzcd}$$

$(p_1 \lor p_4) \land$
$(p_3 \lor \neg p_4 \lor p_5) \land$
$(\neg p_2 \lor \neg p_3 \lor \neg p_4) \land$
Γ_{extra}



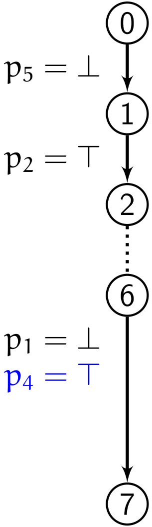
Conflict-driven SAT solvers: Search and Analysis

$(p_1 \vee p_4) \wedge$
$(p_3 \vee \neg p_4 \vee p_5) \wedge$
$(\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge$
Γ_{extra}



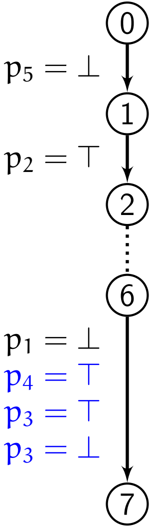
Conflict-driven SAT solvers: Search and Analysis

$(p_1 \vee p_4)\ \wedge$
$(p_3 \vee \neg p_4 \vee p_5)\ \wedge$
$(\neg p_2 \vee \neg p_3 \vee \neg p_4)\ \wedge$
Γ_{extra}



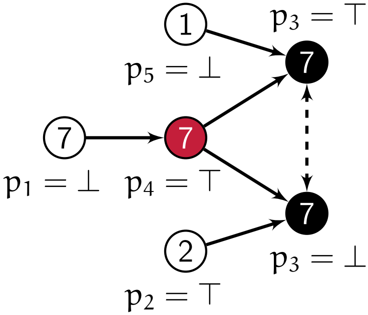
Conflict-driven SAT solvers: Search and Analysis

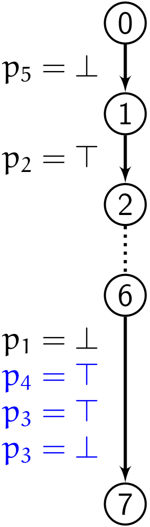
$(p_1 \lor p_4) \land$
$(p_3 \lor \neg p_4 \lor p_5) \land$
$(\neg p_2 \lor \neg p_3 \lor \neg p_4) \land$
Γ_{extra}



Conflict-driven SAT solvers: Search and Analysis

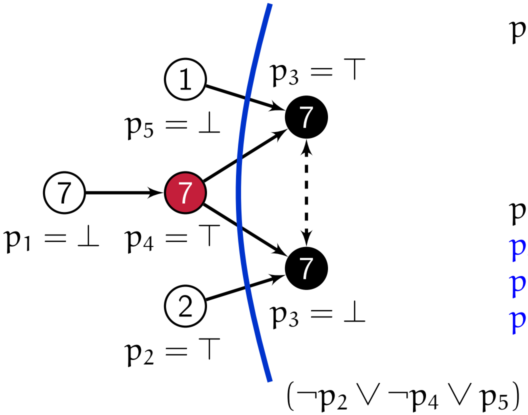
$(p_1 \lor p_4) \land$
$(p_3 \lor \neg p_4 \lor p_5) \land$
$(\neg p_2 \lor \neg p_3 \lor \neg p_4) \land$
Γ_{extra}

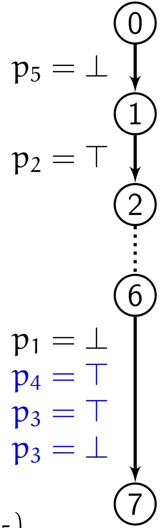




Conflict-driven SAT solvers: Search and Analysis

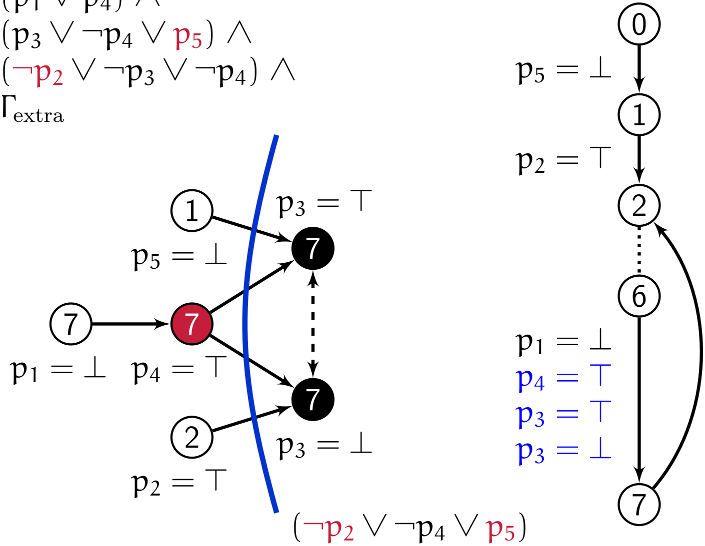
$(p_1 \lor p_4) \land$
$(p_3 \lor \neg p_4 \lor p_5) \land$
$(\neg p_2 \lor \neg p_3 \lor \neg p_4) \land$
Γ_{extra}





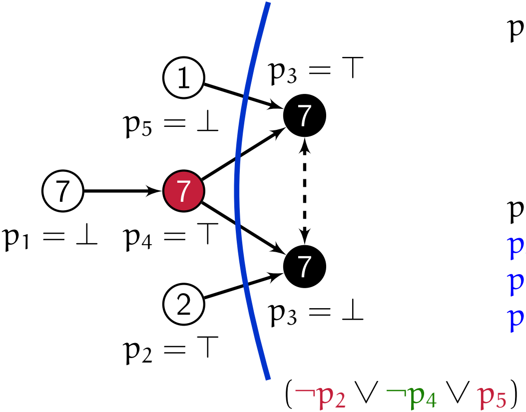
Conflict-driven SAT solvers: Search and Analysis

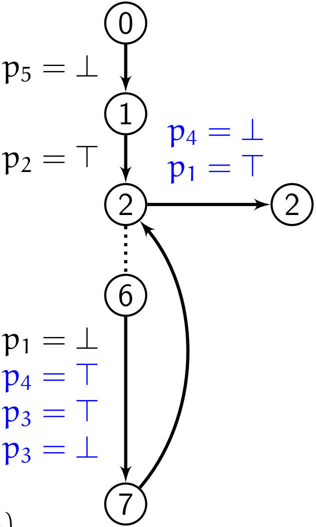
$(p_1 \vee p_4) \wedge$
$(p_3 \vee \neg p_4 \vee p_5) \wedge$
$(\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge$
Γ_{extra}



Conflict-driven SAT solvers: Search and Analysis

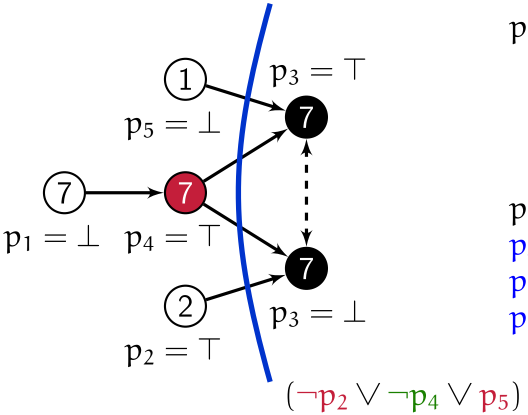
$(p_1 \vee p_4) \wedge$
$(p_3 \vee \neg p_4 \vee p_5) \wedge$
$(\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge$
Γ_{extra}

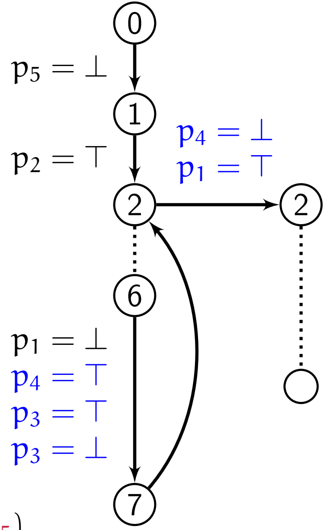




Conflict-driven SAT solvers: Search and Analysis

$(p_1 \vee p_4) \wedge$
$(p_3 \vee \neg p_4 \vee p_5) \wedge$
$(\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge$
Γ_{extra}





Reverse Unit Propagation

Let Γ be a formula. A clause C is implied by Γ via unit propagation (UP) if UP on $\Gamma \wedge \neg C$ results in a conflict.

Example

$$\Gamma = (p_1 \vee p_4) \wedge (p_3 \vee \neg p_4 \vee p_5) \wedge (\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge \ldots$$

Reverse Unit Propagation

Let Γ be a formula. A clause C is implied by Γ via unit propagation (UP) if UP on $\Gamma \wedge \neg C$ results in a conflict.

Example

$$\Gamma = (p_1 \vee p_4) \wedge (p_3 \vee \neg p_4 \vee p_5) \wedge (\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge \ldots$$

clause
units $\neg p_1 \wedge p_2 \wedge \neg p_5$

Reverse Unit Propagation

Let Γ be a formula. A clause C is implied by Γ via unit propagation (UP) if UP on $\Gamma \wedge \neg C$ results in a conflict.

Example

$\Gamma = (p_1 \vee p_4) \wedge (p_3 \vee \neg p_4 \vee p_5) \wedge (\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge \ldots$

clause	$(p_1 \vee p_4)$
units $\neg p_1 \wedge p_2 \wedge \neg p_5$	p_4

Reverse Unit Propagation

Let Γ be a formula. A clause C is implied by Γ via unit propagation (UP) if UP on $\Gamma \wedge \neg C$ results in a conflict.

Example

$\Gamma = (p_1 \vee p_4) \wedge (p_3 \vee \neg p_4 \vee p_5) \wedge (\neg p_2 \vee \neg p_3 \vee \neg p_4) \wedge \ldots$

clause		$(p_1 \vee p_4)$	$(p_3 \vee \neg p_4 \vee p_5)$
units	$\neg p_1 \wedge p_2 \wedge \neg p_5$	p_4	p_3

Reverse Unit Propagation

Let Γ be a formula. A clause C is implied by Γ via unit propagation (UP) if UP on $\Gamma \land \neg C$ results in a conflict.

Example

$\Gamma = (p_1 \lor p_4) \land (p_3 \lor \neg p_4 \lor p_5) \land (\neg p_2 \lor \neg p_3 \lor \neg p_4) \land \ldots$

clause		$(p_1 \lor p_4)$	$(p_3 \lor \neg p_4 \lor p_5)$	$(\neg p_2 \lor \neg p_3 \lor \neg p_4)$
units	$\neg p_1 \land p_2 \land \neg p_5$	p_4	p_3	$\perp$

Reverse Unit Propagation

Let Γ be a formula. A clause C is implied by Γ via unit propagation (UP) if UP on $\Gamma \land \neg C$ results in a conflict.

Example

$$\Gamma = (p_1 \lor p_4) \land (p_3 \lor \neg p_4 \lor p_5) \land (\neg p_2 \lor \neg p_3 \lor \neg p_4) \land \ldots$$

clause		$(p_1 \lor p_4)$	$(p_3 \lor \neg p_4 \lor p_5)$	$(\neg p_2 \lor \neg p_3 \lor \neg p_4)$
units	$\neg p_1 \land p_2 \land \neg p_5$	p_4	p_3	$\bot$

$$\dfrac{\dfrac{(\neg p_2 \lor \neg p_3 \lor \neg p_4) \qquad (p_3 \lor \neg p_4 \lor p_5)}{(\neg p_2 \lor \neg p_4 \lor p_5)} \qquad (p_1 \lor p_4)}{(p_1 \lor \neg p_2 \lor p_5)}$$

CDCL Overview

CDCL in a nutshell:

1. Main loop combines efficient problem simplification with cheap, but effective decision heuristics; ($> 90\%$ of time)
2. Reasoning kicks in if the current state is conflicting;
3. The current state is analyzed and turned into a constraint;
4. The constraint is added to the problem, the heuristics are updated, and the algorithm (partially) restarts.

CDCL Overview

CDCL in a nutshell:

1. Main loop combines efficient problem simplification with cheap, but effective decision heuristics; ($> 90\%$ of time)
2. Reasoning kicks in if the current state is conflicting;
3. The current state is analyzed and turned into a constraint;
4. The constraint is added to the problem, the heuristics are updated, and the algorithm (partially) restarts.

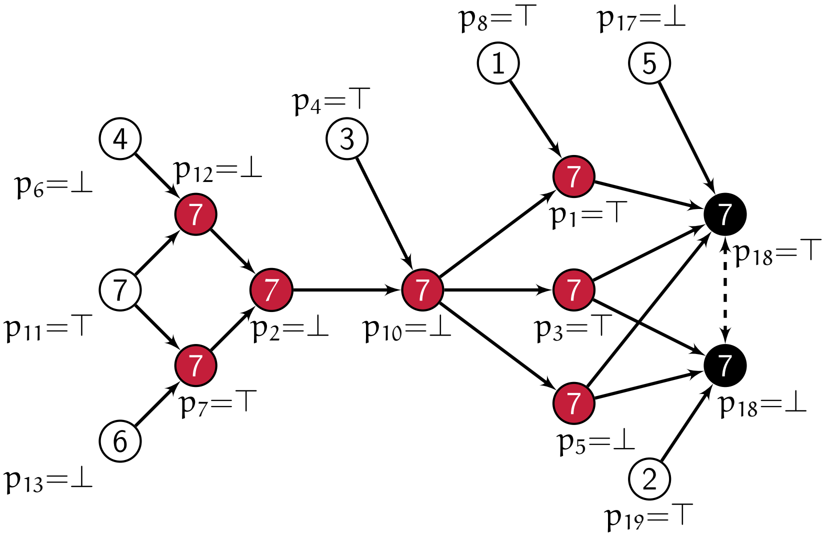
However, it has three weaknesses:

- CDCL is notoriously hard to parallelize;
- the representation impacts CDCL performance; and
- CDCL has exponential runtime on some "simple" problems.

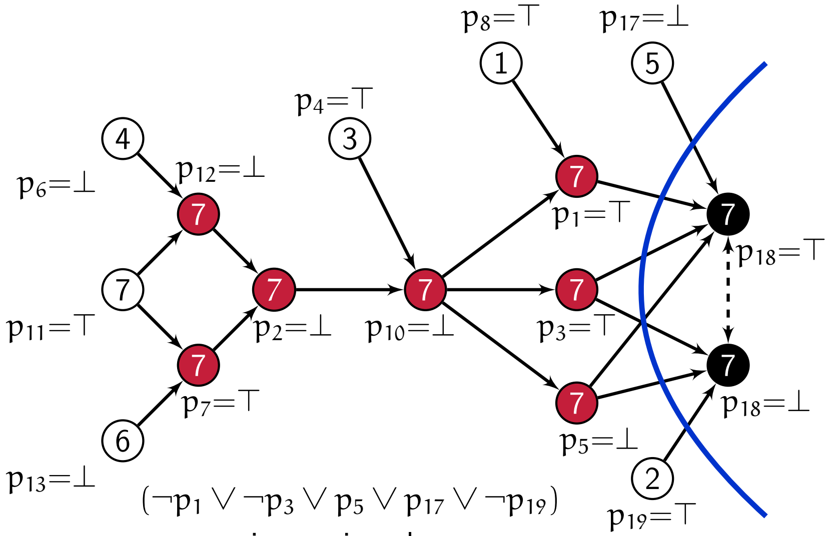
Conflict-driven Clause Learning: Pseudo-code

while TRUE **do**
    l_{decision} := Decide ()
    If no l_{decision} **then return** `satisfiable`
    τ := Simplify $(\tau \cup (l_{\text{decision}} = \top), \Gamma)$
    while $[\![\Gamma]\!]_\tau$ contains $C_{\text{falsified}}$ **do**
        C_{conflict} := Analyze $(C_{\text{falsified}}, \tau)$
        If $C_{\text{conflict}} = \bot$ **then return** `unsatisfiable`
        $\Gamma := \Gamma \cup \{C_{\text{conflict}}\}$
        τ := BackTrack $(\tau, C_{\text{conflict}})$
        τ := Simplify (τ, Γ)
    end while
end while

Learning conflict clauses [Marques-Silva,Sakallah'96]



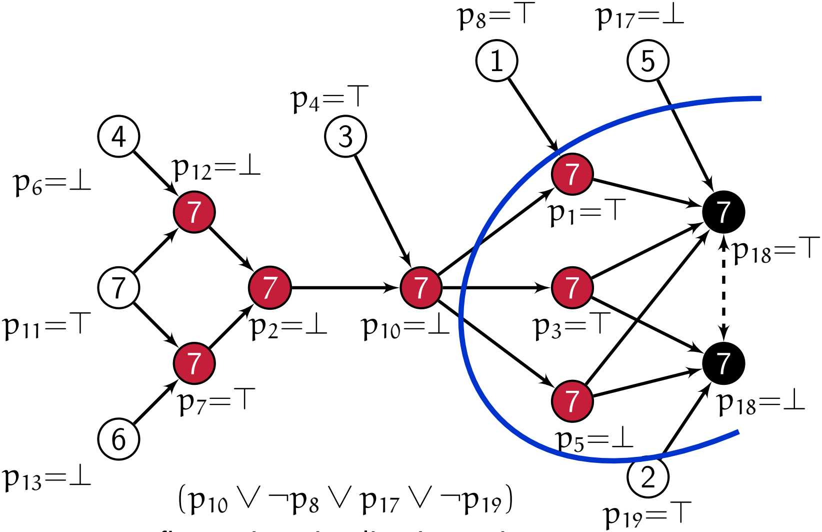
Learning conflict clauses [Marques-Silva,Sakallah'96]



$(\neg p_1 \vee \neg p_3 \vee p_5 \vee p_{17} \vee \neg p_{19})$

tri-asserting clause

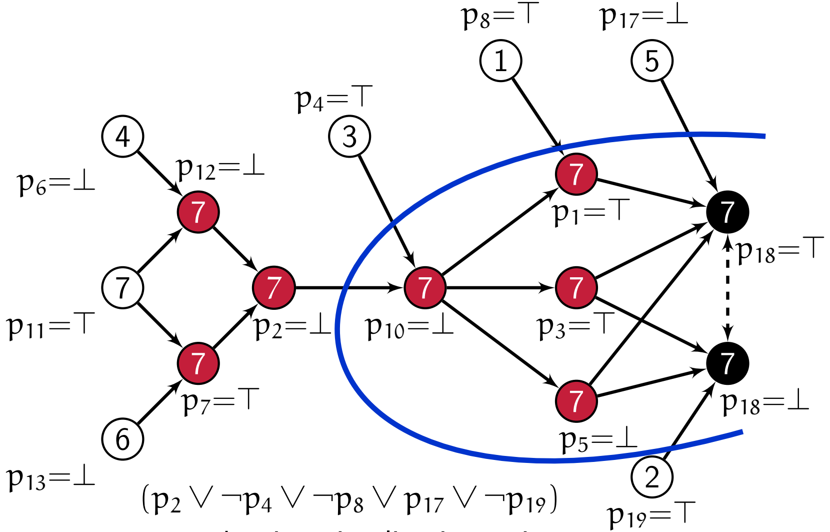
Learning conflict clauses [Marques-Silva,Sakallah'96]



$(p_{10} \vee \neg p_8 \vee p_{17} \vee \neg p_{19})$

first unique implication point

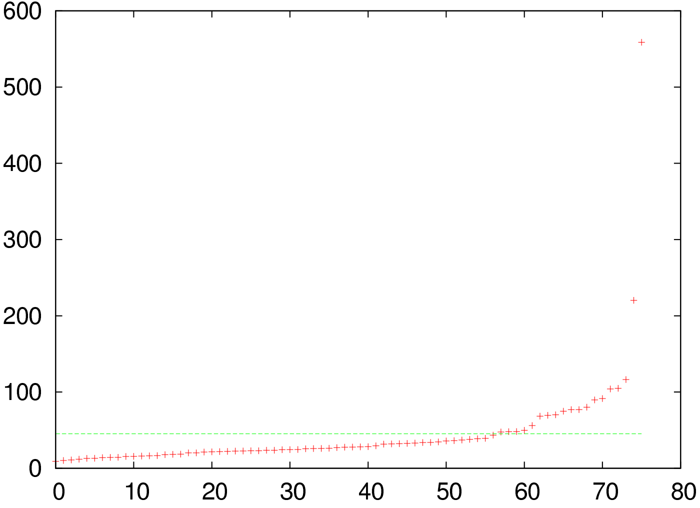
Learning conflict clauses [Marques-Silva,Sakallah'96]



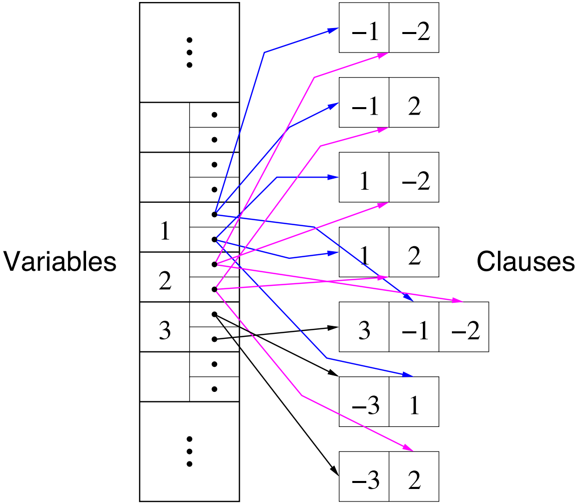
$(p_2 \vee \neg p_4 \vee \neg p_8 \vee p_{17} \vee \neg p_{19})$

second unique implication point

Average Learned Clause Length



Simple data structure for unit propagation



Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = *, p_2 = *, p_3 = *, p_4 = *, p_5 = *, p_6 = *\}$$

$\neg p_1$	p_2	$\neg p_3$	$\neg p_5$	p_6

p_1	$\neg p_3$	p_4	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

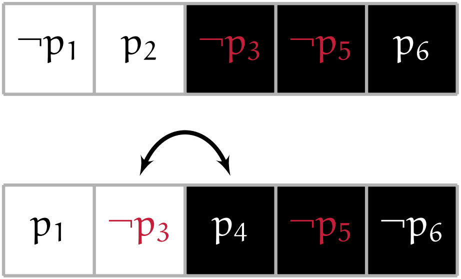
$$\tau = \{p_1 = *, p_2 = *, p_3 = *, p_4 = *, p_5 = \top, p_6 = *\}$$

$\neg p_1$	p_2	$\neg p_3$	$\neg p_5$	p_6

p_1	$\neg p_3$	p_4	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = *, p_2 = *, p_3 = \top, p_4 = *, p_5 = \top, p_6 = *\}$$



Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = *, p_2 = *, p_3 = \top, p_4 = *, p_5 = \top, p_6 = *\}$$

$\neg p_1$	p_2	$\neg p_3$	$\neg p_5$	p_6

p_1	p_4	$\neg p_3$	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = \top, p_2 = *, p_3 = \top, p_4 = *, p_5 = \top, p_6 = *\}$$

$\neg p_1$	p_2	$\neg p_3$	$\neg p_5$	p_6

p_1	p_4	$\neg p_3$	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = \top, p_2 = *, p_3 = \top, p_4 = *, p_5 = \top, p_6 = *\}$$

p_6	p_2	$\neg p_3$	$\neg p_5$	$\neg p_1$

p_1	p_4	$\neg p_3$	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = \top, p_2 = *, p_3 = \top, p_4 = \bot, p_5 = \top, p_6 = *\}$$

p_6	p_2	$\neg p_3$	$\neg p_5$	$\neg p_1$

p_1	p_4	$\neg p_3$	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = \top, p_2 = \bot, p_3 = \top, p_4 = \bot, p_5 = \top, p_6 = *\}$$

p_6	p_2	$\neg p_3$	$\neg p_5$	$\neg p_1$

p_1	p_4	$\neg p_3$	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (1) [MoskewiczMZZM'01]

$$\tau = \{p_1 = \top, p_2 = \bot, p_3 = \top, p_4 = \bot, p_5 = \top, p_6 = \top\}$$

p_6	p_2	$\neg p_3$	$\neg p_5$	$\neg p_1$

p_1	p_4	$\neg p_3$	$\neg p_5$	$\neg p_6$

Conflict-driven: Watch pointers (2) [MoskewiczMZZM'01]

Only examine (get in the cache) a clause when both

- a watch pointer gets falsified
- the other one is not satisfied

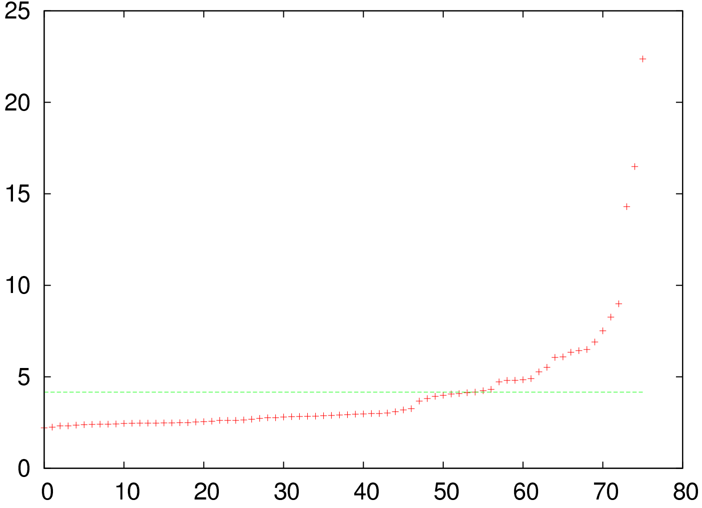
While backjumping, just unassign variables

Conflict clauses $\longrightarrow$ watch pointers

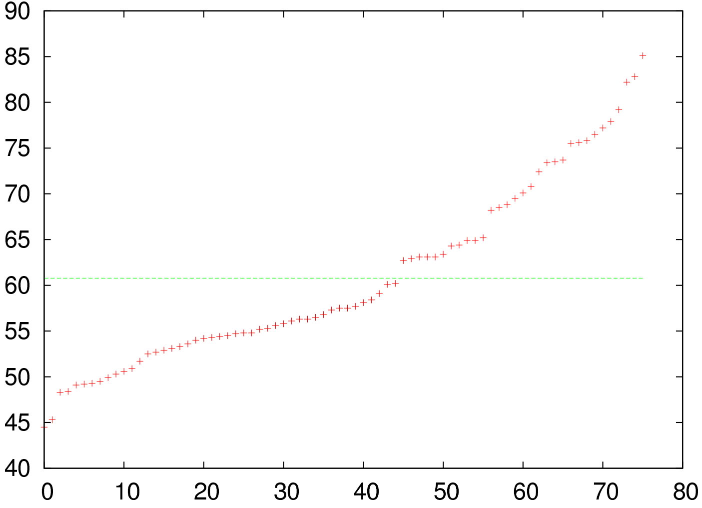
No detailed information available

Not used for binary clauses

Average Number Clauses Visited Per Propagation



Percentage visited clauses with other watched literal true



Clause Learning

Data-structures

Heuristics

Proofs of Unsatisfiability

Most important CDCL heuristics

Variable selection heuristics

- aim: minimize the search space
- plus: could compensate a bad value selection

Most important CDCL heuristics

Variable selection heuristics

- aim: minimize the search space
- plus: could compensate a bad value selection

Value selection heuristics

- aim: guide search towards a solution or conflict
- plus: could compensate a bad variable selection,
  cache solutions of subproblems [PipatsrisawatDarwiche'07]

Most important CDCL heuristics

Variable selection heuristics

- aim: minimize the search space
- plus: could compensate a bad value selection

Value selection heuristics

- aim: guide search towards a solution or conflict
- plus: could compensate a bad variable selection, cache solutions of subproblems [PipatsrisawatDarwiche'07]

Restart strategies

- aim: avoid heavy-tail behavior [GomesSelmanCrato'97]
- plus: focus search on recent conflicts when combined with dynamic heuristics

Variable selection heuristics

Based on the occurrences in the (reduced) formula

- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Variable selection heuristics

Based on the occurrences in the (reduced) formula

- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Variable State Independent Decaying Sum (VSIDS)

- original idea (zChaff): for each conflict, increase the score of involved variables by 1, half all scores each 256 conflicts [MoskewiczMZZM'01]
- improvement (MiniSAT): for each conflict, increase the score of involved variables by δ and increase $\delta := 1.05\delta$ [EenSörensson'03]

Visualization of VSIDS in PicoSAT

http://www.youtube.com/watch?v=MOjhFywLre8

Value selection heuristics

Based on the occurrences in the (reduced) formula

- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Value selection heuristics

Based on the occurrences in the (reduced) formula

- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Based on the encoding / consequently

- negative branching (early MiniSAT) [EenSörensson'03]

Value selection heuristics

Based on the occurrences in the (reduced) formula

- examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads
- not practical for CDCL solver due to watch pointers

Based on the encoding / consequently

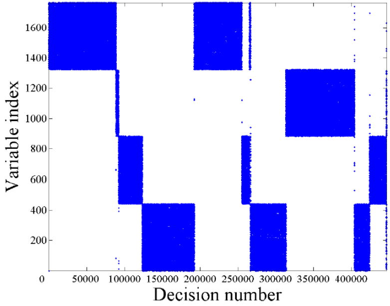
- negative branching (early MiniSAT) [EenSörensson'03]

Based on the last implied value (phase-saving)

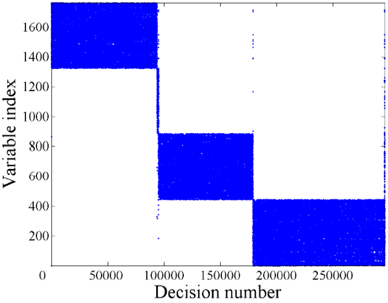
- introduced to CDCL [PipatsrisawatDarwiche'07]
- already used in local search [HirschKojevnikov'01]

Heuristics: Phase-saving [PipatsrisawatDarwiche'07]

Selecting the last implied value remembers solved components



negative branching



phase-saving

Restarts

Restarts in CDCL solvers:

- Counter heavy-tail behavior [GomesSelmanCrato'97]
- Unassign all variables but keep the (dynamic) heuristics

Restarts

Restarts in CDCL solvers:

- Counter heavy-tail behavior [GomesSelmanCrato'97]
- Unassign all variables but keep the (dynamic) heuristics

Restart strategies: [Walsh'99, LubySinclairZuckerman'93]

- Geometrical restart: e.g. $100, 150, 225, 333, 500, 750, \ldots$
- Luby sequence: e.g. $100, 100, 200, 100, 100, 200, 400, \ldots$

Restarts

Restarts in CDCL solvers:

- Counter heavy-tail behavior [GomesSelmanCrato'97]
- Unassign all variables but keep the (dynamic) heuristics

Restart strategies: [Walsh'99, LubySinclairZuckerman'93]

- Geometrical restart: e.g. $100, 150, 225, 333, 500, 750, \ldots$
- Luby sequence: e.g. $100, 100, 200, 100, 100, 200, 400, \ldots$

Rapid restarts by reusing trail: [vanderTakHeuleRamos'11]

- Partial restart same effect as full restart
- Optimal strategy Luby-1: $1, 1, 2, 1, 1, 2, 4, \ldots$

Clause Learning

Data-structures

Heuristics

Proofs of Unsatisfiability

Motivation for Proofs of Unsatisfiability

SAT solvers may have errors and only return yes/no.

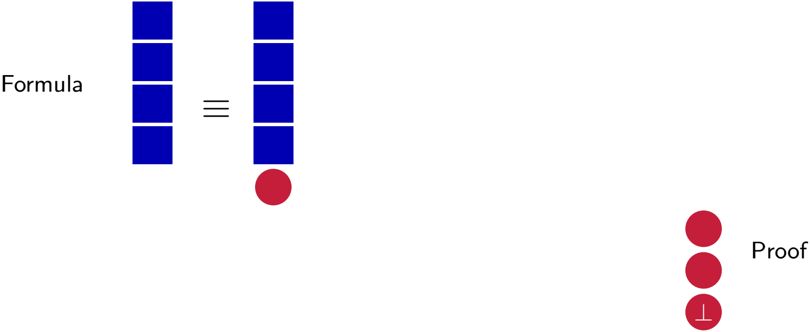
- Documented bugs in SAT, SMT, and QSAT solvers;
  [Brummayer and Biere, 2009; Brummayer et al., 2010]
- Competition winners have contradictory results
  (HWMCC winners from 2011 and 2012)
- Implementation errors often imply conceptual errors;
- Proofs now mandatory for the annual SAT Competitions;
- Mathematical results require a stronger justification than a simple yes/no by a solver. UNSAT must be verifiable.

Clausal Proofs of Unsatisfiability

Reduce the size of the proof by only storing added clauses

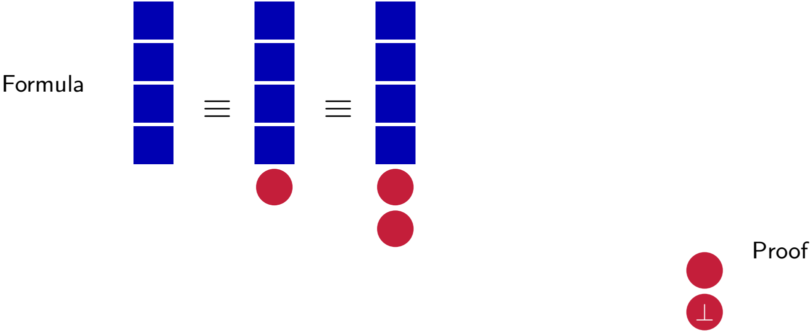
Clausal Proofs of Unsatisfiability

Reduce the size of the proof by only storing added clauses



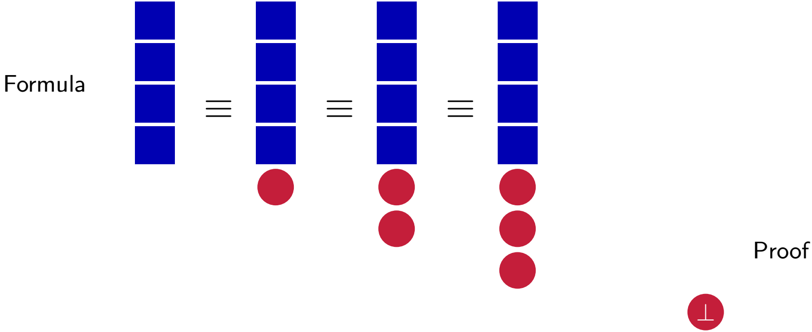
Clausal Proofs of Unsatisfiability

Reduce the size of the proof by only storing added clauses



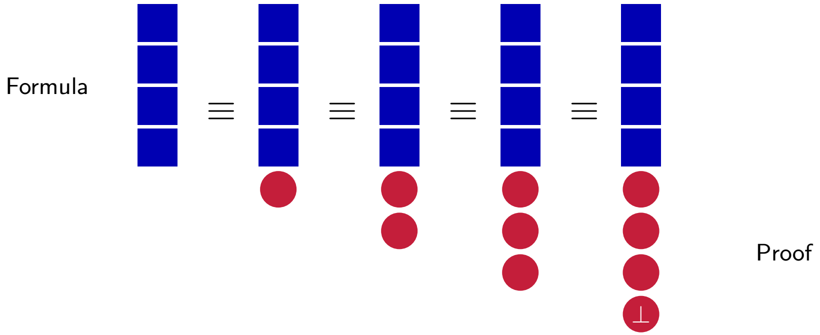
Clausal Proofs of Unsatisfiability

Reduce the size of the proof by only storing added clauses



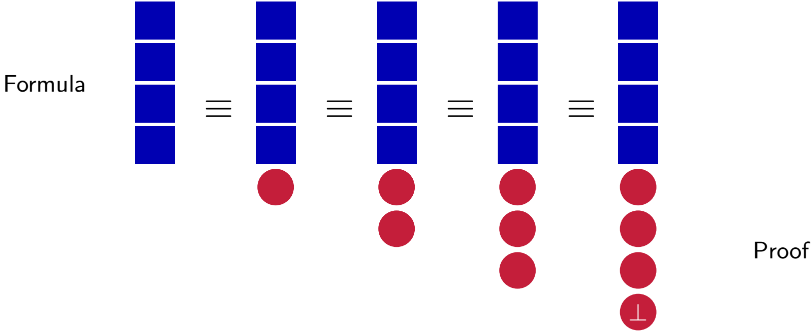
Clausal Proofs of Unsatisfiability

Reduce the size of the proof by only storing added clauses



Clausal Proofs of Unsatisfiability

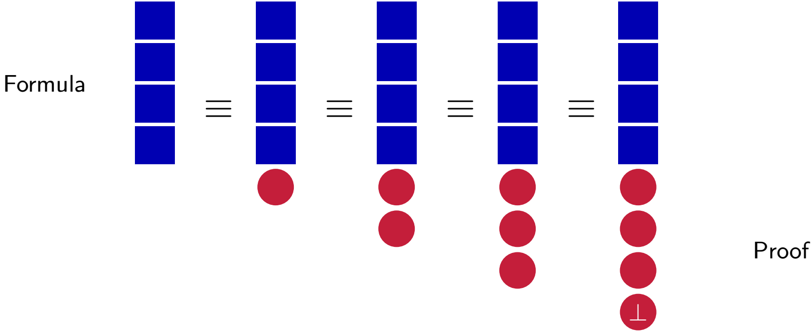
Reduce the size of the proof by only storing added clauses



- Clauses whose addition preserves satisfiability are *redundant*.
- Checking redundancy should be efficient.

Clausal Proofs of Unsatisfiability

Reduce the size of the proof by only storing added clauses



- Clauses whose addition preserves satisfiability are *redundant*.
- Checking redundancy should be efficient.
- Proof systems for this purpose in upcoming lectures.