Conflict-Driven Clause Learning

Current Trends in AI

Bart Bogaerts

March 13, 2020

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

THANKS

- Many slides provided by Joao Marques Silva
- Material of a SAT/SMT summer school `http://satsmt2013.ics.aalto.fi/slides/Marques-Silva.pdf`
- Forms the basis of *Joao Marques-Silva, Sharad Malik: Propositional SAT Solving. Handbook of Model Checking 2018: 247-275*

OVERVIEW

- The SAT problem
- DPLL (1962)
- CDCL (1996)
- What’s hot in SAT?
- Tentacles of CDCL

The SAT problem

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

THE SUCCESS OF SAT

- Given a formula in propositional logic (in CNF), decide whether it is satisfiable

THE SUCCESS OF SAT

- Given a formula in propositional logic (in CNF), decide whether it is satisfiable
- Well-known NP-complete decision problem [C71]

THE SUCCESS OF SAT

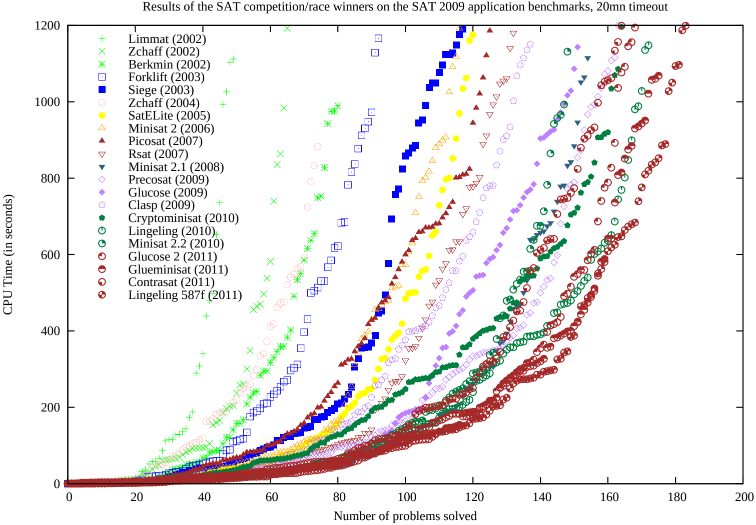
- Given a formula in propositional logic (in CNF), decide whether it is satisfiable
- Well-known NP-complete decision problem [C71]
- In practice, SAT is a success story of Computer Science
  - Hundreds (even more?) of practical applications

THE SUCCESS OF SAT

- Given a formula in propositional logic (in CNF), decide whether it is satisfiable
- Well-known NP-complete decision problem [C71]
- In practice, SAT is a success story of Computer Science
  - Hundreds (even more?) of practical applications

SAT SOLVER IMPROVEMENT

[Source: Le Berre&Biere 2011]



PRELIMINARIES

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- (Partial) assignment: partial/total mapping from variables to $\{0, 1\}$
- Model: (partial) assignment such that at least one literal in every clause is true (1)
- Formula can be SAT/UNSAT

PRELIMINARIES

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- (Partial) assignment: partial/total mapping from variables to $\{0, 1\}$
- Model: (partial) assignment such that at least one literal in every clause is true (1)
- Formula can be SAT/UNSAT
- Example:

$$\mathcal{F} \triangleq (r) \wedge (\bar{r} \vee s) \wedge (\bar{w} \vee a) \wedge (\bar{x} \vee b) \wedge (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$

  - Example models:
    - $\{r, s, a, b, c, d\}$
    - $\{r, s, \bar{x}, y, \bar{w}, z, \bar{a}, b, c, d\}$

RESOLUTION

- Resolution rule: [DP60,R65]

$$\frac{(\alpha \vee x) \qquad\qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

  - Complete proof system for propositional logic

RESOLUTION

- Resolution rule: [DP60,R65]

$$\frac{(\alpha \vee x) \qquad\qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

  - Complete proof system for propositional logic

$$\begin{tikzcd}
(x \vee a) \arrow[d, dash] & & (\bar{x} \vee a) \arrow[dll, dash] & & (\bar{y} \vee \bar{a}) \arrow[drr, dash] & & (y \vee \bar{a}) \arrow[d, dash] \\
(a) \arrow[drrr, dash] & & & & & & (\bar{a}) \arrow[dlll, dash] \\
& & & \perp & & &
\end{tikzcd}$$

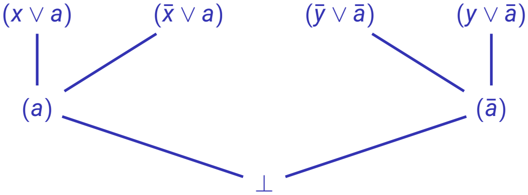
  - Extensively used with (CDCL) SAT solvers

RESOLUTION

- Resolution rule: [DP60,R65]

$$\frac{(\alpha \vee x) \qquad\qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

  - Complete proof system for propositional logic



  - Extensively used with (CDCL) SAT solvers

- Self-subsuming resolution (with $\alpha' \subseteq \alpha$): [e.g. SP04,EB05]

$$\frac{(\alpha \vee x) \qquad\qquad (\alpha' \vee \bar{x})}{(\alpha)}$$

  - (α) subsumes $(\alpha \vee x)$

UNIT PROPAGATION

$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

UNIT PROPAGATION

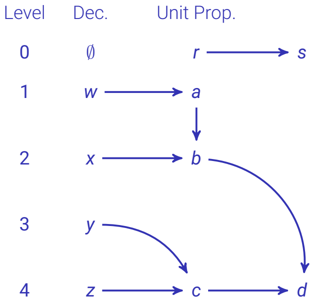
$$\mathcal{F} = (r) \land (\bar{r} \lor s) \land$$
$$(\bar{w} \lor a) \land (\bar{x} \lor \bar{a} \lor b)$$
$$(\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d)$$

- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

UNIT PROPAGATION

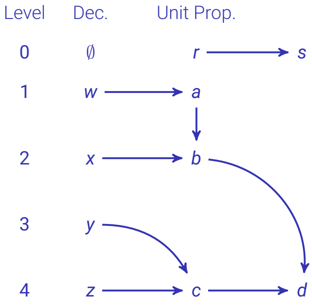
$$
\begin{aligned}
\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\
& (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\
& (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)
\end{aligned}
$$

- Decisions / Variable Branchings: $w = 1, x = 1, y = 1, z = 1$



UNIT PROPAGATION

$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$



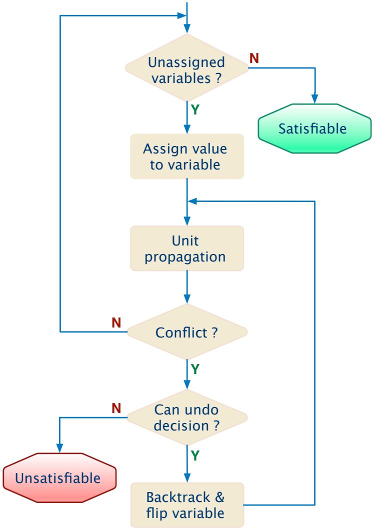
- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

- Additional definitions:
  - Antecedent (or reason) of an implied assignment
    - $(\bar{b} \vee \bar{c} \vee d)$ for d
  - Associate assignment with decision levels
    - $w = 1 @ 1, x = 1 @ 2, y = 1 @ 3, z = 1 @ 4$
    - $r = 1 @ 0, d = 1 @ 4, \ldots$

DPLL

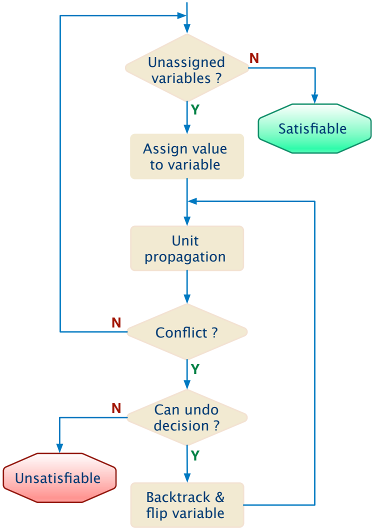
ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

THE DPLL ALGORITHM



- Optional: pure literal rule

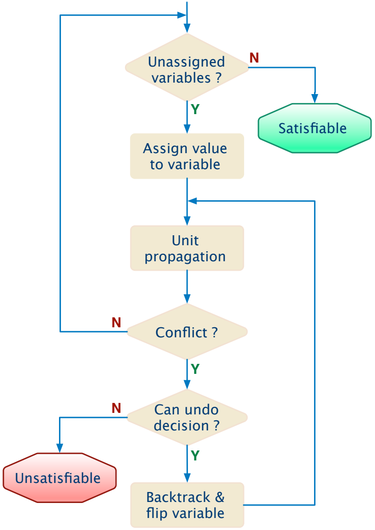
THE DPLL ALGORITHM



$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

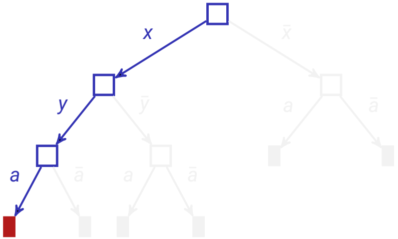
- Optional: pure literal rule

THE DPLL ALGORITHM



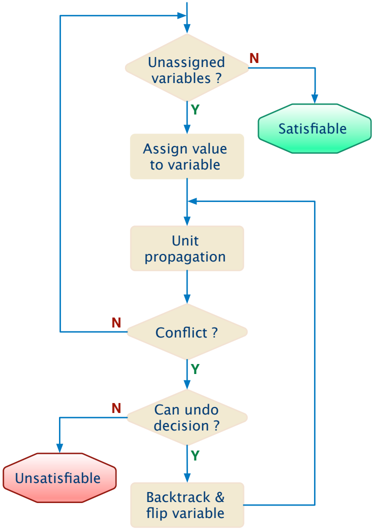
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	a	$\rightarrow b \rightarrow \perp$



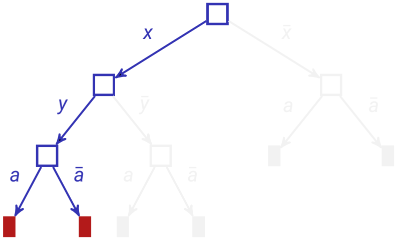
- Optional: pure literal rule

THE DPLL ALGORITHM



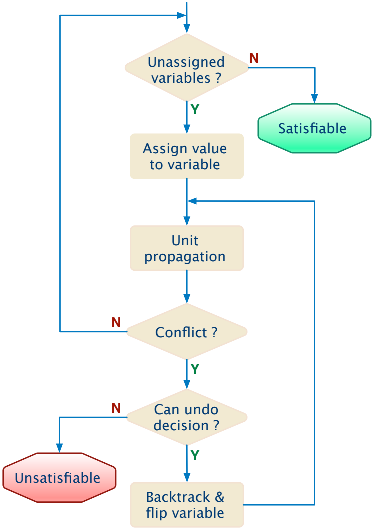
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	$\bar{a}$	$\longrightarrow \bar{b} \longrightarrow \perp$



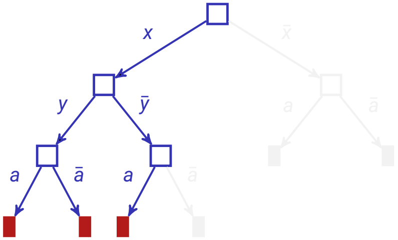
- Optional: pure literal rule

THE DPLL ALGORITHM



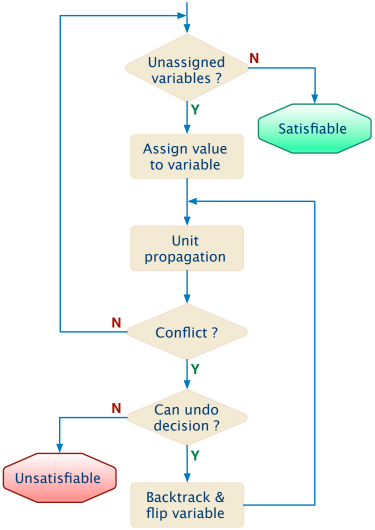
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	a	$\to b \to \perp$



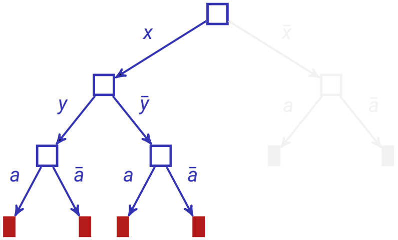
- Optional: pure literal rule

THE DPLL ALGORITHM



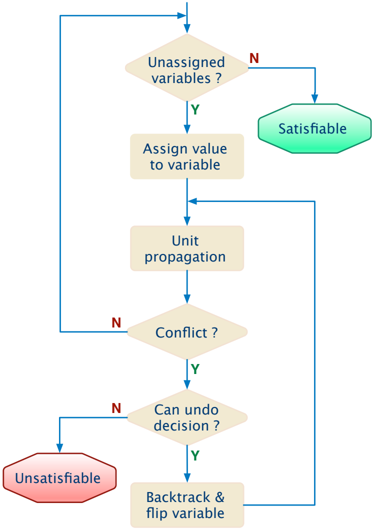
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	$\bar{a}$	$\longrightarrow \bar{b} \longrightarrow \perp$



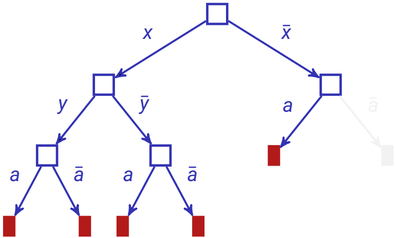
- Optional: pure literal rule

THE DPLL ALGORITHM



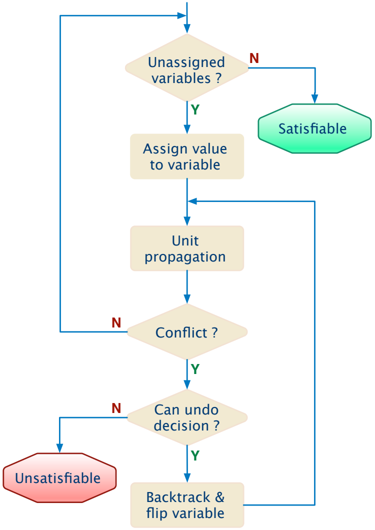
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	$\bar{x}$	$\longrightarrow y$
2	a	$\longrightarrow b \longrightarrow \perp$



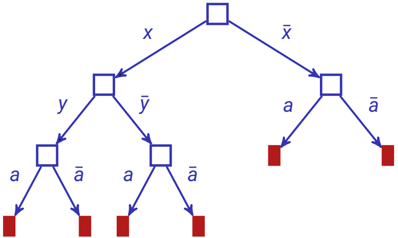
- Optional: pure literal rule

THE DPLL ALGORITHM



$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	$\bar{x} \longrightarrow$	y
2	$\bar{a} \longrightarrow$	$\bar{b} \longrightarrow \perp$



- Optional: pure literal rule

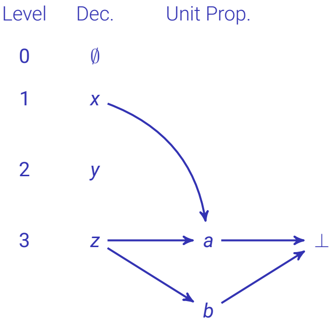
CDCL

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

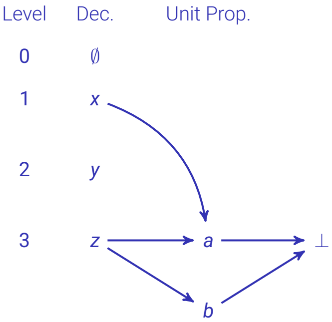
WHAT IS A CDCL SAT SOLVER?

- Extend DPLL SAT solver with: [DP60,DLL62]
  - Clause learning & non-chronological backtracking [MSS96,BS97,Z97]
    - Exploit UIPs [MSS96,SSS12]
    - Minimize learned clauses [SB09,VG09]
    - Opportunistically delete clauses [MSS96,MSS99,GN02]
  - Search restarts [GSK98,BMS00,H07,B08]
  - Lazy data structures
    - Watched literals [MMZZM01]
  - Conflict-guided branching
    - Lightweight branching heuristics [MMZZM01]
    - Phase saving [PD07]
  - ...

CLAUSE LEARNING

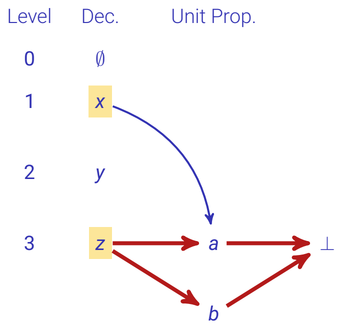


CLAUSE LEARNING



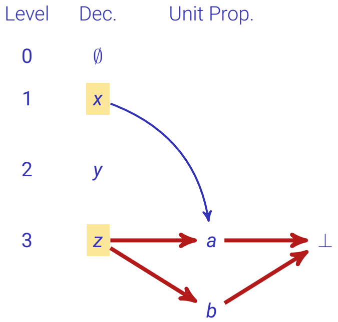
- Analyze conflict

CLAUSE LEARNING



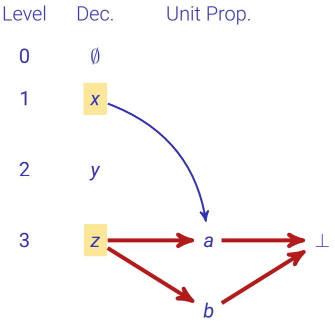
- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels

CLAUSE LEARNING



- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$

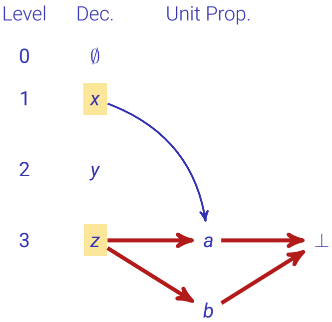
CLAUSE LEARNING

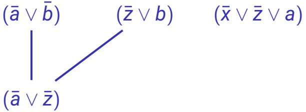


$(\bar{a} \vee \bar{b})$ $(\bar{z} \vee b)$ $(\bar{x} \vee \bar{z} \vee a)$

- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

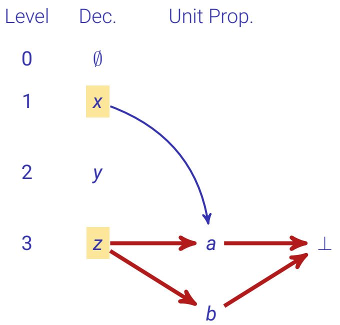
CLAUSE LEARNING

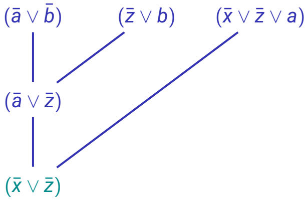




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

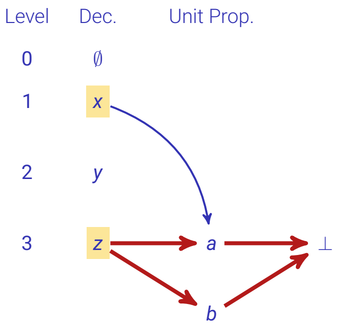
CLAUSE LEARNING

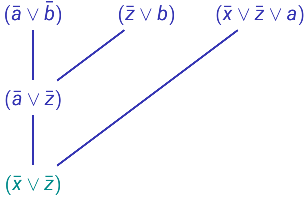




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

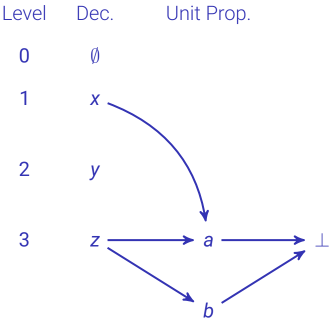
CLAUSE LEARNING



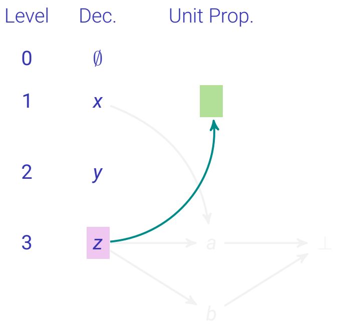


- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

CLAUSE LEARNING – AFTER BRACKTRACKING

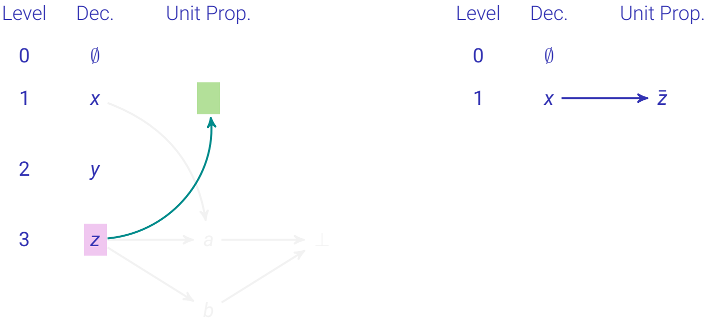


CLAUSE LEARNING – AFTER BRACKTRACKING



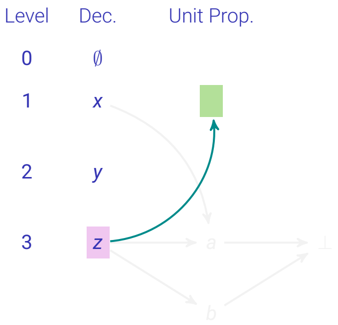
- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

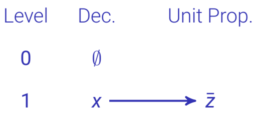
CLAUSE LEARNING – AFTER BRACKTRACKING



- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

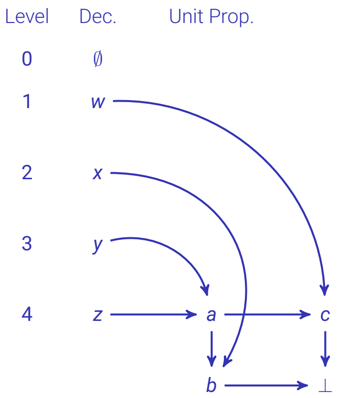
CLAUSE LEARNING – AFTER BRACKTRACKING



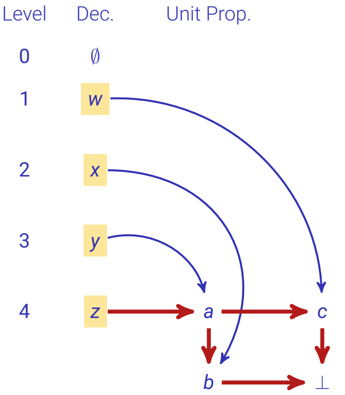


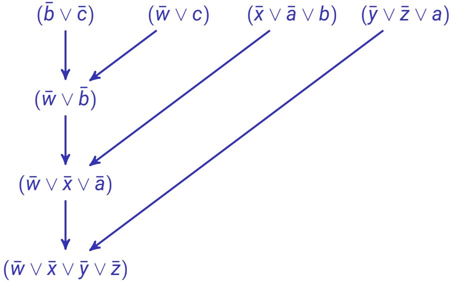
- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1
- Learned clauses are always asserting [MSS96,MSS99]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict [MMZZM01]

UNIQUE IMPLICATION POINTS (UIPS)



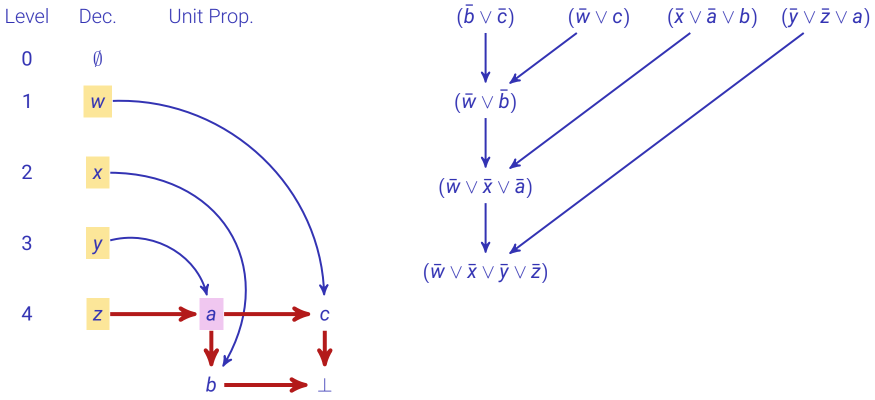
UNIQUE IMPLICATION POINTS (UIPS)





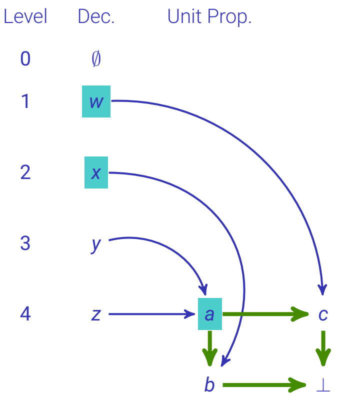
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

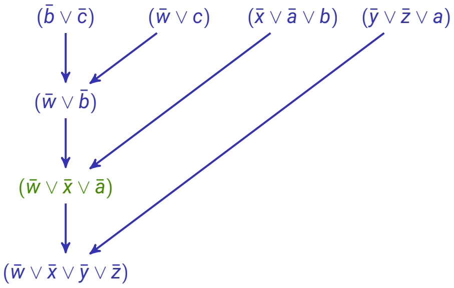
UNIQUE IMPLICATION POINTS (UIPS)



- Learn clause $(\bar{w} \lor \bar{x} \lor \bar{y} \lor \bar{z})$
- But *a* is a UIP

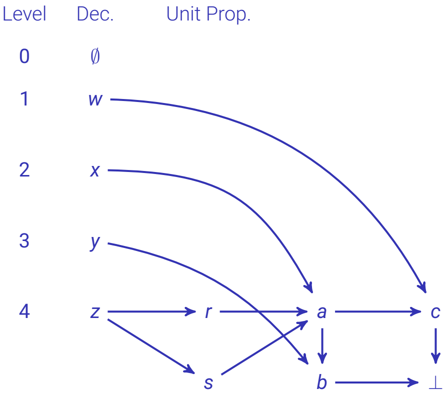
UNIQUE IMPLICATION POINTS (UIPS)



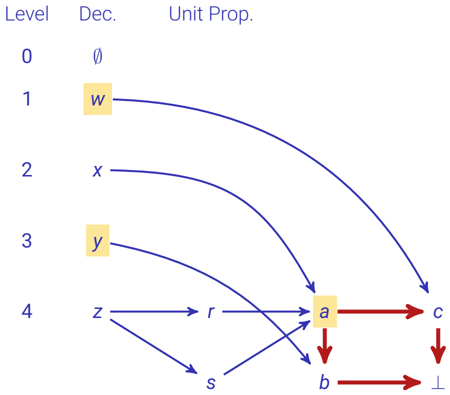


- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But *a* is a UIP
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

MULTIPLE UIPS

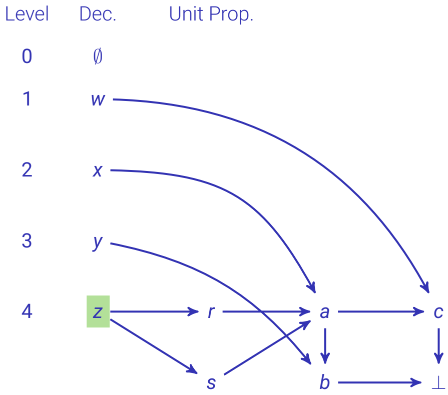


MULTIPLE UIPS



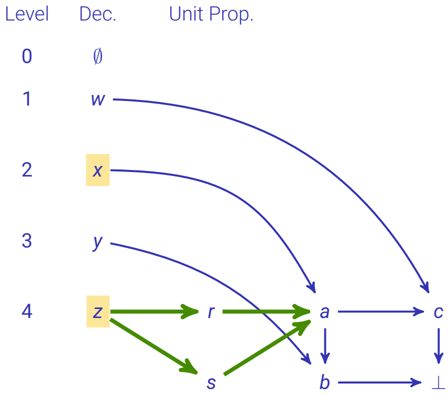
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$

MULTIPLE UIPS



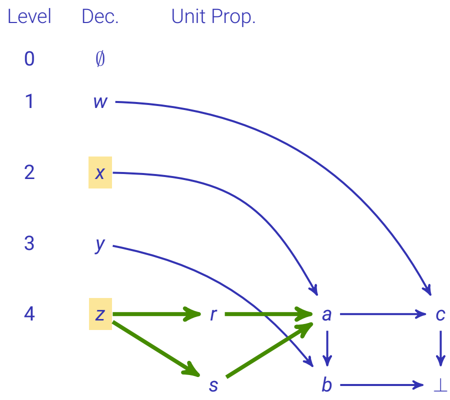
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP

MULTIPLE UIPS



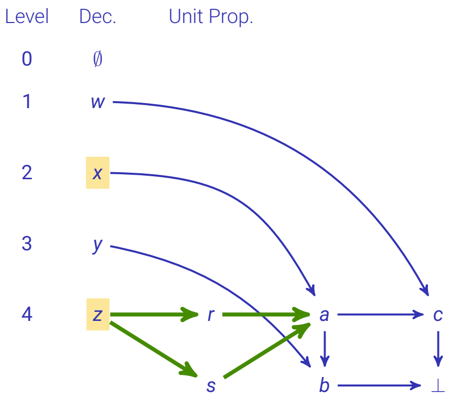
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$

MULTIPLE UIPS



- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$
- In practice smaller clauses more effective
  - Compare with $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

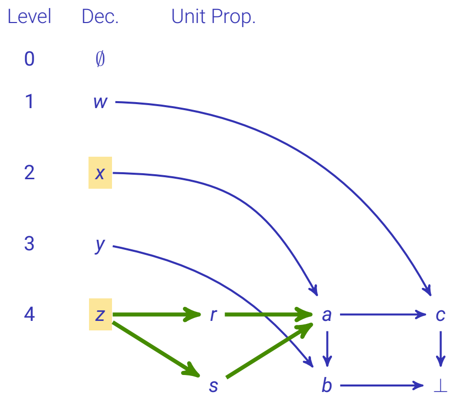
MULTIPLE UIPS



- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$
- In practice smaller clauses more effective
  - Compare with $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

- Multiple UIPs proposed in GRASP [MSS96]
  - First UIP learning proposed in Chaff [MMZZM01]
- Not used in recent state of the art CDCL SAT solvers

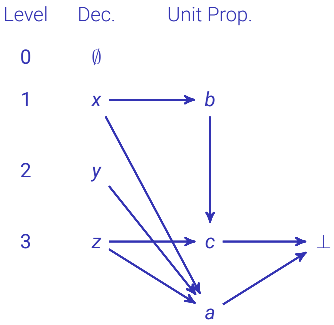
MULTIPLE UIPS



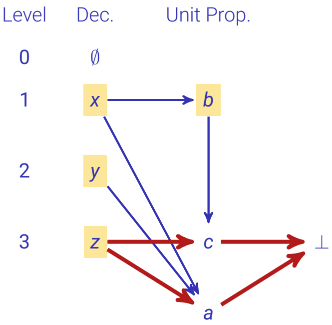
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$
- In practice smaller clauses more effective
  - Compare with $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

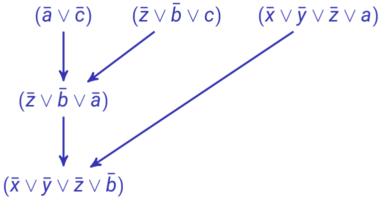
- Multiple UIPs proposed in GRASP [MSS96]
  - First UIP learning proposed in Chaff [MMZZM01]
- Not used in recent state of the art CDCL SAT solvers
- Recent results show it can be beneficial on current instances [SSS12]

CLAUSE MINIMIZATION I



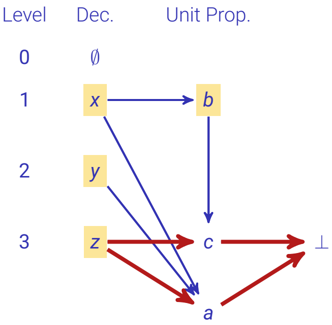
CLAUSE MINIMIZATION I

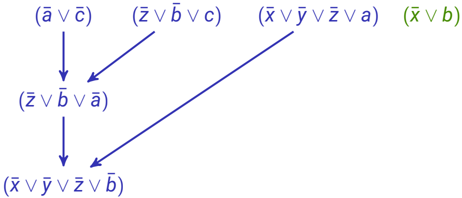




- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$

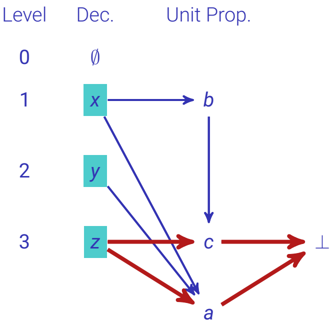
CLAUSE MINIMIZATION I

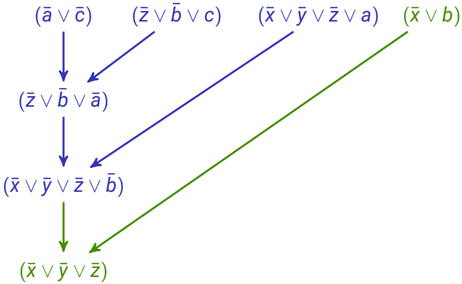




- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$
- Apply self-subsuming resolution (i.e. local minimization) [SB09]

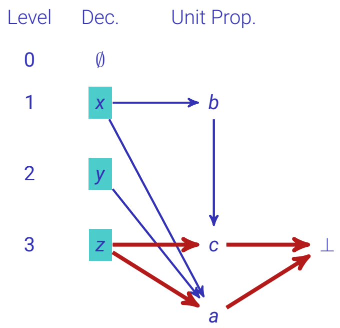
CLAUSE MINIMIZATION I

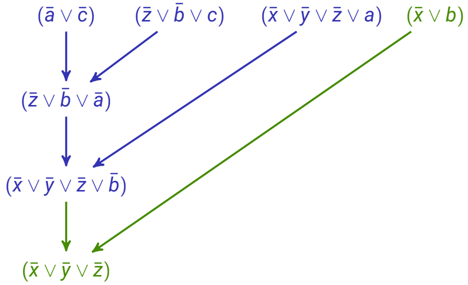




- ~~Learn clause~~ $\cancel{(\bar{x} \lor \bar{y} \lor \bar{z} \lor \bar{b})}$
- Apply self-subsuming resolution (i.e. local minimization)

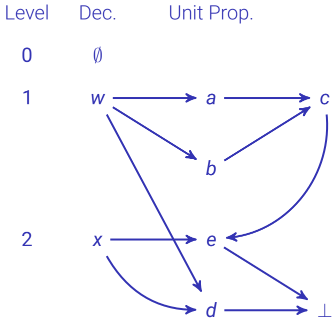
CLAUSE MINIMIZATION I



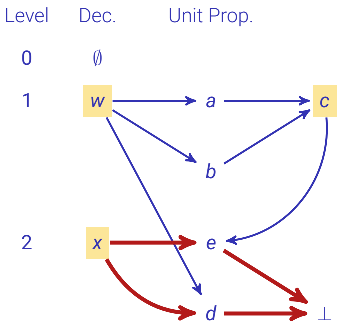


- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization)
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z})$

CLAUSE MINIMIZATION II

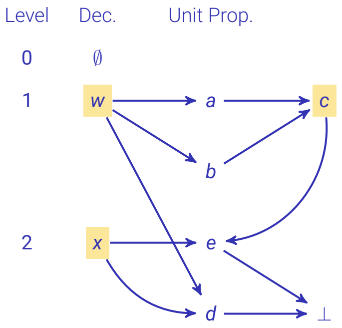


CLAUSE MINIMIZATION II



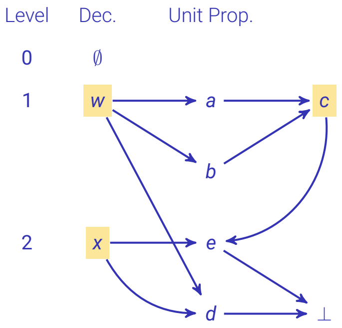
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$

CLAUSE MINIMIZATION II



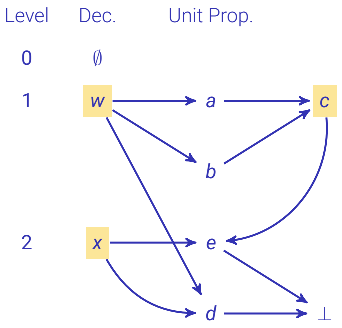
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$

CLAUSE MINIMIZATION II



- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization

CLAUSE MINIMIZATION II

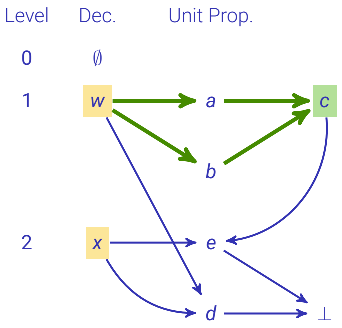


- ~~Learn clause ($\bar{w} \vee \bar{x} \vee \bar{c}$)~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields ($\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b}$)
- Can apply recursive minimization

- Marked nodes: literals in learned clause

[SB09]

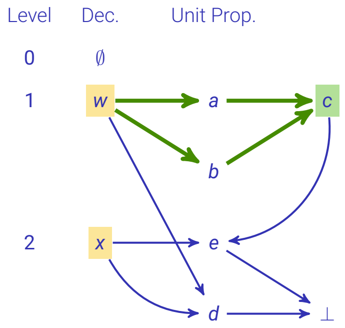
CLAUSE MINIMIZATION II



- ~~Learn clause~~ $\cancel{(\bar{w} \vee \bar{x} \vee \bar{c})}$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization

- Marked nodes: literals in learned clause [SB09]
- Trace back from c until marked nodes or new nodes / decisions
  - Learn clause if only marked nodes visited

CLAUSE MINIMIZATION II

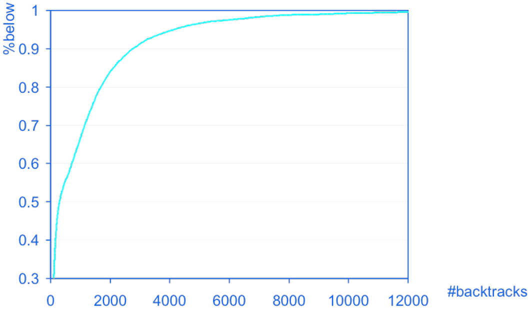


- ~~Learn clause ($\bar{w} \vee \bar{x} \vee \bar{c}$)~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields ($\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b}$)
- Can apply recursive minimization
- Learn clause ($\bar{w} \vee \bar{x}$)

- Marked nodes: literals in learned clause [SB09]
- Trace back from c until marked nodes or new nodes / decisions
  - Learn clause if only marked nodes visited

SEARCH RESTARTS I

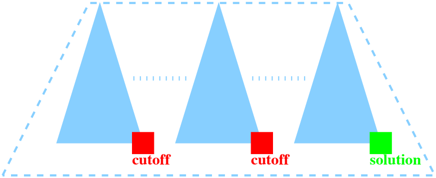
- Heavy-tail behavior: [GSK98]



- 10000 runs, branching randomization on industrial instance
  - Use rapid randomized restarts (search restarts)

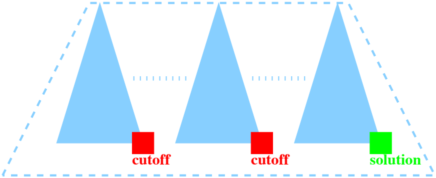
SEARCH RESTARTS II

- Restart search after a number of conflicts



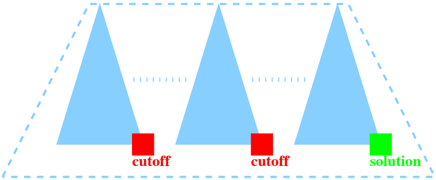
SEARCH RESTARTS II

- Restart search after a number of conflicts
- Increase cutoff after each restart
  - Guarantees completeness
  - Different policies exist (see refs)



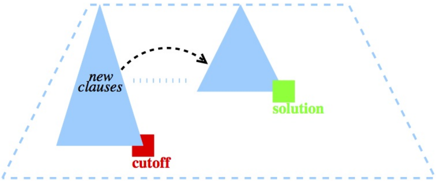
SEARCH RESTARTS II

- Restart search after a number of conflicts
- Increase cutoff after each restart
  - Guarantees completeness
  - Different policies exist (see refs)
- Works for SAT & UNSAT instances. Why?



SEARCH RESTARTS II

- Restart search after a number of conflicts
- Increase cutoff after each restart
  - Guarantees completeness
  - Different policies exist (see refs)
- Works for SAT & UNSAT instances. Why?
- Learned clauses effective after restart(s)



DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why?

DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why? Unit propagation

DATA STRUCTURES BASICS

- Each literal *l* should access clauses containing *l*
  - Why? Unit propagation
- Clause with *k* literals results in *k* references, from literals to the clause

DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L

DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why? Unit propagation

- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$

DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$

DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**

DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**
- Clause learning to be effective requires a more efficient representation:

DATA STRUCTURES BASICS

- Each literal l should access clauses containing l
  - Why? Unit propagation

- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**

- Clause learning to be effective requires a more efficient representation: **Watched Literals**

DATA STRUCTURES BASICS

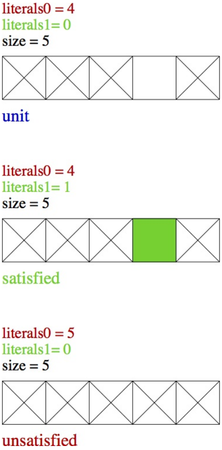
- Each literal l should access clauses containing l
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\, n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**
- Clause learning to be effective requires a more efficient representation: **Watched Literals**
  - Watched literals are one example of lazy data structures
    - But there are others

WATCHED LITERALS

[MMZZM01]

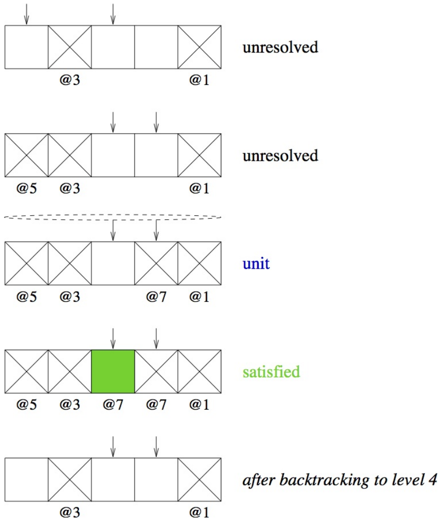
- Important states of a clause



WATCHED LITERALS

[MMZZM01]

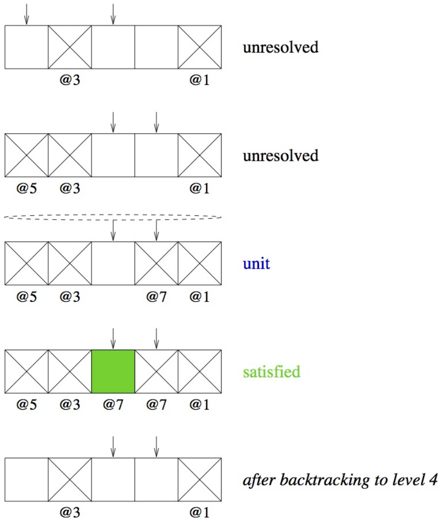
- Important states of a clause
- Associate **2** references with each clause



WATCHED LITERALS

[MMZZM01]

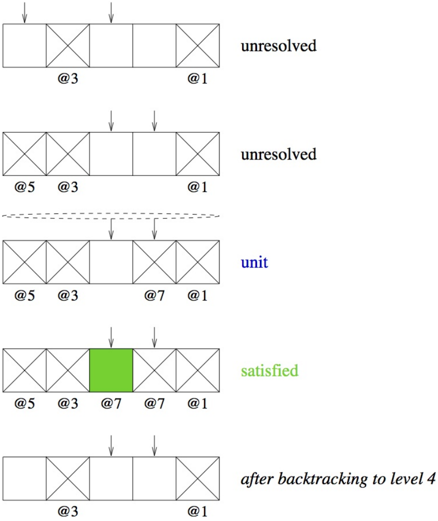
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals



WATCHED LITERALS

[MMZZM01]

- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals
- References **unchanged** when backtracking



ADDITIONAL KEY TECHNIQUES

- Lightweight branching [e.g. MMZZM01]
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores

ADDITIONAL KEY TECHNIQUES

- Lightweight branching [e.g. MMZZM01]
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores
- Clause deletion policies
  - Not practical to keep all learned clauses
  - Delete less used clauses [e.g. MSS96,GN02,ES03]

ADDITIONAL KEY TECHNIQUES

- Lightweight branching [e.g. MMZZM01]
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores
- Clause deletion policies
  - Not practical to keep all learned clauses
  - Delete less used clauses [e.g. MSS96,GN02,ES03]
- Proven recent techniques:
  - Phase saving [PD07]
  - Literal blocks distance [AS09]

What´s hot in SAT

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

WHAT'S HOT IN SAT?

- Clause learning techniques [e.g. ABHJS08,AS09]
  - Clause learning is the key technique in CDCL SAT solvers
  - Many recent papers propose improvements to the basic clause learning approach
- Preprocessing & inprocessing
  - Many recent papers [e.g. JHB12,HJB11]
  - Lots of recent work on symmetry exploitation (static/dynamic) [e.g. DBB17,JKKK17]
  - Essential in some applications

WHAT'S HOT IN SAT?

- Proofs
  - Proof logging (RUP, RAT, DRAT) [HHKW17]
  - Proof complexity [VEGGN18]
- Other Inference Methods
  - (Probabilistic) Model counting [e.g. AHT18]
  - Optimisation (E.g., MAXSAT – more later) [e.g. LM09]
  - Enumeration
  - MUSes / MCSes
- Applications
  - In various domains

Tentacles of CDCL

ARTIFICIAL
INTELLIGENCE
RESEARCH GROUP

SOME TENTACLES OF CDCL

- Lazy Clause Generation for Constraint solving or SAT modulo theories
- Conflict-driven pseudo-Boolean solving
- Incremental SAT solving for MAXSAT & QBF.

SAT ENCODINGS

- Many different problems can easily be encoded into SAT
- For instance, finite-domain Constraint Solving
- Various encoding options:
  - Equality: encode variable $X \in [-100, 100]$ by Boolean variables $[\![x{=}-100]\!]$, $[\![x{=}-99]\!]$, ... with uniqueness constraints
  - Bound: encode variable $X \in [-100, 100]$ by Boolean variables $[\![x{\leq}-100]\!]$, $[\![x{\leq}-99]\!]$, ... with constraints

$$\overline{[\![x{\leq}-100]\!]} \vee [\![x{\leq}-99]\!], \qquad \overline{[\![x{\leq}-99]\!]} \vee [\![x{\leq}-98]\!], \ldots$$

  - Log: encode variable $X \in [-100, 100]$ by means of bitvectors
- This talk assumes the Bound encoding.
- For each type of constraints, an encoding has to be invented

SAT ENCODINGS – EXAMPLE

$$X, Y, Z, U, V \in [-100, 100] \tag{1}$$
$$4U - X - Y \geq 4 \tag{2}$$
$$V \geq U \tag{3}$$
$$Z \geq 5V \tag{4}$$
$$Y + Z \leq 24 \tag{5}$$

$$(\overline{[\![X{\leq}-100]\!]} \vee [\![X{\leq}-99]\!]) \wedge (\overline{[\![X{\leq}-99]\!]} \vee [\![X{\leq}-98]\!]) \wedge \cdots \wedge (\overline{[\![Y{\leq}-100]\!]} \vee [\![Y{\leq}-99]\!]) \wedge \ldots$$
$$\cdots \wedge ([\![X{\leq}-3]\!] \vee [\![Y{\leq}9]\!] \vee \overline{[\![U{\leq}2]\!]}) \wedge \cdots \wedge ([\![X{\leq}9]\!] \vee [\![Y{\leq}9]\!] \vee \overline{[\![U{\leq}5]\!]}) \wedge \ldots$$
$$(\overline{[\![V{\leq}100]\!]} \vee [\![U{\leq}100]\!]) \wedge (\overline{[\![V{\leq}99]\!]} \vee [\![U{\leq}99]\!]) \wedge \cdots \wedge (\overline{[\![V{\leq}5]\!]} \vee [\![U{\leq}5]\!]) \wedge \ldots$$
$$\cdots \wedge ([\![V{\leq}0]\!] \vee \overline{[\![Z{\leq}4]\!]}) \wedge \cdots \wedge ([\![V{\leq}2]\!] \vee \overline{[\![Z{\leq}14]\!]}) \wedge \ldots$$
$$\cdots \wedge ([\![Y{\leq}9]\!] \vee [\![Z{\leq}14]\!]) \wedge \ldots$$

CONSTRAINT PROGRAMMING USING SAT

- If the SAT encoding of a CP program is not too large (at least: fits in memory), we can create it eagerly and use a CDCL solver to solve it.
- But... we can also generate it lazily = Lazy Clause Generation (LCG)
  - Many constraint propagators work by search + domain propagation
  - Idea: generate parts of the encoding only when CDCL solvers needs it:
    - During propagation
    - During explanation

CONSTRAINT PROGRAMMING USING SAT

- If the SAT encoding of a CP program is not too large (at least: fits in memory), we can create it eagerly and use a CDCL solver to solve it.
- But... we can also generate it lazily = Lazy Clause Generation (LCG)
  - Many constraint propagators work by search + domain propagation
  - Idea: generate parts of the encoding only when CDCL solvers needs it:
    - During propagation
    - During explanation

      Can use structure in constraints to learn better clauses **!**

      Example on Blackboard

CONSTRAINT PROGRAMMING USING SAT

- If the SAT encoding of a CP program is not too large (at least: fits in memory), we can create it eagerly and use a CDCL solver to solve it.
- But... we can also generate it lazily = Lazy Clause Generation (LCG)
  - Many constraint propagators work by search + domain propagation
  - Idea: generate parts of the encoding only when CDCL solvers needs it:
    - During propagation
    - During explanation

      Can use structure in constraints to learn better clauses **!**

      Example on Blackboard
- Many more interesting phenomena going on in LCG (see you next week!)

PSEUDO-BOOLEAN SOLVING

Observations:

- Resolution proof system is weak (cfr Pigeonhole)
- Stronger proof systems exist, for instance cutting planes makes use of (linear) pseudo-Boolean constraints (linear constraints over literals) [CCT87]
  - A clause $a \vee \bar{b} \vee c$ corresponds to a PB constraint

$$a + \bar{b} + c \geq 1$$

  - A PB constraint

$$a + \bar{b} + 2 \cdot c + d \geq 2$$

    cannot be translated into a single clause

CUTTING PLANE PROOF SYSTEM

$$\frac{}{l \geq 0} \quad \text{(literal axiom)}$$

$$\frac{\sum_i a_i l_i \geq A \qquad \sum_i b_i l_i \geq B}{\sum_i (ca_i + db_i) l_i \geq cA + dB} \quad \text{(linear combination)}$$

$$\frac{\sum_i a_i l_i \geq A}{\sum_i \lceil a_i / c \rceil l_i \geq \lceil A / c \rceil} \quad \text{(division)}$$

CUTTING PLANES VS RESOLUTION

- In theory, learning cutting planes could allow to derive unsat proofs much faster
- In practice, CDCL solvers seem to outperform cutting plane solvers
- Very recently, new cutting-plane solvers, inspired by CDCL are arising [GNY19]
  - Various issues show up: generalizing CDCL, 1UIP, ... far from obvious!

INCREMENTAL SAT SOLVING & SAT ORACLES

- Incremental SAT Solving: [ES01]
  - Allow calling a solver with a set of assumptions
    - Variables whose value is set before the search start (never backtrack over them!)
  - Often used: replace each clause C_i with $C_i \vee \neg a_i$
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
  - Enables clause reuse

INCREMENTAL SAT SOLVING & SAT ORACLES

- Incremental SAT Solving: [ES01]
  - Allow calling a solver with a set of assumptions
    - Variables whose value is set before the search start (never backtrack over them!)
  - Often used: replace each clause C_i with $C_i \vee \neg a_i$
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
  - Enables clause reuse
- Answer of a SAT solver:
  - SAT + satisfying assignment
  - UNSAT + unsat core (MUS)

INCREMENTAL SAT SOLVING & SAT ORACLES

- Incremental SAT Solving: [ES01]
  - Allow calling a solver with a set of assumptions
    - Variables whose value is set before the search start (never backtrack over them!)
  - Often used: replace each clause C_i with $C_i \vee \neg a_i$
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
  - Enables clause reuse
- Answer of a SAT solver:
  - SAT + satisfying assignment
  - UNSAT + unsat core (MUS)
- Use: SAT solver as oracle in encompassing algorithm
  - For optimization (MAXSAT)
  - For tackling problems arbitrary high up the polynomial hierarchy (QBF)
  - Cores/Assignments often used in encompassing algorithm (which might be a CDCL/LCG solver itself!)

THANK YOU

If you are interested in doing research in this direction (Master thesis / PhD), don't hesitate to e-mail me, or drop by my office

`bart.bogaerts@vub.be`
Pleinlaan 9, 3.67

REFERENCES

DP60 M. Davis, H. Putnam: A Computing Procedure for Quantification Theory. J. ACM 7(3): 201-215 (1960)

DLL62 M. Davis, G. Logemann, D. Loveland: A machine program for theorem-proving. Commun. ACM 5(7): 394-397 (1962)

MSS96 J. Marques-Silva, K. Sakallah: GRASP - a new search algorithm for satisfiability. ICCAD 1996: 220-227

BS97 R. Bayardo Jr., R. Schrag: Using CSP Look-Back Techniques to Solve Real-World SAT Instances. AAAI/IAAI 1997: 203-208

Z97 H. Zhang: SATO: An Efficient Propositional Prover. CADE 1997: 272-275

GSK98 C. Gomes, B. Selman, H. Kautz: Boosting Combinatorial Search Through Randomization. AAAI 1998: 431-437

MSS99 J. Marques-Silva, K. Sakallah: GRASP: A Search Algorithm for Propositional Satisfiability. IEEE Trans. Computers 48(5): 506-521 (1999)

BMS00 L. Baptista, J. Marques-Silva: Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. CP 2000: 489-494

MMZZM01 M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik: Chaff: Engineering an Efficient SAT Solver. DAC 2001: 530-535

REFERENCES

GN02	E. Goldberg, Y. Novikov: BerkMin: A Fast and Robust Sat-Solver. DATE 2002: 142-149
ES03	N. Een, Niklas Sorensson: An Extensible SAT-solver. SAT 2003: 502-518
PD07	K. Pipatsrisawat, A. Darwiche: A Lightweight Component Caching Scheme for Satisfiability Solvers. SAT 2007: 294-299
H07	J. Huang: The Effect of Restarts on the Efficiency of Clause Learning. IJCAI 2007: 2318-2323
ABHJS08	G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, L. Sais: A Generalized Framework for Conflict Analysis. SAT 2008: 21-27
B08	A. Biere: PicoSAT Essentials. JSAT 4(2-4): 75-97 (2008)
SB09	N. Sorensson, A. Biere: Minimizing Learned Clauses. SAT 2009: 237-243
VG09	A. Van Gelder: Improved Conflict-Clause Minimization Leads to Improved Propositional Proof Traces. SAT 2009: 141-146
AS09	G. Audemard, L. Simon: Predicting Learnt Clauses Quality in Modern SAT Solvers. IJCAI 2009: 399-404
SSS12	A. Sabharwal, H. Samulowitz, M. Sellmann: Learning Back-Clauses in SAT. SAT 2012: 498-499

REFERENCES

DBB17 Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe: Symmetric Explanation Learning: Effective Dynamic Symmetry Handling for SAT. SAT 2017: 83-100

JKKK17 Tommi A. Junttila, Matti Karppa, Petteri Kaski, Jukka Kohonen: An Adaptive Prefix-Assignment Technique for Symmetry Reduction.SAT 2017:101-118

HHKW17 Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, Nathan Wetzler: Efficient, Verified Checking of Propositional Proofs. ITP 2017: 269-284

VEGGN18 Marc Vinyals, Jan Elffers, JesÃºs GirÃ¡ldez-Cru, Stephan Gocht, Jakob Nordström: In Between Resolution and Cutting Planes: A Study of Proof Systems for Pseudo-Boolean SAT Solving. SAT 2018: 292-310

AHT18 Dimitris Achlioptas, Zayd Hammoudeh, Panos Theodoropoulos: Fast and Flexible Probabilistic Model Counting. SAT 2018: 148-164

CCT87 William J. Cook, Collette R. Coullard, György Turan: On the complexity of cutting-plane proofs. Discret. Appl. Math. 18(1): 25-38 (1987)

LP09 Chu Min Li, Felip ManyÃ : MaxSAT, Hard and Soft Constraints. Handbok of SAT 613-631 (2009)

GNY19 Stephan Gocht, Jakob Nordström, Amir Yehudayoff: On Division Versus Saturation in Pseudo-Boolean Solving. IJCAI 2019: 1711-1718