CDCL SAT Solvers

Joao Marques-Silva

INESC-ID, IST, ULisbon, Portugal

Theory and Practice of SAT Solving
Dagstuhl Workshop

April 2015

The Success of SAT

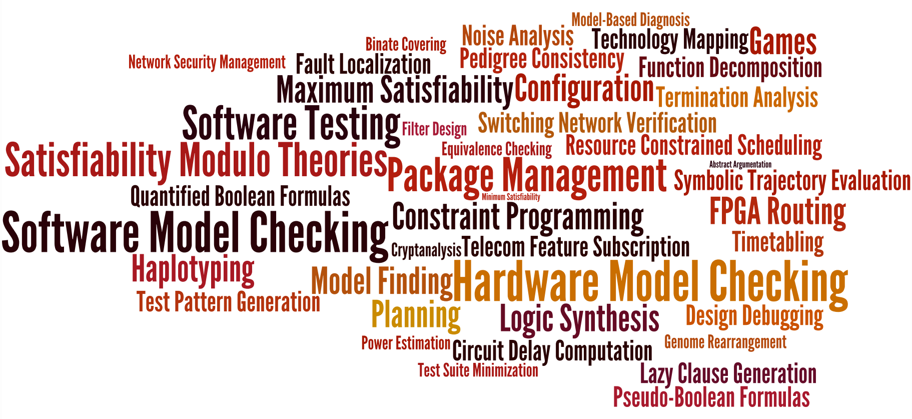
- Well-known NP-complete decision problem [C71]

The Success of SAT

- Well-known NP-complete decision problem [C71]
- In practice, SAT is a success story of Computer Science
  - Hundreds (even more?) of practical applications

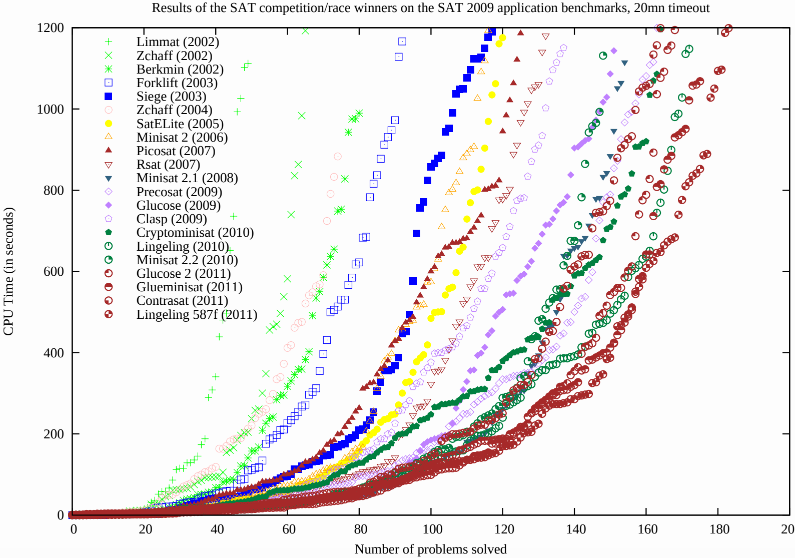
The Success of SAT

- Well-known NP-complete decision problem [C71]
- In practice, SAT is a success story of Computer Science
  - Hundreds (even more?) of practical applications

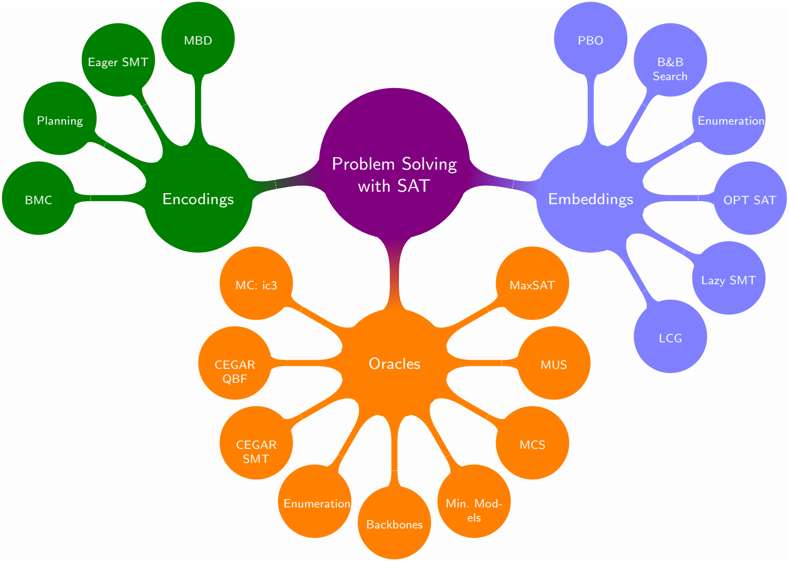


SAT Solver Improvement

[Source: Le Berre&Biere 2011]



SAT-Based Problem Solving



This Talk

- Review key ideas in implementing CDCL SAT solvers
  - Review standard concepts
    - Unit propagation
    - Resolution
    - Proof traces
    - ...
  - Outline organization of DPLL/CDCL SAT solvers
  - Survey most effective techniques
    - Clause learning & non-chronological bakctracking
    - UIPs
    - Clause minimization
    - Search restarts
    - Several heuristics
    - ...

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

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
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- 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
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- 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]

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

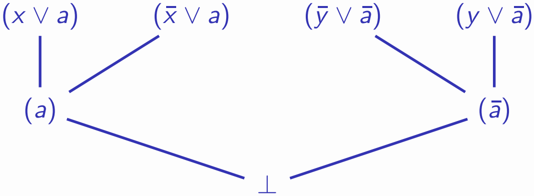
  - Complete proof system for propositional logic

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



$(x \vee a)$ $(\bar{x} \vee a)$ $(\bar{y} \vee \bar{a})$ $(y \vee \bar{a})$

(a) $(\bar{a})$

$\perp$

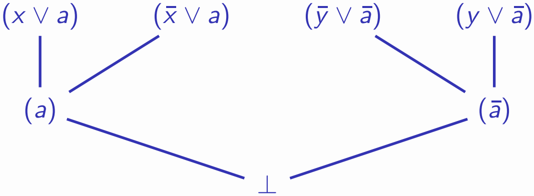
  - Extensively used with (CDCL) SAT solvers

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



$(x \vee a)$ $(\bar{x} \vee a)$ $(\bar{y} \vee \bar{a})$ $(y \vee \bar{a})$

(a) $(\bar{a})$

$\perp$

  - 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

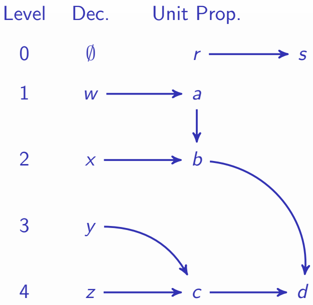
$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

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

Unit Propagation

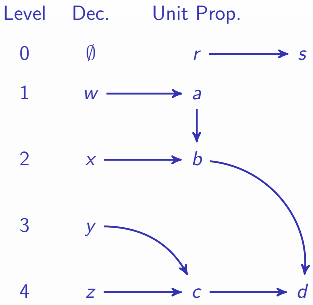
$$\begin{aligned}\mathcal{F} &= (r) \wedge (\bar{r} \vee s) \wedge \\ &\quad (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\ &\quad (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

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



Unit Propagation

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



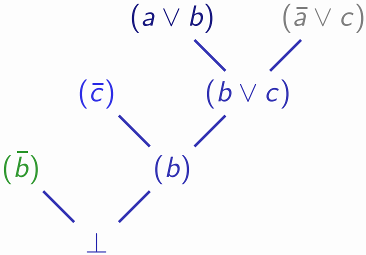
Level	Dec.	Unit Prop.
0	$\emptyset$	r s
1	w	a
2	x	b
3	y	
4	z	c d

- 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
    - $w = 1 @ 1,\ x = 1 @ 2,\ y = 1 @ 3,\ z = 1 @ 4$
    - $r = 1 @ 0,\ d = 1 @ 4,\ \ldots$

Resolution Proofs

- Refutation of unsatisfiable formula by iterated resolution operations produces resolution proof
- An example:
  $\mathcal{F} = (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$
- Resolution proof:



$(a \vee b)$ $(\bar{a} \vee c)$

$(\bar{c})$ $(b \vee c)$

$(\bar{b})$ (b)

$\perp$

- A modern SAT solver can generate resolution proofs using clauses learned by the solver

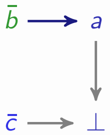
[ZM03]

Unsatisfiable Cores & Proof Traces

- CNF formula:

$$\mathcal{F} \;=\; (\bar{c}) \land (\bar{b}) \land (\bar{a} \lor c) \land (a \lor b) \land (a \lor \bar{d}) \land (\bar{a} \lor \bar{d})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	



Implication graph with conflict

Unsatisfiable Cores & Proof Traces

- CNF formula:

$$\mathcal{F} \quad = \quad (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	

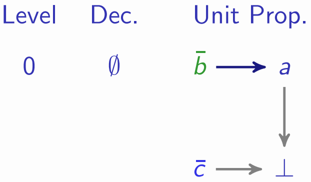
$$\begin{tikzcd}
\bar{b} \arrow[r] & a \arrow[d] \\
\bar{c} \arrow[r] & \bot
\end{tikzcd}$$

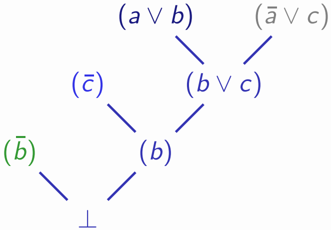
Proof trace $\bot$: $(\bar{a} \vee c)$ $(a \vee b)$ $(\bar{c})$ $(\bar{b})$

Unsatisfiable Cores & Proof Traces

- CNF formula:

$$\mathcal{F} \;=\; (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$$



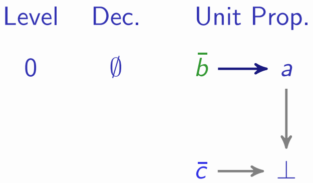


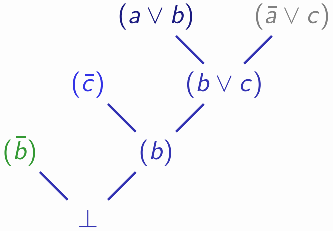
Resolution proof follows structure of conflicts

Unsatisfiable Cores & Proof Traces

- CNF formula:

$$\mathcal{F} \quad = \quad (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$$





Unsatisfiable subformula (core): $(\bar{c}), (\bar{b}), (\bar{a} \vee c), (a \vee b)$

Outline

Basic Definitions

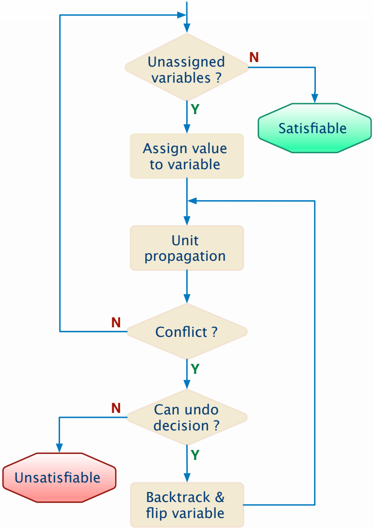
DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

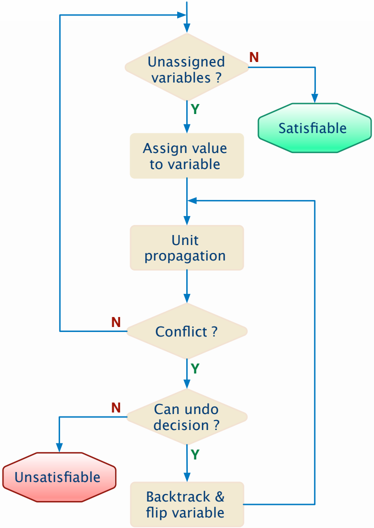
CNF Encodings

The DPLL Algorithm



- Optional: pure literal rule

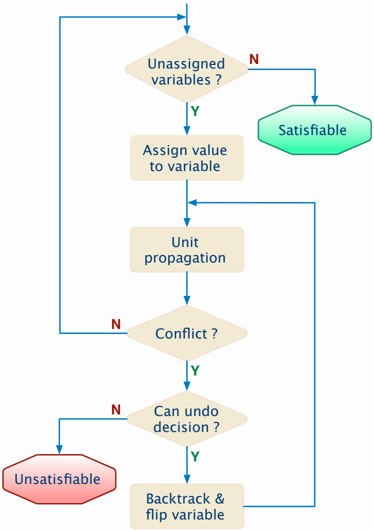
The DPLL Algorithm



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

- Optional: pure literal rule

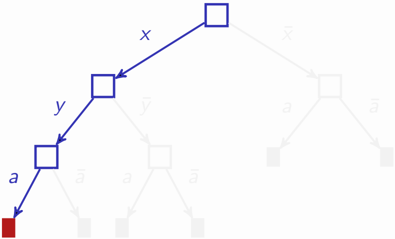
The DPLL Algorithm



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

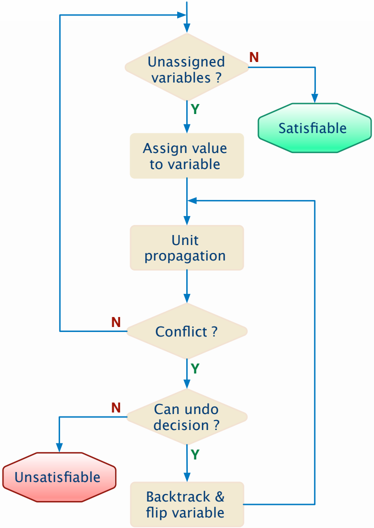
Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	

$$\begin{tikzcd} 3 & a \arrow[r] \arrow[rr, bend right] & b \arrow[r] & \perp \end{tikzcd}$$



- Optional: pure literal rule

The DPLL Algorithm



Unassigned variables ?

N

Satisfiable

Y

Assign value to variable

Unit propagation

Conflict ?

N

Y

Can undo decision ?

N

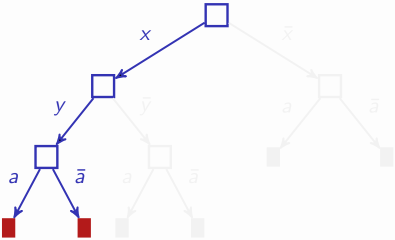
Unsatisfiable

Y

Backtrack & flip variable

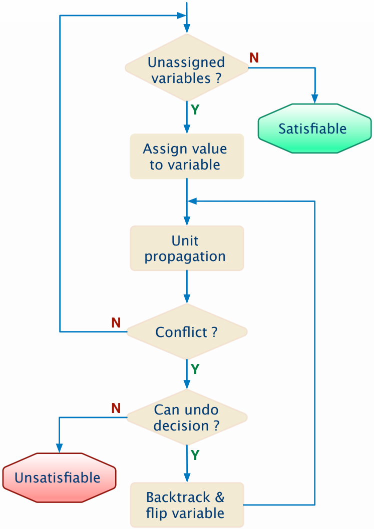
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	$\begin{tikzcd} \bar{a} \arrow[r] \arrow[rr, bend right] & \bar{b} \arrow[r] & \perp \end{tikzcd}$	



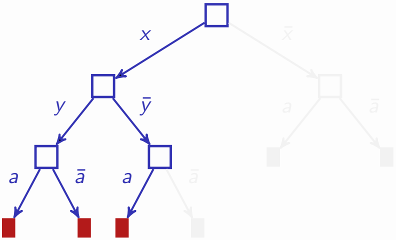
- Optional: pure literal rule

The DPLL Algorithm



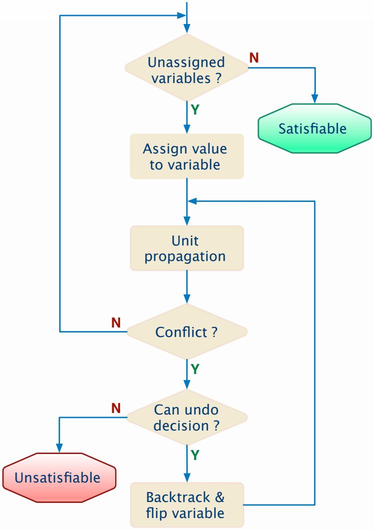
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



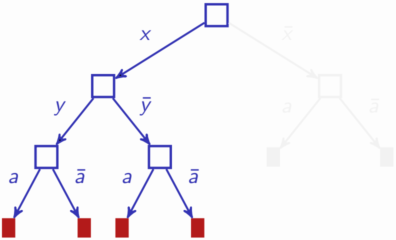
- Optional: pure literal rule

The DPLL Algorithm



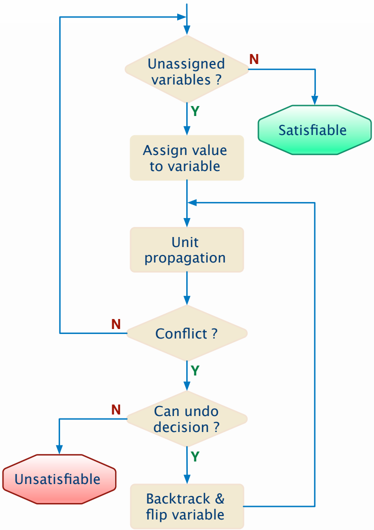
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

<table>
<tr><th>Level</th><th>Dec.</th><th>Unit Prop.</th></tr>
<tr><td>0</td><td>$\emptyset$</td><td></td></tr>
<tr><td>1</td><td>x</td><td></td></tr>
<tr><td>2</td><td>$\bar{y}$</td><td></td></tr>
<tr><td>3</td><td colspan="2">$\begin{tikzcd} \bar{a} \arrow[r] \arrow[rr, bend right] & \bar{b} \arrow[r] & \perp \end{tikzcd}$</td></tr>
</table>



- Optional: pure literal rule

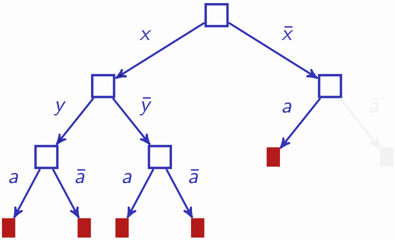
The DPLL Algorithm



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

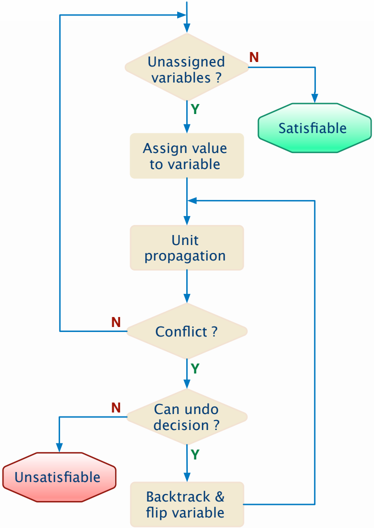
Level	Dec.	Unit Prop.
0	$\emptyset$	
1	$\bar{x}$	y
2	a	b $\perp$

$$\begin{tikzcd}
\bar{x} \arrow[r] & y & \\
a \arrow[r] \arrow[rr, bend right] & b \arrow[r] & \perp
\end{tikzcd}$$



- Optional: pure literal rule

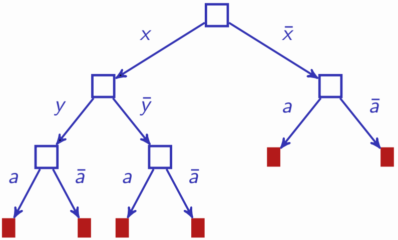
The DPLL Algorithm



- Optional: pure literal rule

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

$$\begin{tikzcd}
\text{Level} & \text{Dec.} & \text{Unit Prop.} & & \\
0 & \emptyset & & & \\
1 & \bar{x} \arrow[r] & y & & \\
2 & \bar{a} \arrow[r] \arrow[rr, bend right] & \bar{b} \arrow[r] & \perp &
\end{tikzcd}$$



Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

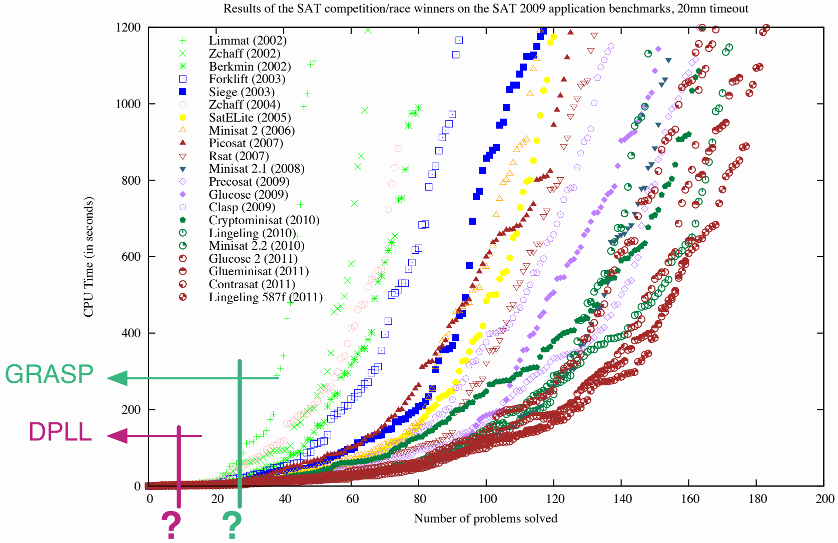
What is a CDCL SAT Solver?

- Extend DPLL SAT solver with: [DP60,DLL62]
  - Clause learning & non-chronological backtracking [MSS96,BS97,Z97]
  - Search restarts [GSK98,BMS00,H07,B08]
  - Lazy data structures
  - Conflict-guided branching
  - ...

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]
  - ...

How Significant are CDCL SAT Solvers?



Outline

Basic Definitions

DPLL Solvers

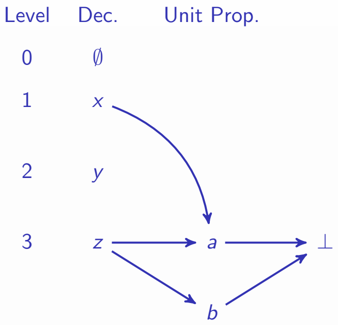
CDCL Solvers

- Clause Learning, UIPs & Minimization
- Search Restarts & Lazy Data Structures

What Next in CDCL Solvers?

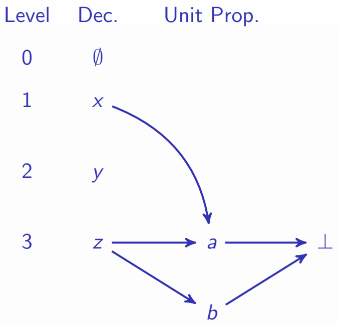
CNF Encodings

Clause Learning



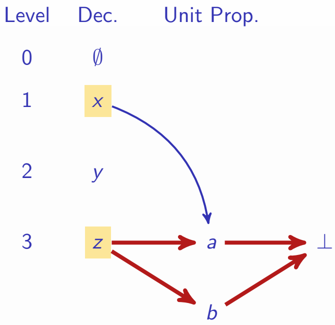
Level	Dec.	Unit Prop.	
0	$\emptyset$		
1	x		
2	y		
3	z	a	$\perp$
		b	

Clause Learning



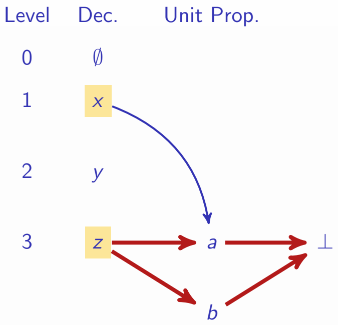
- Analyze conflict

Clause Learning



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

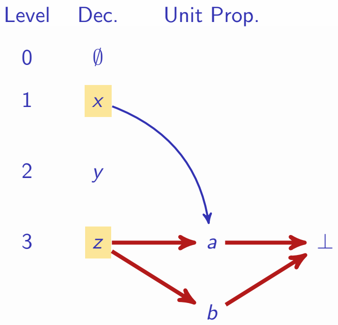
Clause Learning



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

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

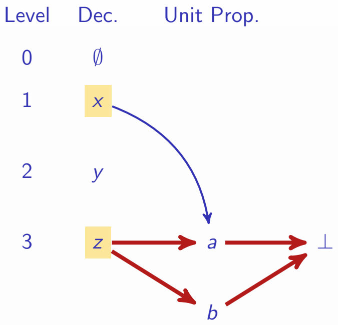
Clause Learning

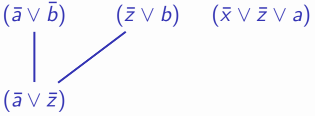


$(\bar{a} \vee \bar{b})$ $(\bar{z} \vee b)$ $(\bar{x} \vee \bar{z} \vee a)$

- 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

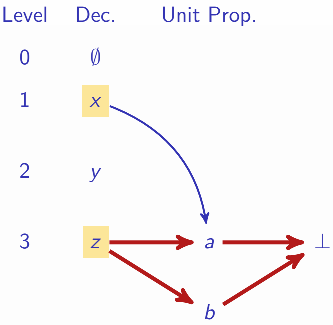
Clause Learning





- 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



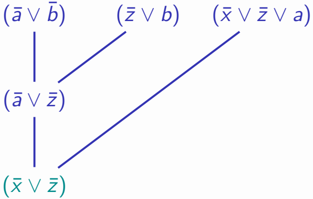
Level Dec. Unit Prop.

0 $\emptyset$

1 x

2 y

3 z a b $\perp$



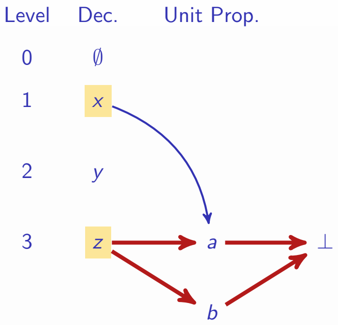
$(\bar{a} \vee \bar{b})$ $(\bar{z} \vee b)$ $(\bar{x} \vee \bar{z} \vee a)$

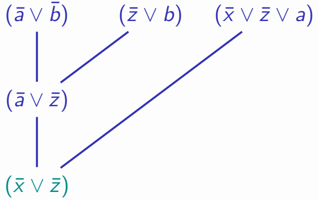
$(\bar{a} \vee \bar{z})$

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

- 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

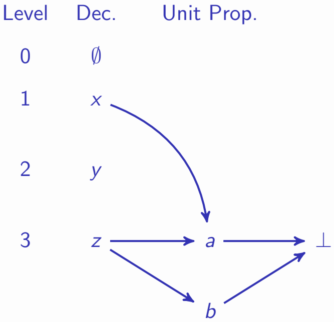
Clause Learning





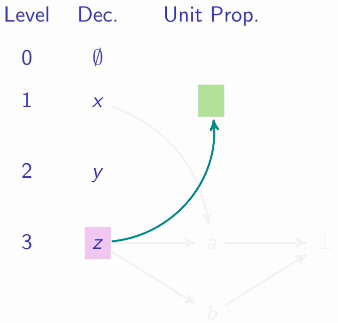
- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

Clause Learning – After Backtracking



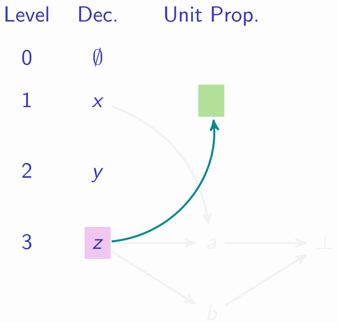
Level	Dec.	Unit Prop.	
0	$\emptyset$		
1	x		
2	y		
3	z	a	$\bot$
		b	

Clause Learning – After Backtracking



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

Clause Learning – After Backtracking



Level Dec. Unit Prop.

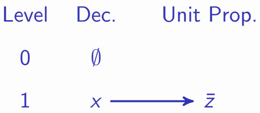
0 $\emptyset$

1 x

2 y

3 z a $\perp$

b



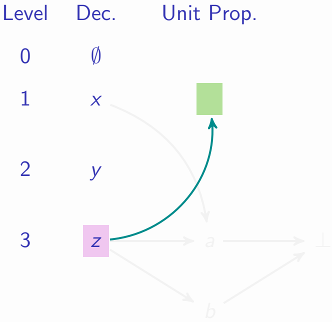
Level Dec. Unit Prop.

0 $\emptyset$

1 $x \longrightarrow \bar{z}$

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

Clause Learning – After Backtracking



Level Dec. Unit Prop.

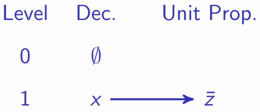
0 $\emptyset$

1 x

2 y

3 z a $\perp$

b



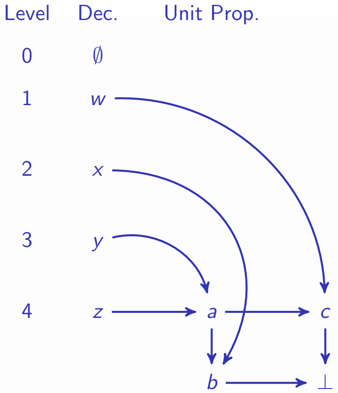
Level Dec. Unit Prop.

0 $\emptyset$

1 $x \longrightarrow \bar{z}$

- 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)



Level Dec. Unit Prop.

0 $\emptyset$

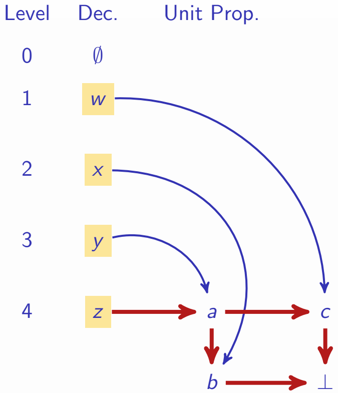
1 w

2 x

3 y

4 z

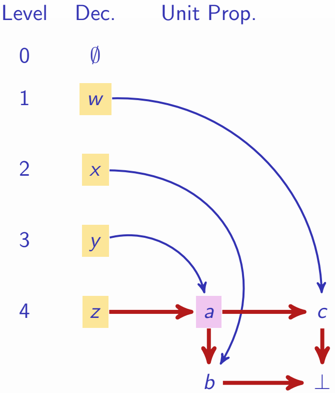
Unique Implication Points (UIPs)

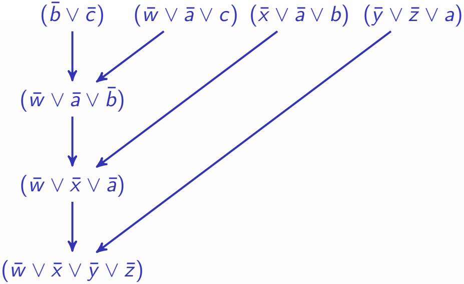


$$\begin{tikzcd}
(\bar{b} \vee \bar{c}) \arrow[d] & (\bar{w} \vee \bar{a} \vee c) \arrow[dl] & (\bar{x} \vee \bar{a} \vee b) \arrow[ddll] & (\bar{y} \vee \bar{z} \vee a) \arrow[dddlll] \\
(\bar{w} \vee \bar{a} \vee \bar{b}) \arrow[d] & & & \\
(\bar{w} \vee \bar{x} \vee \bar{a}) \arrow[d] & & & \\
(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z}) & & &
\end{tikzcd}$$

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

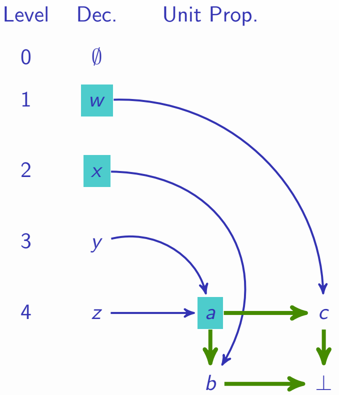
Unique Implication Points (UIPs)

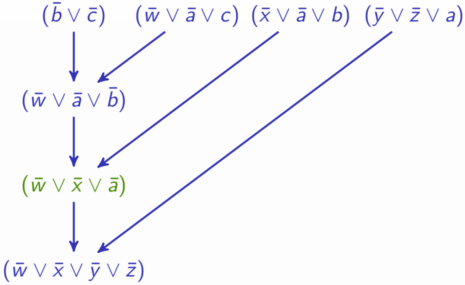



$(\bar{b} \vee \bar{c})$
$(\bar{w} \vee \bar{a} \vee c)$
$(\bar{x} \vee \bar{a} \vee b)$
$(\bar{y} \vee \bar{z} \vee a)$
$(\bar{w} \vee \bar{a} \vee \bar{b})$
$(\bar{w} \vee \bar{x} \vee \bar{a})$
$(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$
- But a is an UIP
  - Dominator in DAG for level 4

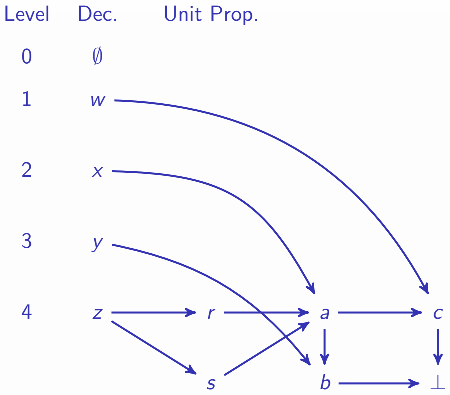
Unique Implication Points (UIPs)



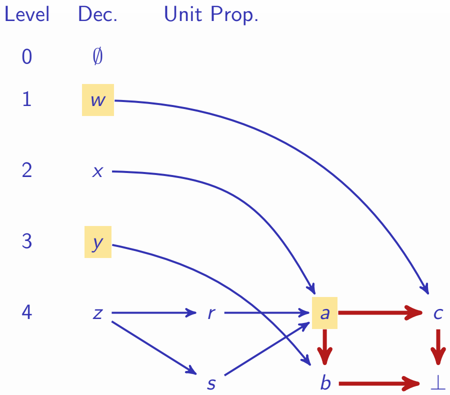

$(\bar{b} \vee \bar{c})$
$(\bar{w} \vee \bar{a} \vee c)$
$(\bar{x} \vee \bar{a} \vee b)$
$(\bar{y} \vee \bar{z} \vee a)$
$(\bar{w} \vee \bar{a} \vee \bar{b})$
$(\bar{w} \vee \bar{x} \vee \bar{a})$
$(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But a is an UIP
  - Dominator in DAG for level 4
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

Multiple UIPs

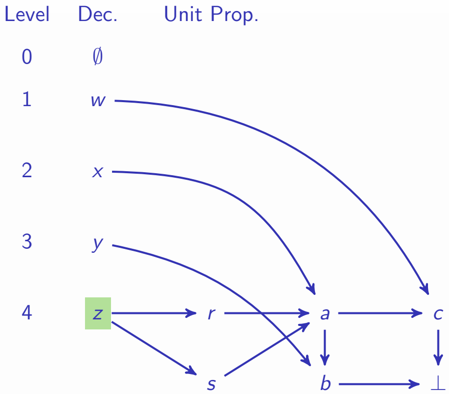


Multiple UIPs



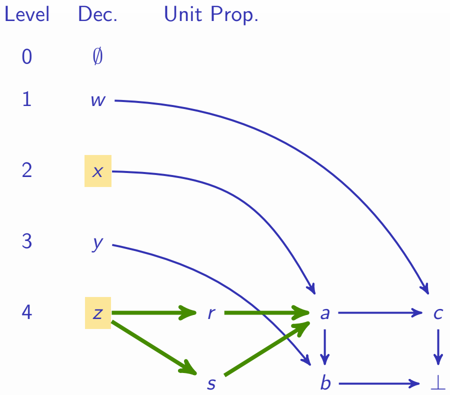
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

Multiple UIPs



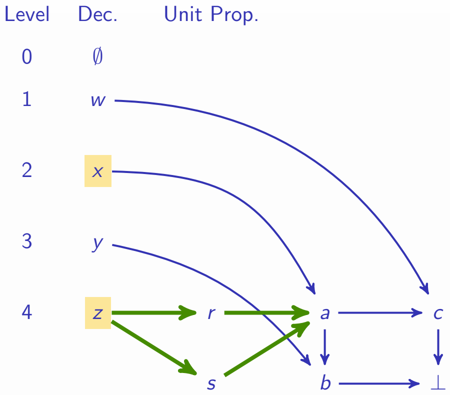
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$
- But there can be more than 1 UIP

Multiple UIPs



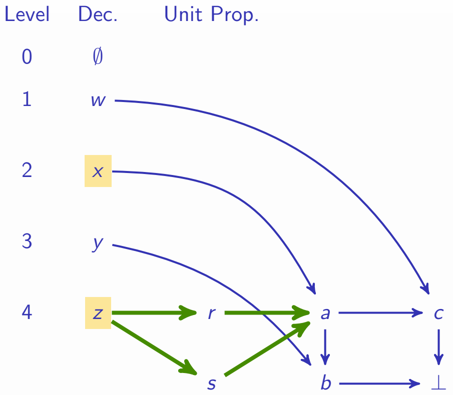
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$

Multiple UIPs



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

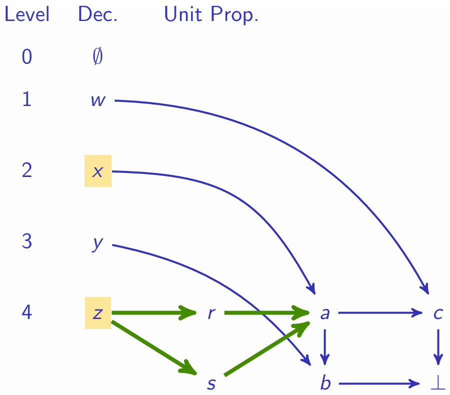
Multiple UIPs



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

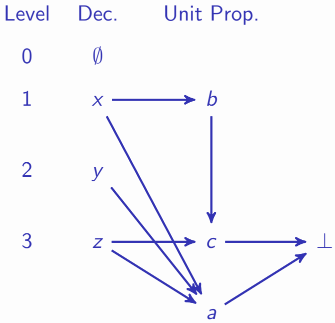
Multiple UIPs



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



Level Dec. Unit Prop.

0 $\emptyset$

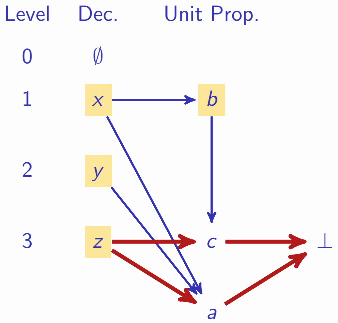
1 x b

2 y

3 z c $\perp$

a

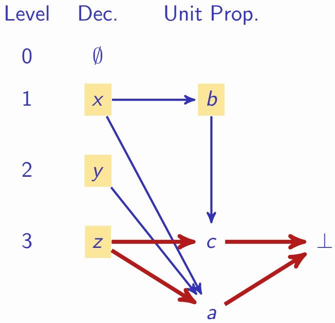
Clause Minimization I

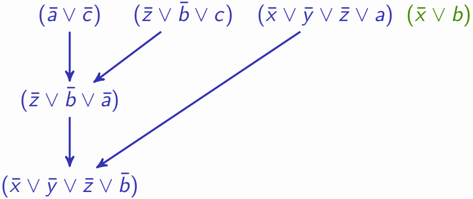


$$\begin{tikzcd}
(\bar{a} \vee \bar{c}) \arrow[d] & (\bar{z} \vee \bar{b} \vee c) \arrow[dl] & (\bar{x} \vee \bar{y} \vee \bar{z} \vee a) \arrow[ddll] \\
(\bar{z} \vee \bar{b} \vee \bar{a}) \arrow[d] & & \\
(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b}) & &
\end{tikzcd}$$

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

Clause Minimization I

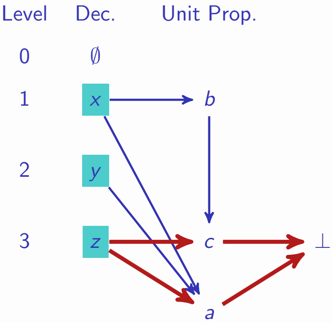


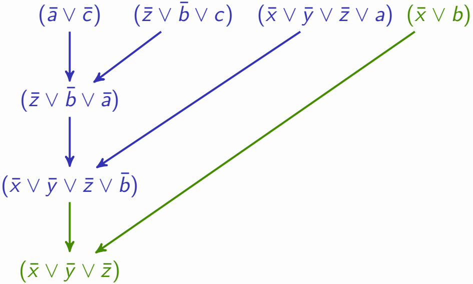


$(\bar{a} \vee \bar{c})$
$(\bar{z} \vee \bar{b} \vee c)$
$(\bar{x} \vee \bar{y} \vee \bar{z} \vee a)$
$(\bar{x} \vee b)$
$(\bar{z} \vee \bar{b} \vee \bar{a})$
$(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$

- 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

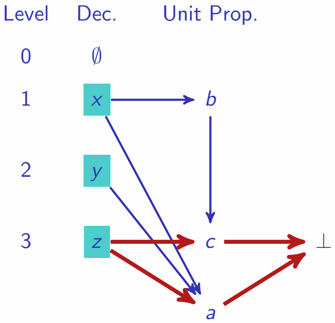


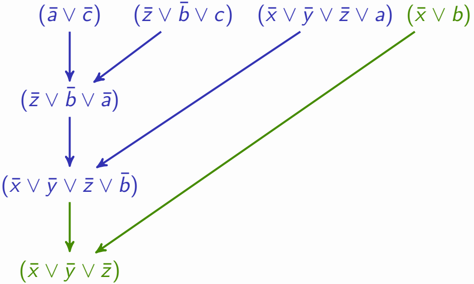


$(\bar a \vee \bar c)$
$(\bar z \vee \bar b \vee c)$
$(\bar x \vee \bar y \vee \bar z \vee a)$
$(\bar x \vee b)$
$(\bar z \vee \bar b \vee \bar a)$
$(\bar x \vee \bar y \vee \bar z \vee \bar b)$
$(\bar x \vee \bar y \vee \bar z)$

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

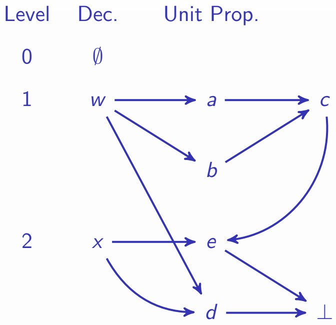
Clause Minimization I




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

- ~~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



Level Dec. Unit Prop.

0 $\emptyset$

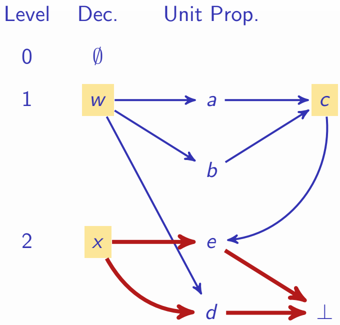
1 w a c

b

2 x e

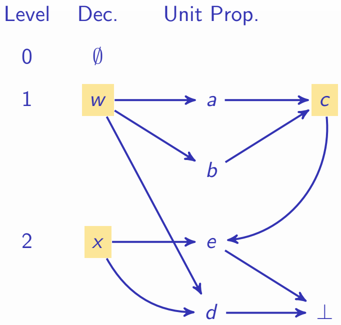
d $\perp$

Clause Minimization II



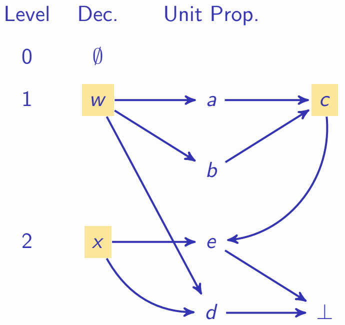
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$

Clause Minimization II



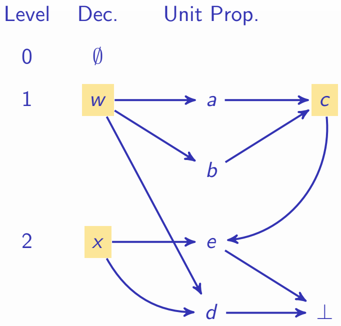
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$

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

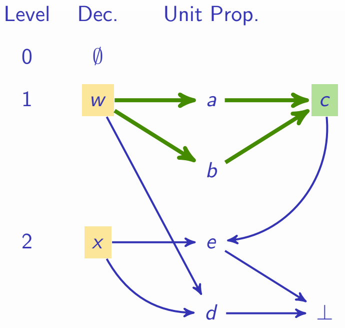


- ~~Learn clause ($\bar{w} \vee \bar{x} \vee \bar{c}$)~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields ($\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b}$)
- Can apply recursive minimization

- Marked nodes: literals in learned clause

[SB09]

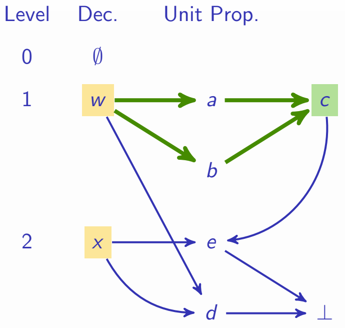
Clause Minimization II



- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- 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 decision nodes
  - Drop literal c if only marked nodes visited

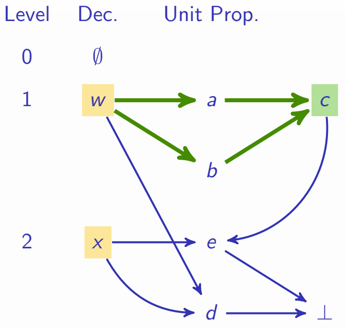
Clause Minimization II



- ~~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 decision nodes
  - Drop literal c 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 decision nodes
  - Drop literal c if only marked nodes visited
- Complexity of recursive minimization?

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

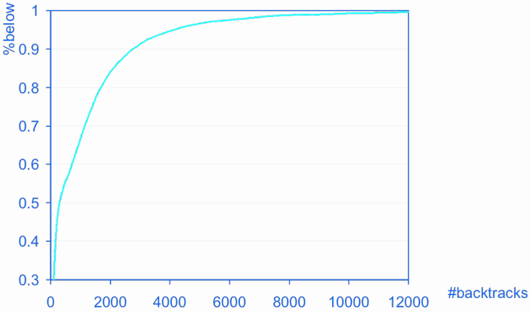
- Clause Learning, UIPs & Minimization
- Search Restarts & Lazy Data Structures

What Next in CDCL Solvers?

CNF Encodings

Search Restarts I

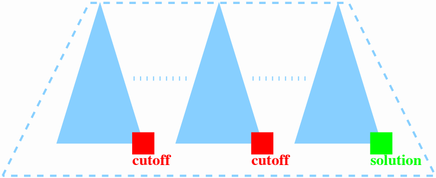
- Heavy-tail behavior: [GSK98]



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

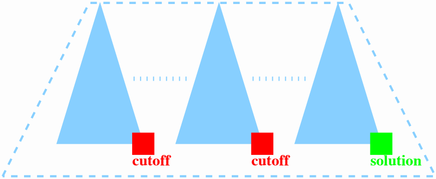
Search Restarts II

- Restart search after a number of conflicts



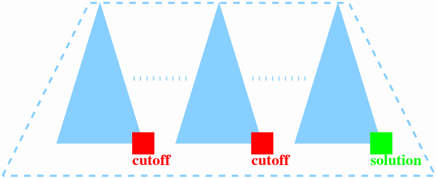
Search Restarts II

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



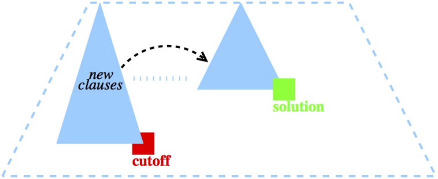
Search Restarts II

- 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,

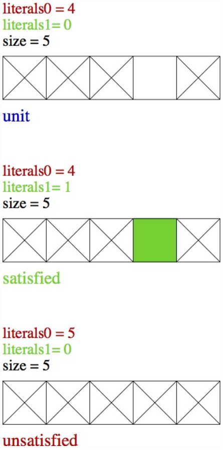
    Unit propagation slow-down worse than linear as clauses are learned **!**

- 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



literals0 = 4
literals1= 0
size = 5

unit

literals0 = 4
literals1= 1
size = 5

satisfied

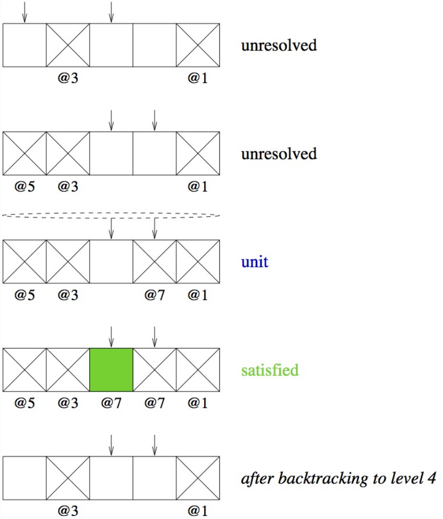
literals0 = 5
literals1= 0
size = 5

unsatisfied

Watched Literals

[MMZZM01]

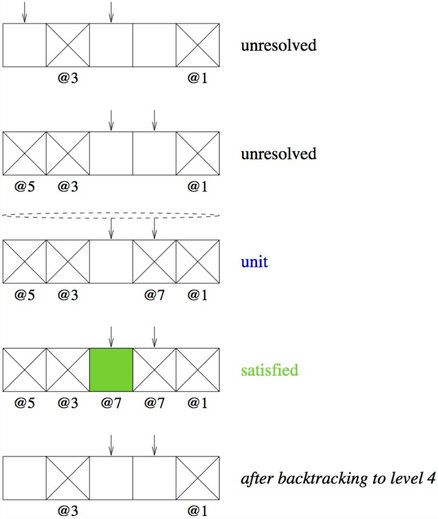
- Important states of a clause
- Associate 2 references with each clause



Watched Literals

[MMZZM01]

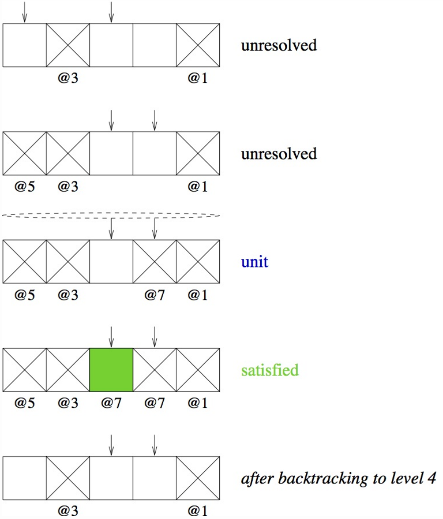
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals



Watched Literals

[MMZZM01]

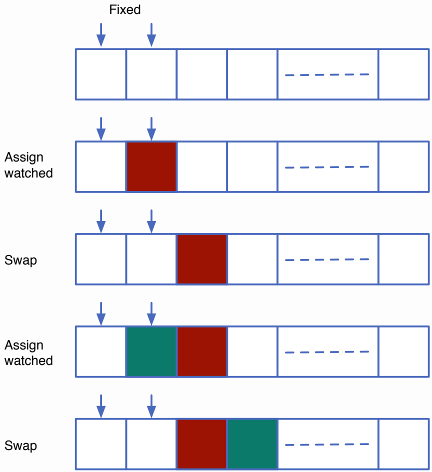
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals
- References **unchanged** when backtracking



Watched Literals, in Practice

[ES03,G13]

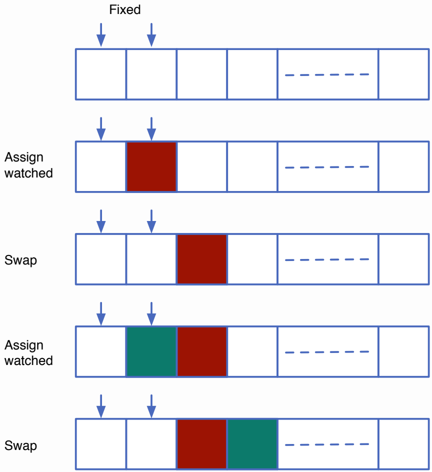
- In practice, first two positions of clause are watched



Watched Literals, in Practice

[ES03,G13]

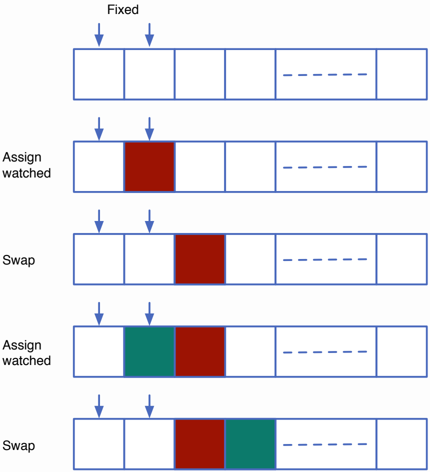
- In practice, first two positions of clause are watched
- May require to traverse already assigned literals, multiple times



Watched Literals, in Practice

[ES03,G13]

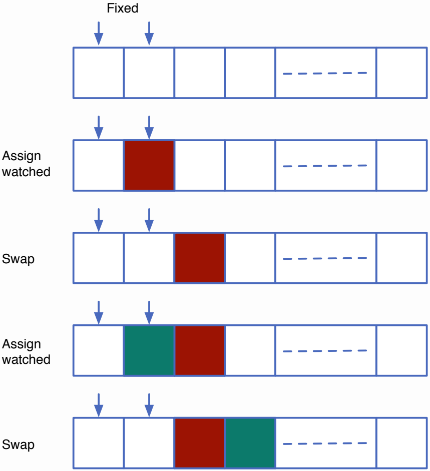
- In practice, first two positions of clause are watched
- May require to traverse already assigned literals, multiple times
- Worst-case time of unit propagation is **quadratic** on clause size and so on number of literals



Watched Literals, in Practice

[ES03,G13]

- In practice, first two positions of clause are watched
- May require to traverse already assigned literals, multiple times
- Worst-case time of unit propagation is **quadratic** on clause size and so on number of literals
- In practice, no gains observed from considering alternative implementations (see previous slide)



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 larger clauses [e.g. MSS96]
  - Delete less used clauses [e.g. 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 larger clauses [e.g. MSS96]
  - Delete less used clauses [e.g. GN02,ES03]
- Proven recent techniques:
  - Phase saving [PD07]
  - Luby restarts [H07]
  - Literal blocks distance [AS09]

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

CDCL – A Glimpse of the Future

- Clause learning techniques [e.g. ABHJS08]
  - Clause learning is the key technique in CDCL SAT solvers
  - Many recent papers propose improvements to the basic clause learning approach

CDCL – A Glimpse of the Future

- Clause learning techniques [e.g. ABHJS08]
  - 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]
  - Essential in some applications

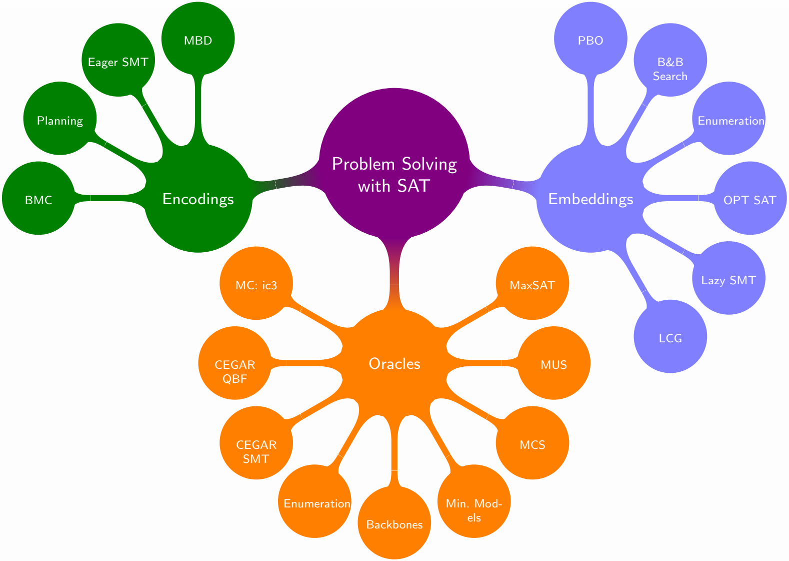
CDCL – A Glimpse of the Future

- Clause learning techniques [e.g. ABHJS08]
  - 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]
  - Essential in some applications

- Application-driven improvements
  - Incremental SAT
    - Handling of assumptions due to MUS extractors [LB13]
    - Changing SAT solvers to better suit applications [AS13]

But Also, SAT-Based Problem Solving



Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

Encoding to CNF

- What to encode?
  - Boolean formulas
    - Tseitin's encoding
    - Plaisted&Greenbaum's encoding
    - ...
  - Cardinality constraints
  - Pseudo-Boolean (PB) constraints
  - Can also translate to SAT:
    - Constraint Satisfaction Problems (CSPs)
    - Answer Set Programming (ASP)
    - Model Finding
    - ...

- Key issues:
  - Encoding size
  - Arc-consistency?

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

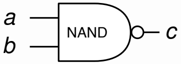
CNF Encodings

- Boolean Formulas
- Cardinality Constraints
- Pseudo-Boolean Constraints
- Encoding CSPs

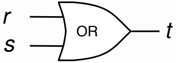
Representing Boolean Formulas / Circuits I

- Satisfiability problems can be defined on Boolean circuits/formulas
- Can represent circuits/formulas as CNF formulas [T68,PG86]
  - For each (simple) gate, CNF formula encodes the consistent assignments to the gate's inputs and output
    - Given $z = \mathrm{OP}(x, y)$, represent in CNF $z \leftrightarrow \mathrm{OP}(x, y)$
  - CNF formula for the circuit is the conjunction of CNF formula for each gate

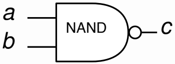
$$\mathcal{F}_c = (a \vee c) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$$

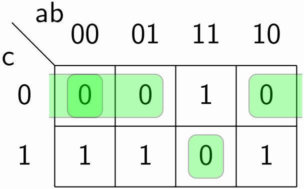


$$\mathcal{F}_t = (\bar{r} \vee t) \wedge (\bar{s} \vee t) \wedge (r \vee s \vee \bar{t})$$



Representing Boolean Formulas / Circuits II





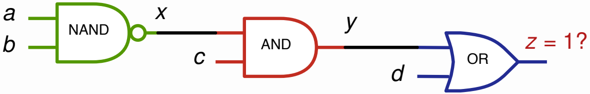
c \ ab	00	01	11	10
0	0	0	1	0
1	1	1	0	1

a	b	c	$\mathcal{F}_c$(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$\mathcal{F}_c = (a \vee c) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$$

Representing Boolean Formulas / Circuits III

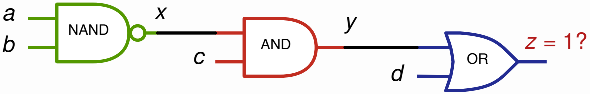
- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses



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

Representing Boolean Formulas / Circuits III

- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses



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

- Note: $z = d \vee (c \wedge (\neg(a \wedge b)))$
  - **No** distinction between Boolean circuits and formulas

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

- Boolean Formulas
- Cardinality Constraints
- Pseudo-Boolean Constraints
- Encoding CSPs

Cardinality Constraints

- How to handle cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$?
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

- Solution #1:
  - Use native PB solver, e.g. BSOLO, PBS, Galena, Pueblo, etc.
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best solvers already encode to CNF
    - E.g. Minisat+, WBO, etc.

Cardinality Constraints

- How to handle cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$?
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

- Solution #1:
  - Use native PB solver, e.g. BSOLO, PBS, Galena, Pueblo, etc.
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best solvers already encode to CNF
    - E.g. Minisat+, WBO, etc.

- Solution #2:
  - Encode cardinality constraints to CNF
  - Use SAT solver

Equals1, AtLeast1 & AtMost1 Constraints

- $\sum_{j=1}^{n} x_j = 1$: encode with $(\sum_{j=1}^{n} x_j \leq 1) \wedge (\sum_{j=1}^{n} x_j \geq 1)$
- $\sum_{j=1}^{n} x_j \geq 1$: encode with $(x_1 \vee x_2 \vee \ldots \vee x_n)$
- $\sum_{j=1}^{n} x_j \leq 1$ encode with:
  - Pairwise encoding
    - Clauses: $\mathcal{O}(n^2)$; No auxiliary variables
  - Sequential counter [S05]
    - Clauses: $\mathcal{O}(n)$; Auxiliary variables: $\mathcal{O}(n)$
  - Bitwise encoding [P07,FP01]
    - Clauses: $\mathcal{O}(n \log n)$; Auxiliary variables: $\mathcal{O}(\log n)$
  - ...

Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:

- An example: $x_1 + x_2 + x_3 \leq 1$

Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$

    $x_j \rightarrow (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}) \Leftrightarrow (\bar{x}_j \vee (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}))$

- An example: $x_1 + x_2 + x_3 \leq 1$

	$j-1$	$v_1 v_0$
x_1	0	00
x_2	1	01
x_3	2	10

Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j-1$

    $x_j \rightarrow (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}) \Leftrightarrow (\bar{x}_j \vee (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}))$
  - Clauses $(\bar{x}_j \vee (v_i \leftrightarrow b_i)) = (\bar{x}_j \vee l_i)$, $i = 0, \ldots, r-1$, where
    - $l_i \equiv v_i$, if $b_i = 1$
    - $l_i \equiv \bar{v}_i$, otherwise

- An example: $x_1 + x_2 + x_3 \leq 1$

	$j-1$	$v_1 v_0$
x_1	0	00
x_2	1	01
x_3	2	10

$$(\bar{x}_1 \vee \bar{v}_1) \wedge (\bar{x}_1 \vee \bar{v}_0)$$
$$(\bar{x}_2 \vee \bar{v}_1) \wedge (\bar{x}_2 \vee v_0)$$
$$(\bar{x}_3 \vee v_1) \wedge (\bar{x}_3 \vee \bar{v}_0)$$

Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$

    $x_j \rightarrow (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}) \Leftrightarrow (\bar{x}_j \vee (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}))$
  - Clauses $(\bar{x}_j \vee (v_i \leftrightarrow b_i)) = (\bar{x}_j \vee l_i)$, $i = 0, \ldots, r-1$, where
    - $l_i \equiv v_i$, if $b_i = 1$
    - $l_i \equiv \bar{v}_i$, otherwise
  - If $x_j = 1$, assignment to v_i variables must encode $j - 1$
    - All other x variables must take value 0
  - If all $x_j = 0$, any assignment to v_i variables is consistent
  - $\mathcal{O}(n \log n)$ clauses ; $\mathcal{O}(\log n)$ auxiliary variables

- An example: $x_1 + x_2 + x_3 \leq 1$

	$j-1$	$v_1 v_0$
x_1	0	00
x_2	1	01
x_3	2	10

$$(\bar{x}_1 \vee \bar{v}_1) \wedge (\bar{x}_1 \vee \bar{v}_0)$$
$$(\bar{x}_2 \vee \bar{v}_1) \wedge (\bar{x}_2 \vee v_0)$$
$$(\bar{x}_3 \vee v_1) \wedge (\bar{x}_3 \vee \bar{v}_0)$$

General Cardinality Constraints

- General form: $\sum_{j=1}^{n} x_j \leq k$ (or $\sum_{j=1}^{n} x_j \geq k$)
  - Sequential counters [S05]
    - Clauses/Variables: $\mathcal{O}(n\,k)$
  - BDDs [ES06]
    - Clauses/Variables: $\mathcal{O}(n\,k)$
  - Sorting networks [ES06]
    - Clauses/Variables: $\mathcal{O}(n \log^2 n)$
  - Cardinality Networks: [ANORC09,ANORC11a]
    - Clauses/Variables: $\mathcal{O}(n \log^2 k)$
  - Pairwise Cardinality Networks: [CZI10]
  - ...

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

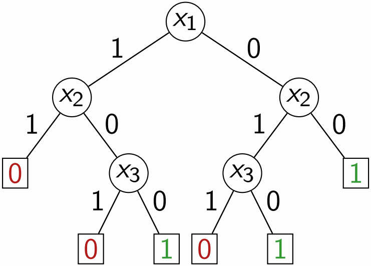
- Boolean Formulas
- Cardinality Constraints
- Pseudo-Boolean Constraints
- Encoding CSPs

Pseudo-Boolean Constraints

- General form: $\sum_{j=1}^{n} a_j \, x_j \leq b$
  - Operational encoding [W98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not guarantee arc-consistency
  - BDDs [ES06]
    - Worst-case exponential number of clauses
  - Polynomial watchdog encoding [BBR09]
    - Let $\nu(n) = \log(n) \log(a_{max})$
    - Clauses: $\mathcal{O}(n^3 \nu(n))$; Aux variables: $\mathcal{O}(n^2 \nu(n))$
  - Improved polynomial watchdog encoding [ANORC11b]
    - Clauses & aux variables: $\mathcal{O}(n^3 \log(a_{max}))$
  - ...

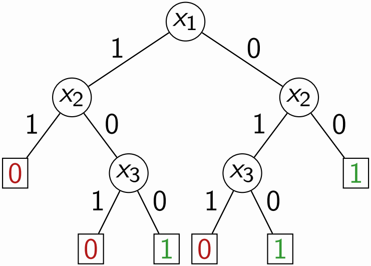
Encoding PB Constraints with BDDs I

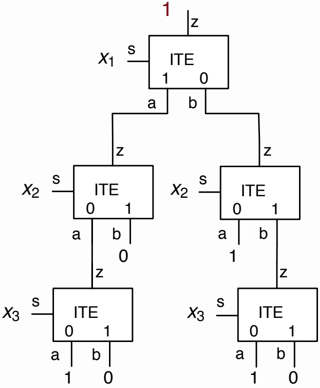
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD



Encoding PB Constraints with BDDs I

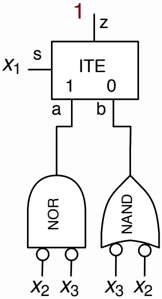
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD





Encoding PB Constraints with BDDs II

- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Extract ITE-based circuit from BDD
- Simplify and create final circuit:



More on PB Constraints

- How about $\sum_{j=1}^{n} a_j \, x_j = k$?

More on PB Constraints

- How about $\sum_{j=1}^{n} a_j\, x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j\, x_j \geq k) \wedge (\sum_{j=1}^{n} a_j\, x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j\, x_j = k$ is a subset-sum constraint (special case of a knapsack constraint)

More on PB Constraints

- How about $\sum_{j=1}^{n} a_j\, x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j\, x_j \geq k) \wedge (\sum_{j=1}^{n} a_j\, x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j\, x_j = k$ is a subset-sum constraint (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

More on PB Constraints

- How about $\sum_{j=1}^{n} a_j\, x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j\, x_j \geq k) \wedge (\sum_{j=1}^{n} a_j\, x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j\, x_j = k$ is a subset-sum constraint
      (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- Example:

$$4x_1 + 4x_2 + 3x_3 + 2x_4 = 5$$

More on PB Constraints

- How about $\sum_{j=1}^{n} a_j\, x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j\, x_j \geq k) \land (\sum_{j=1}^{n} a_j\, x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j\, x_j = k$ is a subset-sum constraint

      (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- Example:

$$4x_1 + 4x_2 + 3x_3 + 2x_4 = 5$$

  - Replace by $(4x_1 + 4x_2 + 3x_3 + 2x_4 \geq 5) \land (4x_1 + 4x_2 + 3x_3 + 2x_4 \leq 5)$

More on PB Constraints

- How about $\sum_{j=1}^{n} a_j\, x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j\, x_j \geq k) \wedge (\sum_{j=1}^{n} a_j\, x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j\, x_j = k$ is a subset-sum constraint

      (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- Example:

$$4x_1 + 4x_2 + 3x_3 + 2x_4 = 5$$

  - Replace by $(4x_1 + 4x_2 + 3x_3 + 2x_4 \geq 5) \wedge (4x_1 + 4x_2 + 3x_3 + 2x_4 \leq 5)$
  - Let $x_3 = 0$

More on PB Constraints

- How about $\sum_{j=1}^{n} a_j\, x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j\, x_j \geq k) \wedge (\sum_{j=1}^{n} a_j\, x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j\, x_j = k$ is a subset-sum constraint

      (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- Example:

$$4x_1 + 4x_2 + 3x_3 + 2x_4 = 5$$

  - Replace by $(4x_1 + 4x_2 + 3x_3 + 2x_4 \geq 5) \wedge (4x_1 + 4x_2 + 3x_3 + 2x_4 \leq 5)$
  - Let $x_3 = 0$
  - Either constraint can still be satisfied, but **not** both

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

CNF Encodings

- Boolean Formulas
- Cardinality Constraints
- Pseudo-Boolean Constraints
- Encoding CSPs

CSP Constraints

- Many possible encodings:
  - Direct encoding [dK89,GJ96,W00]
  - Log encoding [W00]
  - Support encoding [K90,G02]
  - Log-Support encoding [G07]
  - Order encoding for finite linear CSPs [TTKB09]
  - ...

Direct Encoding for CSP w/ Binary Constraints

- Variable x_i with domain D_i, with $m_i = |D_i|$
- Represent values of x_i with Boolean variables $x_{i,1}, \ldots, x_{i,m_i}$
- Require $\sum_{k=1}^{m_i} x_{i,k} = 1$
  - Suffices to require $\sum_{k=1}^{m_i} x_{i,k} \geq 1$ [W00]
- If the pair of assignments $x_i = v_i \wedge x_j = v_j$ is not allowed, add binary clause $(\bar{x}_{i,v_i} \vee \bar{x}_{j,v_j})$

Thanks!