SAT-Based Problem Solving

Joao Marques-Silva[1,2]

[1]University College Dublin, Ireland

[2]IST/INESC-ID, Lisbon, Portugal

University of Calabria, Italy

February 2015

The success of SAT

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

The success of SAT

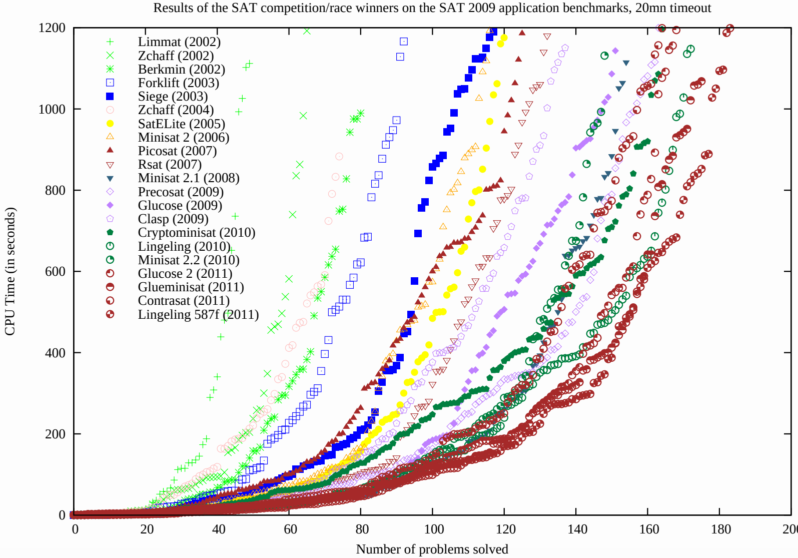
- Well-known NP-complete decision problem [C71]
- In practice, SAT is a success story of Computer Science
  - Hundreds (even more?) of practical applications

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]



These lectures

- Lecture #1: Modern SAT solvers & problem solving with SAT
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Note:** Overview for non-experts

These lectures

- Lecture #1: Modern SAT solvers & problem solving with SAT
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Note:** Overview for non-experts

- Lecture #2: Solving minimal set problems with SAT oracles
  - What are minimal sets?
    - Minimal unsatisfiability (MUS)
    - Maximal satisfiability (MSS/MCS)
    - Prime implicants/implicates
    - Minimal models / backbones / autarkies / ...

These lectures

- Lecture #1: Modern SAT solvers & problem solving with SAT
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Note:** Overview for non-experts
- Lecture #2: Solving minimal set problems with SAT oracles
  - What are minimal sets?
    - Minimal unsatisfiability (MUS)
    - Maximal satisfiability (MSS/MCS)
    - Prime implicants/implicates
    - Minimal models / backbones / autarkies / ...
- Lecture #3: Solving optimization problems with SAT oracles
  - Which optimization problems?
    - Maximum satisfiability (MaxSAT)
    - Minimal satisfiability (MinSAT)
    - Pseudo-Boolean optimization / Weighted Boolean optimization / ...

SAT-Based Problem Solving

Lecture #1:
SAT Solvers & Problem Solving with SAT

Joao Marques-Silva[1,2]

[1]University College Dublin, Ireland

[2]IST/INESC-ID, Lisbon, Portugal

University of Calabria, Italy

February 2015

Part I

CDCL SAT Solvers

Outline

Outline

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]

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

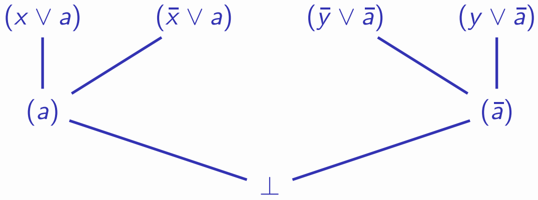
  – Complete proof system for propositional logic

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



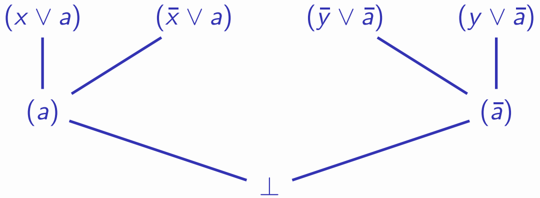
  - Extensively used with (CDCL) SAT solvers

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

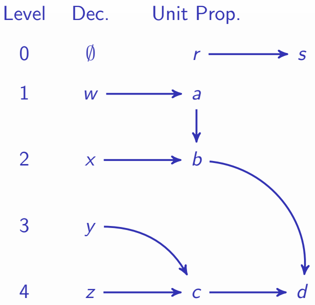
$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

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

Unit Propagation

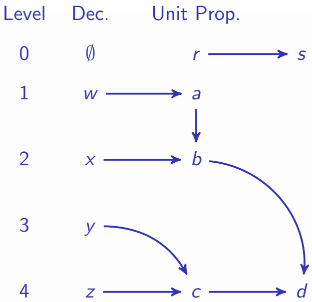
$$
\begin{aligned}
\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\
& (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\
& (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)
\end{aligned}
$$

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



Unit Propagation

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

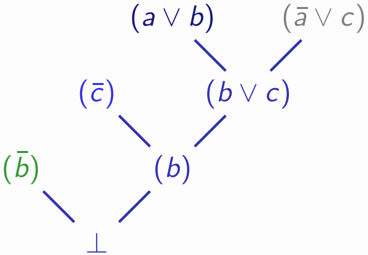


- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

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

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

[ZM03]

Unsatisfiable Cores & Proof Traces

- CNF formula:

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

Level	Dec.	Unit Prop.
0	$\emptyset$	$\bar{b} \longrightarrow a$

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

Implication graph with conflict

Unsatisfiable Cores & Proof Traces

- CNF formula:

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

Level	Dec.	Unit Prop.
0	$\emptyset$	$\begin{tikzcd} \bar{b} \arrow[r] & a \arrow[d] \\ \bar{c} \arrow[r] & \bot \end{tikzcd}$

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

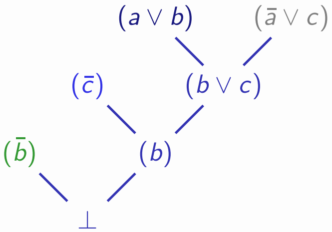
Unsatisfiable Cores & Proof Traces

- CNF formula:

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

Level	Dec.	Unit Prop.
0	$\emptyset$	$\bar{b} \to a$

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



Resolution proof follows structure of conflicts

Unsatisfiable Cores & Proof Traces

- CNF formula:

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

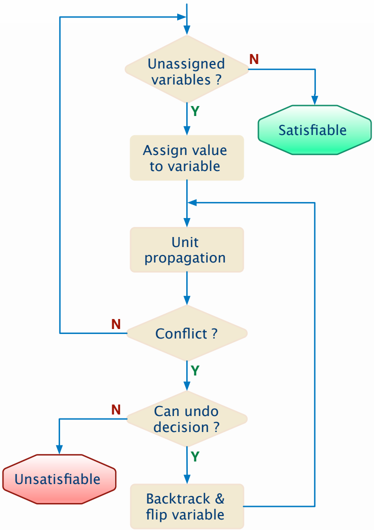
Level	Dec.	Unit Prop.
0	$\emptyset$	$\begin{tikzcd} \bar{b} \arrow[r] & a \arrow[d] \\ \bar{c} \arrow[r] & \perp \end{tikzcd}$

$$\begin{tikzcd}[column sep=small] & & & (a \vee b) \arrow[dr, dash] & & (\bar{a} \vee c) \arrow[dl, dash] \\ & & (\bar{c}) \arrow[dr, dash] & & (b \vee c) \arrow[dl, dash] & \\ & (\bar{b}) \arrow[dr, dash] & & (b) \arrow[dl, dash] & & \\ & & \perp & & & \end{tikzcd}$$

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

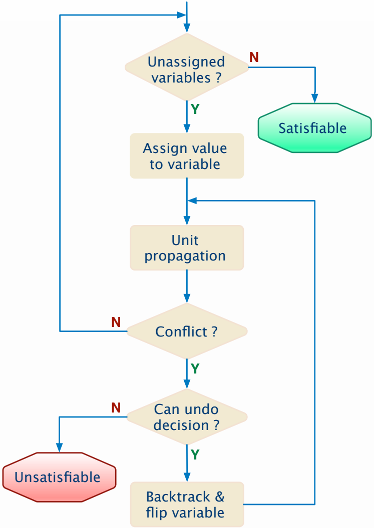
Outline

The DPLL Algorithm



- Optional: pure literal rule

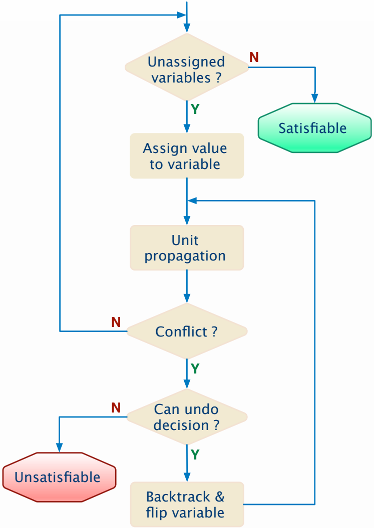
The DPLL Algorithm



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

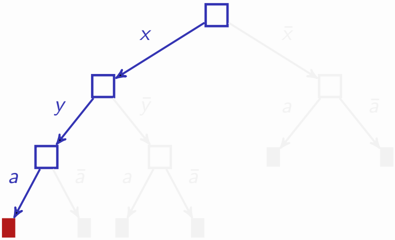
- Optional: pure literal rule

The DPLL Algorithm



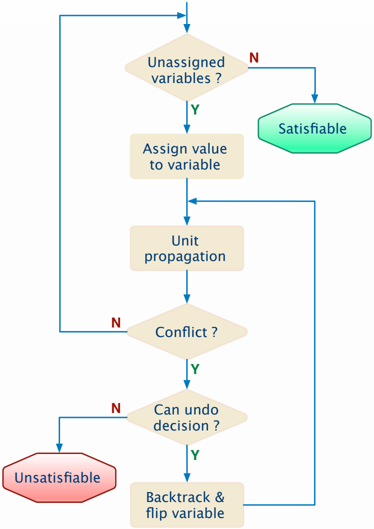
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



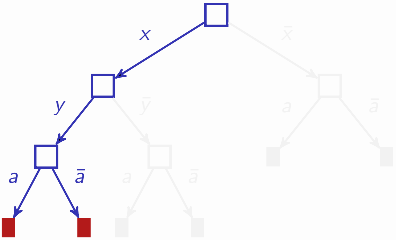
- Optional: pure literal rule

The DPLL Algorithm



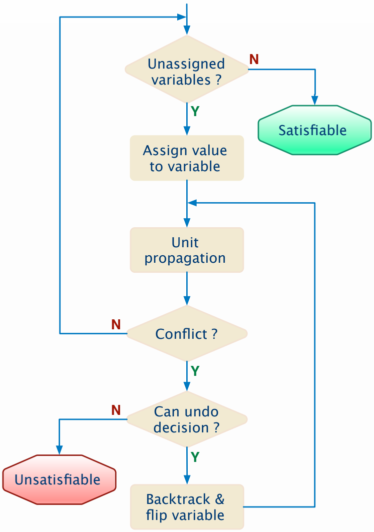
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



- Optional: pure literal rule

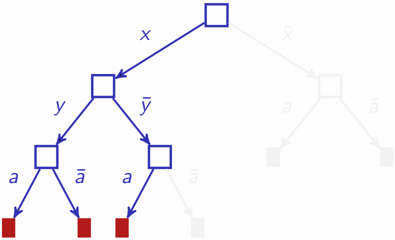
The DPLL Algorithm



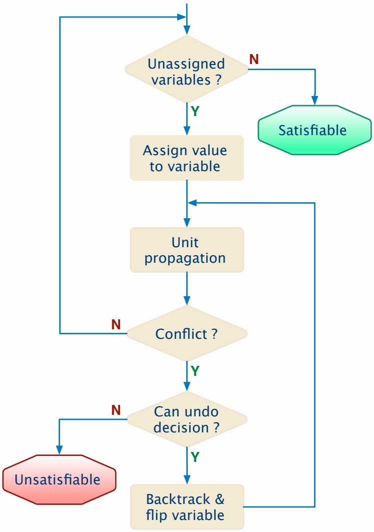
- Optional: pure literal rule

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

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

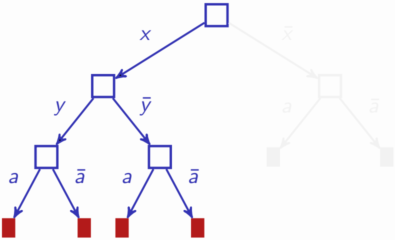


The DPLL Algorithm



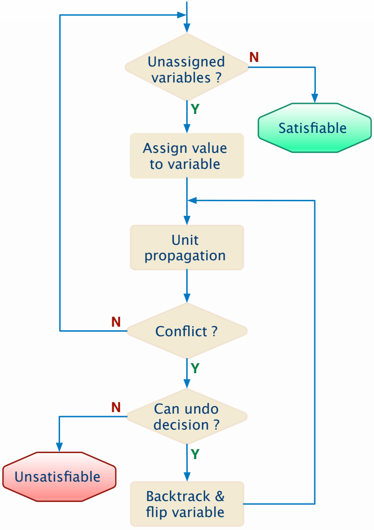
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



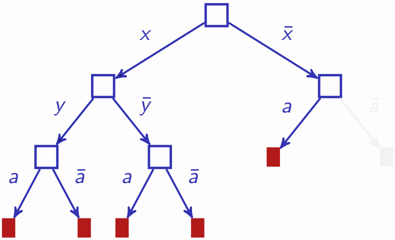
- Optional: pure literal rule

The DPLL Algorithm



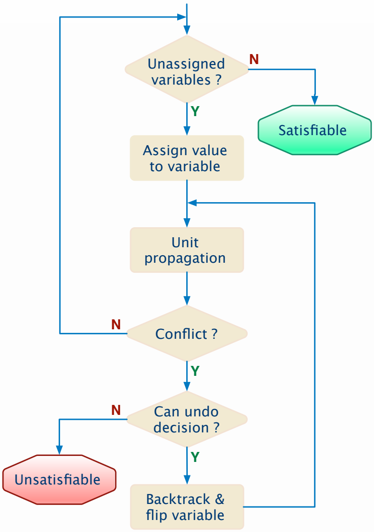
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



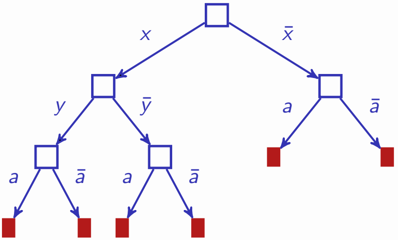
- Optional: pure literal rule

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	$\bar{a}$	$\longrightarrow \bar{b} \longrightarrow \perp$



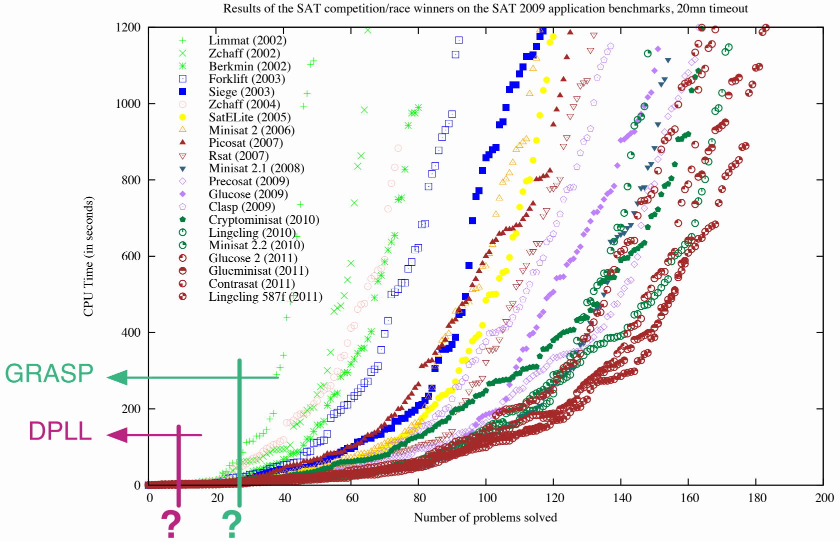
- Optional: pure literal rule

Outline

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

Clause Learning

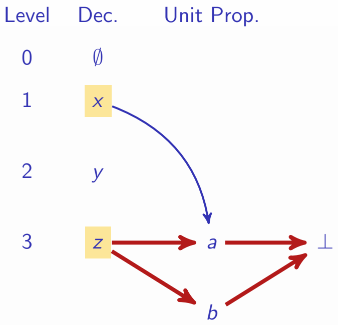
$$
\begin{tikzcd}[row sep=small]
\text{Level} & \text{Dec.} & \text{Unit Prop.} & \\
0 & \emptyset & & \\
1 & x \arrow[rdd, bend left] & & \\
2 & y & & \\
3 & z \arrow[r] \arrow[rd] & a \arrow[r] & \perp \\
 & & b \arrow[ru] &
\end{tikzcd}
$$

Clause Learning

$$
\begin{tikzcd}
\text{Level} & \text{Dec.} & \text{Unit Prop.} & \\
0 & \emptyset & & \\
1 & x \arrow[ddr, bend left] & & \\
2 & y & & \\
3 & z \arrow[r] \arrow[dr] & a \arrow[r] & \perp \\
 & & b \arrow[ur] &
\end{tikzcd}
$$

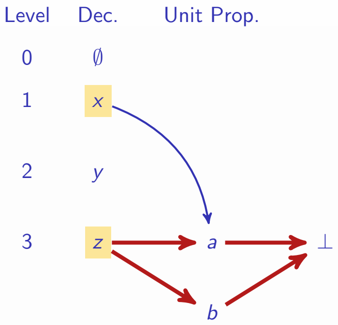
- Analyze conflict

Clause Learning



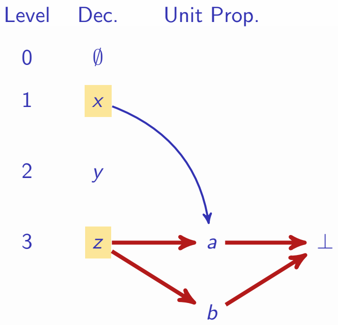
- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels

Clause Learning



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

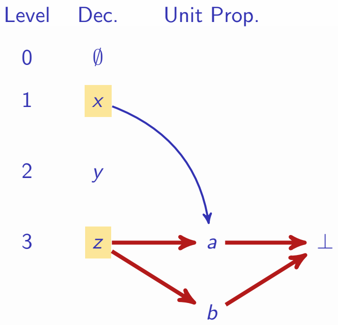
Clause Learning

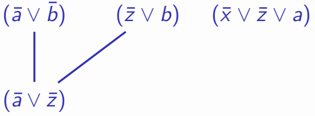


$(\bar{a} \vee \bar{b})$ $(\bar{z} \vee b)$ $(\bar{x} \vee \bar{z} \vee a)$

- 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

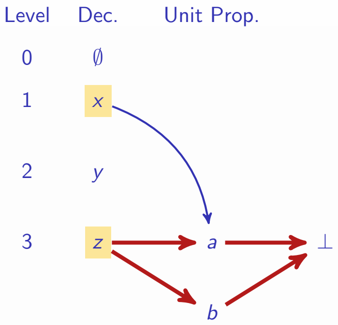
Clause Learning

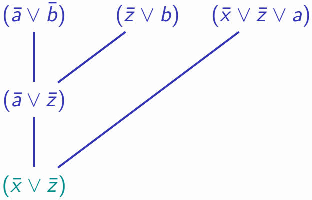




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

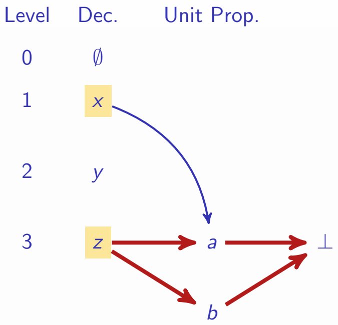
Clause Learning

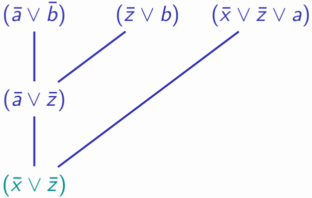




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

Clause Learning



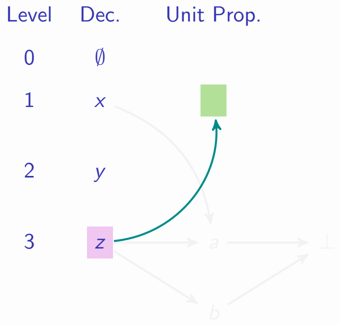


- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

Clause Learning – After Bracktracking

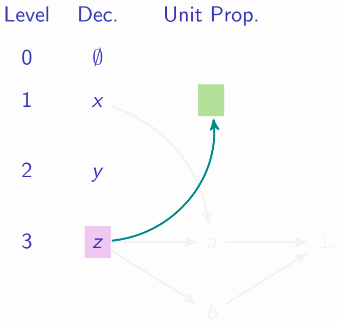
$$
\begin{tikzcd}[row sep=small]
\text{Level} & \text{Dec.} & \text{Unit Prop.} & \\
0 & \emptyset & & \\
1 & x \arrow[ddr, bend left] & & \\
2 & y & & \\
3 & z \arrow[r] \arrow[dr] & a \arrow[r] & \perp \\
 & & b \arrow[ur] &
\end{tikzcd}
$$

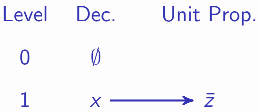
Clause Learning – After Bracktracking



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

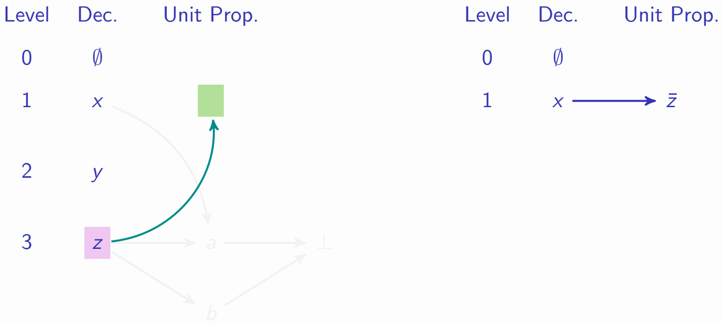
Clause Learning – After Bracktracking





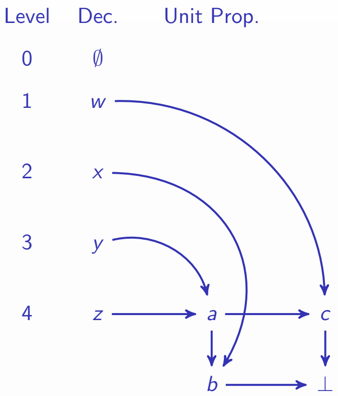
- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

Clause Learning – After Bracktracking

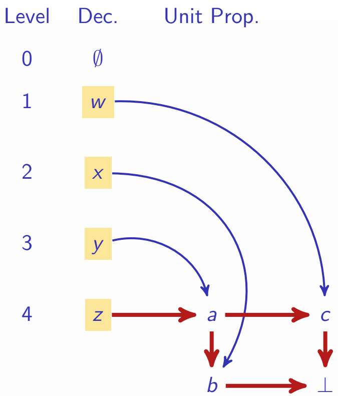


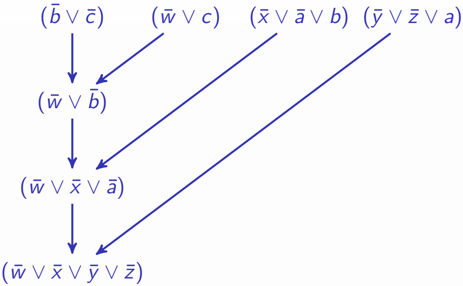
- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1
- Learned clauses are always asserting [MSS96,MSS99]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict [MMZZM01]

Unique Implication Points (UIPs)



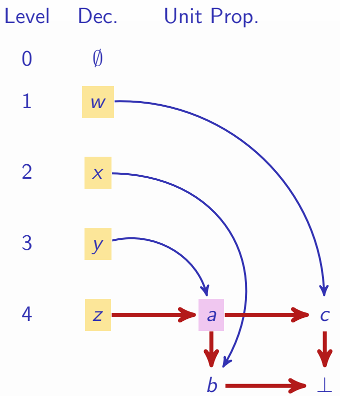
Unique Implication Points (UIPs)

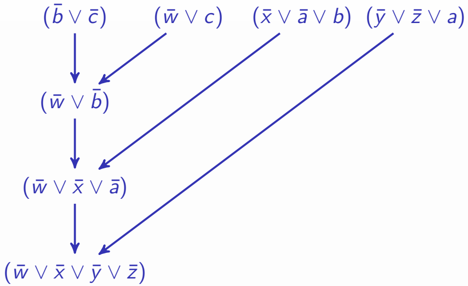




- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

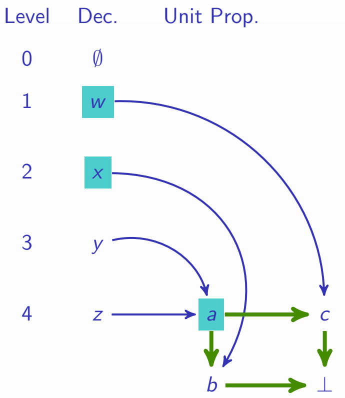
Unique Implication Points (UIPs)

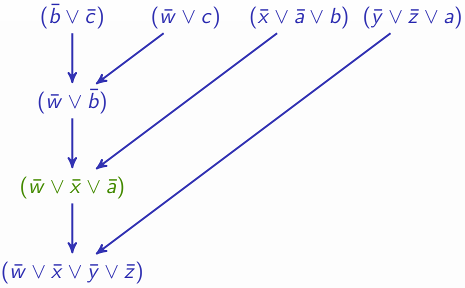




- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$
- But a is an UIP

Unique Implication Points (UIPs)





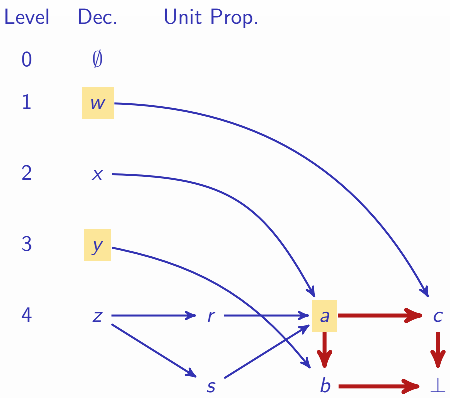
- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But a is an UIP
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

Multiple UIPs

Level Dec. Unit Prop.

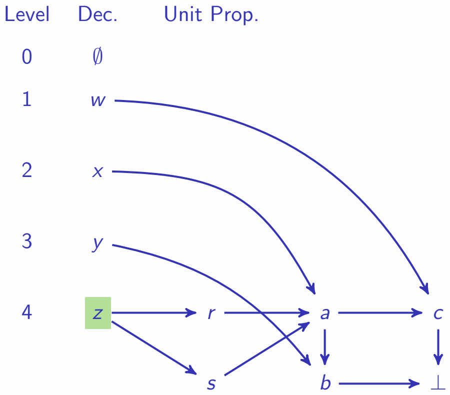
$$
\begin{tikzcd}
0 & \emptyset & & & \\
1 & w \arrow[rrrddd, bend left] & & & \\
2 & x \arrow[rrdd, bend left] & & & \\
3 & y \arrow[rrdd, bend left] & & & \\
4 & z \arrow[r] \arrow[dr] & r \arrow[r] & a \arrow[r] \arrow[d] & c \arrow[d] \\
 & & s \arrow[ur] & b \arrow[r] & \perp
\end{tikzcd}
$$

Multiple UIPs



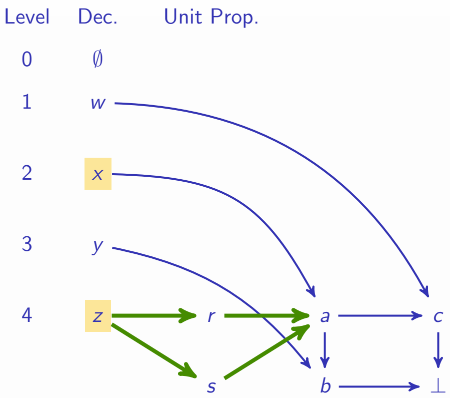
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$

Multiple UIPs



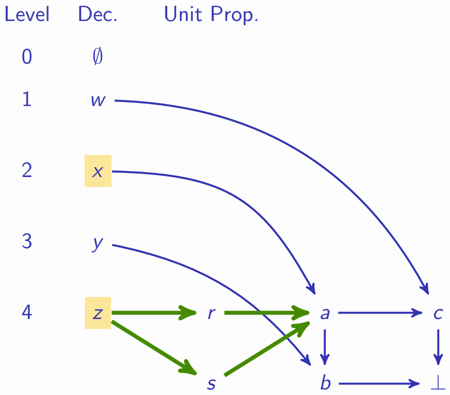
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP

Multiple UIPs



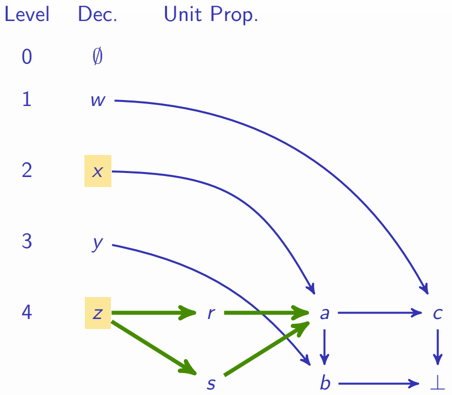
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$

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

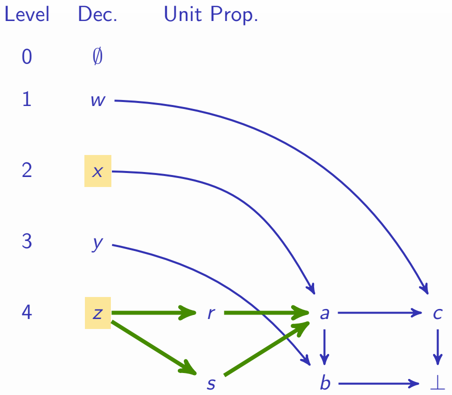
Multiple UIPs



- 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

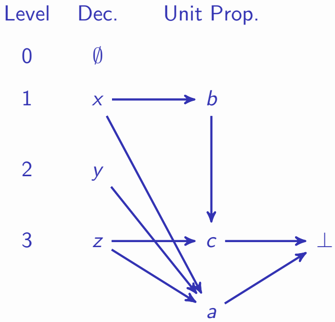
Multiple UIPs



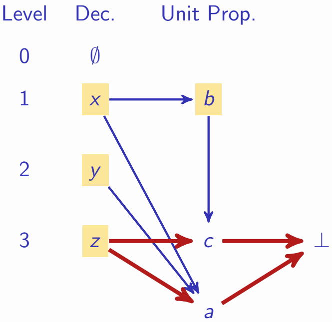
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$
- In practice smaller clauses more effective
  - Compare with $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

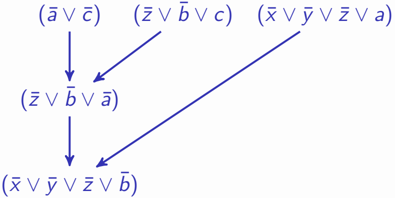
- Multiple UIPs proposed in GRASP [MSS96]
  - First UIP learning proposed in Chaff [MMZZM01]
- Not used in recent state of the art CDCL SAT solvers
- Recent results show it can be beneficial on current instances [SSS12]

Clause Minimization I



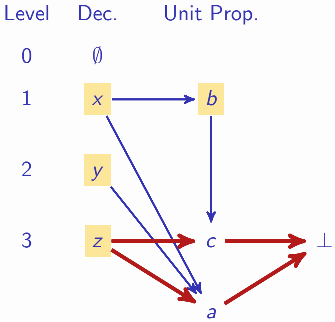
Clause Minimization I

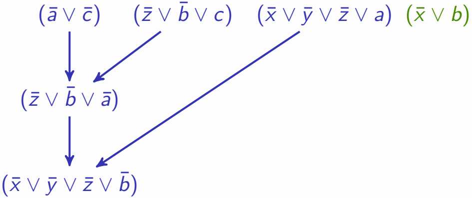




- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$

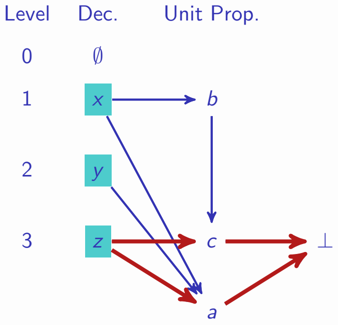
Clause Minimization I

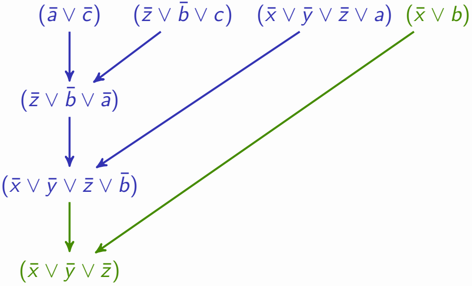




- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$
- Apply self-subsuming resolution (i.e. local minimization) [SB09]

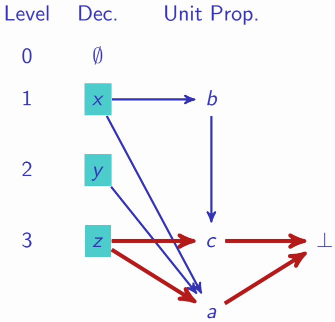
Clause Minimization I

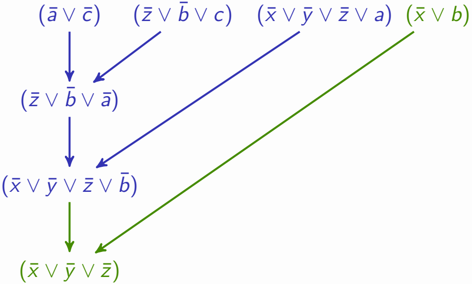




- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization)

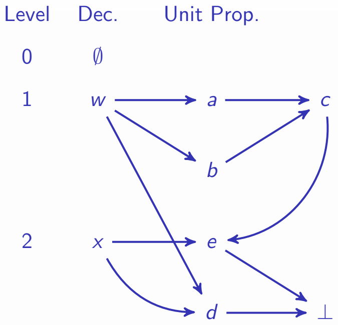
Clause Minimization I



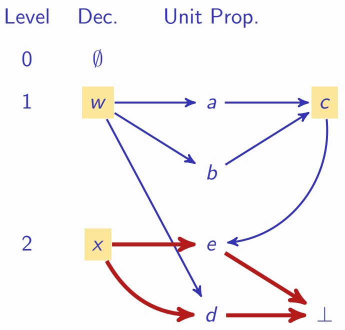


- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization)
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z})$

Clause Minimization II

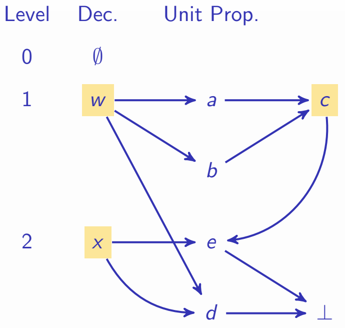


Clause Minimization II



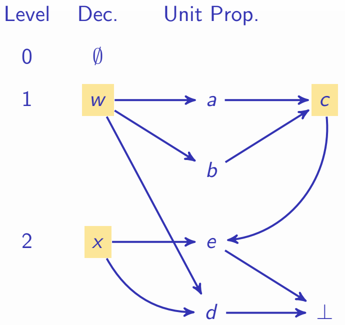
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$

Clause Minimization II



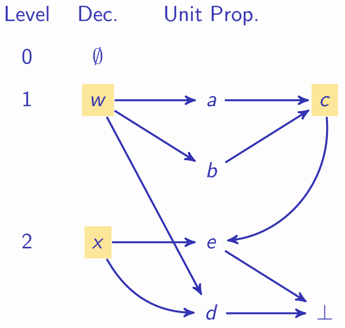
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$

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

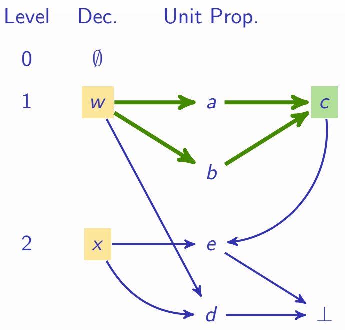


- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization

- Marked nodes: literals in learned clause

[SB09]

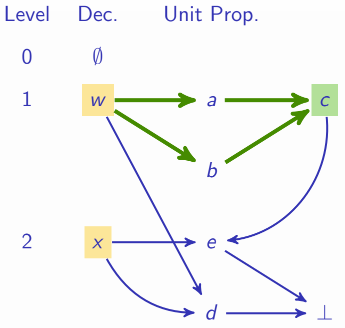
Clause Minimization II



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

Clause Minimization II



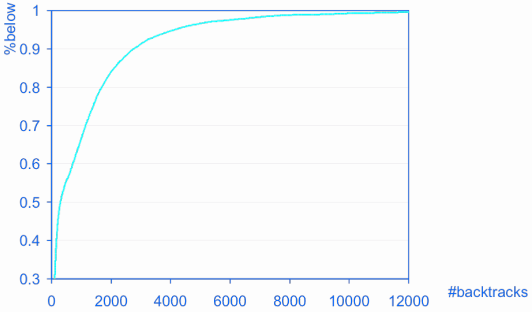
- ~~Learn clause ($\bar{w} \vee \bar{x} \vee \bar{c}$)~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields ($\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b}$)
- Can apply recursive minimization
- Learn clause ($\bar{w} \vee \bar{x}$)

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

Outline

Search Restarts I

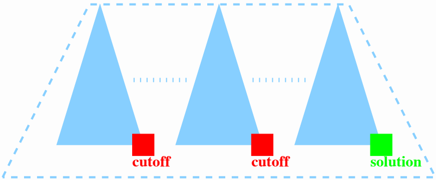
- Heavy-tail behavior: [GSK98]



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

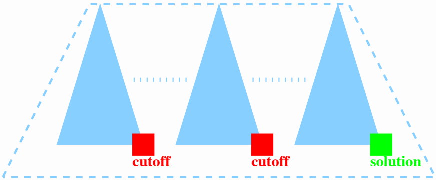
Search Restarts II

- Restart search after a number of conflicts



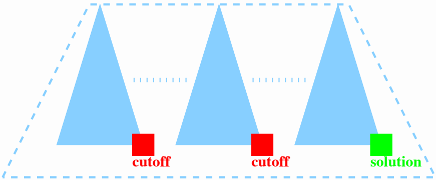
Search Restarts II

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



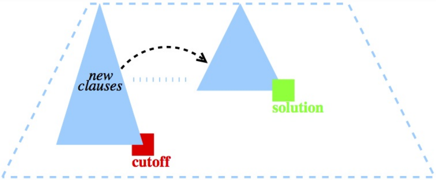
Search Restarts II

- 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,

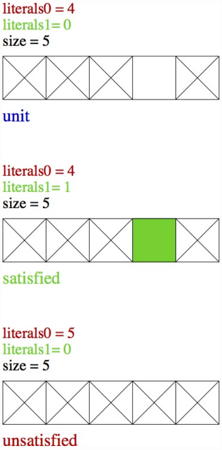
    Unit propagation slow-down worse than linear as clauses are learned **!**

- 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]

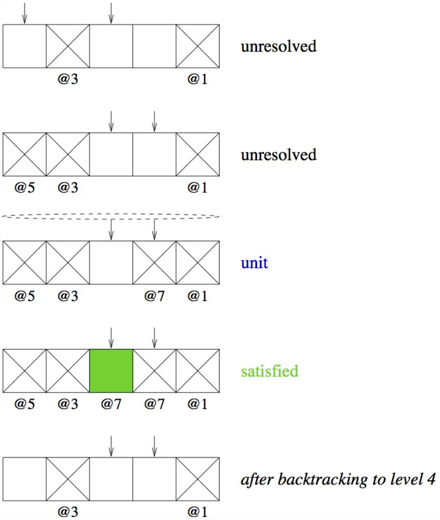
- Important states of a clause



Watched Literals

[MMZZM01]

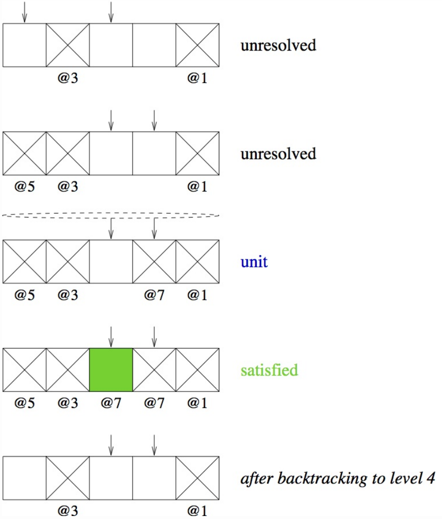
- Important states of a clause
- Associate 2 references with each clause



Watched Literals

[MMZZM01]

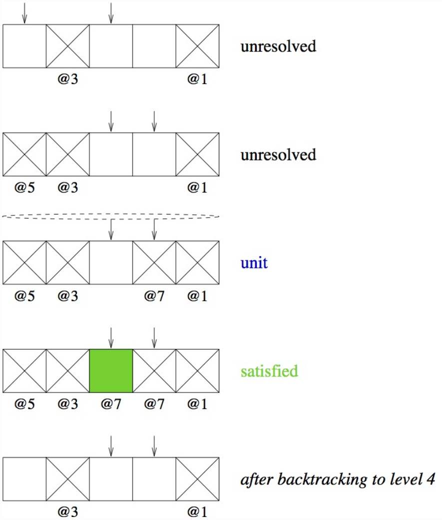
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals



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 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]
  - Literal blocks distance [AS09]

Outline

CDCL – A Glimpse of the Future

- 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]
  - Essential in some applications
- Application-driven improvements
  - Incremental SAT
    - Handling of assumptions due to MUS extractors [LB13]

Part II

SAT-Based Problem Solving

How to Solve Problems with SAT?

- CNF encodings
  - Represent problem as instance of SAT
  - E.g. Eager SMT, Pseudo-Boolean constraints, etc.

How to Solve Problems with SAT?

- CNF encodings
  - Represent problem as instance of SAT
  - E.g. Eager SMT, Pseudo-Boolean constraints, etc.
- Embedding of SAT solvers
  - SAT solver used to implement domain specific algorithm
  - **White-box** integration
  - E.g. Lazy SMT, Pseudo-Boolean constraints/optimization, etc.

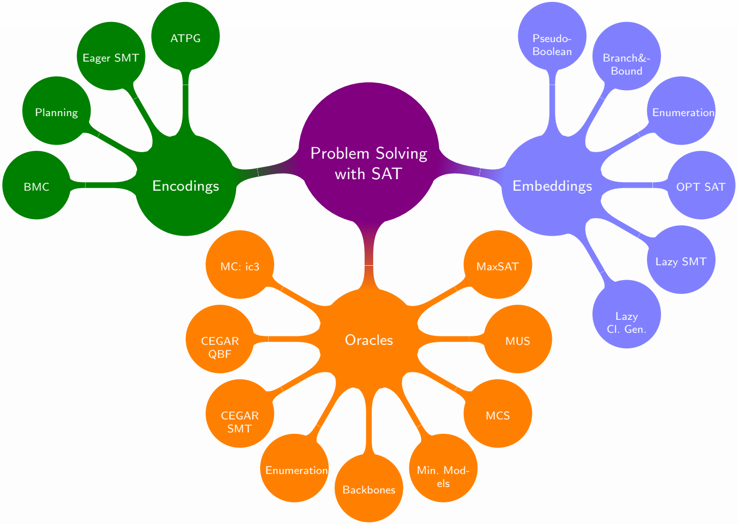
How to Solve Problems with SAT?

- CNF encodings
  - Represent problem as instance of SAT
  - E.g. Eager SMT, Pseudo-Boolean constraints, etc.
- Embedding of SAT solvers
  - SAT solver used to implement domain specific algorithm
  - **White-box** integration
  - E.g. Lazy SMT, Pseudo-Boolean constraints/optimization, etc.
- SAT solvers as oracles
  - Algorithm invokes SAT solver as (a variant of) an NP oracle
  - **Black-box** integration (using standard interface)
  - E.g. MaxSAT, MUSes, (2)QBF, etc.

How to Solve Problems with SAT?

- CNF encodings
  - Represent problem as instance of SAT
  - E.g. Eager SMT, Pseudo-Boolean constraints, etc.
- Embedding of SAT solvers
  - SAT solver used to implement domain specific algorithm
  - **White-box** integration
  - E.g. Lazy SMT, Pseudo-Boolean constraints/optimization, etc.
- SAT solvers as oracles (next lectures)
  - Algorithm invokes SAT solver as (a variant of) an NP oracle
  - **Black-box** integration (using standard interface)
  - E.g. MaxSAT, MUSes, (2)QBF, etc.
- **Note**:
  - CNF encodings most often used with either black-box or white-box approaches
  - SAT techniques adapted in many other domains: QBF, SMT, QBF, CSP, ASP, ILP, ...

SAT-Based Problem Solving



- Some apps associated with more than one concept: planning, BMC, lazy clause generation, etc.

Examples of SAT-Based Problem Solving I

- Function problems in $\mathrm{FP}^{\mathrm{NP}}[\log n]$
  - Unweighted Maximum Satisfiability (MaxSAT)
  - Minimal Correction Subsets (MCSes)
  - Minimal models
  - ...
- Function problems in $\mathrm{FP}^{\mathrm{NP}}$
  - Weighted Maximum Satisfiability (MaxSAT)
  - Minimal Unsatisfiable Subformulas (MUSes)
  - Minimal Equivalent Subformulas (MESes)
  - Prime implicates
  - ...
- Enumeration problems
  - Models
  - MUSes
  - MCSes
  - MaxSAT
  - ...

Examples of SAT-Based Problem Solving II

- Decision problems in Σ_2^P
  - 2QBF
  - ...
- Function problems in $FP^{\Sigma_2^P}$
  - Propositional formula minimization
  - (Weighted) Quantified MaxSAT (QMaxSAT)
  - Smallest MUS (SMUS)
  - ...
- Decision problems in PSPACE
  - QBF
  - ...
- ...

Outline

Outline

CNF Encodings

SAT Embeddings

Conclusions

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
    - ...

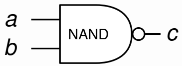
- Key issues:
  - Encoding size
  - Arc-consistency?

Outline

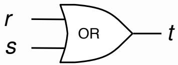
Representing Boolean Formulas / Circuits I

- 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

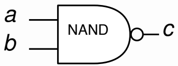
$$\mathcal{F}_c = (a \vee c) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$$



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



Representing Boolean Formulas / Circuits II



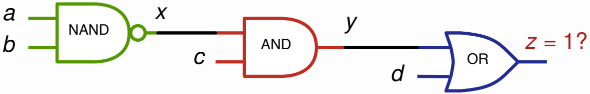
c \ ab	00	01	11	10
0	0	0	1	0
1	1	1	0	1

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

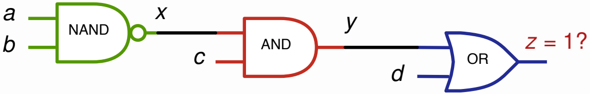
- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses



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

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

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]
  - ...

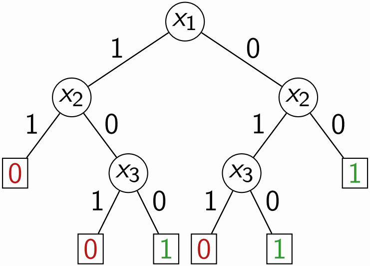
Outline

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: $\mathcal{O}(n^3 \log(a_{max}))$
  - ...

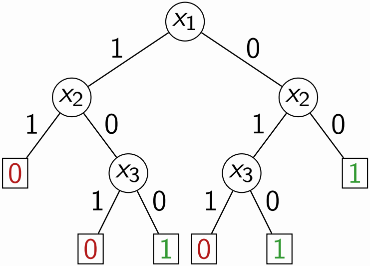
Encoding PB Constraints with BDDs I

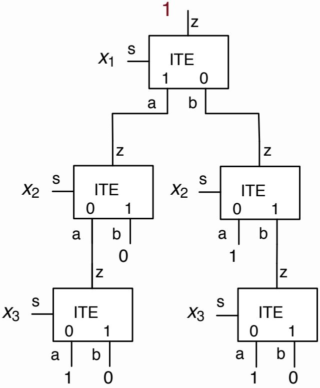
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD



Encoding PB Constraints with BDDs I

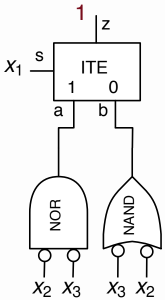
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD





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 $(\sum_{j=1}^{n} a_j\, x_j \geq k) \wedge (\sum_{j=1}^{n} a_j\, x_j \leq k)$, 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 $(4x_1 + 4x_2 + 3x_3 + 2x_4 \geq 5) \wedge (4x_1 + 4x_2 + 3x_3 + 2x_4 \leq 5)$

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

CSP Constraints

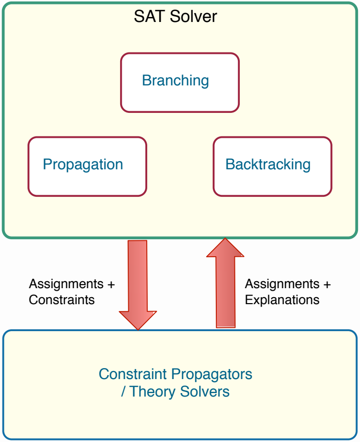
- Many possible encodings:
  - Direct encoding [dK89,GJ96,W00]
  - Log encoding [W00]
  - Support encoding [K90,G02]
  - Log-Support encoding [G07]
  - Order encoding for finite linear CSPs [TTKB09]
  - ...

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 $x_i = v_i \land x_j = v_j$ is not allowed, add binary clause $(\bar{x}_{i,v_i} \lor \bar{x}_{j,v_j})$

Outline

Embedding SAT Solvers



- Modify SAT solver to interface problem-specific propagators (or theory solvers)
- Typical interface:
  - SAT solvers communicates assignments/constraints to propagators
  - Retrieve resulting assignments or explanations for inconsistency
- Well-known examples (many more):
  - Branch&bound PB optimization
  - Non-clausal SAT solvers
  - Lazy SMT solving
- Key problem:
  - Keeping up with improvements in SAT solvers

Pseudo-Boolean Constraints & Optimization

- Pseudo-Boolean Constraints:
  - Boolean variables: $x_1, \ldots, x_n$
  - Linear inequalities:

$$\sum_{j \in N} a_{ij} l_j \geq b_i, \quad l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0,1\}, a_{ij}, b_i \in \mathbb{N}_0^+$$

Pseudo-Boolean Constraints & Optimization

- Pseudo-Boolean Constraints:
  - Boolean variables: $x_1, \ldots, x_n$
  - Linear inequalities:

$$\sum_{j \in N} a_{ij} l_j \geq b_i, \quad l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0, 1\}, a_{ij}, b_i \in \mathbb{N}_0^+$$

- Pseudo-Boolean Optimization (PBO):

$$\begin{array}{ll} \text{minimize} & \sum_{j \in N} c_j \cdot x_j \\ \text{subject to} & \sum_{j \in N} a_{ij} l_j \geq b_i, \\ & l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0, 1\}, a_{ij}, b_i, c_j \in \mathbb{N}_0^+ \end{array}$$

Pseudo-Boolean Constraints & Optimization

- Pseudo-Boolean Constraints:
  - Boolean variables: $x_1, \ldots, x_n$
  - Linear inequalities:

$$\sum_{j \in N} a_{ij} l_j \geq b_i, \quad l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0,1\}, a_{ij}, b_i \in \mathbb{N}_0^+$$

- Pseudo-Boolean Optimization (PBO):

$$\begin{array}{ll} \text{minimize} & \sum_{j \in N} c_j \cdot x_j \\ \text{subject to} & \sum_{j \in N} a_{ij} l_j \geq b_i, \\ & l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0,1\}, a_{ij}, b_i, c_j \in \mathbb{N}_0^+ \end{array}$$

- Branch and bound (B&B) PBO algorithm: [MMS00]
  - Extend SAT solver
  - Must develop propagator for PB constraints
  - B&B search for computing optimum cost function value
    - Trivial upper bound: all $x_j = 1$

Limitations with Embeddings

- B&B MaxSAT solving:
  - Cannot use unit propagation
  - Cannot learn clauses
- MUS extraction:
  - Decision of clauses to include in MUS based on unsatisfiable outcomes
  - No immediate gain from embedding SAT solvers

Outline

Conclusions

- Overview of modern SAT solvers
  - CDCL SAT solvers: clause learning, UIPs, clause minimization, search restarts, etc.
- Introduction to SAT-based problem solving
  - CNF encodings
  - Embedding of SAT solvers
- Next lectures: problem solving with SAT oracles