CDCL SAT Solvers & SAT-Based Problem Solving

Joao Marques-Silva[1,2] & Mikolas Janota[2]

[1]University College Dublin, Ireland

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

SAT/SMT Summer School 2013

Aalto University, Espoo, Finland

The Success of SAT

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

The Success of SAT

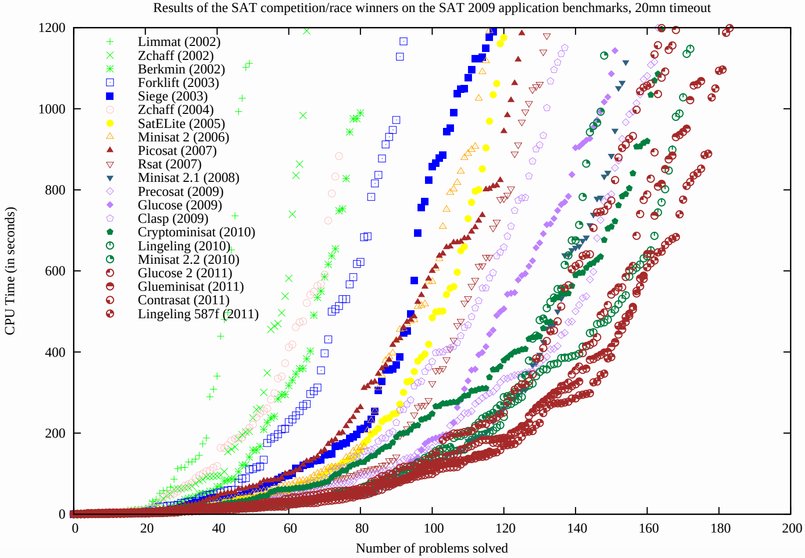
- Well-known NP-complete decision problem [C71]
- In practice, SAT is a success story of Computer Science
  - Hundreds (even more?) of practical applications

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]



This Lecture

- Overview modern SAT solvers
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Note:** Overview for non-experts

This Lecture

- Overview modern SAT solvers
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Note:** Overview for non-experts
- SAT-based problem solving in practice
  - How to do it?

This Lecture

- Overview modern SAT solvers
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Note:** Overview for non-experts
- SAT-based problem solving in practice
  - How to do it?
    - **Encode** problems to SAT
    - **Embed** SAT solvers in applications
    - Iteratively use a SAT solver (i.e. as an **NP oracle**)

Part I

CDCL SAT Solvers

Outline

Outline

Basic Definitions

DPLL Solvers

CDCL Solvers

What Next in CDCL Solvers?

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 $\{0, 1\}$
- 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 $\{0, 1\}$
- Formula can be SAT/UNSAT
- Example:

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

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

Resolution

- Resolution rule: [DP60,R65]

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

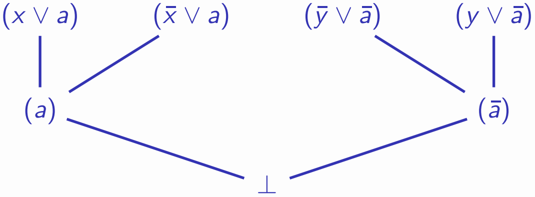
  - Complete proof system for propositional logic

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



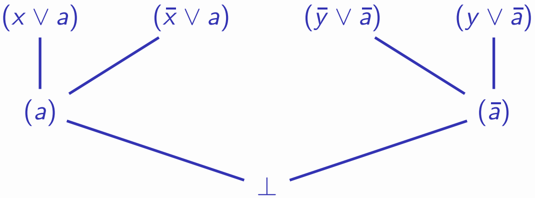
  - Extensively used with (CDCL) SAT solvers

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

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

Unit Propagation

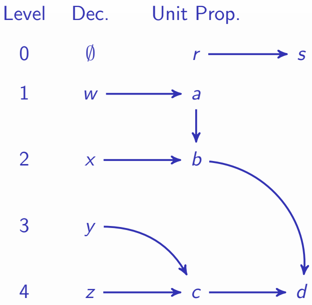
$$
\begin{aligned}
\mathcal{F} \;=\; & (r) \wedge (\bar{r} \vee s) \wedge \\
& (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\
& (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)
\end{aligned}
$$

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

Unit Propagation

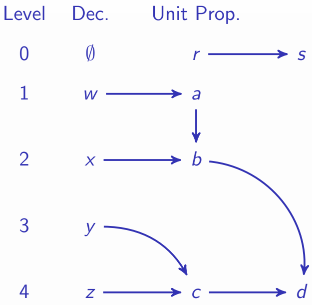
$$\mathcal{F} = (r) \wedge (\bar{r} \vee s) \wedge$$
$$(\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b)$$
$$(\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$

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



Unit Propagation

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

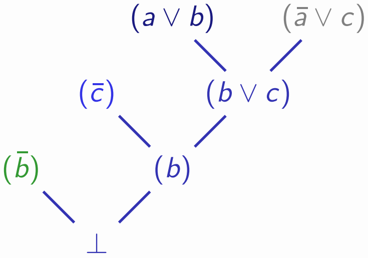


- Decisions / Variable Branchings: $w = 1, x = 1, y = 1, z = 1$

- 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:



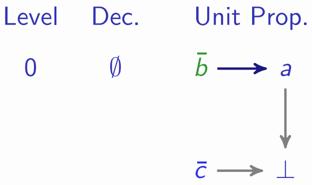
- A modern SAT solver can generate resolution proofs using clauses learned by the solver

[ZM03]

Unsatisfiable Cores & Proof Traces

- CNF formula:

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



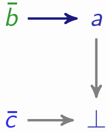
Implication graph with conflict

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	$\emptyset$	

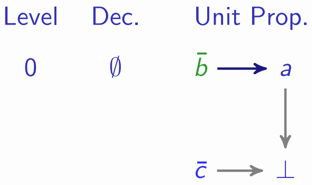


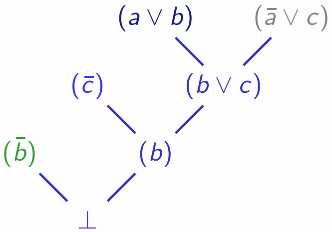
Proof trace $\perp$: $(\bar{a} \vee c)$ $(a \vee b)$ $(\bar{c})$ $(\bar{b})$

Unsatisfiable Cores & Proof Traces

- CNF formula:

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



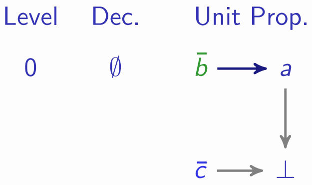


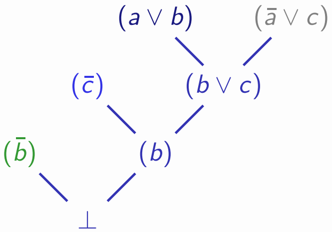
Resolution proof follows structure of conflicts

Unsatisfiable Cores & Proof Traces

- CNF formula:

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

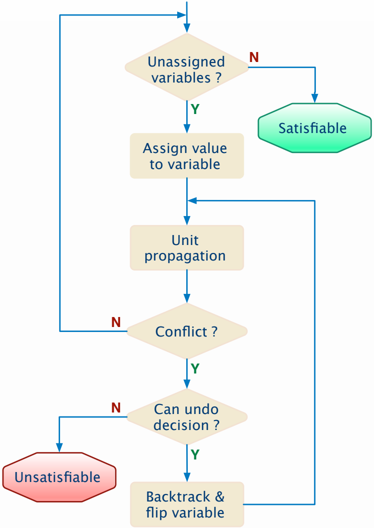




Unsatisfiable subformula (core): $(\bar{c}), (\bar{b}), (\bar{a} \vee c), (a \vee b)$

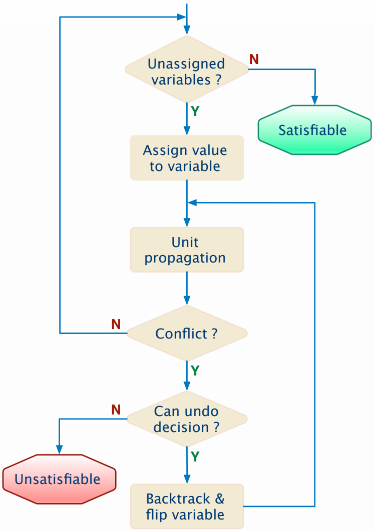
Outline

The DPLL Algorithm



- Optional: pure literal rule

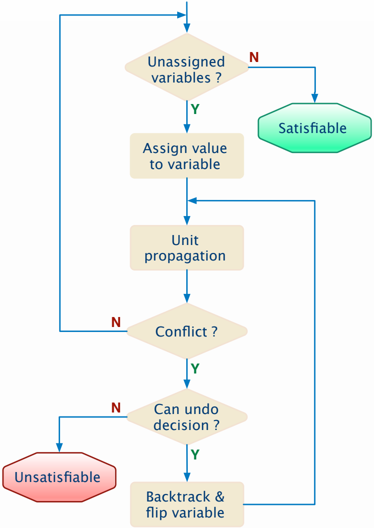
The DPLL Algorithm



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

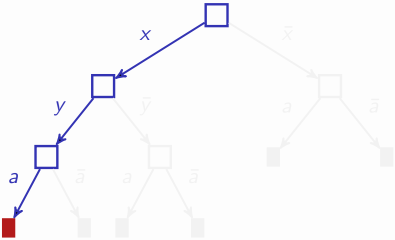
- Optional: pure literal rule

The DPLL Algorithm



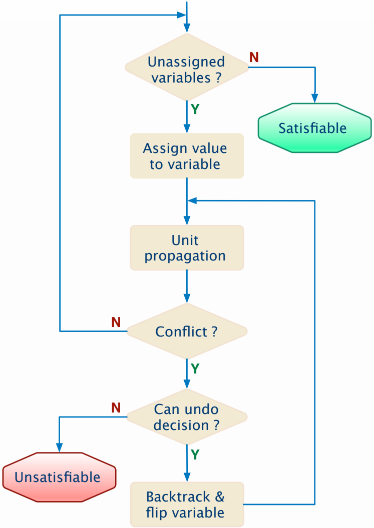
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



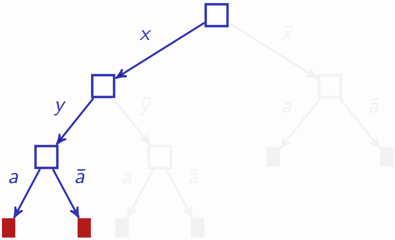
- Optional: pure literal rule

The DPLL Algorithm



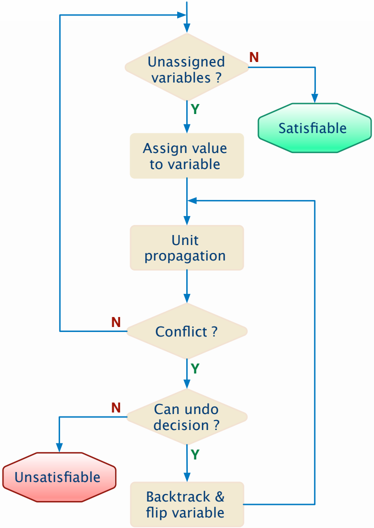
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



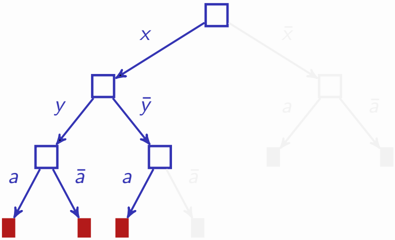
- Optional: pure literal rule

The DPLL Algorithm



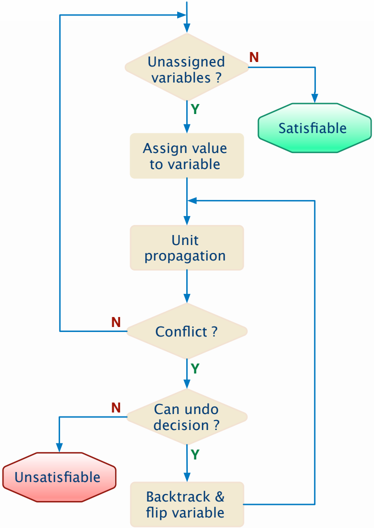
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



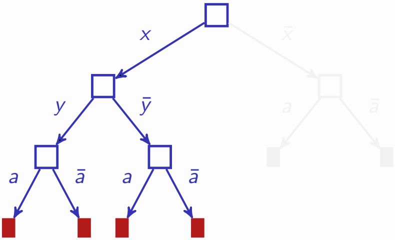
- Optional: pure literal rule

The DPLL Algorithm



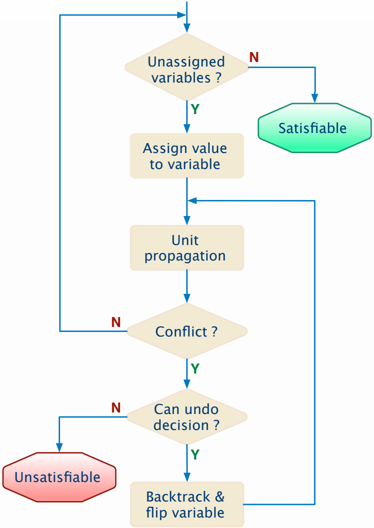
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



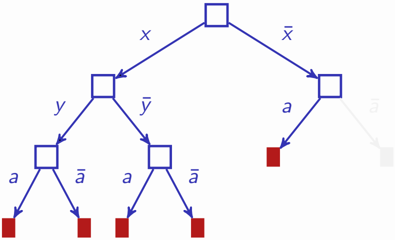
- Optional: pure literal rule

The DPLL Algorithm



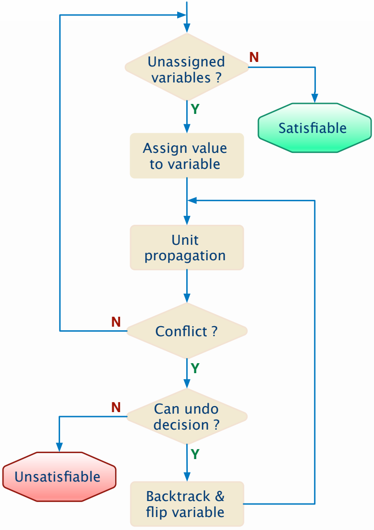
$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$

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

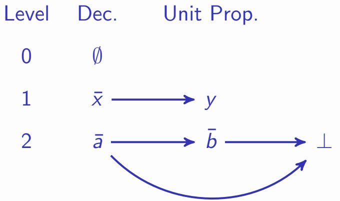


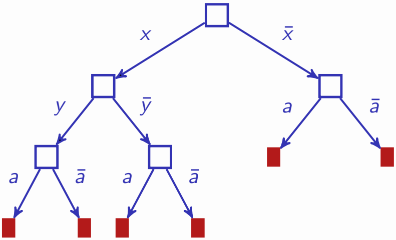
- Optional: pure literal rule

The DPLL Algorithm



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





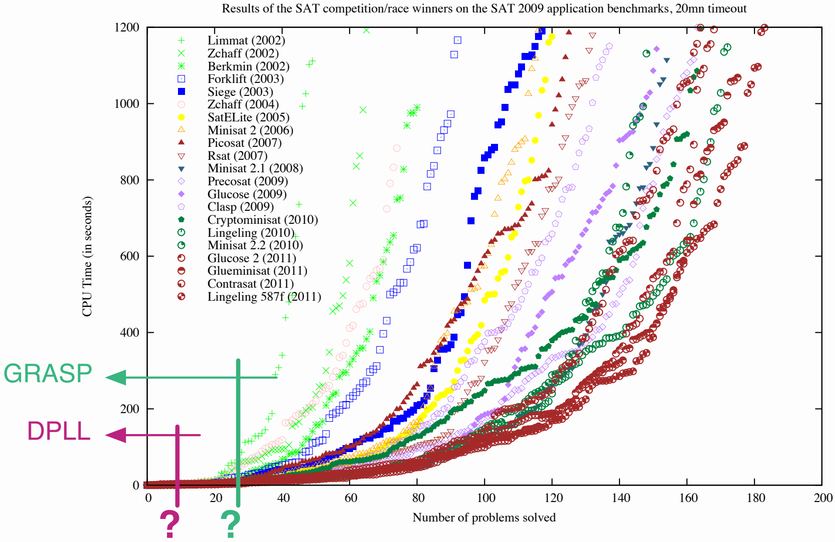
- Optional: pure literal rule

Outline

What is a CDCL SAT Solver?

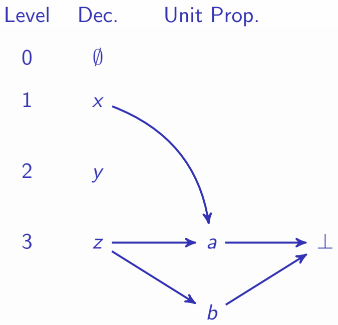
- Extend DPLL SAT solver with: [DP60,DLL62]
  - Clause learning & non-chronological backtracking [MSS96,BS97,Z97]
    - Exploit UIPs [MSS96,SSS12]
    - Minimize learned clauses [SB09,VG09]
    - Opportunistically delete clauses [MSS96,MSS99,GN02]
  - Search restarts [GSK98,BMS00,H07,B08]
  - Lazy data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

How Significant are CDCL SAT Solvers?

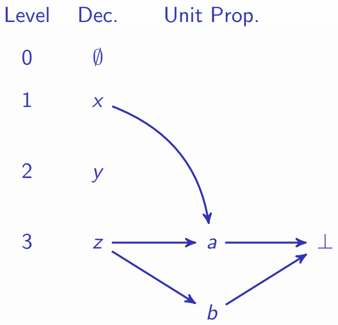


Outline

Clause Learning

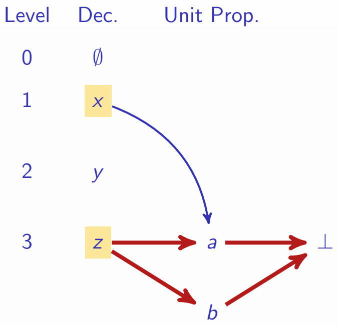


Clause Learning



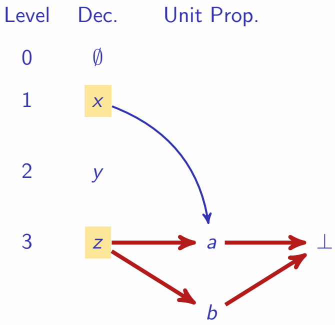
- Analyze conflict

Clause Learning



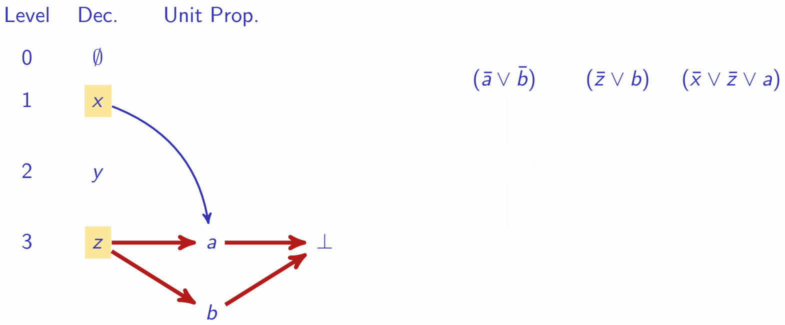
- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels

Clause Learning



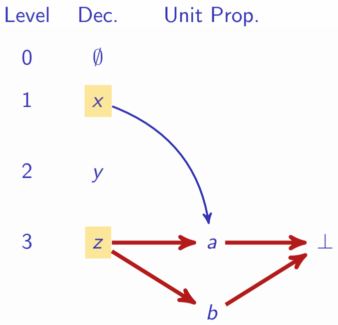
- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$

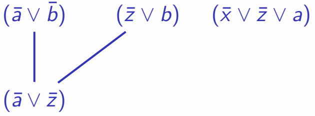
Clause Learning



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

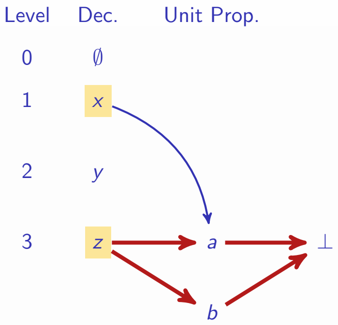
Clause Learning

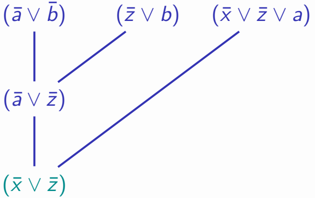




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

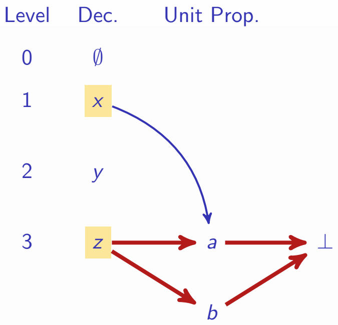
Clause Learning

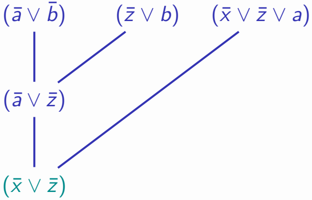




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

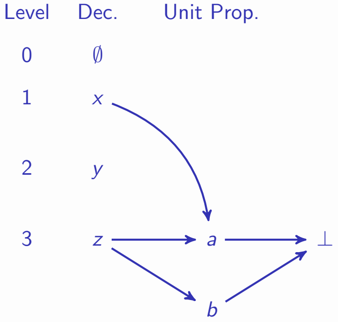
Clause Learning



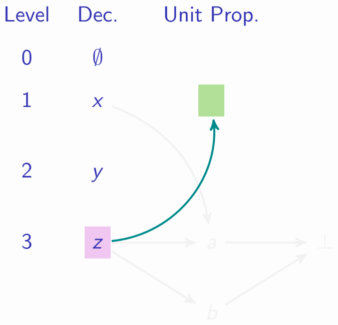


- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

Clause Learning – After Bracktracking

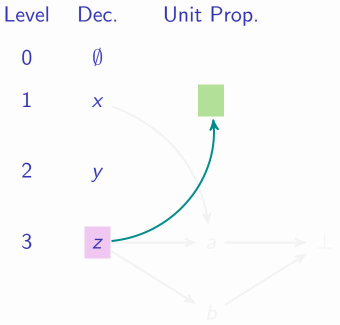


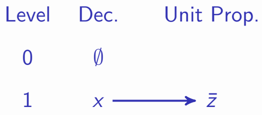
Clause Learning – After Bracktracking



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

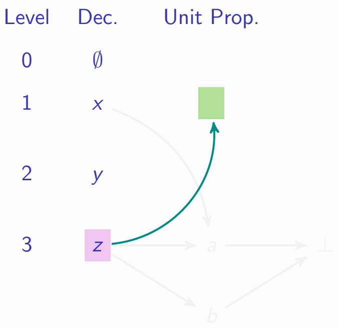
Clause Learning – After Bracktracking

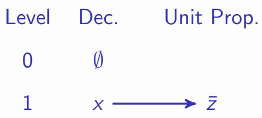




- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

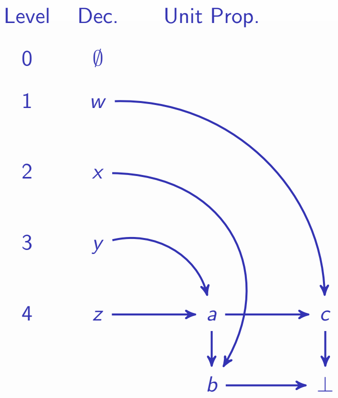
Clause Learning – After Bracktracking



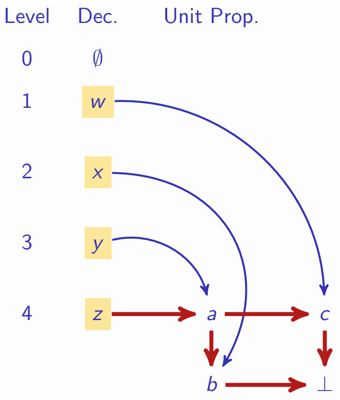


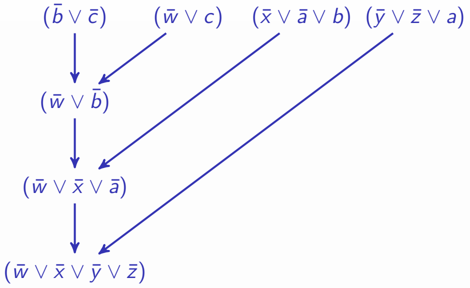
- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1
- Learned clauses are always asserting [MSS96,MSS99]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict [MMZZM01]

Unique Implication Points (UIPs)



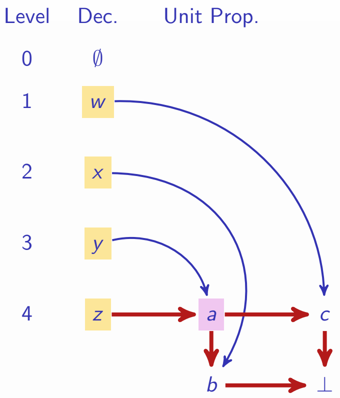
Unique Implication Points (UIPs)

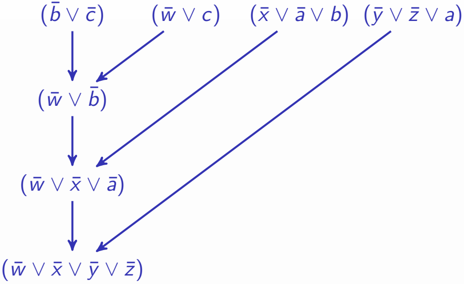




- Learn clause $(\bar{w} \lor \bar{x} \lor \bar{y} \lor \bar{z})$

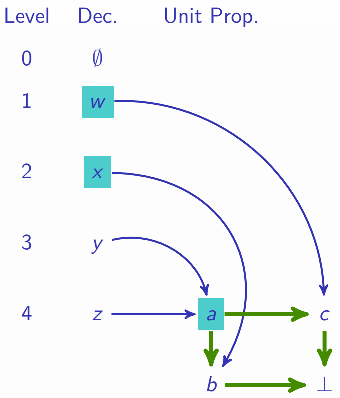
Unique Implication Points (UIPs)

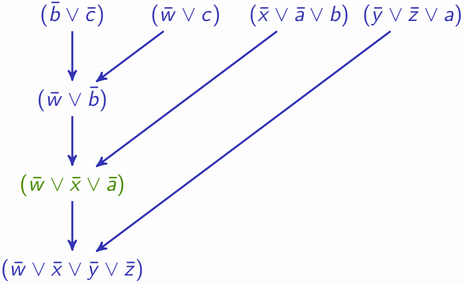




- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$
- But a is an UIP

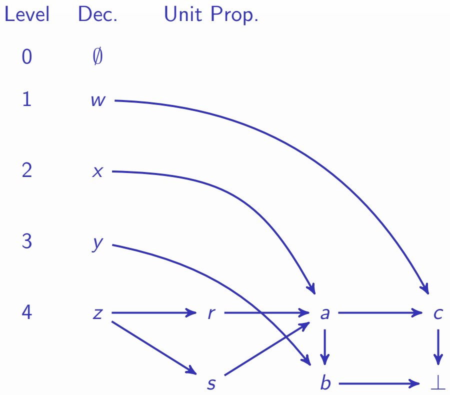
Unique Implication Points (UIPs)



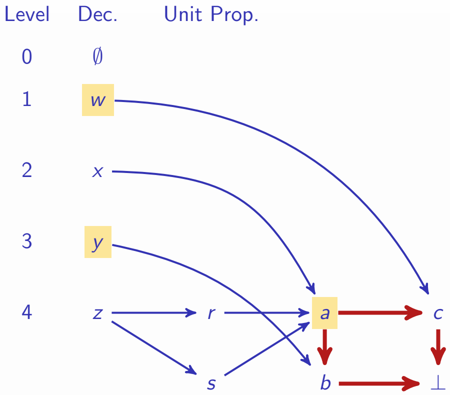


- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But a is an UIP
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

Multiple UIPs

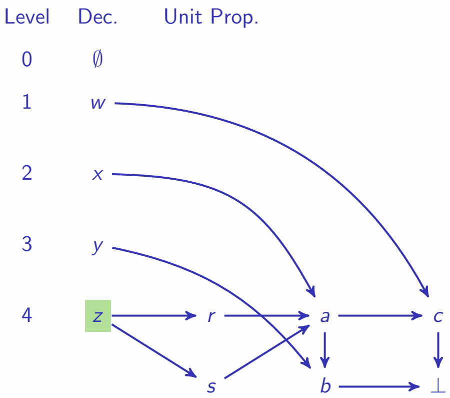


Multiple UIPs



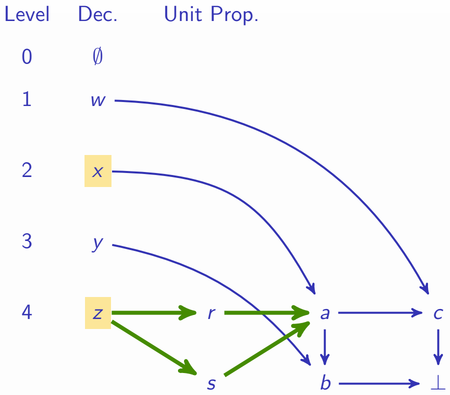
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$

Multiple UIPs



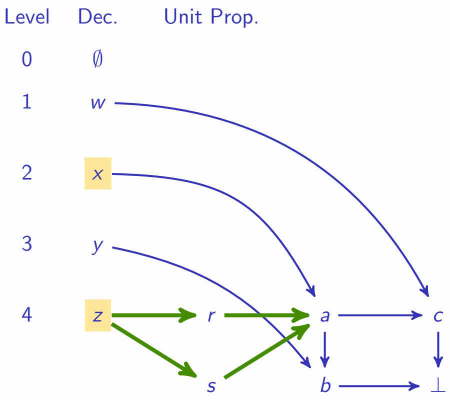
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP

Multiple UIPs



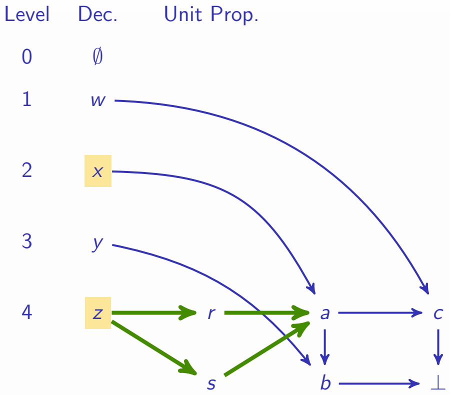
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$

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

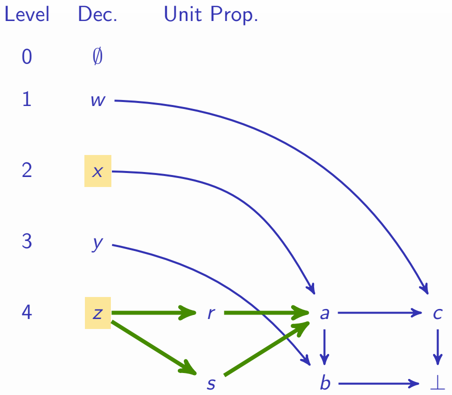
Multiple UIPs



- 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

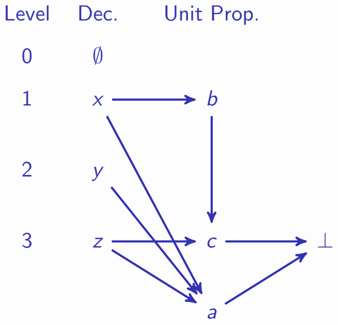
Multiple UIPs



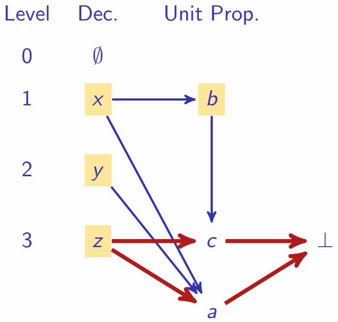
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$
- In practice smaller clauses more effective
  - Compare with $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

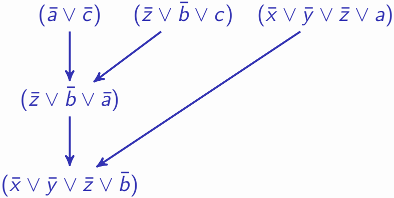
- Multiple UIPs proposed in GRASP [MSS96]
  - First UIP learning proposed in Chaff [MMZZM01]
- Not used in recent state of the art CDCL SAT solvers
- Recent results show it can be beneficial on current instances [SSS12]

Clause Minimization I



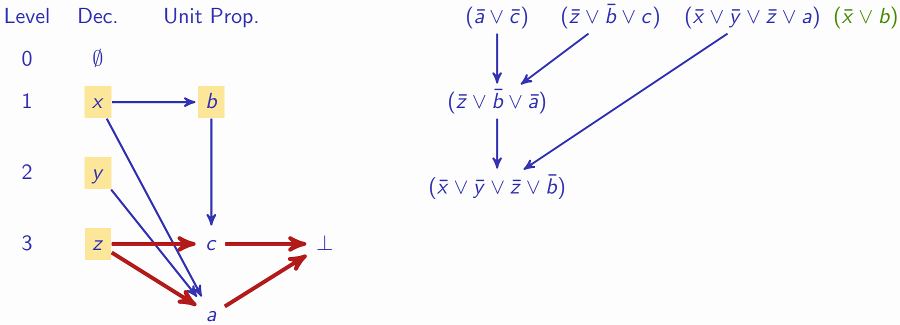
Clause Minimization I





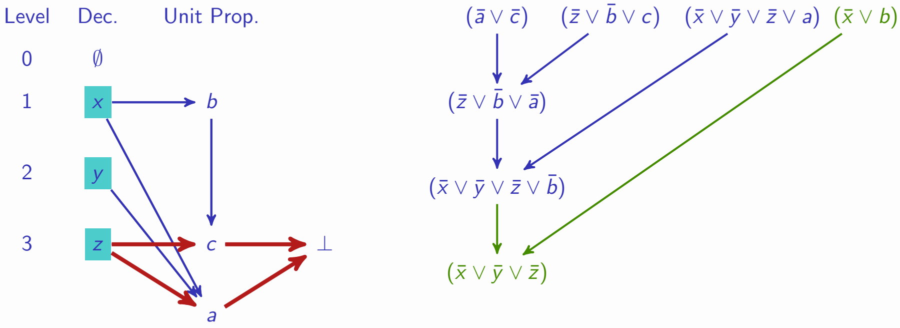
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$

Clause Minimization I



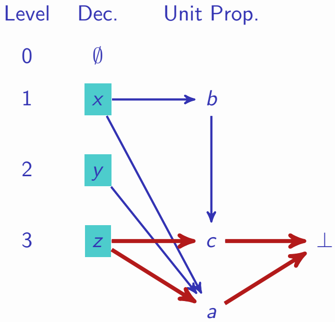
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$
- Apply self-subsuming resolution (i.e. local minimization) [SB09]

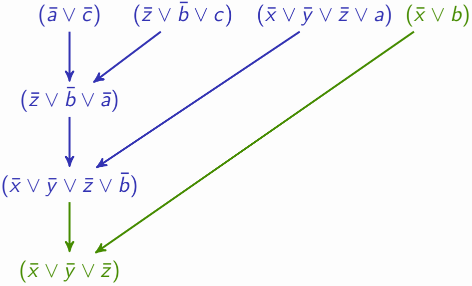
Clause Minimization I



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

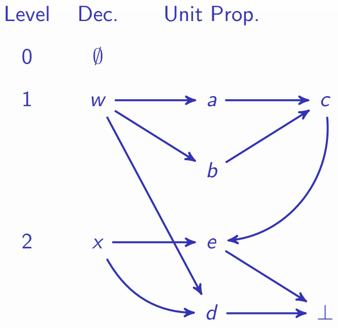
Clause Minimization I



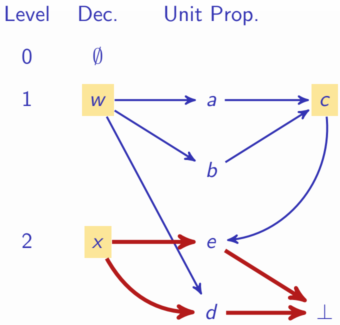


- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization)
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z})$

Clause Minimization II

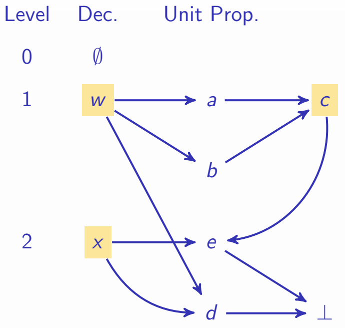


Clause Minimization II



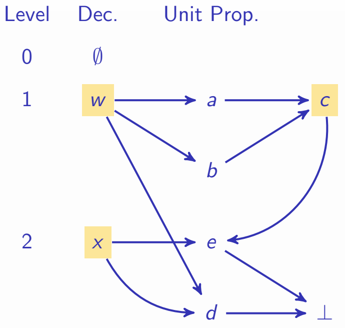
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$

Clause Minimization II



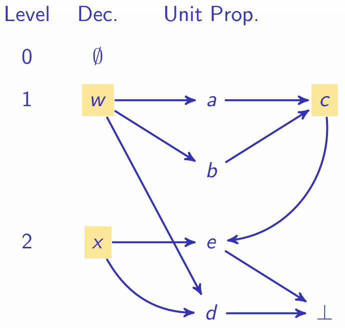
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$

Clause Minimization II



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

Clause Minimization II

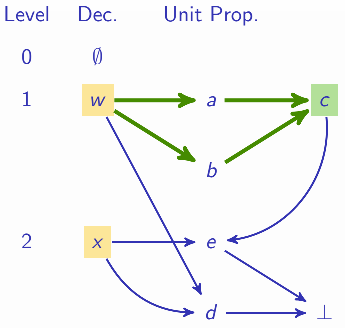


- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization

- Marked nodes: literals in learned clause

[SB09]

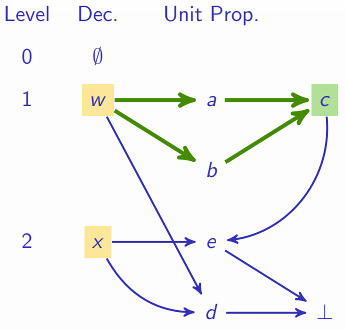
Clause Minimization II



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

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

Clause Minimization II



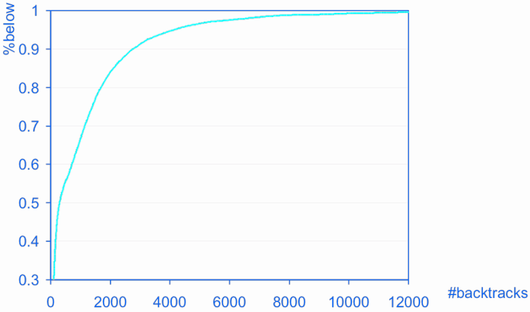
- ~~Learn clause ($\bar{w} \lor \bar{x} \lor \bar{c}$)~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields ($\bar{w} \lor \bar{x} \lor \bar{a} \lor \bar{b}$)
- Can apply recursive minimization
- Learn clause ($\bar{w} \lor \bar{x}$)

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

Outline

Search Restarts I

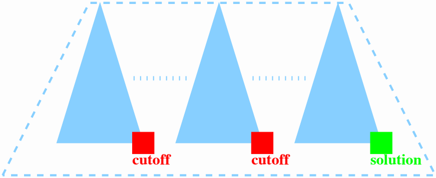
- Heavy-tail behavior: [GSK98]



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

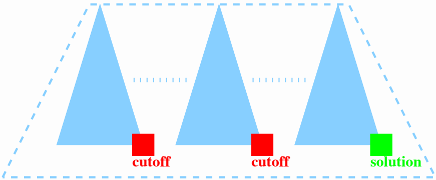
Search Restarts II

- Restart search after a number of conflicts



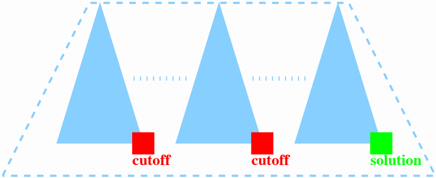
Search Restarts II

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



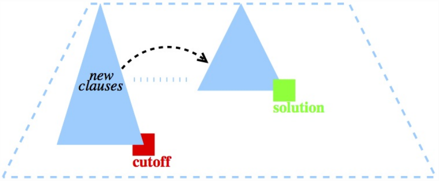
Search Restarts II

- 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,

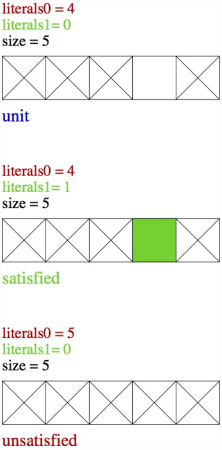
    Unit propagation slow-down worse than linear as clauses are learned **!**

- 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]

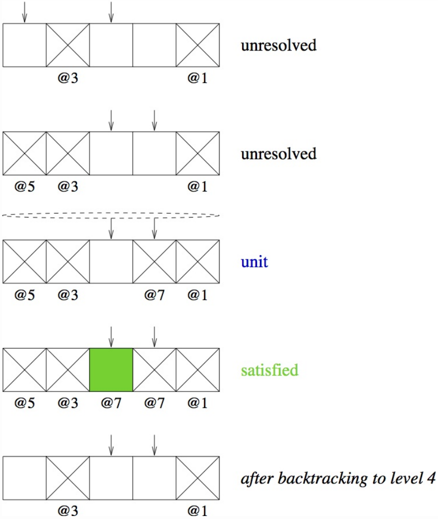
- Important states of a clause



Watched Literals

[MMZZM01]

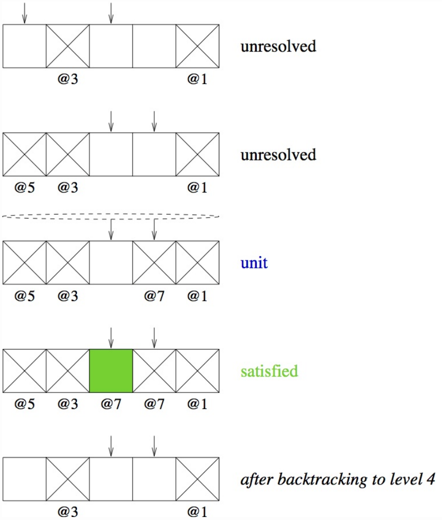
- Important states of a clause
- Associate **2** references with each clause



Watched Literals

[MMZZM01]

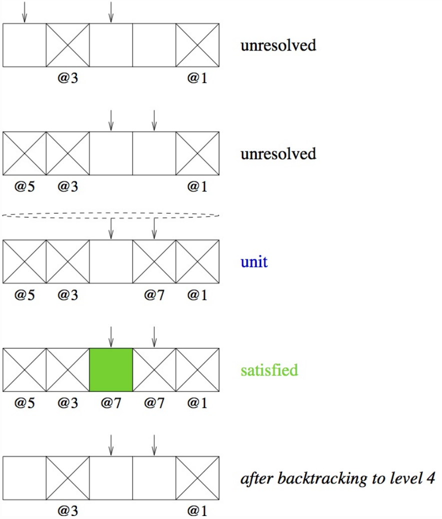
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals



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 less used clauses [e.g. MSS96,GN02,ES03]

Additional Key Techniques

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

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?

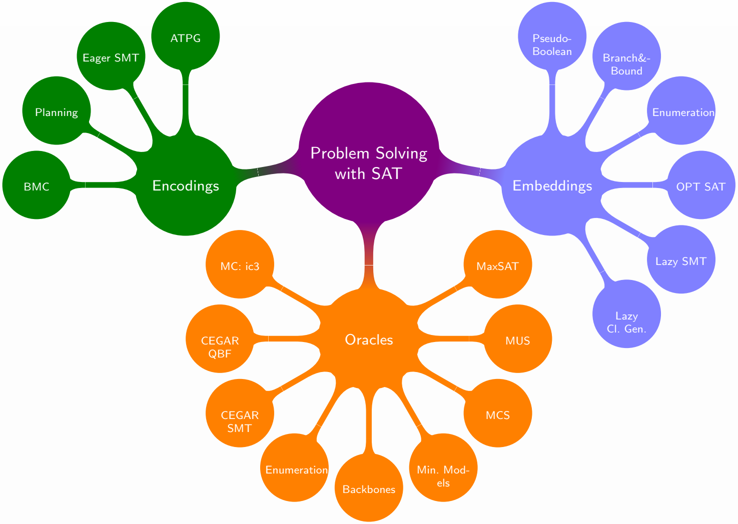
- CNF encodings
  - Represent problem as instance of SAT
  - E.g. Eager SMT, Pseudo-Boolean constraints, etc.
- Embedding of SAT solvers
  - SAT solver used to implement domain specific algorithm
  - White-box integration
  - E.g. Lazy SMT, Pseudo-Boolean constraints/optimization, etc.
- SAT solvers as oracles
  - Algorithm invokes SAT solver as an NP oracle
  - Black-box integration
  - 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
  - Algorithm invokes SAT solver as an NP oracle
  - Black-box integration
  - 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, ASP, ILP, CSP, ...

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 $\text{FP}^{\text{NP}}[\log n]$
  - Unweighted Maximum Satisfiability (MaxSAT)
  - Minimal Correction Subsets (MCSes)
  - Minimal models
  - ...
- Function problems in FP^{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 $FP^{\Sigma_2^P}$
  - (Weighted) Quantified MaxSAT (QMaxSAT) [IJMS13]
  - Smallest MUS (SMUS) [IJMS13]
  - ...
- Decision problems in PSPACE
  - QBF
  - ...
- ...

Outline

Outline

CNF Encodings

SAT Embeddings

SAT Oracles

What Next in SAT-Based Problem Solving?

Encoding to CNF

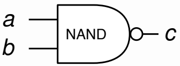
- What to encode?
  - Boolean formulas
    - Tseitin's encoding
    - Plaisted&Greenbaum's encoding
    - ...
  - Cardinality constraints
  - Pseudo-Boolean (PB) constraints
  - Can also translate to SAT:
    - Constraint Satisfaction Problems (CSPs)
    - Answer Set Programming (ASP)
    - Model Finding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

Outline

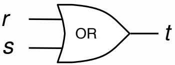
Representing Boolean Formulas / Circuits I

- 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

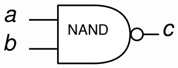
$$\mathcal{F}_c = (a \vee c) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$$



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



Representing Boolean Formulas / Circuits II

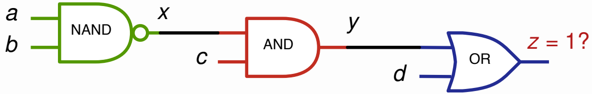


a	b	c	$\mathcal{F}_c$(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

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

Representing Boolean Formulas / Circuits III

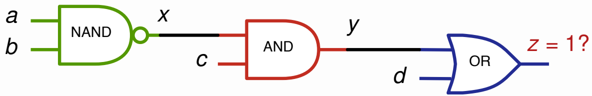
- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses



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

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



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

- 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 PB solver
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best solvers already encode to CNF
    - E.g. Minisat+, but also QMaxSat, MSUnCore, (W)PM2

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 PB solver
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best solvers already encode to CNF
    - E.g. Minisat+, but also QMaxSat, MSUnCore, (W)PM2

- 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 $v_0 \ldots v_{j-1} = b_0 \ldots b_{j-1}$, the binary encoding of $j - 1$

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

- 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 $v_0 \ldots v_{j-1} = b_0 \ldots b_{j-1}$, the binary encoding of $j - 1$

    $x_j \rightarrow (v_0 = b_0) \wedge \ldots \wedge (v_{j-1} = b_{j-1}) \Leftrightarrow (\bar{x}_j \vee (v_0 = b_0) \wedge \ldots \wedge (v_{j-1} = b_{j-1}))$
  - 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 $v_0 \ldots v_{j-1} = b_0 \ldots b_{j-1}$, the binary encoding of $j - 1$

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

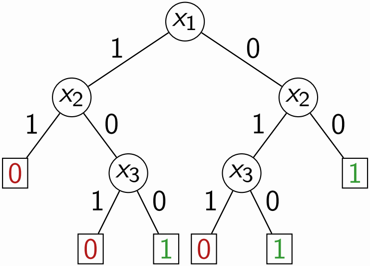
Outline

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

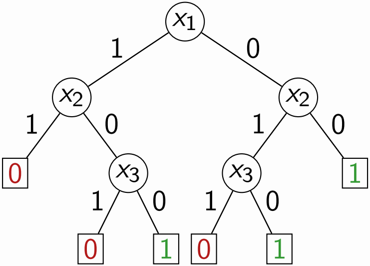
Encoding PB Constraints with BDDs I

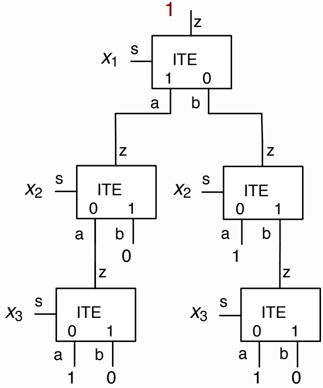
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD



Encoding PB Constraints with BDDs I

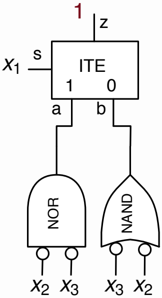
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD





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

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:

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

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:

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

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

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:

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

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

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:

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

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

Outline

CNF Encodings

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

SAT Embeddings

SAT Oracles

What Next in SAT-Based Problem Solving?

CSP Constraints

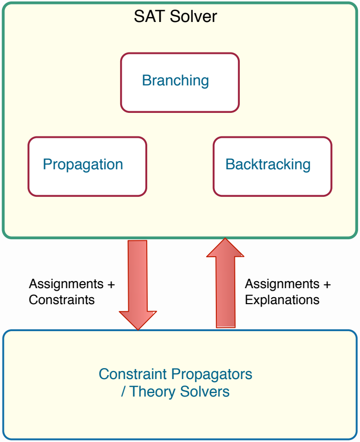
- Many possible encodings:
  - Direct encoding [dK89,GJ96,W00]
  - Log encoding [W00]
  - Support encoding [K90,G02]
  - Log-Support encoding [G07]
  - Order encoding for finite linear CSPs [TTKB09]

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 $(\bar{x}_{i,v_i} \vee \bar{x}_{j,v_j})$

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 (see later talks)
- 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

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
  - Non-incremental SAT:

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
  - Non-incremental SAT:

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
    - Add clause $(\bar{a}_i)$ to *delete* C_i
  - Non-incremental SAT:

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
    - Add clause $(\bar{a}_i)$ to *delete* C_i
    - **Note**: incremental SAT enables clause reuse
  - Non-incremental SAT:

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
    - Add clause $(\bar{a}_i)$ to *delete* C_i
    - **Note**: incremental SAT enables clause reuse
  - Non-incremental SAT:
    - Submit complete formula to SAT solver in each iteration
    - **Note**: difficult to instrument clause reuse

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
    - Add clause $(\bar{a}_i)$ to *delete* C_i
    - **Note**: incremental SAT enables clause reuse
  - Non-incremental SAT:
    - Submit complete formula to SAT solver in each iteration
    - **Note**: difficult to instrument clause reuse
- What does the SAT oracle compute/return?
  1. Yes/No: $(st) \leftarrow \text{SAT}(\mathcal{F})$

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
    - Add clause $(\bar{a}_i)$ to *delete* C_i
    - **Note**: incremental SAT enables clause reuse
  - Non-incremental SAT:
    - Submit complete formula to SAT solver in each iteration
    - **Note**: difficult to instrument clause reuse
- What does the SAT oracle compute/return?
  1. Yes/No: $(st) \leftarrow \text{SAT}(\mathcal{F})$
  2. Compute model: $(st, \mu) \leftarrow \text{SAT}(\mathcal{F})$

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \vee \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
    - Add clause $(\bar{a}_i)$ to *delete* C_i
    - **Note**: incremental SAT enables clause reuse
  - Non-incremental SAT:
    - Submit complete formula to SAT solver in each iteration
    - **Note**: difficult to instrument clause reuse
- What does the SAT oracle compute/return?
  1. Yes/No: $(st) \leftarrow \text{SAT}(\mathcal{F})$
  2. Compute model: $(st, \mu) \leftarrow \text{SAT}(\mathcal{F})$
  3. Compute unsatisfiable cores: $(st, \mu, \mathcal{U}) \leftarrow \text{SAT}(\mathcal{F})$

Practical Aspects of Using SAT Oracles

- Incremental vs. non-incremental SAT [ES03]
  - Incremental SAT:
    - Replace each clause (C_i) with $(C_i \lor \bar{a}_i)$, where a_i is assumption variable
    - When calling SAT solver, each assumption can be assigned 1, 0, or be left unassigned
    - $a_i = 1$ to activate clause C_i
    - $a_i = 0$ to deactivate clause C_i
    - Add clause $(\bar{a}_i)$ to *delete* C_i
    - **Note**: incremental SAT enables clause reuse
  - Non-incremental SAT:
    - Submit complete formula to SAT solver in each iteration
    - **Note**: difficult to instrument clause reuse
- What does the SAT oracle compute/return?
  1. Yes/No: $(st) \leftarrow \text{SAT}(\mathcal{F})$
  2. Compute model: $(st, \mu) \leftarrow \text{SAT}(\mathcal{F})$
  3. Compute unsatisfiable cores: $(st, \mu, \mathcal{U}) \leftarrow \text{SAT}(\mathcal{F})$
  4. Compute proof traces/resolution proof: $(st, \mu, \mathcal{T}) \leftarrow \text{SAT}(\mathcal{F})$

Outline

Defining MUSes

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Formula is unsatisfiable but not irreducible

Defining MUSes

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Formula is unsatisfiable but not irreducible
- Can remove clauses, and formula still unsatisfiable

Defining MUSes

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Formula is unsatisfiable but not irreducible
- Can remove clauses, and formula still unsatisfiable
- A Minimal Unsatisfiable Subformula (**MUS**) is an unsatisfiable and irreducible subformula

Defining MUSes

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Formula is unsatisfiable but not irreducible
- Can remove clauses, and formula still unsatisfiable
- A Minimal Unsatisfiable Subformula (**MUS**) is an unsatisfiable and irreducible subformula

Defining MUSes

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Formula is unsatisfiable but not irreducible
- Can remove clauses, and formula still unsatisfiable
- A Minimal Unsatisfiable Subformula (**MUS**) is an unsatisfiable and irreducible subformula

Defining MUSes

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Formula is unsatisfiable but not irreducible
- Can remove clauses, and formula still unsatisfiable
- A Minimal Unsatisfiable Subformula (**MUS**) is an unsatisfiable and irreducible subformula
- How to compute an MUS?

Deletion-Based MUS Extraction

```
Input   : Unsatisfiable CNF Formula 𝓕
Output: MUS 𝓜
begin
    𝓜 ← 𝓕                                   // MUS over-approximation
    foreach c ∈ 𝓜 do
        if not SAT(𝓜 \ {c}) then
            𝓜 ← 𝓜 \ {c}     // If UNSAT(𝓜 \ {c}), then c ∉ 𝓜
    return 𝓜                                 // Final 𝓜 is MUS
end
```

- Number of calls to SAT solver: $\mathcal{O}(|\mathcal{F}|)$

Deletion-Based MUS Extraction

```
Input  : Unsatisfiable CNF Formula 𝓕
Output: MUS 𝓜
begin
    𝓜 ← 𝓕                              // MUS over-approximation
    foreach c ∈ 𝓜 do
        if not SAT(𝓜 \ {c}) then
            𝓜 ← 𝓜 \ {c}                // Remove c from 𝓜
    return 𝓜                           // Final 𝓜 is MUS
end
```

- Number of calls to SAT solver: $\mathcal{O}(|\mathcal{F}|)$

An Example

$$(\neg x_1 \vee x_2)$$
$$(\neg x_3 \vee x_2)$$
$$(x_1 \vee x_2)$$
$$(\neg x_3)$$
$$(\neg x_2)$$

UNSAT instance

An Example

$(\neg x_1 \vee x_2)$
$(\neg x_3 \vee x_2)$
$(x_1 \vee x_2)$
$(\neg x_3)$
$(\neg x_2)$

Hide clause $(\neg x_1 \vee x_2)$

An Example

$(\neg x_1 \vee x_2)$
$(\neg x_3 \vee x_2)$
$(x_1 \vee x_2)$
$(\neg x_3)$
$(\neg x_2)$

SAT instance $\rightarrow$ keep clause $(\neg x_1 \vee x_2)$

An Example

$(\neg x_1 \vee x_2)$
$(\neg x_3 \vee x_2)$
$(x_1 \vee x_2)$
$(\neg x_3)$
$(\neg x_2)$

Hide clause $(\neg x_3 \vee x_2)$

An Example

$(\neg x_1 \vee x_2)$

$(x_1 \vee x_2)$

$(\neg x_3)$

$(\neg x_2)$

UNSAT instance $\rightarrow$ remove clause $(\neg x_3 \vee x_2)$

An Example

$(\neg x_1 \vee x_2)$

$(\neg x_3 \vee x_2)$

$(x_1 \vee x_2)$

$(\neg x_3)$

$(\neg x_2)$

Hide clause $(x_1 \vee x_2)$

An Example

$(\neg x_1 \vee x_2)$

$(\neg x_3 \vee x_2)$

$(x_1 \vee x_2)$

$(\neg x_3)$

$(\neg x_2)$

SAT instance → keep clause $(x_1 \vee x_2)$

An Example

$(\neg x_1 \vee x_2)$

$(\neg x_3 \vee x_2)$

$(x_1 \vee x_2)$

$(\neg x_3)$

$(\neg x_2)$

Hide clause $(\neg x_3)$

An Example

$$(\neg x_1 \vee x_2)$$

$$(x_1 \vee x_2)$$

$$(\neg x_2)$$

UNSAT instance $\rightarrow$ remove clause $(\neg x_3)$

An Example

$$(\neg x_1 \vee x_2)$$
$$(\neg x_3 \vee x_2)$$
$$(x_1 \vee x_2)$$
$$(\neg x_3)$$
$$(\neg x_2)$$

Hide clause $(\neg x_2)$

An Example

$$(\neg x_1 \vee x_2)$$

$$(x_1 \vee x_2)$$

SAT instance $\rightarrow$ keep clause $(\neg x_2)$

An Example

$$(\neg x_1 \vee x_2)$$
$$(x_1 \vee x_2)$$
$$(\neg x_2)$$

Computed MUS

More on MUS Extraction

Algorithm	# Oracle Calls	Reference
Insertion (Default)	$\mathcal{O}(m \times k)$	[SP88]
Deletion (Default)	$\mathcal{O}(m)$	[CD91,BDTW93]
QuickXplain	$\mathcal{O}(k \times (1 + \log \frac{m}{k}))$	[J01,J04]
Dichotomic	$\mathcal{O}(k \times \log m)$	[HLSB06]
Insertion with Relaxation Variables	$\mathcal{O}(m)$	[MSL11]
Deletion with Model Rotation	$\mathcal{O}(m)$	[BLMS12,MSL11]
Progression	$\mathcal{O}(k \times \log(1 + \frac{m}{k}))$	[MSJB13]

More on MUS Extraction

Algorithm	# Oracle Calls	Reference
Insertion (Default)	$\mathcal{O}(m \times k)$	[SP88]
Deletion (Default)	$\mathcal{O}(m)$	[CD91,BDTW93]
QuickXplain	$\mathcal{O}(k \times (1 + \log \frac{m}{k}))$	[J01,J04]
Dichotomic	$\mathcal{O}(k \times \log m)$	[HLSB06]
Insertion with Relaxation Variables	$\mathcal{O}(m)$	[MSL11]
Deletion with Model Rotation	$\mathcal{O}(m)$	[BLMS12,MSL11]
Progression	$\mathcal{O}(k \times \log(1 + \frac{m}{k}))$	[MSJB13]

- Additional Techniques:
  - Restrict formula to unsatisfiable subsets [BDTW93,HLSB06,DHN06,MSL11]
  - Check redundancy condition [vMW08,MSL11,BLMS12]
  - Model rotation, recursive model rotation, etc. [MSL11,BMS11,BLMS12,W12]

Outline

Defining Maximum Satisfiability

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable

Defining Maximum Satisfiability

$x_6 \lor x_2$	$\neg x_6 \lor x_2$	$\neg x_2 \lor x_1$	$\neg x_1$
$\neg x_6 \lor x_8$	$x_6 \lor \neg x_8$	$x_2 \lor x_4$	$\neg x_4 \lor x_5$
$x_7 \lor x_5$	$\neg x_7 \lor x_5$	$\neg x_5 \lor x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula

Defining Maximum Satisfiability

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest MCSes

MaxSAT Problem(s)

- MaxSAT:
  - All clauses are soft
  - Maximize number of satisfied soft clauses
  - Minimize number of unsatisfied soft clauses

MaxSAT Problem(s)

- MaxSAT:
  - All clauses are soft
  - Maximize number of satisfied soft clauses
  - Minimize number of unsatisfied soft clauses

- Partial MaxSAT:
  - Hard clauses must be satisfied
  - Minimize number of unsatisfied soft clauses

MaxSAT Problem(s)

- MaxSAT:
  - All clauses are soft
  - Maximize number of satisfied soft clauses
  - Minimize number of unsatisfied soft clauses
- Partial MaxSAT:
  - Hard clauses must be satisfied
  - Minimize number of unsatisfied soft clauses
- Weighted MaxSAT
  - Weights associated with (soft) clauses
  - Minimize sum of weights of unsatisfied clauses

MaxSAT Problem(s)

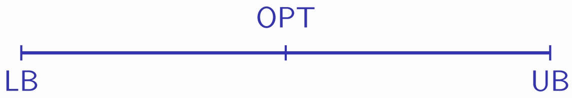
- MaxSAT:
  - All clauses are soft
  - Maximize number of satisfied soft clauses
  - Minimize number of unsatisfied soft clauses
- Partial MaxSAT:
  - Hard clauses must be satisfied
  - Minimize number of unsatisfied soft clauses
- Weighted MaxSAT
  - Weights associated with (soft) clauses
  - Minimize sum of weights of unsatisfied clauses
- Weighted Partial MaxSAT
  - Weights associated with soft clauses
  - Hard clauses must be satisfied
  - Minimize sum of weights of unsatisfied soft clauses

Definitions

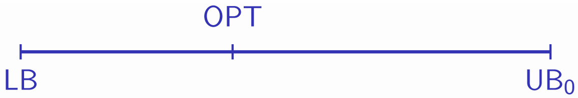
- Cost of assignment:
  - Sum of weights of unsatisfied clauses
- Optimum solution (OPT):
  - Assignment with minimum cost
- Upper Bound (UB):
  - Assignment with cost not less than OPT
  - E.g. $\sum_{c_i \in \varphi} w_i + 1$; hard clauses may be inconsistent
- Lower Bound (LB):
  - No assignment with cost no larger than LB
  - E.g. -1; it may be possible to satisfy all soft clauses

Definitions

- Cost of assignment:
  - Sum of weights of unsatisfied clauses
- Optimum solution (OPT):
  - Assignment with minimum cost
- Upper Bound (UB):
  - Assignment with cost not less than OPT
  - E.g. $\sum_{c_i \in \varphi} w_i + 1$; hard clauses may be inconsistent
- Lower Bound (LB):
  - No assignment with cost no larger than LB
  - E.g. -1; it may be possible to satisfy all soft clauses

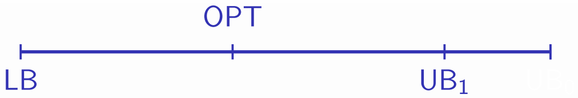


Iterative SAT Solving – Refine UB



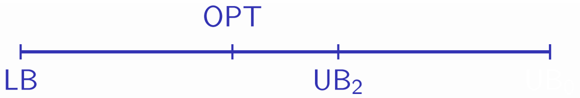
- Require $\sum w_i \, r_i \leq UB_0 - 1$

Iterative SAT Solving – Refine UB



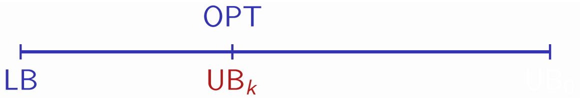
- Require $\sum w_i \, r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i \, r_i$

Iterative SAT Solving – Refine UB



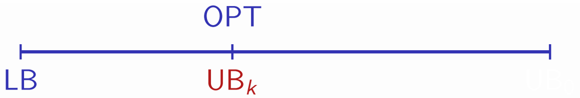
- Require $\sum w_i\, r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i\, r_i$

Iterative SAT Solving – Refine UB



- Require $\sum w_i\, r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i\, r_i$
- Repeat until constraint $\sum w_i\, r_i \leq UB_k - 1$ becomes UNSAT
  - UB_k denotes the optimum value

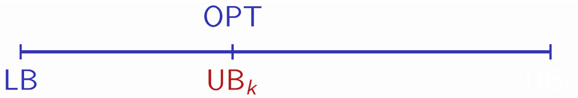
Iterative SAT Solving – Refine UB



- Require $\sum w_i\, r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i\, r_i$
- Repeat until constraint $\sum w_i\, r_i \leq UB_k - 1$ becomes UNSAT
  - UB_k denotes the optimum value

- Worst-case # of iterations **exponential** on instance size

Iterative SAT Solving – Refine UB



- Require $\sum w_i\, r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i\, r_i$
- Repeat until constraint $\sum w_i\, r_i \leq UB_k - 1$ becomes UNSAT
  - UB_k denotes the optimum value

- Worst-case # of iterations **exponential** on instance size

- Example tools:
  - Minisat+: CNF encoding of constraints [ES06]
  - SAT4J: native handling of constraints [LBP10]
  - QMaxSat: CNF encoding of constraints [KZFH12]
  - ...

Fu&Malik's Core-Guided Approach

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

Example CNF formula

Fu&Malik's Core-Guided Approach

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

Formula is UNSAT; $\text{OPT} \leq |\varphi| - 1$; Get unsat core

Fu&Malik's Core-Guided Approach

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1 \vee r_1$	$\neg x_1 \vee r_2$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3 \vee r_5$	$\neg x_3 \vee r_6$

$\sum_{i=1}^{6} r_i \leq 1$

Add relaxation variables and AtMost1 constraint

Fu&Malik's Core-Guided Approach

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1 \vee r_1$	$\neg x_1 \vee r_2$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3 \vee r_5$	$\neg x_3 \vee r_6$

$\sum_{i=1}^{6} r_i \leq 1$

Formula is (again) UNSAT; OPT $\leq |\varphi| - 2$; Get unsat core

Fu&Malik's Core-Guided Approach

$x_6 \vee x_2 \vee r_7$ $\quad \neg x_6 \vee x_2 \vee r_8$ $\quad \neg x_2 \vee x_1 \vee r_1 \vee r_9$ $\quad \neg x_1 \vee r_2 \vee r_{10}$

$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_{11}$ $\quad \neg x_7 \vee x_5 \vee r_{12}$ $\quad \neg x_5 \vee x_3 \vee r_5 \vee r_{13}$ $\quad \neg x_3 \vee r_6 \vee r_{14}$

$\sum_{i=1}^{6} r_i \leq 1$ $\quad \sum_{i=7}^{14} r_i \leq 1$

Add new relaxation variables and AtMost1 constraint

Fu&Malik's Core-Guided Approach

$x_6 \vee x_2 \vee r_7$	$\neg x_6 \vee x_2 \vee r_8$	$\neg x_2 \vee x_1 \vee r_1 \vee r_9$	$\neg x_1 \vee r_2 \vee r_{10}$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5 \vee r_{11}$	$\neg x_7 \vee x_5 \vee r_{12}$	$\neg x_5 \vee x_3 \vee r_5 \vee r_{13}$	$\neg x_3 \vee r_6 \vee r_{14}$
$\sum_{i=1}^{6} r_i \leq 1$	$\sum_{i=7}^{14} r_i \leq 1$		

Instance is now SAT

Fu&Malik's Core-Guided Approach

$x_6 \vee x_2 \vee r_7$	$\neg x_6 \vee x_2 \vee r_8$	$\neg x_2 \vee x_1 \vee r_1 \vee r_9$	$\neg x_1 \vee r_2 \vee r_{10}$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4 \vee r_3$	$\neg x_4 \vee x_5 \vee r_4$
$x_7 \vee x_5 \vee r_{11}$	$\neg x_7 \vee x_5 \vee r_{12}$	$\neg x_5 \vee x_3 \vee r_5 \vee r_{13}$	$\neg x_3 \vee r_6 \vee r_{14}$

$\sum_{i=1}^{6} r_i \leq 1$ $\quad$ $\sum_{i=7}^{14} r_i \leq 1$

MaxSAT solution is $|\varphi| - \mathcal{I} = 12 - 2 = 10$

Organization of Fu&Malik's Algorithm

- Clauses characterized as:
  - Soft: initial set of soft clauses
  - Hard: initially hard, or added during execution of algorithm
    - E.g. clauses from AtMost1 constraints

- While exist unsatisfiable cores [FM06]
  - Add fresh set B of relaxation variables to soft clauses in core
  - Add new AtMost1 constraint

$$\sum_{b_i \in B} b_i \leq 1$$

    - At most 1 relaxation variable from set B can take value 1

- (Partial) MaxSAT solution is $|\varphi| - \mathcal{I}$
  - $\mathcal{I}$: number of iterations ($\equiv$ number of computed unsat cores)

Organization of Fu&Malik's Algorithm

- Clauses characterized as:
  - Soft: initial set of soft clauses
  - Hard: initially hard, or added during execution of algorithm
    - E.g. clauses from AtMost1 constraints
- While exist unsatisfiable cores [FM06]
  - Add fresh set B of relaxation variables to soft clauses in core
  - Add new AtMost1 constraint

$$\sum_{b_i \in B} b_i \leq 1$$

    - At most 1 relaxation variable from set B can take value 1
- (Partial) MaxSAT solution is $|\varphi| - \mathcal{I}$
  - $\mathcal{I}$: number of iterations ($\equiv$ number of computed unsat cores)
- Can be adapted for weighted MaxSAT [ABL09a,MMSP09]

Oracle-Based MaxSAT Solving I

- Iterative: [MHLPMS13]
  - Linear search SAT/UNSAT (refine UB) [e.g. LBP10]
  - Linear search UNSAT/SAT (refine LB)
  - Binary search [e.g. FM06]
  - Bit-based
  - Mixed linear/binary search [e.g. KZFH12]
- Core-Guided: [MHLPMS13,ABL13]
  - FM/(W)MSU1.X/WPM1 [FM06,MSM08,MMSP09,ABL09a,ABGL12]
  - (W)MSU3 [MSP07]
  - (W)MSU4 [MSP08]
  - (W)PM2 [ABL09a,ABL09b,ABL10,ABGL13]
  - Core-guided binary search (w/ disjoint cores) [HMMS11,MHMS12]
    - Bin-Core, Bin-Core-Dis, Bin-Core-Dis2
- Iterative subsetting [DB11,DB13a,DB13b]

Oracle MaxSAT Solving II

- A sample of recent algorithms:

Algorithm	# Oracle Calls	Reference
Linear search SU	Exponential	[e.g. LP10]
Binary search	Linear	[e.g. FM06]
WMSU1/WPM1	Exponential*	[FM06,MSM08,MMSP09,ABL09a,ABGL12]
WPM2	Exponential*	[ABL10,ABGL13]
Bin-Core-Dis	Linear	[HMMS11,MHMS12]
Iterative subsetting	Exponential	[DB11,DB13a,DB13b]

* Weighted case; depends on computed cores

- Example MaxSAT solvers:
  - MSUnCore; WPM1, WPM2; QMaxSAT; SAT4J; etc.

Outline

Problem Statement

[GMN09]

Given: $\exists X \forall Y . \phi$, where ϕ is a propositional formula

Question: Is there an assignment τ to X such that $\forall Y . \phi[X/\tau]$?

Problem Statement

[GMN09]

Given: $\exists X \forall Y.\, \phi$, where ϕ is a propositional formula
Question: Is there an assignment τ to X such that $\forall Y.\, \phi[X/\tau]$?

Example

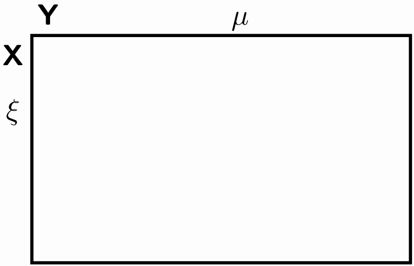
$$\exists x_1, x_2\ \forall y_1, y_2.\, (x_1 \rightarrow y_1) \wedge (x_2 \rightarrow y_2)$$

solution: $x_1 = 0, x_2 = 0$

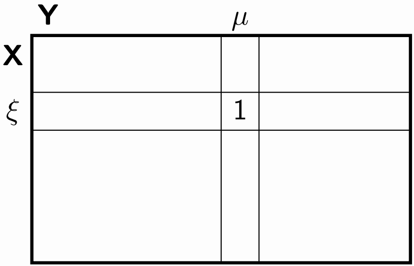
Motivation

- Σ_2^P complete
- interesting problems in this class, e.g. certain nonmonotonic reasoning, aspects of model checking, conformant planning
- separate track at QBF Eval

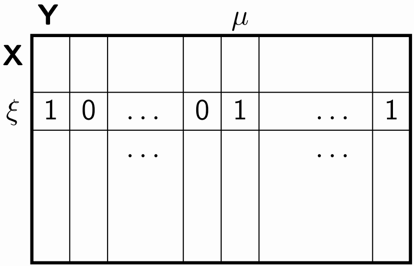
Looking at Assignments



Looking at Assignments



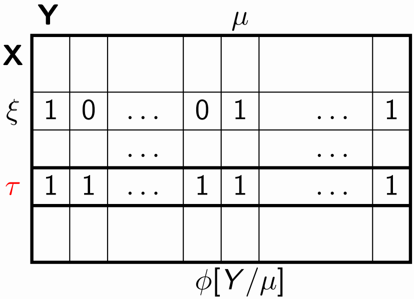
Looking at Assignments



Looking at Assignments

	Y				μ		
X							
ξ	1	0	$\dots$	0	1	$\dots$	1
			$\dots$			$\dots$	
τ	1	1	$\dots$	1	1	$\dots$	1

Looking at Assignments



Expanding $\exists X \forall Y.\, \phi$ into SAT

$$\exists X \forall Y.\, \phi \;\longrightarrow\; \mathsf{SAT}\left(\bigwedge_{\mu \in \mathcal{B}^{|Y|}} \phi[Y/\mu] \right)$$

Expanding $\exists X \forall Y.\, \phi$ into SAT

$$\exists X \forall Y.\, \phi \;\longrightarrow\; \mathsf{SAT}\left(\bigwedge_{\mu \in \mathcal{B}^{|Y|}} \phi[Y/\mu] \right)$$

Example

$$\exists x_1, x_2 \forall y_1, y_2.\, (x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$$

Expansion:

$$\begin{array}{ll} & (x_1 \leftrightarrow 0) \wedge (x_2 \leftrightarrow 0) \wedge (\bar{x}_1 \vee \bar{x}_2) \\ \wedge & (x_1 \leftrightarrow 0) \wedge (x_2 \leftrightarrow 1) \wedge (\bar{x}_1 \vee \bar{x}_2) \\ \wedge & (x_1 \leftrightarrow 1) \wedge (x_2 \leftrightarrow 0) \wedge (\bar{x}_1 \vee \bar{x}_2) \\ \wedge & (x_1 \leftrightarrow 1) \wedge (x_2 \leftrightarrow 1) \wedge (\bar{x}_1 \vee \bar{x}_2) \end{array}$$

Expanding $\exists X \forall Y.\, \phi$ into SAT

$$\exists X \forall Y.\, \phi \;\longrightarrow\; \mathsf{SAT}\left(\bigwedge_{\mu \in \mathcal{B}^{|Y|}} \phi[Y/\mu]\right)$$

Example

$$\exists x_1, x_2 \forall y_1, y_2.\, (x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$$

Expansion:

$$\begin{array}{ll} & (x_1 \leftrightarrow 0) \wedge (x_2 \leftrightarrow 0) \wedge (\bar{x}_1 \vee \bar{x}_2) \\ \wedge & (x_1 \leftrightarrow 0) \wedge (x_2 \leftrightarrow 1) \wedge (\bar{x}_1 \vee \bar{x}_2) \\ \wedge & (x_1 \leftrightarrow 1) \wedge (x_2 \leftrightarrow 0) \wedge (\bar{x}_1 \vee \bar{x}_2) \\ \wedge & (x_1 \leftrightarrow 1) \wedge (x_2 \leftrightarrow 1) \wedge (\bar{x}_1 \vee \bar{x}_2) \end{array}$$

Abstraction of $\exists X \forall Y . \phi$

- Consider only some set of assignments $\omega \subseteq \mathcal{B}^{|Y|}$

$$\bigwedge_{\mu \in \omega} \phi[Y/\mu]$$

Abstraction of $\exists X \forall Y . \phi$

- Consider only some set of assignments $\omega \subseteq \mathcal{B}^{|Y|}$

$$\bigwedge_{\mu \in \omega} \phi[Y/\mu]$$

- If a solution to the problem is a solution to the abstraction

$$\bigwedge_{\mu \in \mathcal{B}^{|Y|}} \phi[Y/\mu] \Rightarrow \bigwedge_{\mu \in \omega} \phi[Y/\mu]$$

Abstraction of $\exists X \forall Y . \phi$

- Consider only some set of assignments $\omega \subseteq \mathcal{B}^{|Y|}$

$$\bigwedge_{\mu \in \omega} \phi[Y/\mu]$$

- If a solution to the problem is a solution to the abstraction

$$\bigwedge_{\mu \in \mathcal{B}^{|Y|}} \phi[Y/\mu] \;\Rightarrow\; \bigwedge_{\mu \in \omega} \phi[Y/\mu]$$

- But not the other way around, a solution to an abstraction is not necessarily a solution to the original problem.

CEGAR Loop

input : $\exists X \forall Y . \phi$
output: (true, τ) if there exists τ s.t. $\forall Y . \phi[X/\tau]$,
(false, –) otherwise

```
ω ← ∅;
while true do
    (outc₁, τ) ← SAT(⋀_{μ∈ω} φ[Y/μ]);          // find a candidate
    if outc₁ = false then
        return (false,–);                        // no candidate found
    end
    if "τ is a solution";                        // solution check
    then
        return (true, τ)
    else
        "Grow ω";                                // refinement
    end
end
```

CEGAR Loop

input : $\exists X \forall Y . \phi$
output: (true, τ) if there exists τ s.t. $\forall Y . \phi[X/\tau]$,
(false, –) otherwise

```
ω ← ∅;
while true do
    (outc₁, τ) ← SAT(⋀_{μ∈ω} φ[Y/μ]);          // find a candidate
    if outc₁ = false then
        return (false,–);                       // no candidate found
    end
    if "τ is a solution";                       // solution check
    then
        return (true, τ)
    else
        "Grow ω";                               // refinement
    end
end
```

Testing for Solution

A value τ is a solution to $\exists X \forall Y . \phi$ iff

$$\forall Y . \phi[X/\tau] \quad \textit{iff} \quad \text{UNSAT}(\neg\phi[X/\tau])$$

Testing for Solution

A value τ is a solution to $\exists X \forall Y.\, \phi$ iff

$$\forall Y.\, \phi[X/\tau] \quad \textit{iff} \quad \mathsf{UNSAT}(\neg\phi[X/\tau])$$

If $\mathsf{SAT}(\neg\phi[X/\tau])$ by some μ, then μ is a counterexample to τ

Testing for Solution

A value τ is a solution to $\exists X \forall Y . \phi$ iff

$$\forall Y . \phi[X/\tau] \;\; \textit{iff} \;\; \text{UNSAT}(\neg\phi[X/\tau])$$

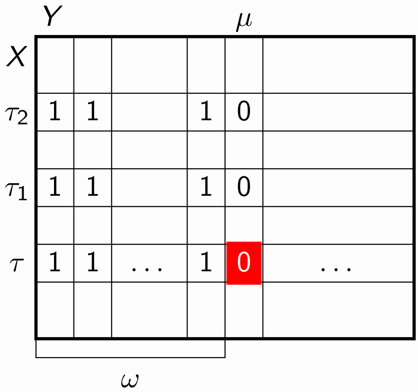
If SAT($\neg\phi[X/\tau]$) by some μ, then μ is a counterexample to τ

Example

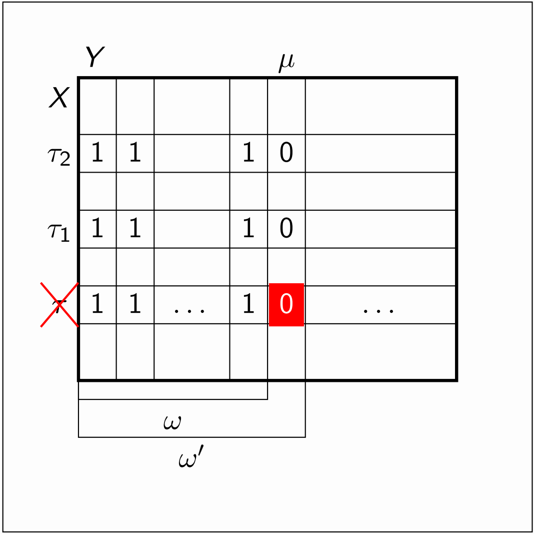
$\exists x_1, x_2 \; \forall y_1, y_2 . \; (x_1 \rightarrow y_1) \wedge (x_2 \rightarrow y_2)$

- candidate: $x_1 = 1, x_2 = 1$
- counterexamples: $y_1 = 0, y_2 = 0$
  $y_1 = 0, y_2 = 1$
  $y_1 = 1, y_2 = 0$

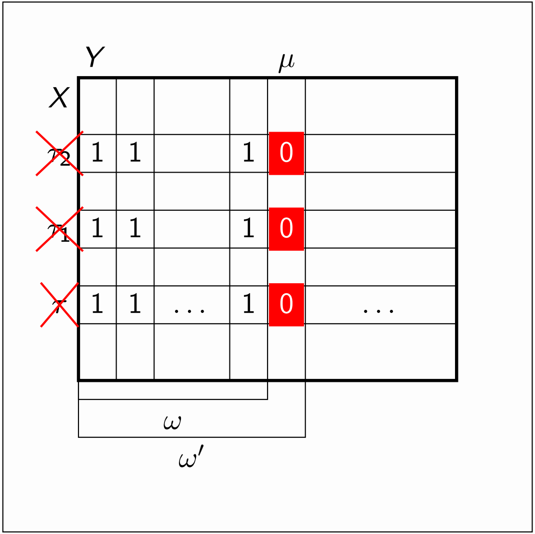
Refinement



Refinement



Refinement



AReQS (Abstraction Refinement-based QBF Solver)

```
input  : ∃X∀Y. φ
output: (true, τ) if there exists τ s.t. ∀Y. φ[X/τ],
        (false, –) otherwise

ω ← ∅;                                  // start with the empty expansion
while true do
    (outc₁, τ) ← SAT(⋀_{μ∈ω} φ[Y/μ]);              // find a candidate
    if outc₁ = false then
        return (false,–);                          // no candidate found
    end
    (outc₂, μ) ← SAT(¬φ[X/τ]);               // find a counterexample
    if outc₂ = false then
        return (true, τ) ;                   // candidate is a solution
    end
    ω ← ω ∪ {μ} ;                                         // refine
end
```

AReQS — Conclusions

- … is a CEGAR-based algorithm for 2QBF [JMS11]

AReQS — Conclusions

- … is a CEGAR-based algorithm for 2QBF [JMS11]
- … uses SAT solver as an oracle

AReQS — Conclusions

- … is a CEGAR-based algorithm for 2QBF [JMS11]
- … uses SAT solver as an oracle
- … gradually expands given 2QBF into a SAT formula

AReQS — Conclusions

- ... is a CEGAR-based algorithm for 2QBF [JMS11]
- ... uses SAT solver as an oracle
- ... gradually expands given 2QBF into a SAT formula
- Can be extended to arbitrary number of levels by recursion (RAReQS) [JKMSC12]

Outline

CNF Encodings

SAT Embeddings

SAT Oracles

What Next in SAT-Based Problem Solving?

SAT-Based Problem Solving – A Glimpse of the Future

- Remarkable (and increasing) number of applications of SAT
- Can use SAT for solving problems in different complexity classes
  - $\text{FP}^{\text{NP}}[\log n]$, FP^{NP}, ...
  - E.g. tackling problems in the polynomial hierarchy
- Many new recent algorithms for concrete problems
  - MaxSAT
  - MUSes
  - MCSes
  - Enumeration problems
  - ...
- Better encodings?
- White-box vs. black-box approaches?
  - But use of oracles inevitable in many cases

Thank You

References – DPLL & CDCL SAT Solvers I

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

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

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

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

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

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

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

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

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

References – DPLL & CDCL SAT Solvers II

GN02 E. Goldberg, Y. Novikov: BerkMin: A Fast and Robust Sat-Solver. DATE 2002: 142-149

ES03 N. Een, Niklas Sorensson: An Extensible SAT-solver. SAT 2003: 502-518

PD07 K. Pipatsrisawat, A. Darwiche: A Lightweight Component Caching Scheme for Satisfiability Solvers. SAT 2007: 294-299

H07 J. Huang: The Effect of Restarts on the Efficiency of Clause Learning. IJCAI 2007: 2318-2323

ABHJS08 G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, L. Sais: A Generalized Framework for Conflict Analysis. SAT 2008: 21-27

B08 A. Biere: PicoSAT Essentials. JSAT 4(2-4): 75-97 (2008)

SB09 N. Sorensson, A. Biere: Minimizing Learned Clauses. SAT 2009: 237-243

VG09 A. Van Gelder: Improved Conflict-Clause Minimization Leads to Improved Propositional Proof Traces. SAT 2009: 141-146

AS09 G. Audemard, L. Simon: Predicting Learnt Clauses Quality in Modern SAT Solvers. IJCAI 2009: 399-404

SSS12 A. Sabharwal, H. Samulowitz, M. Sellmann: Learning Back-Clauses in SAT. SAT 2012: 498-499

References – CNF Encodings I

T68	G. Tseitin: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Studies in Constructive Mathematics and Mathematical Logic, pp. 115-125 (1970)
B68	K. Batcher: Sorting Networks and Their Applications. AFIPS Spring Joint Computing Conference 1968: 307-314
PG86	David A. Plaisted, Steven Greenbaum: A Structure-Preserving Clause Form Translation. J. Symb. Comput. 2(3): 293-304 (1986)
dK89	Johan de Kleer: A Comparison of ATMS and CSP Techniques. IJCAI 1989: 290-296
GJ96	R. Genisson, P. Jegou: Davis and Putnam were Already Checking Forward. ECAI 1996: 180-184
W98	J. Warners: A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. Inf. Process. Lett. 68(2): 63-69 (1998)
W00	T. Walsh: SAT v CSP. CP 2000: 441-456

References – CNF Encodings II

FP01	A. Frisch, T. Peugniez: Solving Non-Boolean Satisfiability Problems with Stochastic Local Search. IJCAI 2001: 282-290
FS02	T. Fahle, M. Sellmann: Cost Based Filtering for the Constrained Knapsack Problem. Annals OR 115(1-4): 73-93 (2002)
S03	M. Sellmann: Approximated Consistency for Knapsack Constraints. CP 2003: 679-693
F03	M. Trick: A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. Annals OR 118(1-4): 73-84 (2003)
S05	C. Sinz: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. CP 2005: 827-831
ES06	N. Een, N. Sorensson: Translating Pseudo-Boolean Constraints into SAT. JSAT 2(1-4): 1-26 (2006)
G07	M. Gavanelli: The Log-Support Encoding of CSP into SAT. CP 2007: 815-822
P07	S. Prestwich: Variable Dependency in Local Search: Prevention Is Better Than Cure. SAT 2007: 107-120

References – CNF Encodings III

ANORC09 R. Asin, R. Nieuwenhuis, A. Oliveras, E. RodrÃguez-Carbonell: Cardinality Networks and Their Applications. SAT 2009: 167-180

BBR09 O. Bailleux, Y. Boufkhad, O. Roussel: New Encodings of Pseudo-Boolean Constraints into CNF. SAT 2009: 181-194

TTKB09 Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, Mutsunori Banbara: Compiling finite linear CSP into SAT. Constraints 14(2): 254-272 (2009)

CZI10 M. Codish, M. Zazon-Ivry: Pairwise Cardinality Networks. LPAR (Dakar) 2010: 154-172

ANORC11a R. Asin, R. Nieuwenhuis, A. Oliveras, E. RodrÃguez-Carbonell: Cardinality Networks: a theoretical and empirical study. Constraints 16(2): 195-221 (2011)

ANORC11b I. Abio, R. Nieuwenhuis, A. Oliveras, E. Rodriguez-Carbonell: BDDs for Pseudo-Boolean Constraints - Revisited. SAT 2011

References – Embedding SAT Solvers, Iterative MaxSAT & PBO

MMS06	V. Manquinho, J. Marques-Silva: On Using Cutting Planes in Pseudo-Boolean Optimization. JSAT 2(1-4): 209-219 (2006)
NOT06	R. Nieuwenhuis, A. Oliveras, C. Tinelli: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM 53(6): 937-977 (2006)
S07	R. Sebastiani: Lazy Satisability Modulo Theories. JSAT 3(3-4): 141-224 (2007)
BSST09	C. Barrett, R. Sebastiani, S. Seshia, C. Tinelli: Satisfiability Modulo Theories. Handbook of Satisfiability 2009: 825-885
LBP10	D. Le Berre, A. Parrain: The Sat4j library, release 2.2. JSAT 7(2-3): 59-6 (2010)
KZFH12	M. Koshimura, T. Zhang, H. Fujita, R. Hasegawa: QMaxSAT: A Partial Max-SAT Solver. JSAT 8(1/2): 95-100 (2012)

References – MUSes I

SP88 J. de Siqueira, J.-F. Puget: Explanation-Based Generalisation of Failures. ECAI 1988: 339-344

CD91 J. Chinneck, E. Dravnieks: Locating Minimal Infeasible Constraint Sets in Linear Programs. INFORMS Journal on Computing 3(2): 157-168 (1991)

BDTW93 R. R. Bakker, F. Dikker, F. Tempelman, P. M. Wognum: Diagnosing and Solving Over-Determined Constraint Satisfaction Problems. IJCAI 1993: 276-281

J01 U. Junker: QUICKXPLAIN: Conflict Detection for Arbitrary Constraint Propagation Algorithms, WMSPC IJCAI 2001

J04 U. Junker: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. AAAI 2004: 167-172

HLSB06 F. Hemery, C. Lecoutre, L. Sais, F. Boussemart: Extracting MUCs from Constraint Networks. ECAI 2006: 113-117

CMP07 E. Gregoire, B. Mazure, C. Piette: Local-search Extraction of MUSes. Constraints 12(3): 325-344 (2007)

vMW08 H. van Maaren, S. Wieringa: Finding Guaranteed MUSes Fast. SAT 2008: 291-304

References – MUSes II

DGHP09 C. Desrosiers, P. Galinier, A. Hertz, S. Paroz: Using heuristics to find minimal unsatisfiable subformulas in satisfiability problems. J. Comb. Optim. 18(2): 124-150 (2009)

MS10 J. Marques-Silva: Minimal Unsatisfiability: Models, Algorithms and Applications. ISMVL 2010: 9-14

MSL11 J. Marques-Silva, I. Lynce: On Improving MUS Extraction Algorithms. SAT 2011: 159-173

BMS11 A. Belov, J. Marques-Silva: Accelerating MUS extraction with recursive model rotation. FMCAD 2011: 37-40

BLMS12 A. Belov, I. Lynce, J. Marques-Silva: Towards Efficient MUS Extraction. AI Communications, 25(2): 97-116 (2012)

BMS12 A. Belov, J. Marques-Silva: MUSer2: An Efficient MUS Extractor, System Description. Journal on Satisfiability, Boolean Modeling and Computation, 8: 123-128 (2012)

W12 Siert Wieringa: Understanding, Improving and Parallelizing MUS Finding Using Model Rotation. CP 2012: 672-687

MSJB13 J. Marques-Silva, M. Janota, A. Belov: Minimal Sets over Monotone Predicates in Boolean Formulae. CAV 2013

References – Core-Guided MaxSAT I

FM06	Z. Fu, S. Malik: On Solving the Partial MAX-SAT Problem. SAT 2006: 252-265
MSP07	J. Marques-Silva, J. Planes: On Using Unsatisfiability for Solving Maximum Satisfiability CoRR abs/0712.1097: (2007)
MSP08	J. Marques-Silva, Jordi Planes: Algorithms for Maximum Satisfiability using Unsatisfiable Cores. DATE 2008: 408-413
MSM08	J. Marques-Silva, V. Manquinho: Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms. SAT 2008: 225-230
MMSP09	V. Manquinho, J. Marques Silva, J. Planes: Algorithms for Weighted Boolean Optimization. SAT 2009: 495-508
ABL09a	C. Ansotegui, M. Bonet, J. Levy: Solving (Weighted) Partial MaxSAT through Satisfiability Testing. SAT 2009: 427-440
ABL09b	C. Ansotegui, M. L. Bonet, J. Levy: On Solving MaxSAT Through SAT. CCIA 2009: 284-292
ABL10	C. Ansotegui, M. Bonet, J. Levy: A New Algorithm for Weighted Partial MaxSAT. AAAI 2010
HMMS11	F. Heras, A. Morgado, J. Marques-Silva: Core-Guided Binary Search Algorithms for Maximum Satisfiability. AAAI 2011.

References – Core-Guided MaxSAT

DB11 J. Davies, F. Bacchus: Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. CP 2011: 225-239

MHMS12 A. Morgado, F. Heras, J. Marques-Silva: Improvements to Core-Guided Binary Search for MaxSAT. SAT 2012.

ABGL12 C. Ansotegui, M. Bonet, J. Gabas, J. Levy: Improving SAT-Based Weighted MaxSAT Solvers. CP 2012: 86-101

DB13a J. Davies, F. Bacchus: Exploiting the Power of MIP Solvers in MaxSAT. SAT 2013: 166-181

ABL13 C. Ansotegui, M. Bonet, J. Levy: SAT-based MaxSAT algorithms. Artif. Intell. 196: 77-105 (2013

ABGL13 C. Ansotegui, M. Bonet, J. Gabas and J. Levy: Improving WPM2 for (Weighted) Partial MaxSAT. CP 2013

DB13b J. Davies and F. Bacchus: Postponing Optimization to Speed Up MaxSAT Solving. CP 2013

MHLPMS13 A. Morgado, F. Heras, M. Liffiton, J. Planes, J. Marques-Silva: Iterative and Core-Guided MaxSAT Solving: A Survey and Assessment. Constraints: An International Journal. In Press (2013)

References – 2QBF & QBF

GMN09 E. Giunchiglia, P. Marin, M. Narizzano: Reasoning with Quantified Boolean Formulas. Handbook of Satisfiability 2009: 761-780

JMS11 M. Janota, J. Marques-Silva: Abstraction-Based Algorithm for 2QBF. SAT 2011: 230-244

JKMSC12 M. Janota, W. Klieber, J. Marques-Silva, E. Clarke: Solving QBF with Counterexample Guided Refinement. SAT 2012: 114-128

KJMSC13 W. Klieber, M. Janota, J. Marques-Silva, E. Clarke: Solving QBF with Free Variables. CP 2013

References – Additional References

R65	J. Robinson: A Machine-Oriented Logic Based on the Resolution Principle. J. ACM 12(1): 23-41 (1965)
C71	S. Cook: The Complexity of Theorem-Proving Procedures. STOC 1971: 151-158
ZM03	L. Zhang, S. Malik: Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. DATE 2003: 10880-10885
SP04	S. Subbarayan, D. Pradhan: NiVER: Non-increasing Variable Elimination Resolution for Preprocessing SAT Instances. SAT 2004: 276-291
EB05	N. Een, A. Biere: Effective Preprocessing in SAT Through Variable and Clause Elimination. SAT 2005: 61-75
HJB11	M. Heule, M. Jarvisalo, A. Biere: Efficient CNF Simplification Based on Binary Implication Graphs. SAT 2011: 201-215
JHB12	M. Jarvisalo, M. Heule, A. Biere: Inprocessing Rules. IJCAR 2012: 355-370
IJMS13	A. Ignatiev, M. Janota, J. Marques-Silva: Quantified Maximum Satisfiability: - A Core-Guided Approach. SAT 2013: 250-266
LB13	J.-M. Lagniez, A. Biere: Factoring Out Assumptions to Speed Up MUS Extraction. SAT 2013: 276-292