



# Constraint Solving

René Thiemann      and      Fabian Mitterwallner

based on a previous course by Aart Middeldorp

# Outline

- 1. Summary of Previous Lecture**
- 2. Example SAT Reduction: Shakashaka**
- 3. Maximal Satisfiability**
- 4. Incremental SAT Solving**
- 5. Further Reading**

## SAT Solving

- conflict graph is used to compute backjump clauses
- two new inference rules: learn (a backjump clause) and restart

## Theorem (Cook-Levin)

SAT is **NP-complete**

## SAT Variation

$k$ -SAT: every clause has (at most)  $k$  literals

## Theorem

- 3SAT is NP-complete
- 2SAT is solvable in polynomial time

## Planar 3SAT

instance is 3SAT formula  $\varphi$  whose **incidence graph** is planar

- $\varphi$  with clauses  $\mathcal{C} = \{C_1, \dots, C_m\}$  over variables  $\mathcal{V} = \{x_1, \dots, x_n\}$
- bipartite graph  $(\mathcal{C} \cup \mathcal{V}, \mathcal{E})$  with  $\mathcal{E}$  containing edge  $C_i - x_j$  if and only if  $C_i$  contains  $x_j$  or  $\neg x_j$

## Theorem (Lichtenstein 1982)

planar 3SAT is NP-complete

## Remark

planar 3SAT is often used in reductions to show NP-hardness of particular problems

# Outline

1. Summary of Previous Lecture
- 2. Example SAT Reduction: Shakashaka**
3. Maximal Satisfiability
4. Incremental SAT Solving
5. Further Reading

# Shakashaka

|   |   |   |   |   |   |  |   |
|---|---|---|---|---|---|--|---|
| 2 |   |   |   | 1 | 0 |  |   |
|   |   |   |   |   | 2 |  |   |
|   |   |   |   |   |   |  | 1 |
| 1 | 1 | 2 |   |   |   |  | 2 |
|   |   |   |   |   |   |  |   |
|   |   |   |   | 4 |   |  |   |
|   |   |   |   |   |   |  |   |
| 2 |   |   | 2 |   |   |  | 0 |

# Shakashaka

|   |   |   |   |   |   |  |   |
|---|---|---|---|---|---|--|---|
| 2 |   |   |   | 1 | 0 |  |   |
|   |   |   |   |   | 2 |  |   |
|   |   |   |   |   |   |  | 1 |
| 1 | 1 | 2 |   |   |   |  | 2 |
|   |   |   |   |   |   |  |   |
|   |   |   |   | 4 |   |  |   |
|   |   |   |   |   |   |  |   |
| 2 |   |   | 2 |   |   |  | 0 |

- fill grid with triangles  to obtain white rectangles

# Shakashaka

|   |   |   |   |   |   |  |   |
|---|---|---|---|---|---|--|---|
| 2 |   |   |   | 1 | 0 |  |   |
|   |   |   |   |   | 2 |  |   |
|   |   |   |   |   |   |  | 1 |
| 1 | 1 | 2 |   |   |   |  | 2 |
|   |   |   |   |   |   |  |   |
|   |   |   |   | 4 |   |  |   |
|   |   |   |   |   |   |  |   |
| 2 |   |   | 2 |   |   |  | 0 |

- fill grid with triangles  to obtain white rectangles
- numbered cells must have right number of neighbouring triangles



# Shakashaka



- fill grid with triangles  to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

# Shakashaka



- fill grid with triangles to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

# Shakashaka



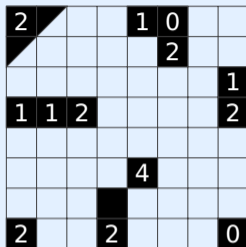
- fill grid with triangles to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

# Shakashaka



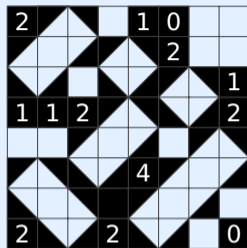
- fill grid with triangles  to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

# Shakashaka



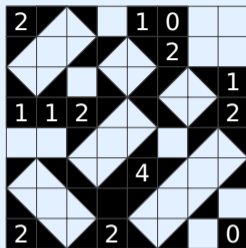
- fill grid with triangles  to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

# Shakashaka



- fill grid with triangles  to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

# Shakashaka

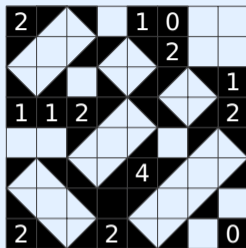


- fill grid with triangles  to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

## Theorem

Shakashaka is NP-hard

## Shakashaka



- fill grid with triangles  to obtain white rectangles
- numbered cells must have right number of neighbouring triangles

## Theorem

Shakashaka is NP-hard

## Proof

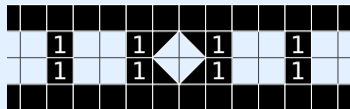
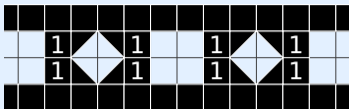
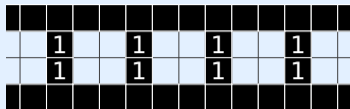
reduction from planar 3SAT



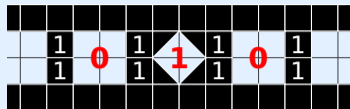
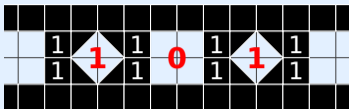
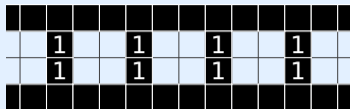
# Wires

|  |   |  |  |   |  |  |   |  |  |   |  |  |  |  |
|--|---|--|--|---|--|--|---|--|--|---|--|--|--|--|
|  |   |  |  |   |  |  |   |  |  |   |  |  |  |  |
|  | 1 |  |  | 1 |  |  | 1 |  |  | 1 |  |  |  |  |
|  | 1 |  |  | 1 |  |  | 1 |  |  | 1 |  |  |  |  |
|  |   |  |  |   |  |  |   |  |  |   |  |  |  |  |

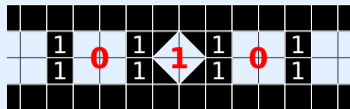
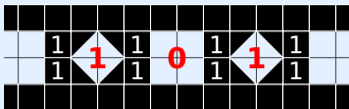
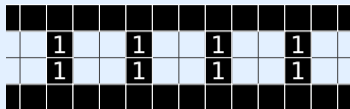
# Wires



# Wires



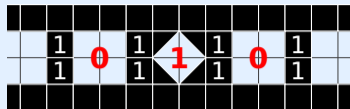
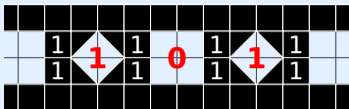
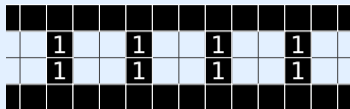
## Wires



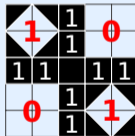
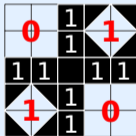
## Variable Gadgets



## Wires

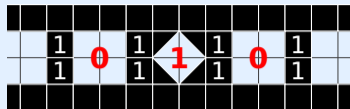
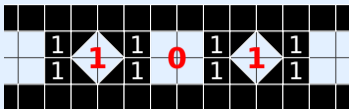
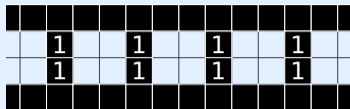


## Variable Gadgets

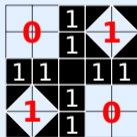


two possible solutions  $\approx$  two possible valuations

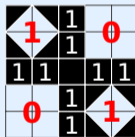
## Wires



## Variable Gadgets



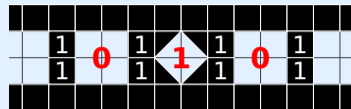
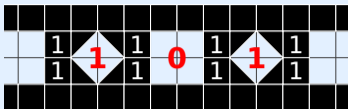
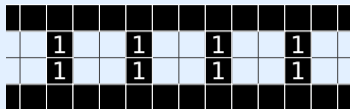
F



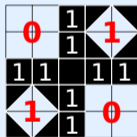
T

two possible solutions  $\approx$  two possible valuations

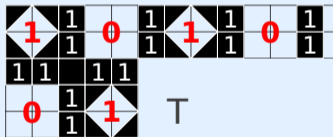
## Wires



## Variable Gadgets



F



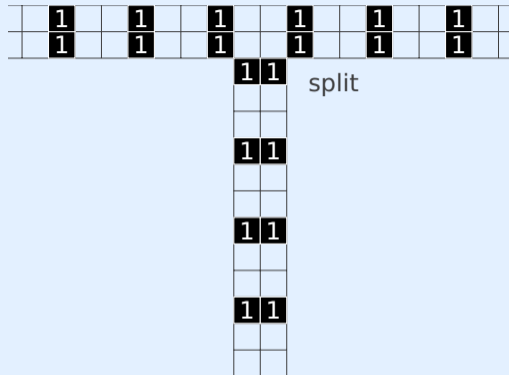
T

two possible solutions  $\approx$  two possible valuations

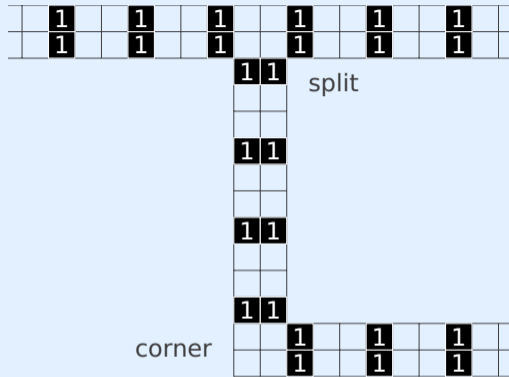




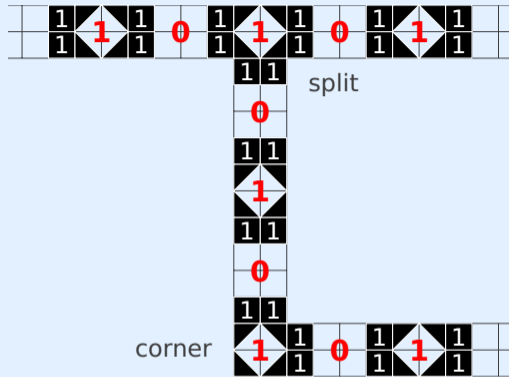
## Split and Corner Gadgets



## Split and Corner Gadgets

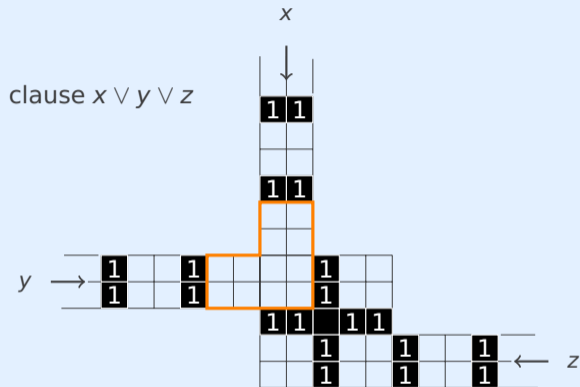


## Split and Corner Gadgets

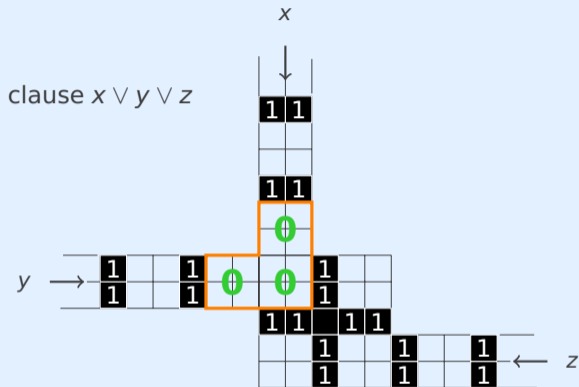




## Clause Gadget

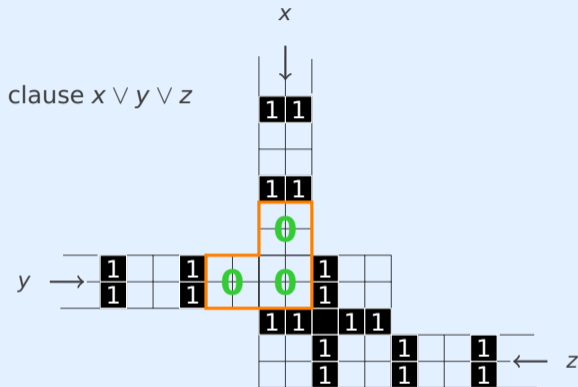


# Clause Gadget



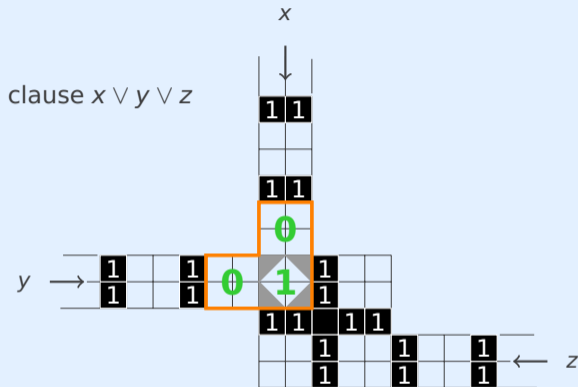
| x | y | z |
|---|---|---|
| 0 | 0 | 0 |

# Clause Gadget



| x | y | z |                       |
|---|---|---|-----------------------|
| 0 | 0 | 0 | <b>X</b> no rectangle |

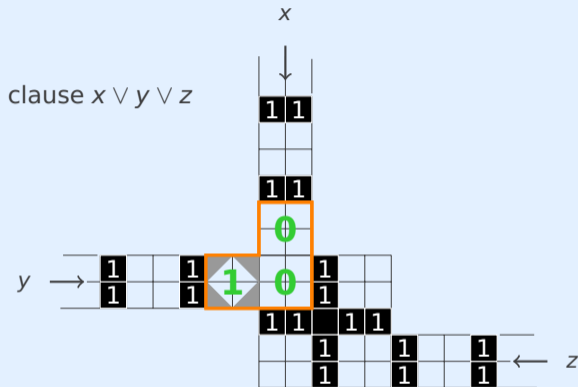
# Clause Gadget



| $x$ | $y$ | $z$ |   |
|-----|-----|-----|---|
| 0   | 0   | 0   | <span style="color: red;">✗</span> no rectangle |
| 0   | 0   | 1   | <span style="color: green;">✓</span>            |

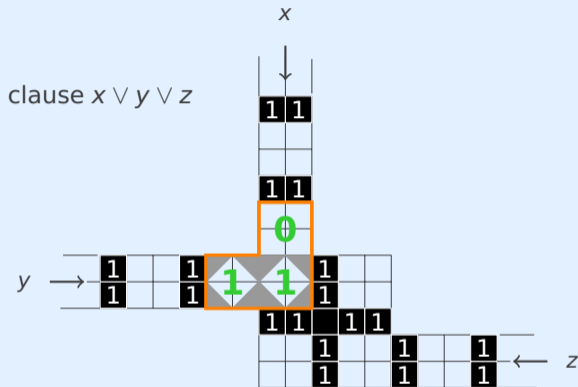


# Clause Gadget



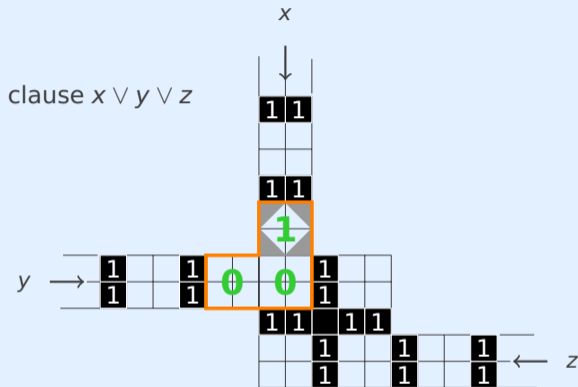
| $x$ | $y$ | $z$ |   |
|-----|-----|-----|---|
| 0   | 0   | 0   | <span style="color: red;">✗</span> no rectangle |
| 0   | 0   | 1   | <span style="color: green;">✓</span>            |
| 0   | 1   | 0   | <span style="color: green;">✓</span>            |

# Clause Gadget



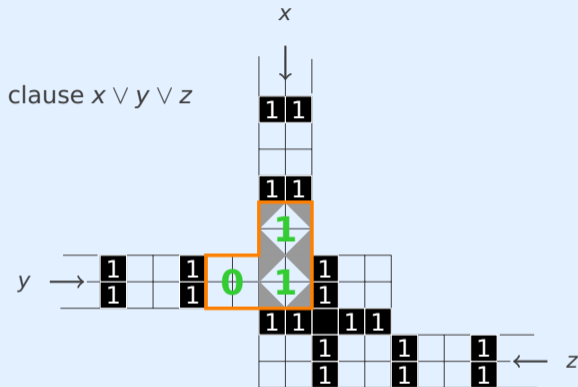
| $x$ | $y$ | $z$ |   |
|-----|-----|-----|---|
| 0   | 0   | 0   | <span style="color: red;">✗</span> no rectangle |
| 0   | 0   | 1   | <span style="color: green;">✓</span>            |
| 0   | 1   | 0   | <span style="color: green;">✓</span>            |
| 0   | 1   | 1   | <span style="color: green;">✓</span>            |

# Clause Gadget



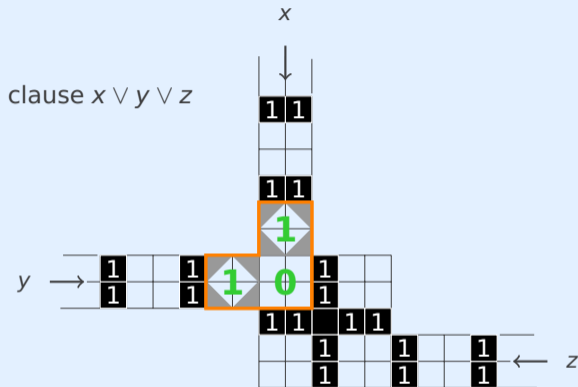
| $x$ | $y$ | $z$ |   |
|-----|-----|-----|---|
| 0   | 0   | 0   | <span style="color: red;">✗</span> no rectangle |
| 0   | 0   | 1   | <span style="color: green;">✓</span>            |
| 0   | 1   | 0   | <span style="color: green;">✓</span>            |
| 0   | 1   | 1   | <span style="color: green;">✓</span>            |
| 1   | 0   | 0   | <span style="color: green;">✓</span>            |

# Clause Gadget



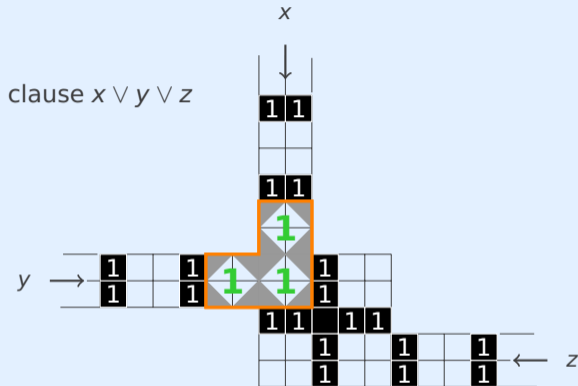
| x | y | z |   |
|---|---|---|---|
| 0 | 0 | 0 | <span style="color: red;">✗</span> no rectangle |
| 0 | 0 | 1 | <span style="color: green;">✓</span>            |
| 0 | 1 | 0 | <span style="color: green;">✓</span>            |
| 0 | 1 | 1 | <span style="color: green;">✓</span>            |
| 1 | 0 | 0 | <span style="color: green;">✓</span>            |
| 1 | 0 | 1 | <span style="color: green;">✓</span>            |

# Clause Gadget



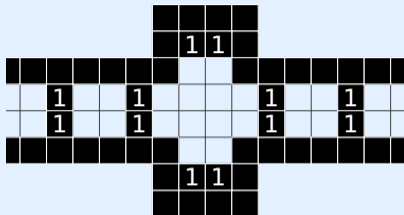
| x | y | z |   |
|---|---|---|---|
| 0 | 0 | 0 | <span style="color: red;">✗</span> no rectangle |
| 0 | 0 | 1 | <span style="color: green;">✓</span>            |
| 0 | 1 | 0 | <span style="color: green;">✓</span>            |
| 0 | 1 | 1 | <span style="color: green;">✓</span>            |
| 1 | 0 | 0 | <span style="color: green;">✓</span>            |
| 1 | 0 | 1 | <span style="color: green;">✓</span>            |
| 1 | 1 | 0 | <span style="color: green;">✓</span>            |

# Clause Gadget

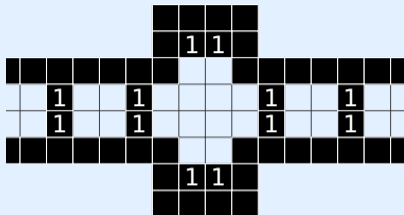


| x | y | z |                       |
|---|---|---|-----------------------|
| 0 | 0 | 0 | <b>X</b> no rectangle |
| 0 | 0 | 1 | ✓                     |
| 0 | 1 | 0 | ✓                     |
| 0 | 1 | 1 | ✓                     |
| 1 | 0 | 0 | ✓                     |
| 1 | 0 | 1 | ✓                     |
| 1 | 1 | 0 | ✓                     |
| 1 | 1 | 1 | ✓                     |

## Parity Gadget



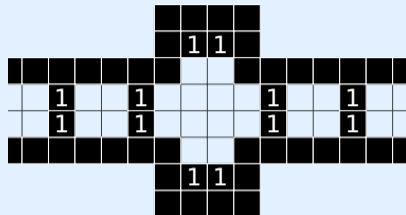
## Parity Gadget



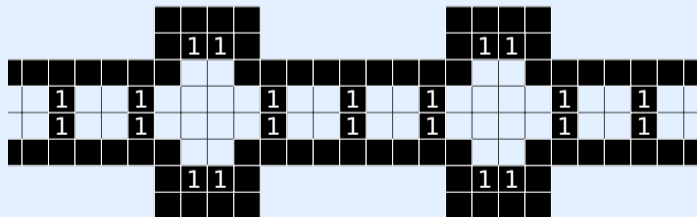
to change position of wires



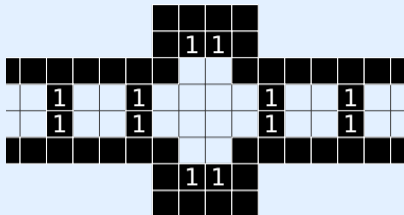
## Parity Gadget



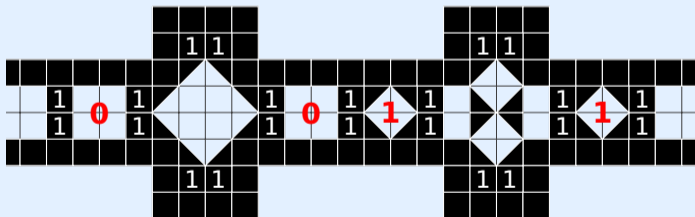
to change position of wires



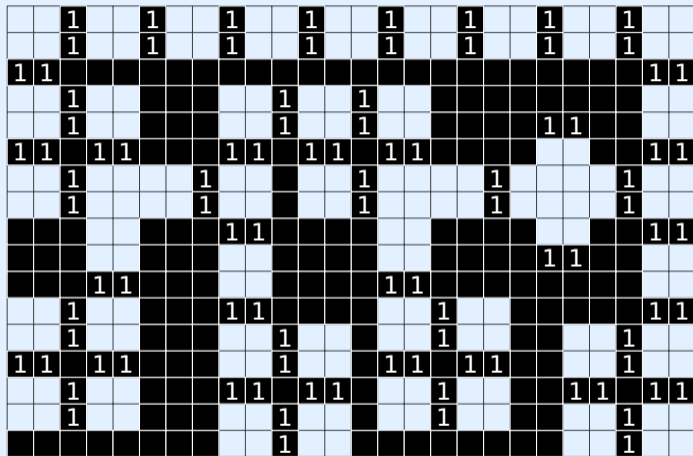
# Parity Gadget



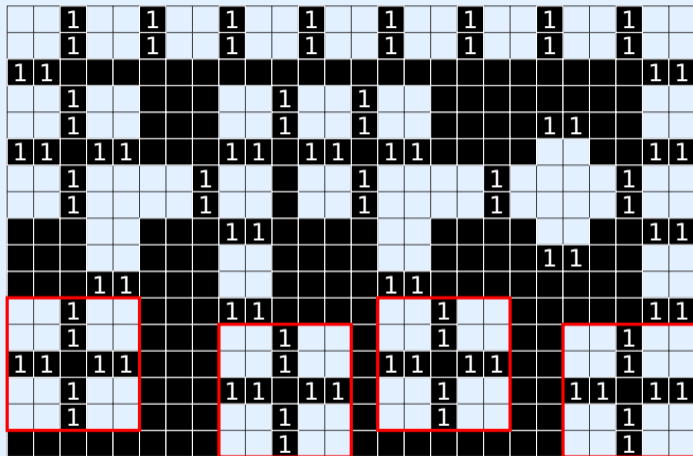
to change position of wires



# Example

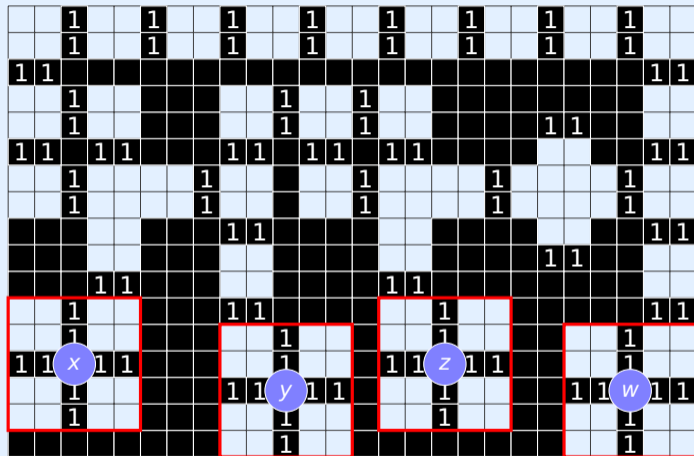


## Example



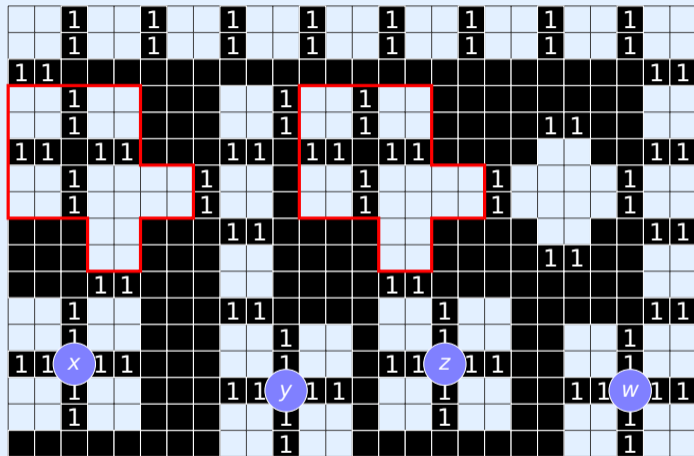
- 4 variables

## Example



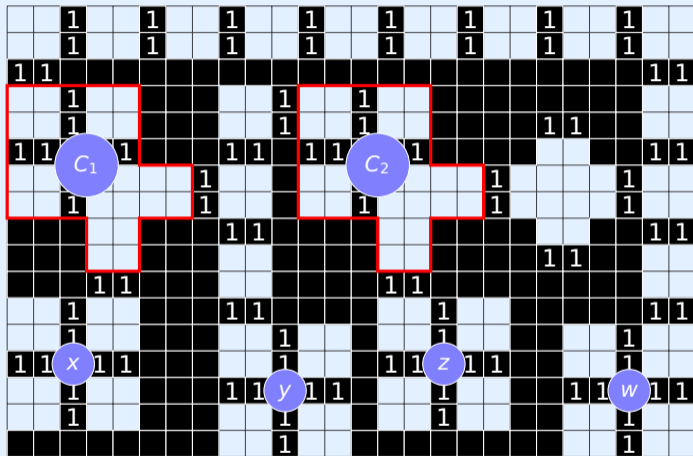
- 4 variables  $x y z w$

## Example



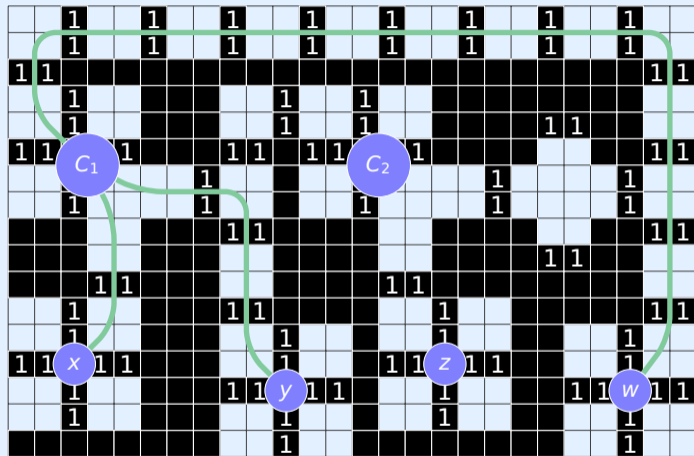
- 4 variables  $x y z w$
- 2 clauses

## Example



- 4 variables  $x y z w$
- 2 clauses  $C_1 C_2$

## Example

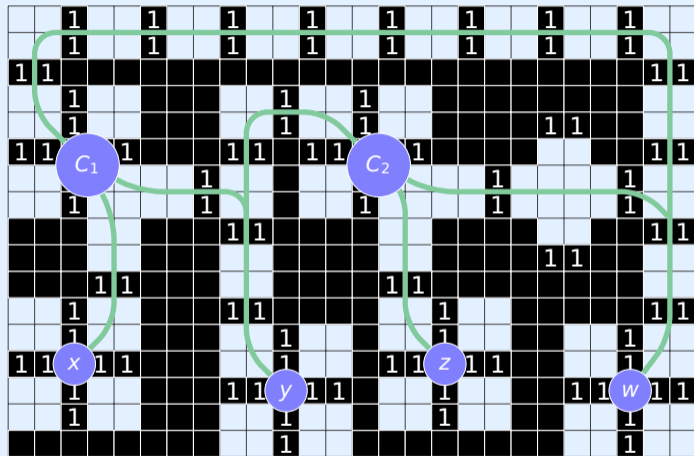


- 4 variables  $x y z w$
- 2 clauses  $C_1 C_2$

$$C_1 = \{x, \neg y, w\}$$



## Example



- 4 variables  $x y z w$
- 2 clauses  $C_1 C_2$

$$C_1 = \{x, \neg y, w\}$$

$$C_2 = \{y, \neg z, \neg w\}$$

# Outline

1. Summary of Previous Lecture
2. Example SAT Reduction: Shakashaka
- 3. Maximal Satisfiability**
4. Incremental SAT Solving
5. Further Reading

## MaxSAT

input: propositional CNF  $\varphi$

output: valuation  $v$  that maximizes number of satisfied clauses in  $\varphi$

## MaxSAT

input: propositional CNF  $\varphi$

output: valuation  $v$  that maximizes number of satisfied clauses in  $\varphi$

### Example

- CNF  $(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee r) \wedge \neg p \wedge (\neg p \vee \neg r \vee q)$  is unsatisfiable

## MaxSAT

input: propositional CNF  $\varphi$

output: valuation  $v$  that maximizes number of satisfied clauses in  $\varphi$

### Example

- CNF  $(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee r) \wedge \neg p \wedge (\neg p \vee \neg r \vee q)$  is unsatisfiable
- $v(p) = v(q) = v(r) = T$  satisfies 5 out of 6 clauses

## MaxSAT

input: propositional CNF  $\varphi$

output: valuation  $v$  that maximizes number of satisfied clauses in  $\varphi$

## Example

- CNF  $(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee r) \wedge \neg p \wedge (\neg p \vee \neg r \vee q)$  is unsatisfiable
- $v(p) = v(q) = v(r) = T$  satisfies 5 out of 6 clauses

## Variation

- **hard** clauses that must be satisfied

## MaxSAT

input: propositional CNF  $\varphi$

output: valuation  $v$  that maximizes number of satisfied clauses in  $\varphi$

## Example

- CNF  $(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee r) \wedge \neg p \wedge (\neg p \vee \neg r \vee q)$  is unsatisfiable
- $v(p) = v(q) = v(r) = T$  satisfies 5 out of 6 clauses

## Variation

- hard clauses that must be satisfied
- **soft** clauses that are desirable to be satisfied

## MaxSAT

input: propositional CNF  $\varphi$

output: valuation  $v$  that maximizes number of satisfied clauses in  $\varphi$

## Example

- CNF  $(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee r) \wedge \neg p \wedge (\neg p \vee \neg r \vee q)$  is unsatisfiable
- $v(p) = v(q) = v(r) = T$  satisfies 5 out of 6 clauses

## Variation

- hard clauses that must be satisfied
- soft clauses that are desirable to be satisfied
- **weights** for soft clauses



## MaxSAT

input: propositional CNF  $\varphi$

output: valuation  $v$  that maximizes number of satisfied clauses in  $\varphi$

## Example

- CNF  $(q \vee \neg r) \wedge (\neg q \vee r) \wedge p \wedge (\neg p \vee r) \wedge \neg p \wedge (\neg p \vee \neg r \vee q)$  is unsatisfiable
- $v(p) = v(q) = v(r) = T$  satisfies 5 out of 6 clauses

## Variation

- hard clauses that must be satisfied
- soft clauses that are desirable to be satisfied
- weights for soft clauses
- goal: maximize sum of weights of soft clauses while satisfying all hard clauses

## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by  $T$

## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by  $\top$

## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by **T**
- $\varphi_{\bar{x}}$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by **F**

## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by T
- $\varphi_{\bar{x}}$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by F
- function **simp**( $\varphi$ )
  - replaces  $\neg T$  by F and  $\neg F$  by T

## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by T
- $\varphi_{\bar{x}}$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by F
- function **simp**( $\varphi$ )
  - replaces  $\neg T$  by F and  $\neg F$  by T
  - removes all clauses which contain T

## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by T
- $\varphi_{\bar{x}}$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by F
- function **simp**( $\varphi$ )
  - replaces  $\neg T$  by F and  $\neg F$  by T
  - removes all clauses which contain T
  - removes F from remaining clauses

## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by T
- $\varphi_{\bar{x}}$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by F
- function  $\text{simp}(\varphi)$ 
  - replaces  $\neg T$  by F and  $\neg F$  by T
  - removes all clauses which contain T
  - removes F from remaining clauses
- **#empty**( $\varphi$ ) denotes number of empty clauses in  $\varphi$



## Branch and Bound — Notation

- $\varphi_x$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by T
- $\varphi_{\bar{x}}$  denotes formula  $\varphi$  with all occurrences of  $x$  replaced by F
- function  $\text{simp}(\varphi)$ 
  - replaces  $\neg T$  by F and  $\neg F$  by T
  - removes all clauses which contain T
  - removes F from remaining clauses
- $\#\text{empty}(\varphi)$  denotes number of empty clauses in  $\varphi$
- CNF  $\varphi$  is presented as **list** of clauses

function **BnB**( $\varphi, b$ )

## Branch & Bound — Algorithm

function **BnB**( $\varphi, b$ )      -- initial call:  $b$  is number  $|\varphi|$  of clauses in  $\varphi$

## Remarks

- **BnB**( $\varphi, |\varphi|$ ) returns **minimum** number of **unsatisfied** clauses ( $\text{minUNSAT}(\varphi)$ )

## Branch & Bound — Algorithm

function **BnB**( $\varphi, b$ )      -- initial call:  $b$  is number  $|\varphi|$  of clauses in  $\varphi$

## Remarks

- $\text{BnB}(\varphi, |\varphi|)$  returns minimum number of unsatisfied clauses ( $\text{minUNSAT}(\varphi)$ )
- $\text{maxSAT}$  answer is  $|\varphi| - \text{BnB}(\varphi, |\varphi|)$

## Branch & Bound — Algorithm

function **BnB**( $\varphi, b$ )      -- initial call:  $b$  is number  $|\varphi|$  of clauses in  $\varphi$   
     $\varphi \leftarrow \text{simp}(\varphi)$

## Remarks

- $\text{BnB}(\varphi, |\varphi|)$  returns minimum number of unsatisfied clauses ( $\text{minUNSAT}(\varphi)$ )
- maxSAT answer is  $|\varphi| - \text{BnB}(\varphi, |\varphi|)$

## Branch & Bound — Algorithm

function **BnB**( $\varphi, b$ )      -- initial call:  $b$  is number  $|\varphi|$  of clauses in  $\varphi$   
     $\varphi \leftarrow \text{simp}(\varphi)$   
    if  $\varphi$  contains only empty clauses then return  $\#\text{empty}(\varphi)$

## Remarks

- $\text{BnB}(\varphi, |\varphi|)$  returns minimum number of unsatisfied clauses ( $\text{minUNSAT}(\varphi)$ )
- maxSAT answer is  $|\varphi| - \text{BnB}(\varphi, |\varphi|)$
- $\#\text{empty}(\varphi)$  denotes number of clauses falsified by current valuation

## Branch & Bound — Algorithm

```
function BnB( $\varphi, b$ )      -- initial call:  $b$  is number  $|\varphi|$  of clauses in  $\varphi$   
   $\varphi \leftarrow \text{simp}(\varphi)$   
  if  $\varphi$  contains only empty clauses then return  $\#\text{empty}(\varphi)$   
  if  $\#\text{empty}(\varphi) \geq b$  then return  $b$ 
```

## Remarks

- $\text{BnB}(\varphi, |\varphi|)$  returns minimum number of unsatisfied clauses ( $\text{minUNSAT}(\varphi)$ )
- $\text{maxSAT}$  answer is  $|\varphi| - \text{BnB}(\varphi, |\varphi|)$
- $\#\text{empty}(\varphi)$  denotes number of clauses falsified by current valuation

## Branch & Bound — Algorithm

```
function BnB( $\varphi, b$ )      -- initial call:  $b$  is number  $|\varphi|$  of clauses in  $\varphi$   
   $\varphi \leftarrow \text{simp}(\varphi)$   
  if  $\varphi$  contains only empty clauses then return  $\#\text{empty}(\varphi)$   
  if  $\#\text{empty}(\varphi) \geq b$  then return  $b$   
   $x \leftarrow$  select variable in  $\varphi$ 
```

## Remarks

- $\text{BnB}(\varphi, |\varphi|)$  returns minimum number of unsatisfied clauses ( $\text{minUNSAT}(\varphi)$ )
- maxSAT answer is  $|\varphi| - \text{BnB}(\varphi, |\varphi|)$
- $\#\text{empty}(\varphi)$  denotes number of clauses falsified by current valuation



## Branch & Bound — Algorithm

```
function BnB( $\varphi, b$ )      -- initial call:  $b$  is number  $|\varphi|$  of clauses in  $\varphi$ 
   $\varphi \leftarrow \text{simp}(\varphi)$ 
  if  $\varphi$  contains only empty clauses then return  $\#\text{empty}(\varphi)$ 
  if  $\#\text{empty}(\varphi) \geq b$  then return  $b$ 
   $x \leftarrow$  select variable in  $\varphi$ 
  return  $\min(b, \text{BnB}(\varphi_x, b), \text{BnB}(\varphi_{\bar{x}}, \text{BnB}(\varphi_x, b)))$ 
```

## Remarks

- $\text{BnB}(\varphi, |\varphi|)$  returns minimum number of unsatisfied clauses ( $\text{minUNSAT}(\varphi)$ )
- maxSAT answer is  $|\varphi| - \text{BnB}(\varphi, |\varphi|)$
- $\#\text{empty}(\varphi)$  denotes number of clauses falsified by current valuation

## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$

## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$

BnB( $\varphi, 6$ )

## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$

BnB( $\varphi, 6$ )

## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$                                    $\# \text{empty} = 0$

BnB( $\varphi, 6$ )

$$\begin{array}{c} | \\ \boxed{0 \geq 6} \end{array}$$

## Example

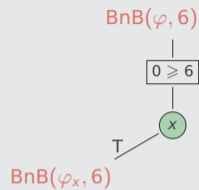
- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$   $\#empty = 0$

BnB( $\varphi, 6$ )



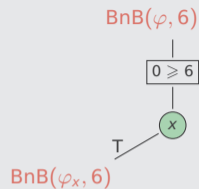
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$



## Example

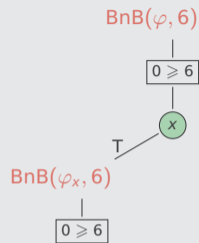
- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$





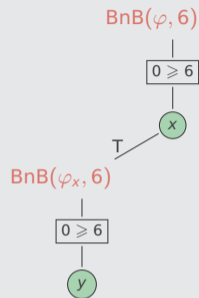
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0



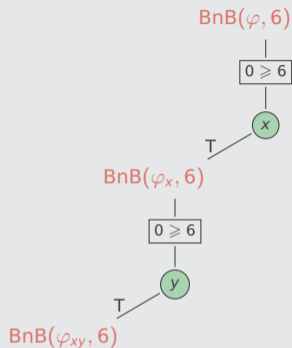
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0



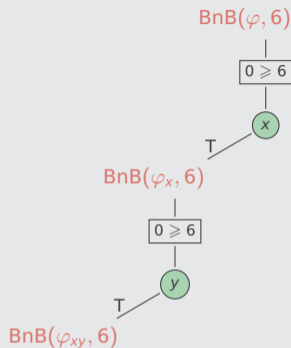
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \top, z \vee \neg \top, \neg \top$



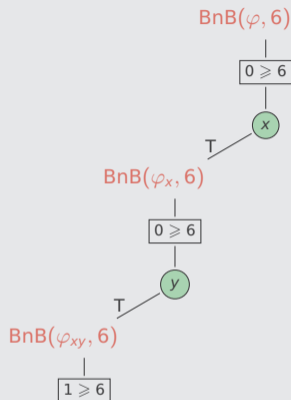
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \top, z \vee \neg \top, \neg \top$   
 $\text{simp}(\varphi_{xy}) = z, \square$



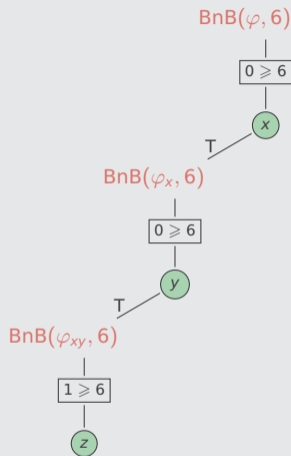
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \top, z \vee \neg \top, \neg \top$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1



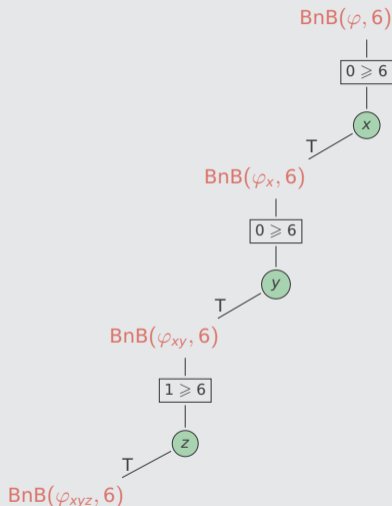
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \top, \neg \top \vee y, z \vee \neg y, \top \vee z, \top \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \top, z \vee \neg \top, \neg \top$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1



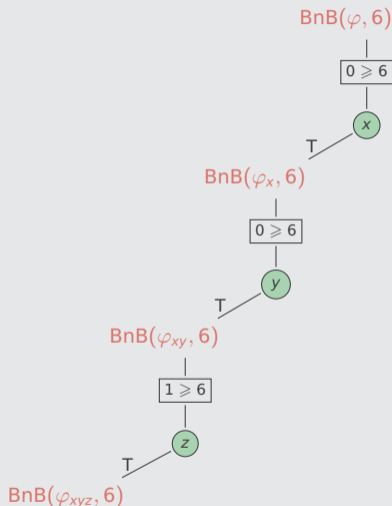
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$



## Example

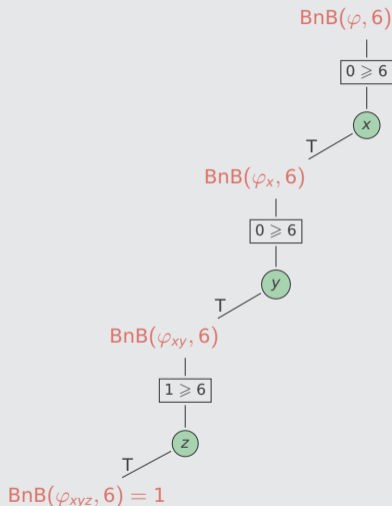
- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$





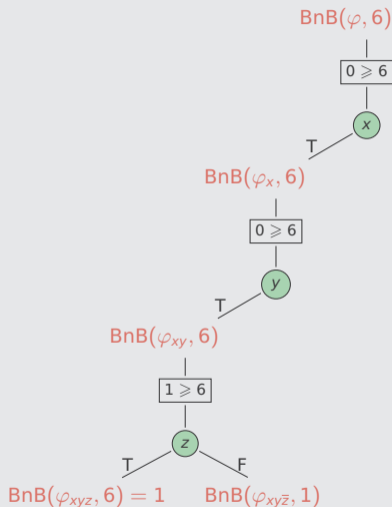
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1



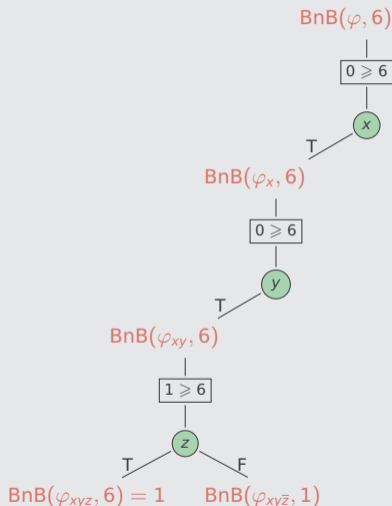
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$



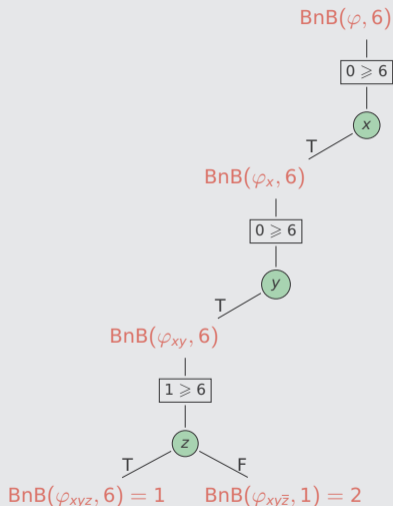
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$



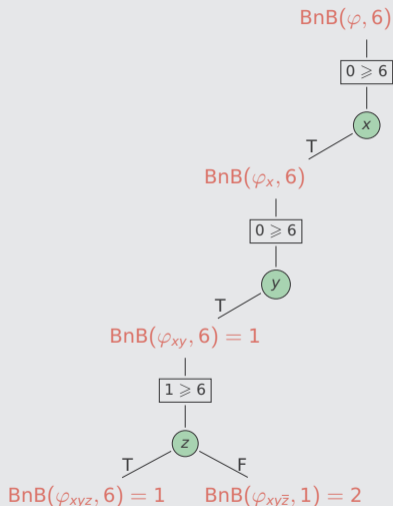
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2



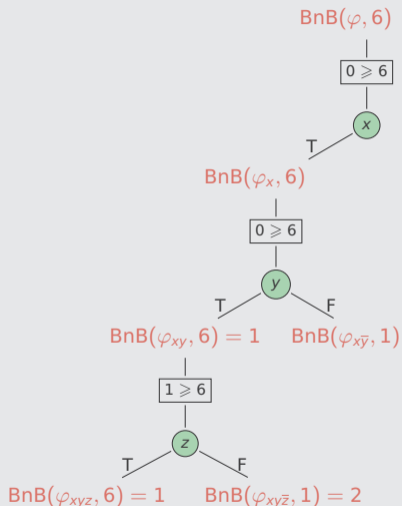
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2



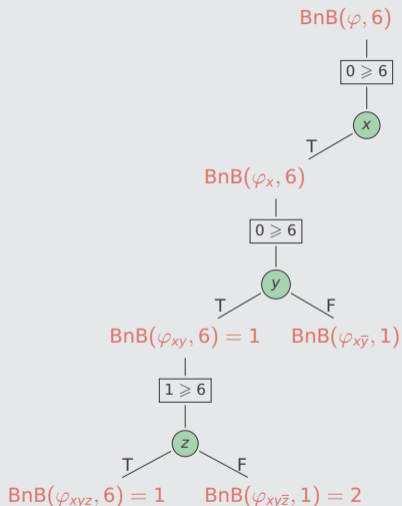
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$



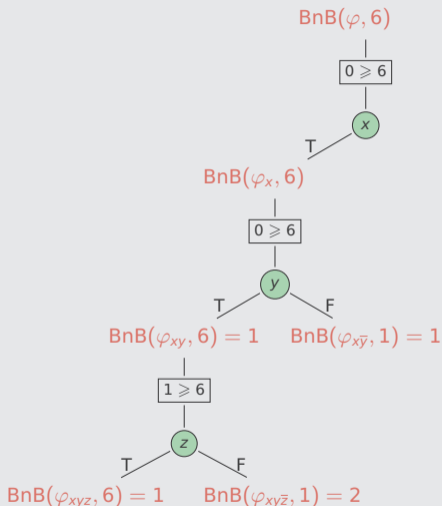
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$



## Example

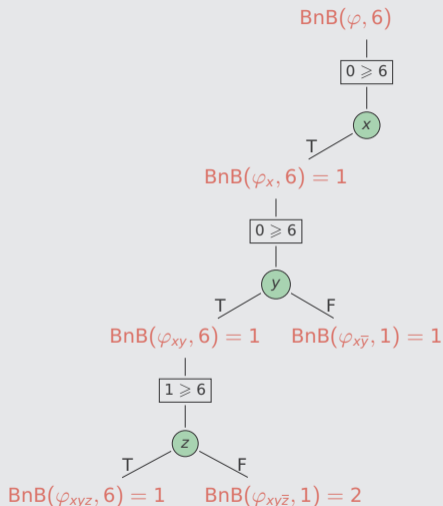
- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1





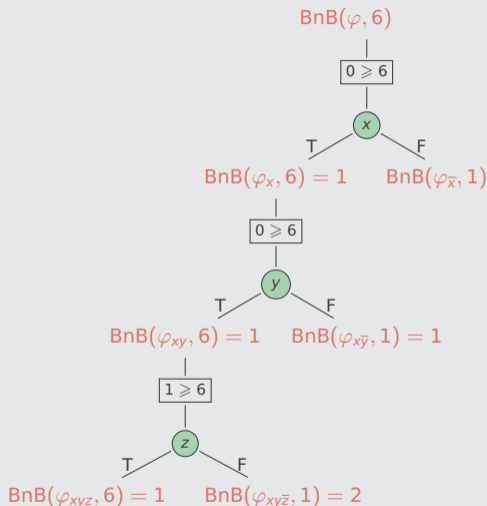
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1



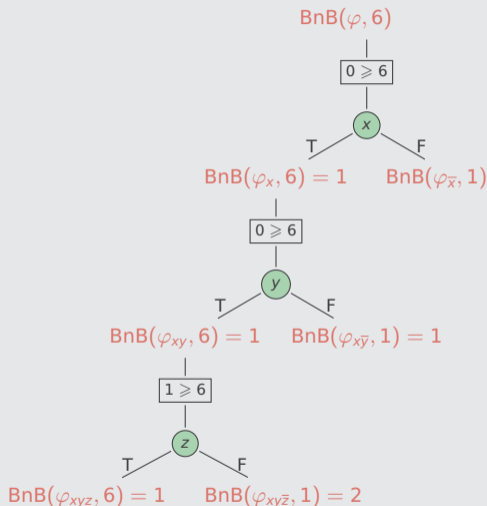
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1
- $\varphi_{\bar{x}} = \mathbf{F}, \neg \mathbf{F} \vee y, z \vee \neg y, \mathbf{F} \vee z, \mathbf{F} \vee y, \neg y$



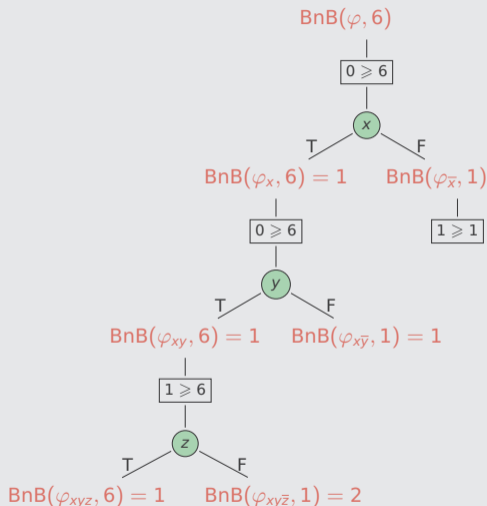
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1
- $\varphi_{\bar{x}} = \mathbf{F}, \neg \mathbf{F} \vee y, z \vee \neg y, \mathbf{F} \vee z, \mathbf{F} \vee y, \neg y$   
 $\text{simp}(\varphi_{\bar{x}}) = \square, z \vee \neg y, z, y, \neg y$



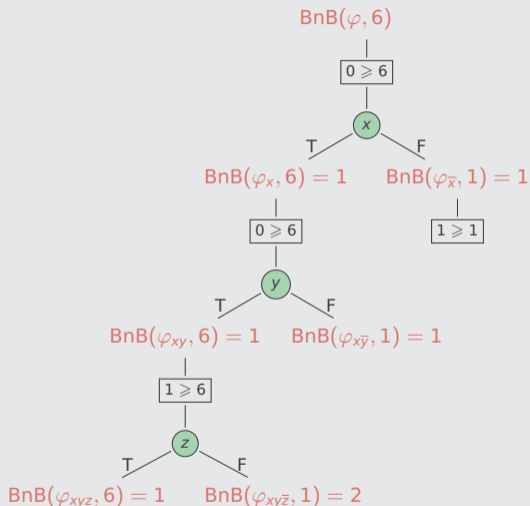
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1
- $\varphi_{\bar{x}} = \mathbf{F}, \neg \mathbf{F} \vee y, z \vee \neg y, \mathbf{F} \vee z, \mathbf{F} \vee y, \neg y$   
 $\text{simp}(\varphi_{\bar{x}}) = \square, z \vee \neg y, z, y, \neg y$  #empty = 1



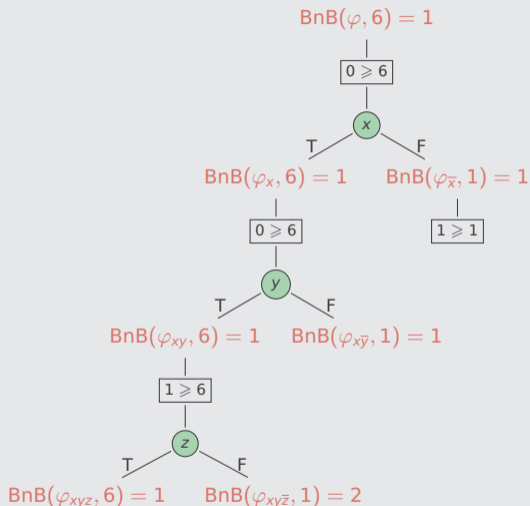
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1
- $\varphi_{\bar{x}} = \mathbf{F}, \neg \mathbf{F} \vee y, z \vee \neg y, \mathbf{F} \vee z, \mathbf{F} \vee y, \neg y$   
 $\text{simp}(\varphi_{\bar{x}}) = \square, z \vee \neg y, z, y, \neg y$  #empty = 1



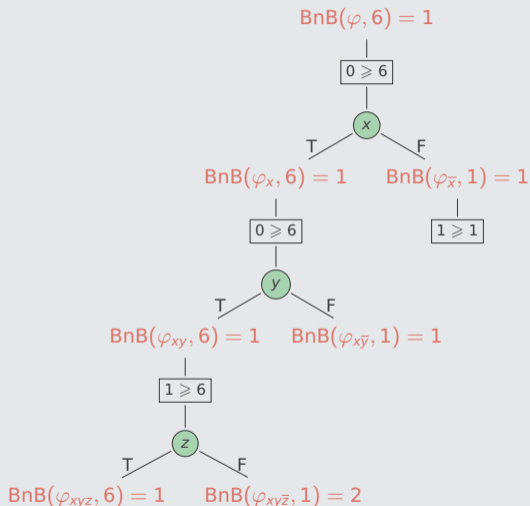
## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1
- $\varphi_{\bar{x}} = \mathbf{F}, \neg \mathbf{F} \vee y, z \vee \neg y, \mathbf{F} \vee z, \mathbf{F} \vee y, \neg y$   
 $\text{simp}(\varphi_{\bar{x}}) = \square, z \vee \neg y, z, y, \neg y$  #empty = 1



## Example

- $\varphi = x, \neg x \vee y, z \vee \neg y, x \vee z, x \vee y, \neg y$   
 $\text{simp}(\varphi) = \varphi$  #empty = 0
- $\varphi_x = \mathbf{T}, \neg \mathbf{T} \vee y, z \vee \neg y, \mathbf{T} \vee z, \mathbf{T} \vee y, \neg y$   
 $\text{simp}(\varphi_x) = y, z \vee \neg y, \neg y$  #empty = 0
- $\varphi_{xy} = \mathbf{T}, z \vee \neg \mathbf{T}, \neg \mathbf{T}$   
 $\text{simp}(\varphi_{xy}) = z, \square$  #empty = 1
- $\varphi_{xyz} = \mathbf{T}, \square$   
 $\text{simp}(\varphi_{xyz}) = \square$  #empty = 1
- $\varphi_{xy\bar{z}} = \mathbf{F}, \square$   
 $\text{simp}(\varphi_{xy\bar{z}}) = \square, \square$  #empty = 2
- $\varphi_{x\bar{y}} = \mathbf{F}, z \vee \neg \mathbf{F}, \neg \mathbf{F}$   
 $\text{simp}(\varphi_{x\bar{y}}) = \square$  #empty = 1
- $\varphi_{\bar{x}} = \mathbf{F}, \neg \mathbf{F} \vee y, z \vee \neg y, \mathbf{F} \vee z, \mathbf{F} \vee y, \neg y$   
 $\text{simp}(\varphi_{\bar{x}}) = \square, z \vee \neg y, z, y, \neg y$  #empty = 1
- $\text{maxSAT}(\varphi) = 6 - \text{BnB}(\varphi, 6) = 5$



## Binary Search — Idea

repeatedly call SAT solver in binary search fashion and return  $\text{minUNSAT}(\varphi)$



## Binary Search — Idea

repeatedly call SAT solver in **binary search fashion** and return  $\text{minUNSAT}(\varphi)$

### Example

consider formula with 18 clauses; can we satisfy ...

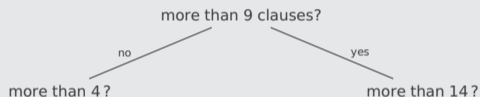
more than 9 clauses?

## Binary Search — Idea

repeatedly call SAT solver in binary search fashion and return  $\min\text{UNSAT}(\varphi)$

### Example

consider formula with 18 clauses; can we satisfy ...

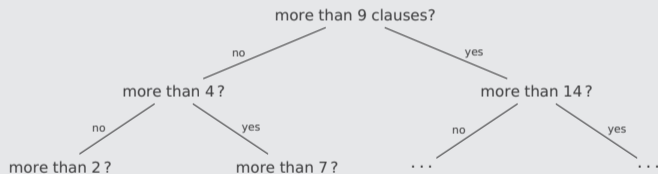


## Binary Search — Idea

repeatedly call SAT solver in binary search fashion and return  $\min\text{UNSAT}(\varphi)$

### Example

consider formula with 18 clauses; can we satisfy ...

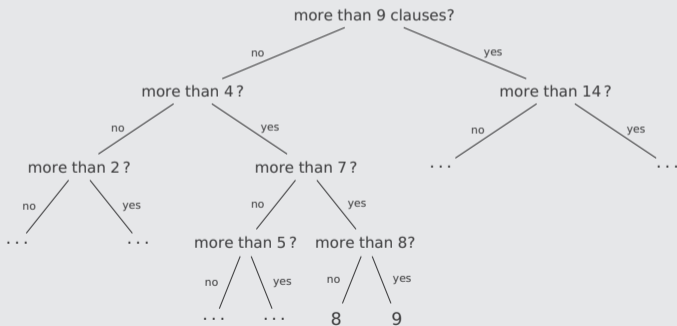


## Binary Search — Idea

repeatedly call SAT solver in binary search fashion and return  $\text{minUNSAT}(\varphi)$

### Example

consider formula with 18 clauses; can we satisfy ...



## Binary Search

function `BinarySearch`( $\{C_1, \dots, C_m\}$ )

## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function **search**( $\varphi, L, U$ )



## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function **search**( $\varphi, L, U$ )

if  $L \geq U$  then return  $U$

## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function **search**( $\varphi, L, U$ )

if  $L \geq U$  then return  $U$

$M = \lfloor \frac{L+U}{2} \rfloor$

## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function **search**( $\varphi, L, U$ )

if  $L \geq U$  then return  $U$

$M = \lfloor \frac{L+U}{2} \rfloor$

if SAT( $\varphi \wedge \text{CNF}(b_1 + \dots + b_m \leq M)$ )

## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function **search**( $\varphi, L, U$ )

if  $L \geq U$  then return  $U$

$M = \lfloor \frac{L+U}{2} \rfloor$

if SAT( $\varphi \wedge \text{CNF}(b_1 + \dots + b_m \leq M)$ ) then return search( $\varphi, L, M$ )

## Binary Search

function **BinarySearch**( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function **search**( $\varphi, L, U$ )

if  $L \geq U$  then return  $U$

$M = \lfloor \frac{L+U}{2} \rfloor$

if SAT( $\varphi \wedge \text{CNF}(b_1 + \dots + b_m \leq M)$ ) then return search( $\varphi, L, M$ )

else return search( $\varphi, M + 1, U$ )

## Binary Search

function BinarySearch( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function search( $\varphi, L, U$ )

if  $L \geq U$  then return  $U$

$M = \lfloor \frac{L+U}{2} \rfloor$

if SAT( $\varphi \wedge \text{CNF}(b_1 + \dots + b_m \leq M)$ ) then return search( $\varphi, L, M$ )

else return search( $\varphi, M + 1, U$ )

## Theorem

BinarySearch( $\varphi$ ) = minUNSAT( $\varphi$ )

## Binary Search

function BinarySearch( $\{C_1, \dots, C_m\}$ )

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$       --  $b_1, \dots, b_m$  are fresh variables

return search( $\varphi, 0, m$ )

function search( $\varphi, L, U$ )

if  $L \geq U$  then return  $U$

$M = \lfloor \frac{L+U}{2} \rfloor$

if SAT( $\varphi \wedge \text{CNF}(b_1 + \dots + b_m \leq M)$ ) then return search( $\varphi, L, M$ )      **cardinality constraint**

else return search( $\varphi, M + 1, U$ )

## Theorem

BinarySearch( $\varphi$ ) = minUNSAT( $\varphi$ )

## Example

$$\varphi = \left\{ \begin{array}{cccc} 6 \vee 2 & \neg 6 \vee 2 & \neg 2 \vee 1 & \neg 1 \\ \neg 6 \vee 8 & 6 \vee \neg 8 & 2 \vee 4 & \neg 4 \vee 5 \\ 7 \vee 5 & \neg 7 \vee 5 & \neg 3 & \neg 5 \vee 3 \end{array} \right\}$$



## Example

$$\varphi = \left\{ \begin{array}{cccc} 6 \vee 2 \vee b_1 & \neg 6 \vee 2 \vee b_2 & \neg 2 \vee 1 \vee b_3 & \neg 1 \vee b_4 \\ \neg 6 \vee 8 \vee b_5 & 6 \vee \neg 8 \vee b_6 & 2 \vee 4 \vee b_7 & \neg 4 \vee 5 \vee b_8 \\ 7 \vee 5 \vee b_9 & \neg 7 \vee 5 \vee b_{10} & \neg 3 \vee b_{11} & \neg 5 \vee 3 \vee b_{12} \end{array} \right\}$$

## Example

$$\varphi = \left\{ \begin{array}{cccc} 6 \vee 2 \vee b_1 & \neg 6 \vee 2 \vee b_2 & \neg 2 \vee 1 \vee b_3 & \neg 1 \vee b_4 \\ \neg 6 \vee 8 \vee b_5 & 6 \vee \neg 8 \vee b_6 & 2 \vee 4 \vee b_7 & \neg 4 \vee 5 \vee b_8 \\ 7 \vee 5 \vee b_9 & \neg 7 \vee 5 \vee b_{10} & \neg 3 \vee b_{11} & \neg 5 \vee 3 \vee b_{12} \end{array} \right\}$$

- $L = 0, U = 12, M = 6$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 6))$  ?

## Example

$$\varphi = \left\{ \begin{array}{llll} 6 \vee 2 \vee b_1 & \neg 6 \vee 2 \vee b_2 & \neg 2 \vee 1 \vee b_3 & \neg 1 \vee b_4 \\ \neg 6 \vee 8 \vee b_5 & 6 \vee \neg 8 \vee b_6 & 2 \vee 4 \vee b_7 & \neg 4 \vee 5 \vee b_8 \\ 7 \vee 5 \vee b_9 & \neg 7 \vee 5 \vee b_{10} & \neg 3 \vee b_{11} & \neg 5 \vee 3 \vee b_{12} \end{array} \right\}$$

- $L = 0, U = 12, M = 6$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 6))$  ?      ✓
- $L = 0, U = 6, M = 3$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 3))$  ?

## Example

$$\varphi = \left\{ \begin{array}{cccc} 6 \vee 2 \vee b_1 & \neg 6 \vee 2 \vee b_2 & \neg 2 \vee 1 \vee b_3 & \neg 1 \vee b_4 \\ \neg 6 \vee 8 \vee b_5 & 6 \vee \neg 8 \vee b_6 & 2 \vee 4 \vee b_7 & \neg 4 \vee 5 \vee b_8 \\ 7 \vee 5 \vee b_9 & \neg 7 \vee 5 \vee b_{10} & \neg 3 \vee b_{11} & \neg 5 \vee 3 \vee b_{12} \end{array} \right\}$$

- $L = 0, U = 12, M = 6$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 6))$  ?      ✓
- $L = 0, U = 6, M = 3$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 3))$  ?      ✓
- $L = 0, U = 3, M = 1$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 1))$  ?

## Example

$$\varphi = \left\{ \begin{array}{cccc} 6 \vee 2 \vee b_1 & \neg 6 \vee 2 \vee b_2 & \neg 2 \vee 1 \vee b_3 & \neg 1 \vee b_4 \\ \neg 6 \vee 8 \vee b_5 & 6 \vee \neg 8 \vee b_6 & 2 \vee 4 \vee b_7 & \neg 4 \vee 5 \vee b_8 \\ 7 \vee 5 \vee b_9 & \neg 7 \vee 5 \vee b_{10} & \neg 3 \vee b_{11} & \neg 5 \vee 3 \vee b_{12} \end{array} \right\}$$

- $L = 0, U = 12, M = 6$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 6))$  ?      ✓
- $L = 0, U = 6, M = 3$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 3))$  ?      ✓
- $L = 0, U = 3, M = 1$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 1))$  ?      ✗
- $L = 2, U = 3, M = 2$        $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 2))$  ?

## Example

$$\varphi = \left\{ \begin{array}{cccc} 6 \vee 2 \vee b_1 & \neg 6 \vee 2 \vee b_2 & \neg 2 \vee 1 \vee b_3 & \neg 1 \vee b_4 \\ \neg 6 \vee 8 \vee b_5 & 6 \vee \neg 8 \vee b_6 & 2 \vee 4 \vee b_7 & \neg 4 \vee 5 \vee b_8 \\ 7 \vee 5 \vee b_9 & \neg 7 \vee 5 \vee b_{10} & \neg 3 \vee b_{11} & \neg 5 \vee 3 \vee b_{12} \end{array} \right\}$$

- $L = 0, U = 12, M = 6$       $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 6))$  ?     ✓
- $L = 0, U = 6, M = 3$       $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 3))$  ?     ✓
- $L = 0, U = 3, M = 1$       $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 1))$  ?     ✗
- $L = 2, U = 3, M = 2$       $\text{SAT}(\varphi \wedge \text{CNF}(b_1 + \dots + b_{12} \leq 2))$  ?     ✓
- $L = 2, U = 2$       $\text{minUNSAT}(\varphi) = 2 \implies \text{maxSAT}(\varphi) = 10$

# MaxSAT Competition

## Further Information

- MaxSAT Evaluation 2023

# Outline

1. Summary of Previous Lecture
2. Example SAT Reduction: Shakashaka
3. Maximal Satisfiability
- 4. Incremental SAT Solving**
5. Further Reading



## Motivation

- applications such as MaxSAT often require **incremental interface**

## Motivation

- applications such as MaxSAT often require **incremental interface**
  - determine satisfiability of clauses  $C_1$

## Motivation

- applications such as MaxSAT often require **incremental interface**
  - determine satisfiability of clauses  $C_1$
  - slightly change  $C_1$  to  $C_2$  and again determine satisfiability
  - ...

## Motivation

- applications such as MaxSAT often require **incremental interface**
  - determine satisfiability of clauses  $C_1$
  - slightly change  $C_1$  to  $C_2$  and again determine satisfiability
  - ...
- aims

## Motivation

- applications such as MaxSAT often require **incremental interface**
  - determine satisfiability of clauses  $C_1$
  - slightly change  $C_1$  to  $C_2$  and again determine satisfiability
  - ...
- aims
  - do **not restart** SAT solver in every iteration

## Motivation

- applications such as MaxSAT often require **incremental interface**
  - determine satisfiability of clauses  $C_1$
  - slightly change  $C_1$  to  $C_2$  and again determine satisfiability
  - ...
- aims
  - do **not restart** SAT solver in every iteration
  - **reuse learned clauses, keep knowledge** of solver (decision heuristics, ...)

## Motivation

- applications such as MaxSAT often require **incremental interface**
  - determine satisfiability of clauses  $C_1$
  - slightly change  $C_1$  to  $C_2$  and again determine satisfiability
  - ...
- aims
  - do **not restart** SAT solver in every iteration
  - **reuse learned clauses, keep knowledge** of solver (decision heuristics, ...)
- challenge
  - keep learned clauses, even if some clauses are removed from input

## Solution: Assumptions

- first introduced in SAT solver MiniSAT



## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\} \cup c$ ; CNF is  $\bigcup_i C_i$

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c$  for each  $c \in C_1$

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c$  for each  $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of  $C_1$

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c$  for each  $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of  $C_1$
  - for switching from clauses  $C_1$  to  $C_2$  perform two steps

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c^d$  for each  $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of  $C_1$
  - for switching from clauses  $C_1$  to  $C_2$  perform two steps
    - **undo decisions**  $x_c^d$  for all  $c \in C_1 \setminus C_2$  (deactivate) and

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c$  for each  $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of  $C_1$
  - for switching from clauses  $C_1$  to  $C_2$  perform two steps
    - **undo decisions**  $x_c$  for all  $c \in C_1 \setminus C_2$  (deactivate) and
    - **add decisions**  $x_c$  for all  $c \in C_2 \setminus C_1$  (activate)



## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c$  for each  $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of  $C_1$
  - for switching from clauses  $C_1$  to  $C_2$  perform two steps
    - **undo decisions**  $x_c$  for all  $c \in C_1 \setminus C_2$  (deactivate) and
    - **add decisions**  $x_c$  for all  $c \in C_2 \setminus C_1$  (activate)
  - then continue with DPLL algorithm as in previous step, continue with  $C_3, \dots$

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c$  for each  $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of  $C_1$
  - for switching from clauses  $C_1$  to  $C_2$  perform two steps
    - **undo decisions**  $x_c^d$  for all  $c \in C_1 \setminus C_2$  (deactivate) and
    - **add decisions**  $x_c^d$  for all  $c \in C_2 \setminus C_1$  (activate)
  - then continue with DPLL algorithm as in previous step, continue with  $C_3, \dots$

## Observation

- since the set of clauses stays identical, learned clauses stay valid

## Solution: Assumptions

- first introduced in SAT solver MiniSAT
- key idea: **assumption literals** (also called **clause selectors**)
  - create new assumption literal  $x_c$  for each clause  $c$  that might be activated or deactivated
  - change every such clause  $c$  in the CNF by  $\{\neg x_c\}_d \cup c$ ; CNF is  $\bigcup_i C_i$
  - run DPLL algorithm for  $C_1$  with **initial decisions**  $x_c$  for each  $c \in C_1$
  - as soon as DPLL algorithm backtracks below this initial decision level, report unsat of  $C_1$
  - for switching from clauses  $C_1$  to  $C_2$  perform two steps
    - **undo decisions**  $x_c^d$  for all  $c \in C_1 \setminus C_2$  (deactivate) and
    - **add decisions**  $x_c^d$  for all  $c \in C_2 \setminus C_1$  (activate)
  - then continue with DPLL algorithm as in previous step, continue with  $C_3, \dots$

## Observation

- since the set of clauses stays identical, learned clauses stay valid
- learned clauses that contain clause selectors can only be applied if these clauses are currently activated; other learned clauses can be used for all runs

# Outline

1. Summary of Previous Lecture
2. Example SAT Reduction: Shakashaka
3. Maximal Satisfiability
4. Incremental SAT Solving
- 5. Further Reading**

## Kröning and Strichmann

- Sections 2.2 and 2.4

## Further Reading

- Erik D. Demaine, Yoshio Okamata, Ryuhei Uehara, and Yushi Uno  
Computational Complexity and an Integer Programming Model of Shakashaka  
Proc. 25th Canadian Conference on Computational Geometry, 2013

## Motivation for extending SAT to first-order theories

predicates instead of propositional variables

## Motivation for extending SAT to first-order theories

predicates instead of propositional variables

### Examples

- equalities and disequalities over the reals

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$

## Motivation for extending SAT to first-order theories

predicates instead of propositional variables

### Examples

- equalities and disequalities over the reals

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$

- boolean combination of linear-arithmethic predicates

$$(x_1 + 2x_3 < 5) \vee \neg(x_3 \leq 1) \wedge (x_1 \geq x_3)$$



## Motivation for extending SAT to first-order theories

predicates instead of propositional variables

### Examples

- equalities and disequalities over the reals

$$(x_1 = x_2 \vee x_1 = x_3) \wedge (x_1 = x_2 \vee x_1 = x_4) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4$$

- boolean combination of linear-arithmethic predicates

$$(x_1 + 2x_3 < 5) \vee \neg(x_3 \leq 1) \wedge (x_1 \geq x_3)$$

- formula over arrays

$$(i = j \wedge a[j] = 1) \wedge \neg(a[i] = 1)$$

## Important Concepts

- assumption literal (clause selector)
- binary search
- branch and bound
- hard and soft clause
- incremental SAT solving
- maximal satisfiability
- minUNSAT
- Shakashaka gadget