

Computer-Aided Reasoning for Software

# CSSE507

## **A Modern SAT Solver**

**Emina Torlak**

[emina@cs.washington.edu](mailto:emina@cs.washington.edu)

# Today

## Last lecture

- Review of propositional logic and the DPLL algorithm

## Today

- The CDCL algorithm at the core of modern SAT solvers:
  - 3 important extensions of DPLL
  - Engineering matters

# A brief review of DPLL

```
// Returns true if the CNF formula F is  
// satisfiable; otherwise returns false.
```

```
DPLL(F)
```

```
  G ← BCP(F)
```

```
  if G =  $\top$  then return true
```

```
  if G =  $\perp$  then return false
```

```
  p ← choose(vars(G))
```

```
  return DPLL(G{p  $\mapsto$   $\top$ }) ||
```

```
    DPLL(G{p  $\mapsto$   $\perp$ })
```

**Boolean constraint propagation** applies *unit resolution* until fixed point:

$$\frac{\beta \quad b_1 \vee \dots \vee b_m \vee \neg\beta}{b_1 \vee \dots \vee b_m}$$

$$\frac{\beta \quad b_1 \vee \dots \vee b_m \vee \beta}{\top}$$

# A brief review of DPLL

```
// Returns true if the CNF formula F is  
// satisfiable; otherwise returns false.
```

```
DPLL(F)  
  G ← BCP(F)  
  if G =  $\top$  then return true  
  if G =  $\perp$  then return false  
  p ← choose(vars(G))  
  return DPLL(G{p  $\mapsto$   $\top$ }) ||  
         DPLL(G{p  $\mapsto$   $\perp$ })
```

**Boolean constraint propagation** applies *unit resolution* until fixed point:

$$\frac{\beta \quad b_1 \vee \dots \vee b_m \vee \neg\beta}{b_1 \vee \dots \vee b_m}$$

$$\frac{\beta \quad b_1 \vee \dots \vee b_m \vee \beta}{\top}$$

Okay for randomly generated CNFs, but not for practical ones. Why?

# A brief review of DPLL

```
// Returns true if the CNF formula F is  
// satisfiable; otherwise returns false.
```

```
DPLL(F)
```

```
  G ← BCP(F)
```

```
  if G =  $\top$  then return true
```

```
  if G =  $\perp$  then return false
```

```
  p ← choose(vars(G))
```

```
  return DPLL(G{p ↦  $\top$ }) ||
```

```
    DPLL(G{p ↦  $\perp$ })
```

**No learning:** throws away all the work performed to conclude that the current partial assignment (PA) is bad. Revisits bad PAs that lead to conflict due to the same root cause.

# A brief review of DPLL

```
// Returns true if the CNF formula F is  
// satisfiable; otherwise returns false.
```

```
DPLL(F)
```

```
  G ← BCP(F)
```

```
  if G =  $\top$  then return true
```

```
  if G =  $\perp$  then return false
```

```
  p ← choose(vars(G))
```

```
  return DPLL(G{p ↦  $\top$ }) ||
```

```
         DPLL(G{p ↦  $\perp$ })
```

**No learning:** throws away all the work performed to conclude that the current partial assignment (PA) is bad. Revisits bad PAs that lead to conflict due to the same root cause.

## **Chronological backtracking:**

backtracks one level, even if it can be deduced that the current PA became doomed at a lower level.

# A brief review of DPLL

```
// Returns true if the CNF formula F is  
// satisfiable; otherwise returns false.
```

```
DPLL(F)
```

```
  G ← BCP(F)
```

```
  if G =  $\top$  then return true
```

```
  if G =  $\perp$  then return false
```

```
  p ← choose(vars(G))
```

```
  return DPLL(G{p  $\mapsto$   $\top$ }) ||  
         DPLL(G{p  $\mapsto$   $\perp$ })
```

**No learning:** throws away all the work performed to conclude that the current partial assignment (PA) is bad. Revisits bad PAs that lead to conflict due to the same root cause.

**Naive decisions:** picks an arbitrary variable to branch on. Fails to consider the state of the search to make heuristically better decisions.

**Chronological backtracking:** backtracks one level, even if it can be deduced that the current PA became doomed at a lower level.

# Conflict-Driven Clause Learning (CDCL)

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```



# Conflict-Driven Clause Learning (CDCL)

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

**Learning:** F augmented with a conflict clause that summarizes the root cause of the conflict.

# Conflict-Driven Clause Learning (CDCL)

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

**Learning:** F augmented with a conflict clause that summarizes the root cause of the conflict.

**Non-chronological backtracking:** backtracks b levels, based on the cause of the conflict.

# Conflict-Driven Clause Learning (CDCL)

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

**Decision heuristics** choose the next literal to add to the current partial assignment based on the state of the search.

**Learning:** F augmented with a conflict clause that summarizes the root cause of the conflict.

**Non-chronological backtracking:** backtracks b levels, based on the cause of the conflict.

# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

...

# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

...

$x_i @ l$

# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1: \neg x_1 \vee x_2 \vee \neg x_4$

$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

$c_3: \neg x_3 \vee \neg x_4$

$c_4: x_4 \vee x_5 \vee x_6$

$c_5: \neg x_5 \vee x_7$

$c_6: \neg x_6 \vee x_7 \vee \neg x_8$

...

...

$x_8@2$

$x_1@1$

# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

...

$x_8@2$

$x_1@1$

$\neg x_7@3$

# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1: \neg x_1 \vee x_2 \vee \neg x_4$

$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

$c_3: \neg x_3 \vee \neg x_4$

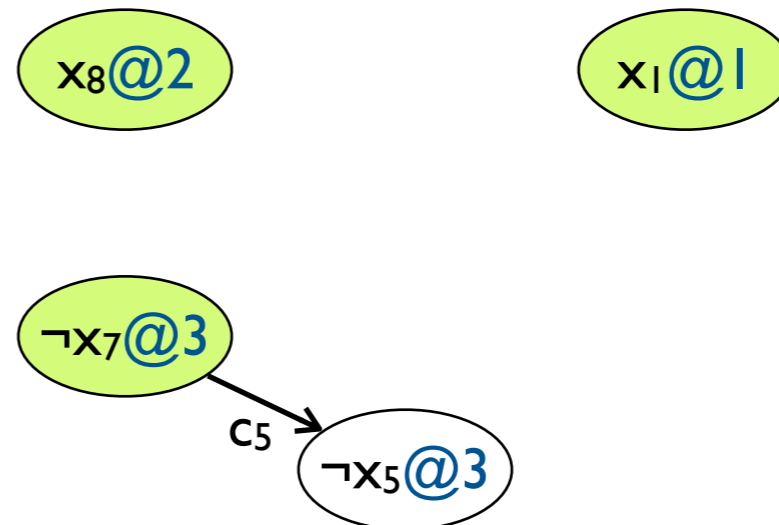
$c_4: x_4 \vee x_5 \vee x_6$

$c_5: \neg x_5 \vee x_7$

$c_6: \neg x_6 \vee x_7 \vee \neg x_8$

...

...





# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1: \neg x_1 \vee x_2 \vee \neg x_4$

$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

$c_3: \neg x_3 \vee \neg x_4$

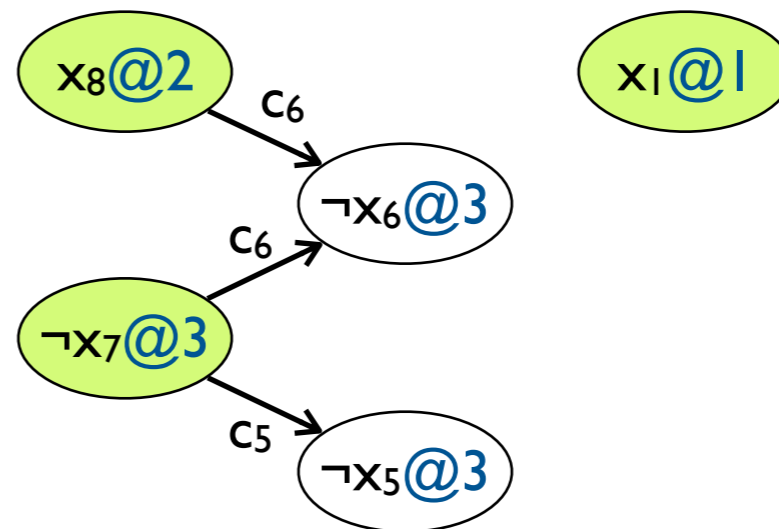
$c_4: x_4 \vee x_5 \vee x_6$

$c_5: \neg x_5 \vee x_7$

$c_6: \neg x_6 \vee x_7 \vee \neg x_8$

...

...



# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
  return true
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1: \neg x_1 \vee x_2 \vee \neg x_4$

$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

$c_3: \neg x_3 \vee \neg x_4$

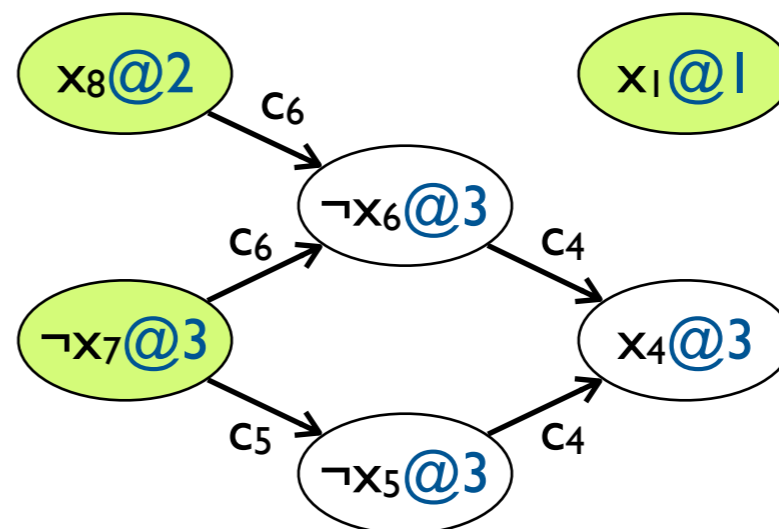
$c_4: x_4 \vee x_5 \vee x_6$

$c_5: \neg x_5 \vee x_7$

$c_6: \neg x_6 \vee x_7 \vee \neg x_8$

...

...



# CDCL by example

```

CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
  
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

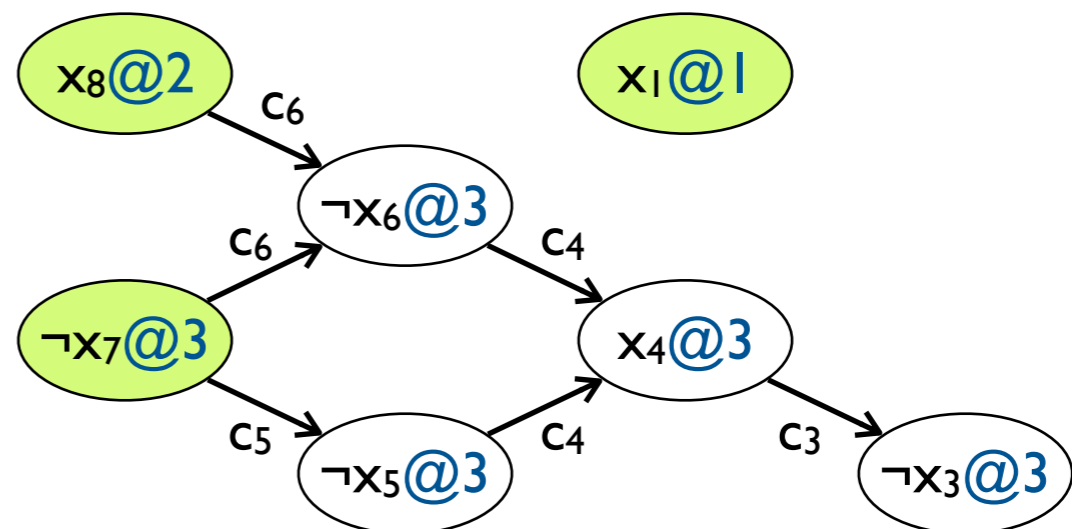
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

...



# CDCL by example

```

CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
  return true
  
```

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1: \neg x_1 \vee x_2 \vee \neg x_4$

$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

$c_3: \neg x_3 \vee \neg x_4$

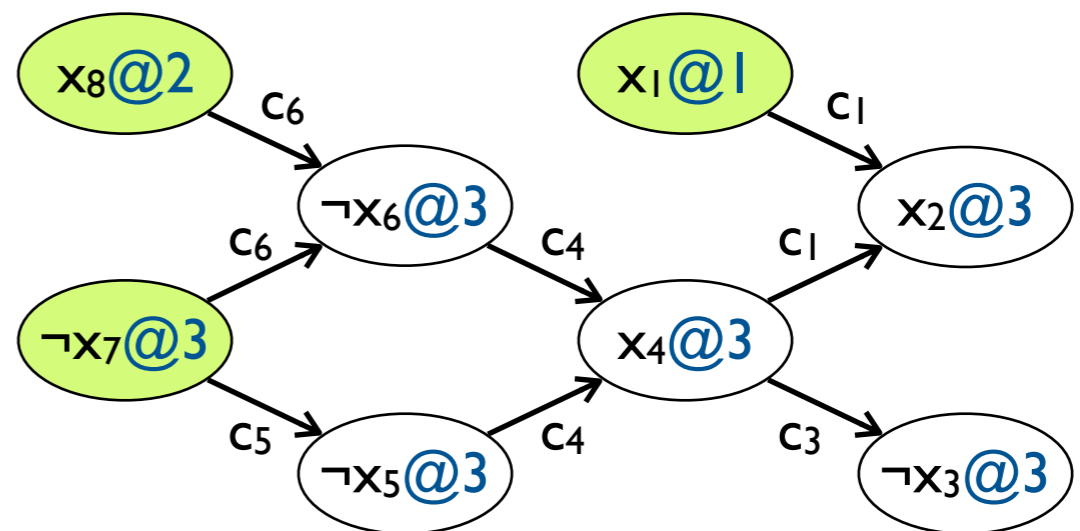
$c_4: x_4 \vee x_5 \vee x_6$

$c_5: \neg x_5 \vee x_7$

$c_6: \neg x_6 \vee x_7 \vee \neg x_8$

...

...



# CDCL by example

CDCL(F)

$A \leftarrow \{ \}$

**if** BCP(F,A) = *conflict* **then return false**

level  $\leftarrow 0$

**while** hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \text{DECIDE}(F,A) \}$

**while** BCP(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \text{ANALYZECONFLICT}()$

$F \leftarrow F \cup \{c\}$

**if**  $b < 0$  **then return false**

**else** BACKTRACK(F,A, b)

level  $\leftarrow b$

**return true**

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

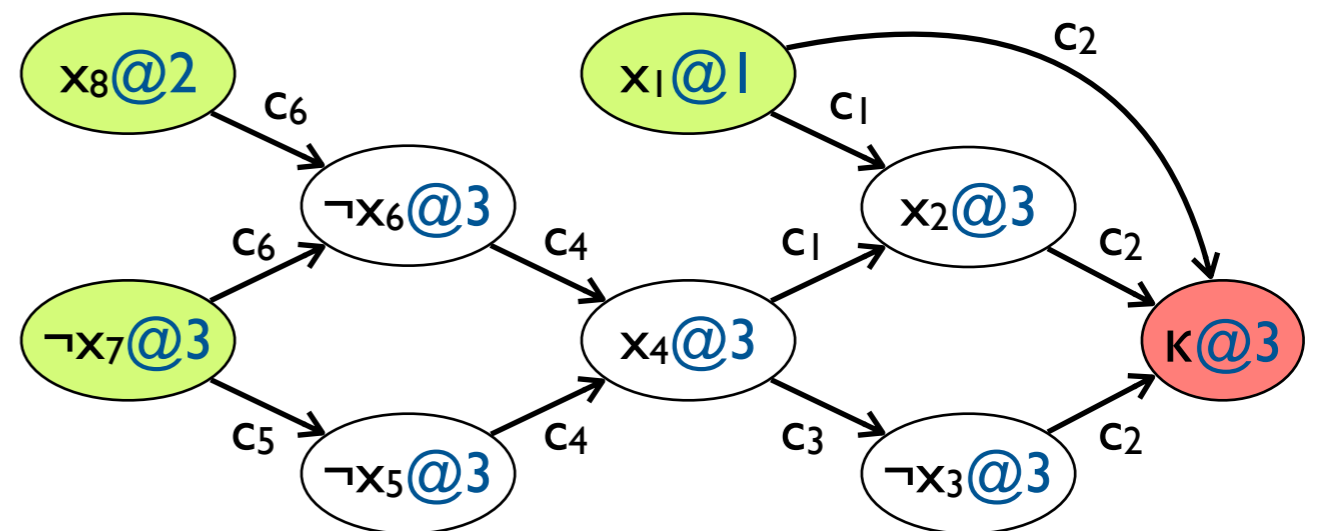
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

...



# CDCL by example

CDCL(F)

$A \leftarrow \{ \}$

**if** **BCP**(F,A) = *conflict* **then return false**

level  $\leftarrow 0$

**while** hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

**while** **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

$F \leftarrow F \cup \{c\}$

**if**  $b < 0$  **then return false**

**else** **BACKTRACK**(F,A, b)

level  $\leftarrow b$

**return true**

$\langle 1, \neg x_1 \vee \neg x_4 \rangle$

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

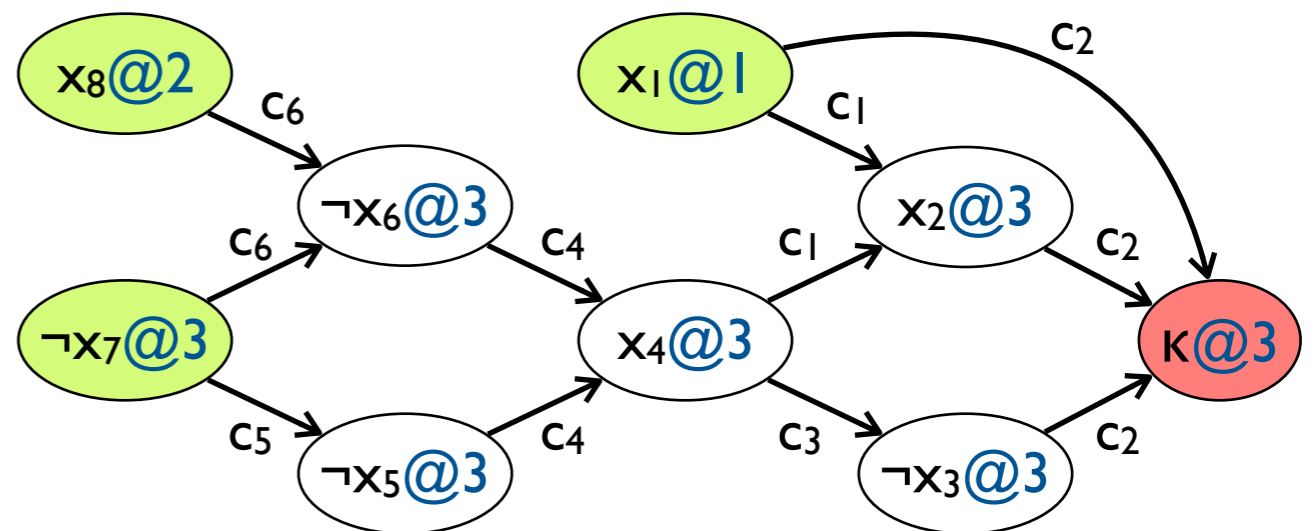
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

...



# CDCL by example

CDCL(F)

$A \leftarrow \{ \}$

**if** **BCP**(F,A) = *conflict* **then return false**

level  $\leftarrow 0$

**while** hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

**while** **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

$F \leftarrow F \cup \{c\}$

**if**  $b < 0$  **then return false**

**else** **BACKTRACK**(F,A, b)

level  $\leftarrow b$

**return true**

$\langle 1, \neg x_1 \vee \neg x_4 \rangle$

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9, c \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

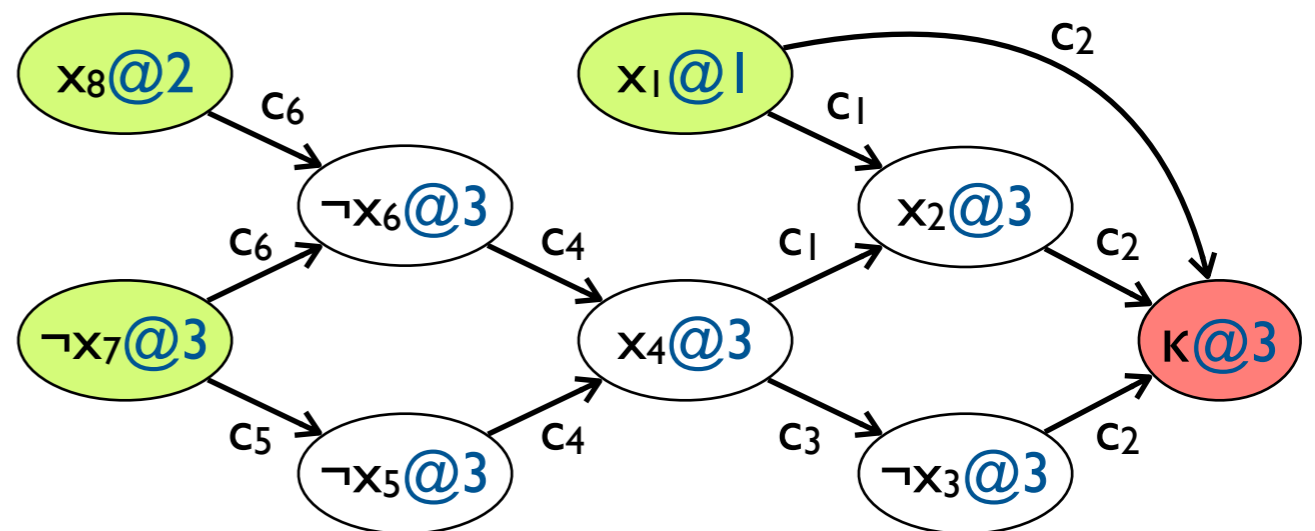
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

$c : \neg x_1 \vee \neg x_4$



# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

⟨1, ¬x<sub>1</sub> ∨ ¬x<sub>4</sub>⟩

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9, c \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

$c : \neg x_1 \vee \neg x_4$

x<sub>1</sub>@1



# CDCL by example

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

⟨1, ¬x<sub>1</sub> ∨ ¬x<sub>4</sub>⟩

$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9, c \}$

$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$

$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$

$c_3 : \neg x_3 \vee \neg x_4$

$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

$c_6 : \neg x_6 \vee x_7 \vee \neg x_8$

...

$c : \neg x_1 \vee \neg x_4$

Conflict clause  
is unit after  
backtracking!

x<sub>1</sub>@1

# CDCL in depth

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

- Definitions
- ANALYZECONFLICT
- DECIDE heuristics
- Implementation

# Basic definitions

Under a given partial assignment (PA), a variable may be

- **assigned** (true/false literal)
- **unassigned**.

$$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$$

$$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$$

$$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$$

...

$$c_8 : x_9 \vee \neg x_2$$

$$c_9 : x_9 \vee x_{10} \vee x_3$$

True literals highlighted in green; false literals highlighted in red.

# Basic definitions

Under a given partial assignment (PA), a variable may be

- **assigned** (true/false literal)
- **unassigned**.

A clause may be

- **satisfied** ( $\geq 1$  true literal)
- **unsatisfied** (all false literals)
- **unit** (one unassigned literal, rest false)
- **unresolved** (otherwise)

$$F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \dots, c_9 \}$$

$$c_1 : \neg x_1 \vee x_2 \vee \neg x_4$$

$$c_2 : \neg x_1 \vee \neg x_2 \vee x_3$$

...

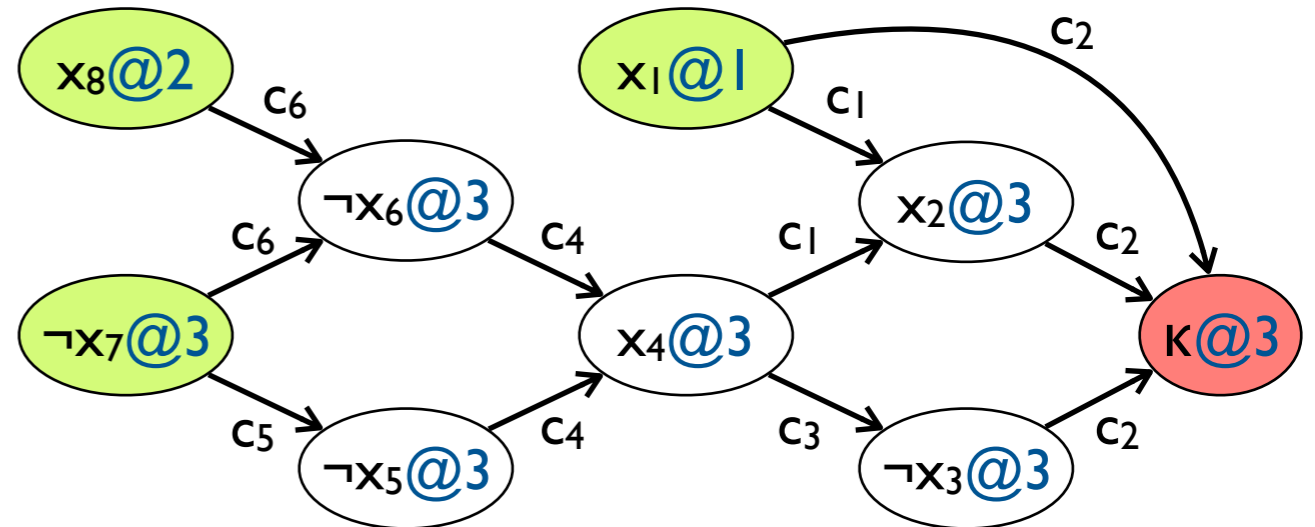
$$c_8 : x_9 \vee \neg x_2$$

$$c_9 : x_9 \vee x_{10} \vee x_3$$

True literals highlighted in green; false literals highlighted in red.

# Implication graph

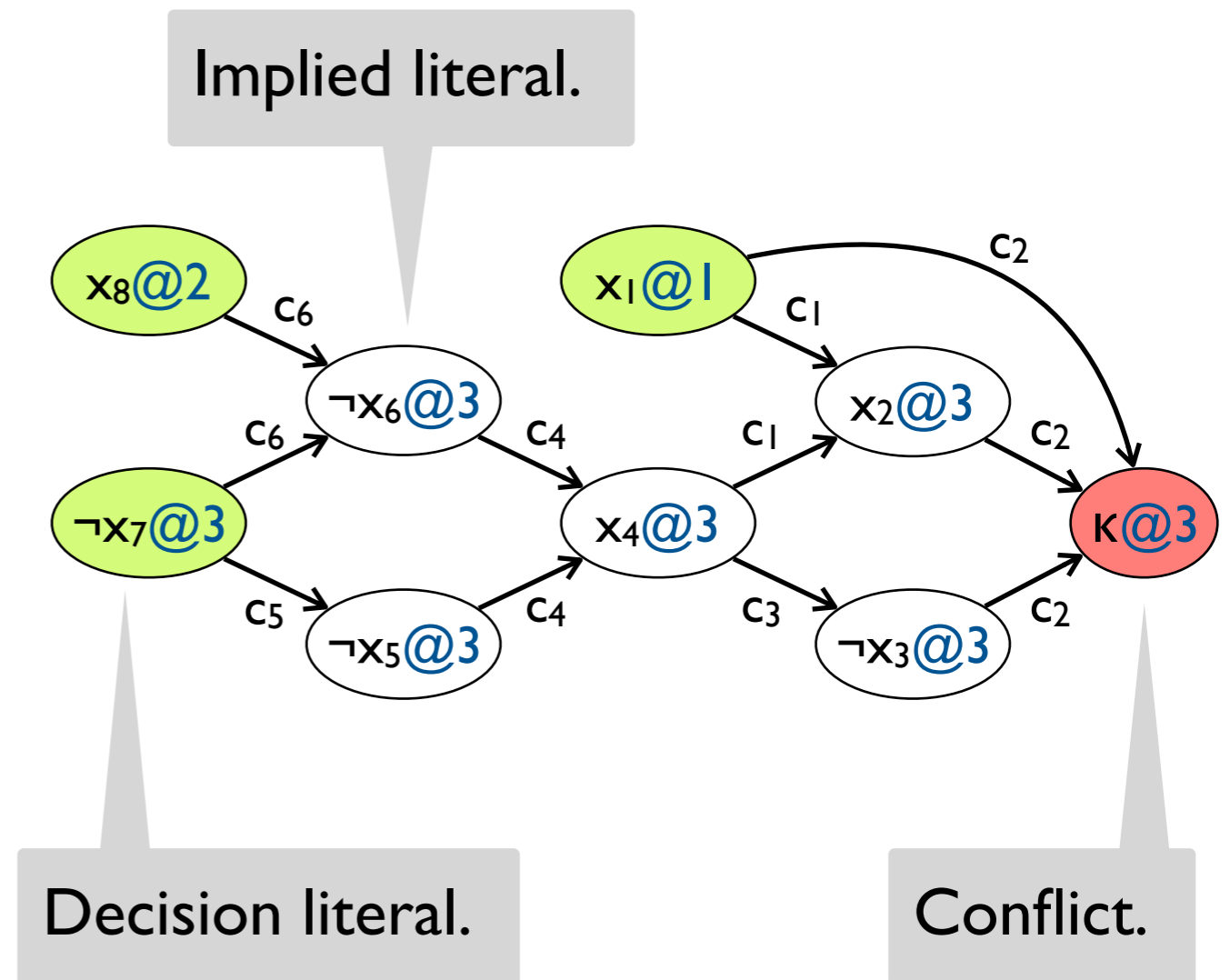
An implication graph  $G = (V, E)$  is a DAG that records the history of decisions and the resulting deductions derived with BCP.



# Implication graph

An implication graph  $G = (V, E)$  is a DAG that records the history of decisions and the resulting deductions derived with BCP.

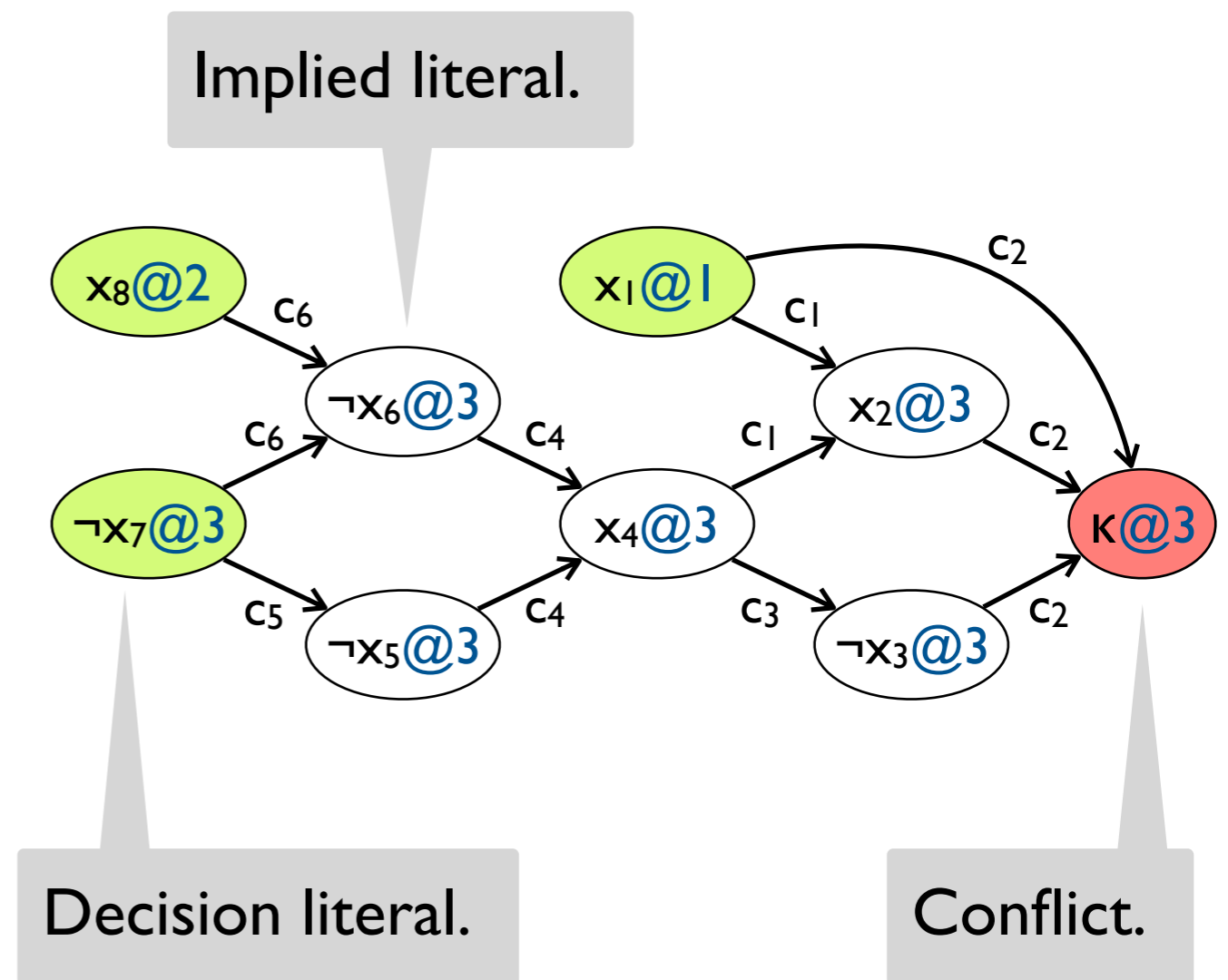
- $v \in V$  is a literal (or  $\kappa$ ) and the decision level at which it entered the current PA.



# Implication graph

An implication graph  $G = (V, E)$  is a DAG that records the history of decisions and the resulting deductions derived with BCP.

- $v \in V$  is a literal (or  $\kappa$ ) and the decision level at which it entered the current PA.
- $\langle v, w \rangle \in E$  iff  $v \neq w$ ,  $\neg v \in \text{antecedent}(w)$ , and  $\langle v, w \rangle$  is labeled with  $\text{antecedent}(w)$

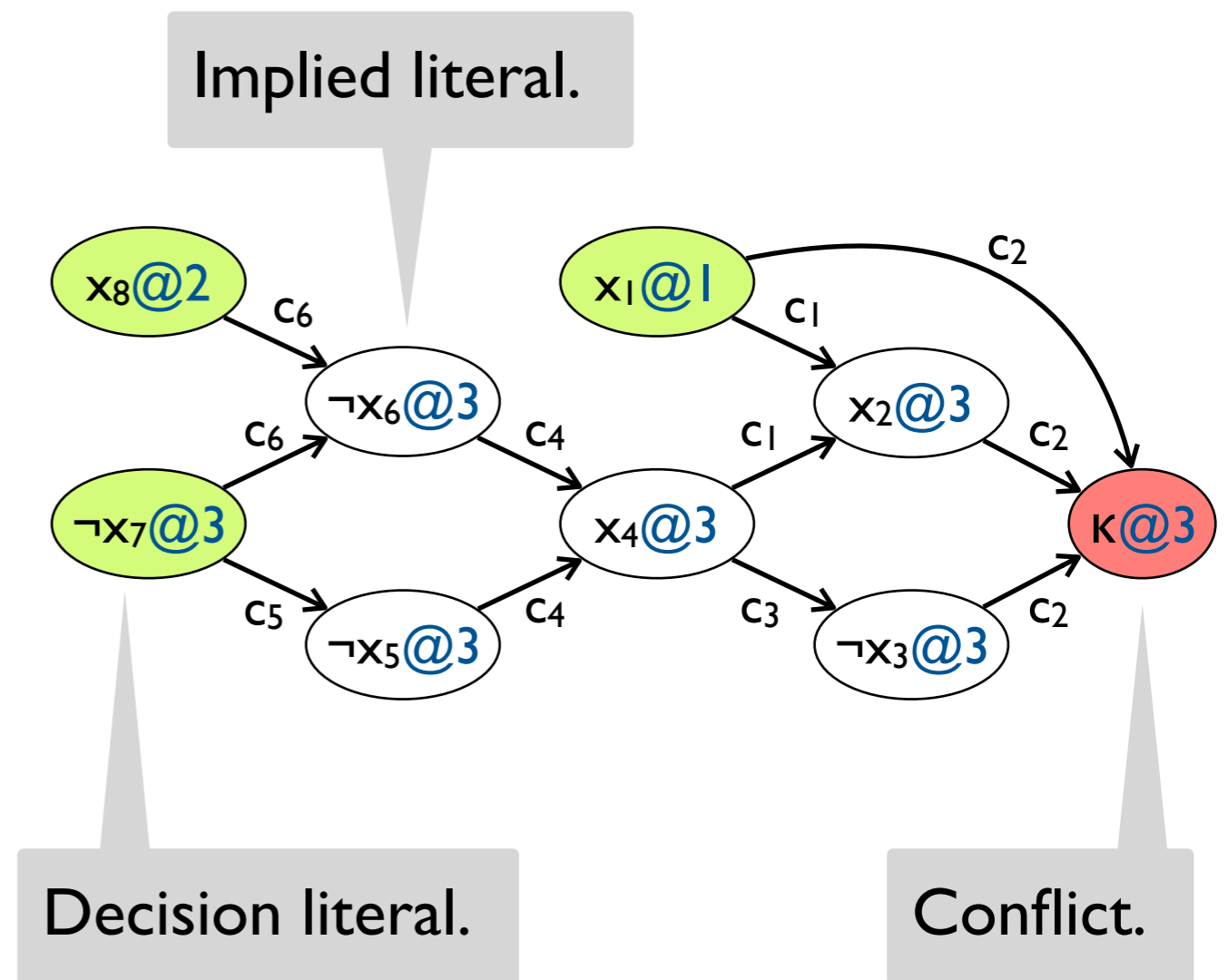


# Implication graph

An implication graph  $G = (V, E)$  is a DAG that records the history of decisions and the resulting deductions derived with BCP.

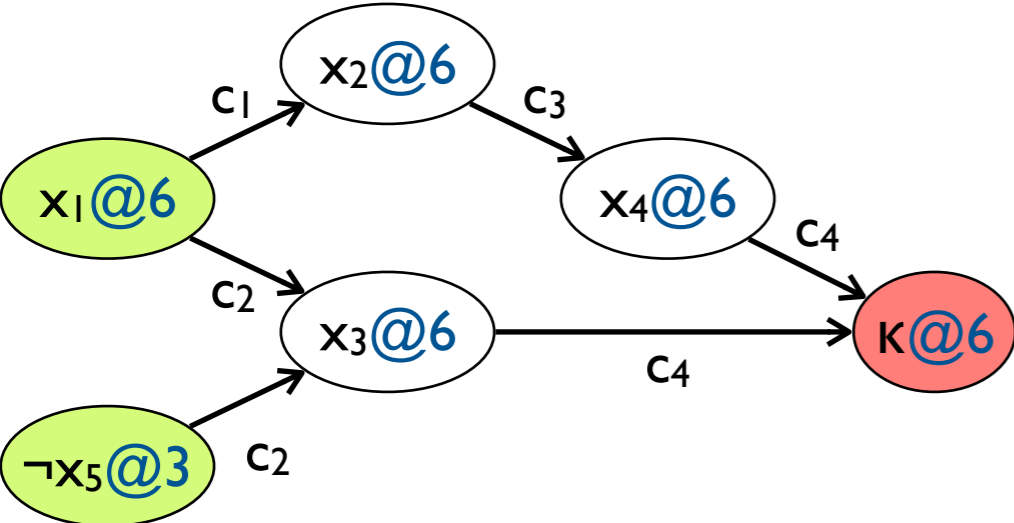
- $v \in V$  is a literal (or  $\kappa$ ) and the decision level at which it entered the current PA.
- $\langle v, w \rangle \in E$  iff  $v \neq w$ ,  $\neg v \in \text{antecedent}(w)$ , and  $\langle v, w \rangle$  is labeled with  $\text{antecedent}(w)$

A unit clause  $c$  is the antecedent of its sole unassigned literal.



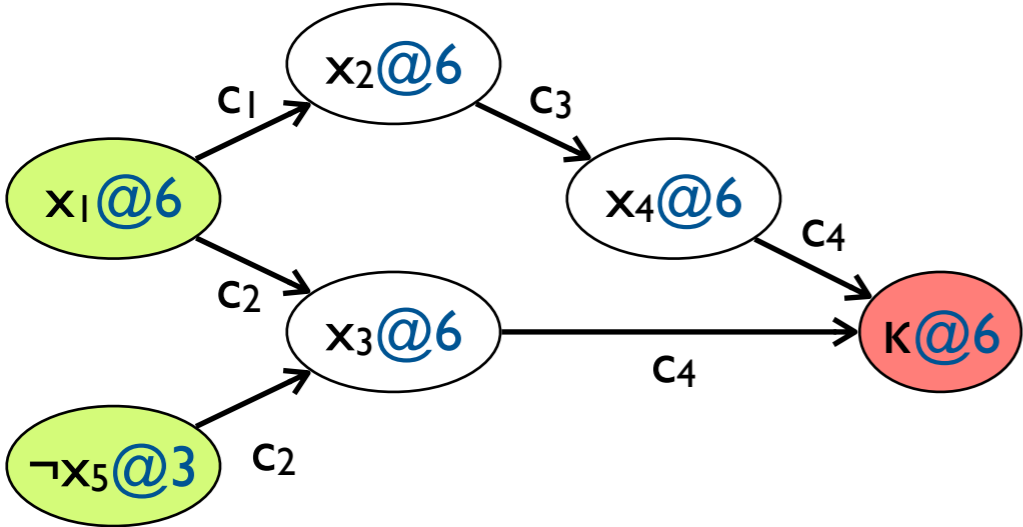


# Implication graph: a quick exercise



What clauses gave rise to this implication graph?

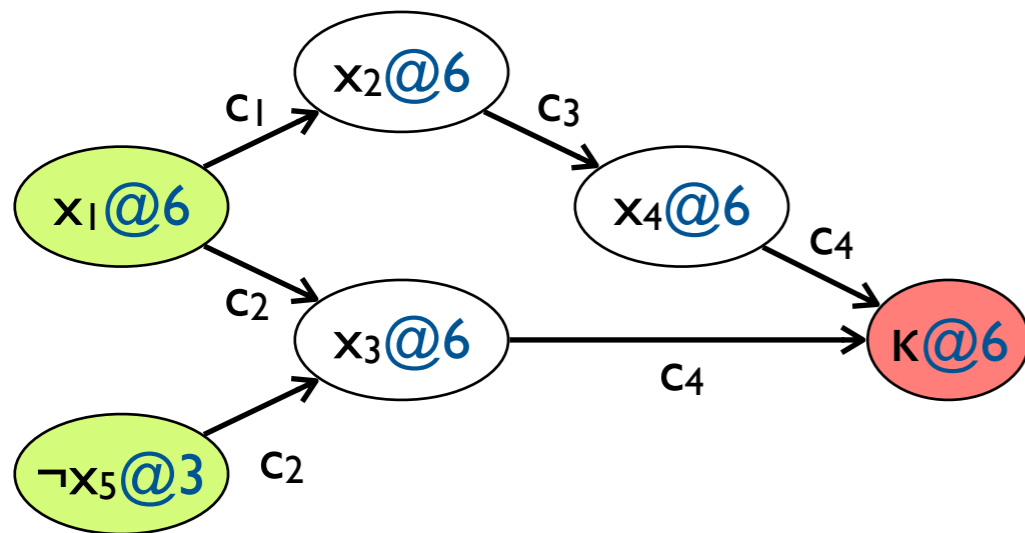
# Implication graph: a quick exercise



What clauses gave rise to this implication graph?

$$c_1 : \neg x_1 \vee x_2$$

# Implication graph: a quick exercise

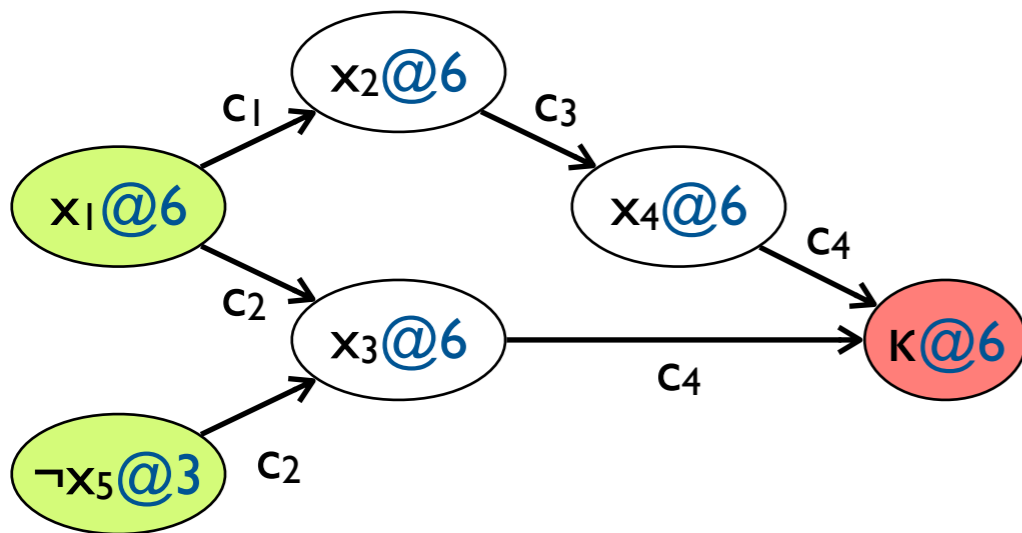


What clauses gave rise to this implication graph?

$$c_1 : \neg x_1 \vee x_2$$

$$c_2 : \neg x_1 \vee x_3 \vee x_5$$

# Implication graph: a quick exercise



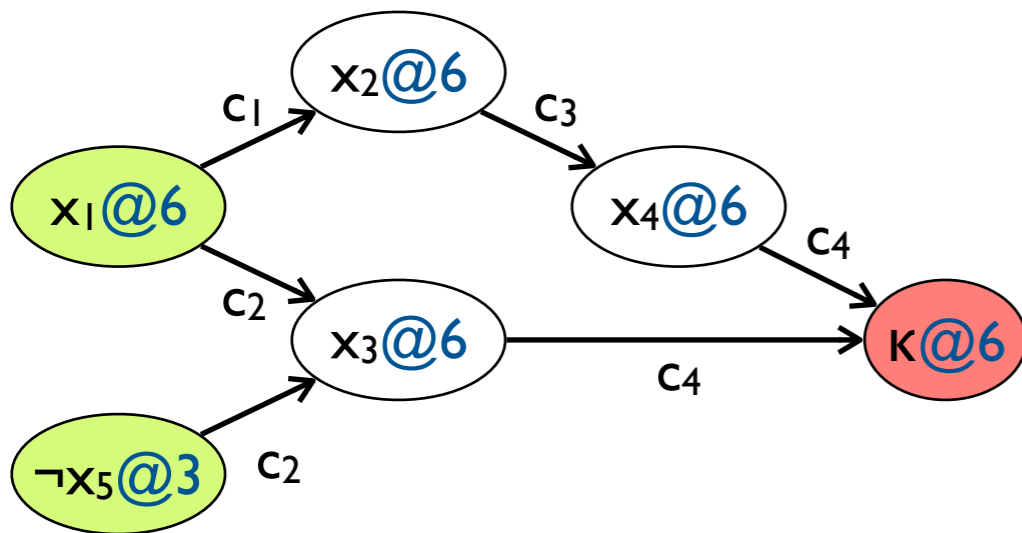
What clauses gave rise to this implication graph?

$$c_1 : \neg x_1 \vee x_2$$

$$c_2 : \neg x_1 \vee x_3 \vee x_5$$

$$c_3 : \neg x_2 \vee x_4$$

# Implication graph: a quick exercise



What clauses gave rise to this implication graph?

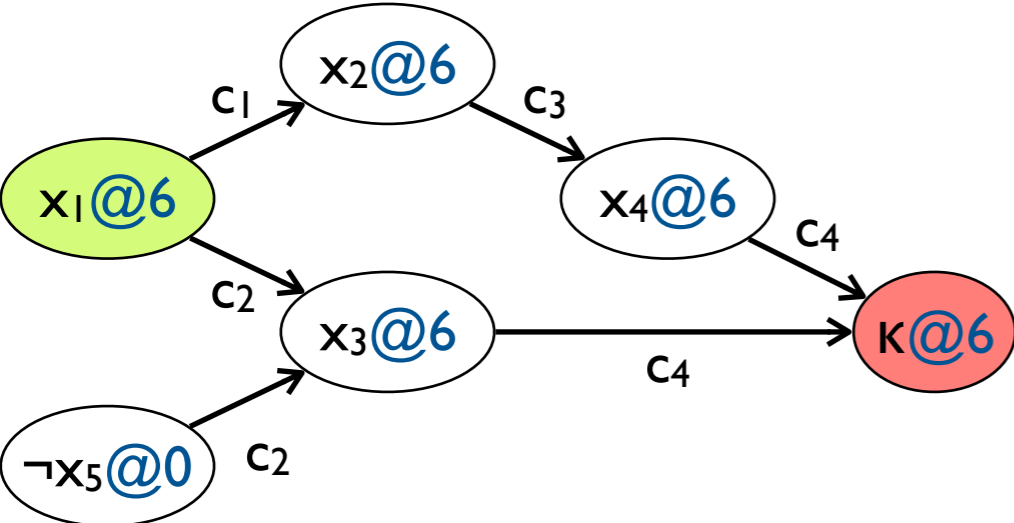
$$c_1 : \neg x_1 \vee x_2$$

$$c_2 : \neg x_1 \vee x_3 \vee x_5$$

$$c_3 : \neg x_2 \vee x_4$$

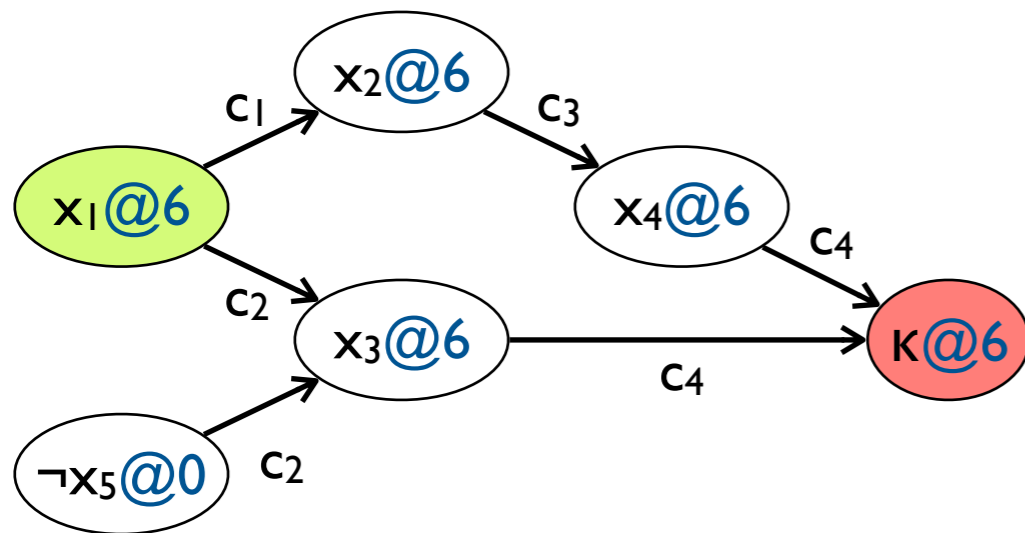
$$c_4 : \neg x_3 \vee \neg x_4$$

# Implication graph: an even quicker exercise



What clauses gave rise to this implication graph?

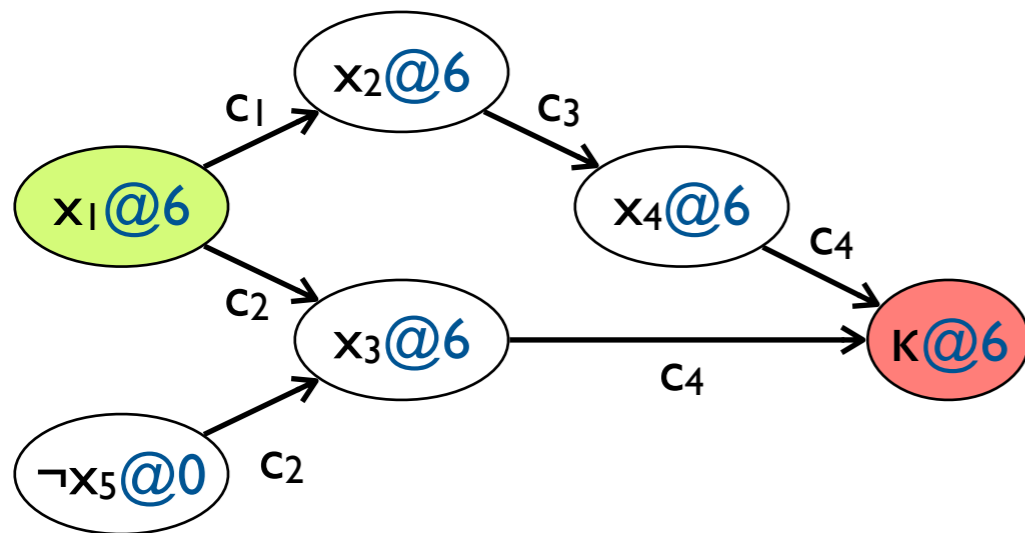
# Implication graph: an even quicker exercise



What clauses gave rise to this implication graph?

$$c_1 : \neg x_1 \vee x_2$$

# Implication graph: an even quicker exercise



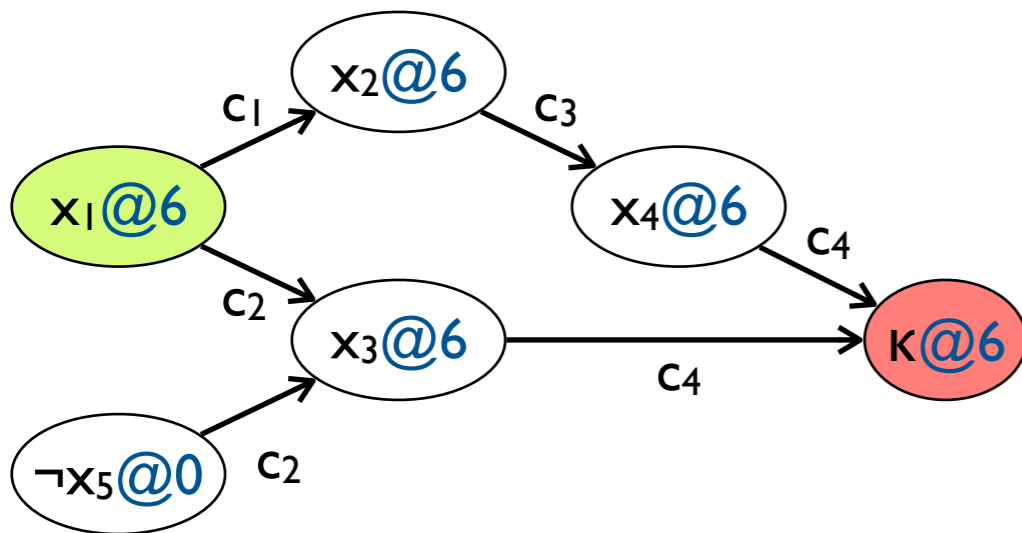
What clauses gave rise to this implication graph?

$$c_1 : \neg x_1 \vee x_2$$

$$c_2 : \neg x_1 \vee x_3 \vee x_5$$



# Implication graph: an even quicker exercise



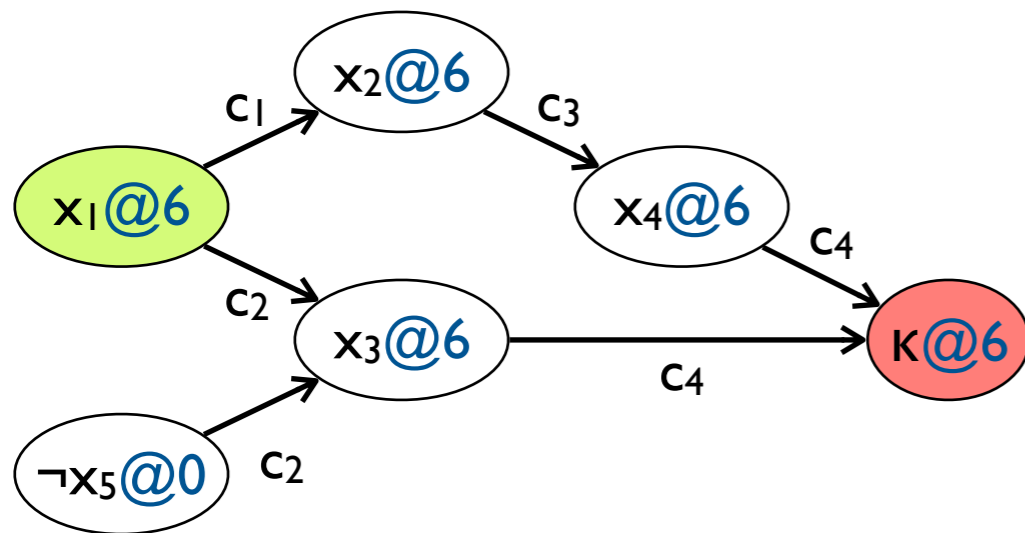
What clauses gave rise to this implication graph?

$$c_1 : \neg x_1 \vee x_2$$

$$c_2 : \neg x_1 \vee x_3 \vee x_5$$

$$c_3 : \neg x_2 \vee x_4$$

# Implication graph: an even quicker exercise



What clauses gave rise to this implication graph?

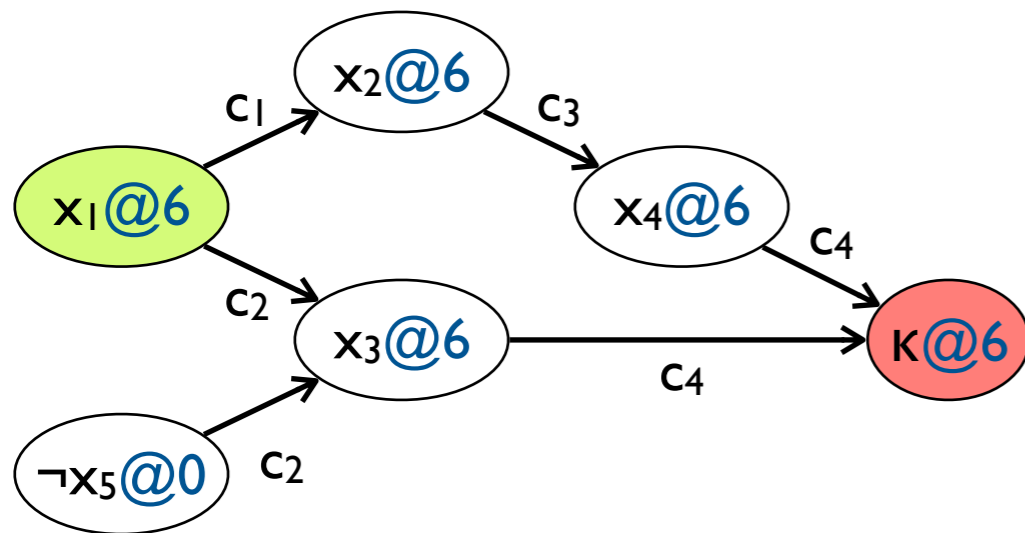
$$c_1 : \neg x_1 \vee x_2$$

$$c_2 : \neg x_1 \vee x_3 \vee x_5$$

$$c_3 : \neg x_2 \vee x_4$$

$$c_4 : \neg x_3 \vee \neg x_4$$

# Implication graph: an even quicker exercise



Assignments at ground (0) level are implied by unary clauses.

What clauses gave rise to this implication graph?

$$c_1 : \neg x_1 \vee x_2$$

$$c_2 : \neg x_1 \vee x_3 \vee x_5$$

$$c_3 : \neg x_2 \vee x_4$$

$$c_4 : \neg x_3 \vee \neg x_4$$

$$c_k : \neg x_5$$

# Using an implication graph to analyze a conflict

CDCL(F)

$A \leftarrow \{\}$

if **BCP**(F,A) = *conflict* then return *false*

level  $\leftarrow 0$

while hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

while **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

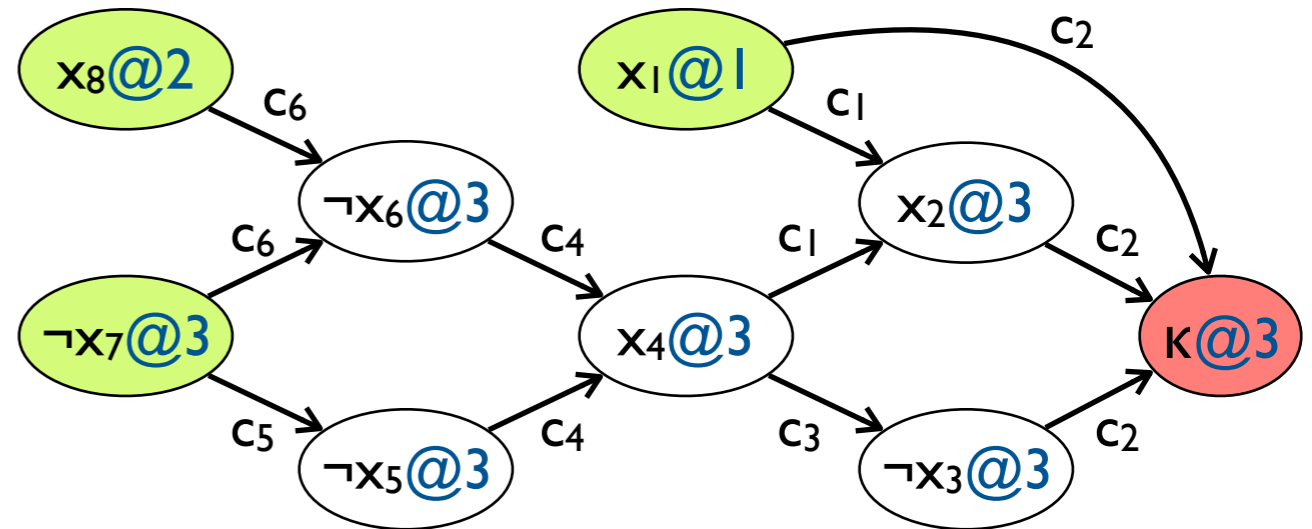
$F \leftarrow F \cup \{c\}$

if  $b < 0$  then return *false*

else **BACKTRACK**(F,A, b)

level  $\leftarrow b$

return *true*



A conflict clause is implied by F and it blocks partial assignments (PAs) that lead to the current conflict.

# Using an implication graph to analyze a conflict

CDCL(F)

$A \leftarrow \{\}$

if **BCP**(F,A) = *conflict* then return *false*

level  $\leftarrow 0$

while hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

while **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

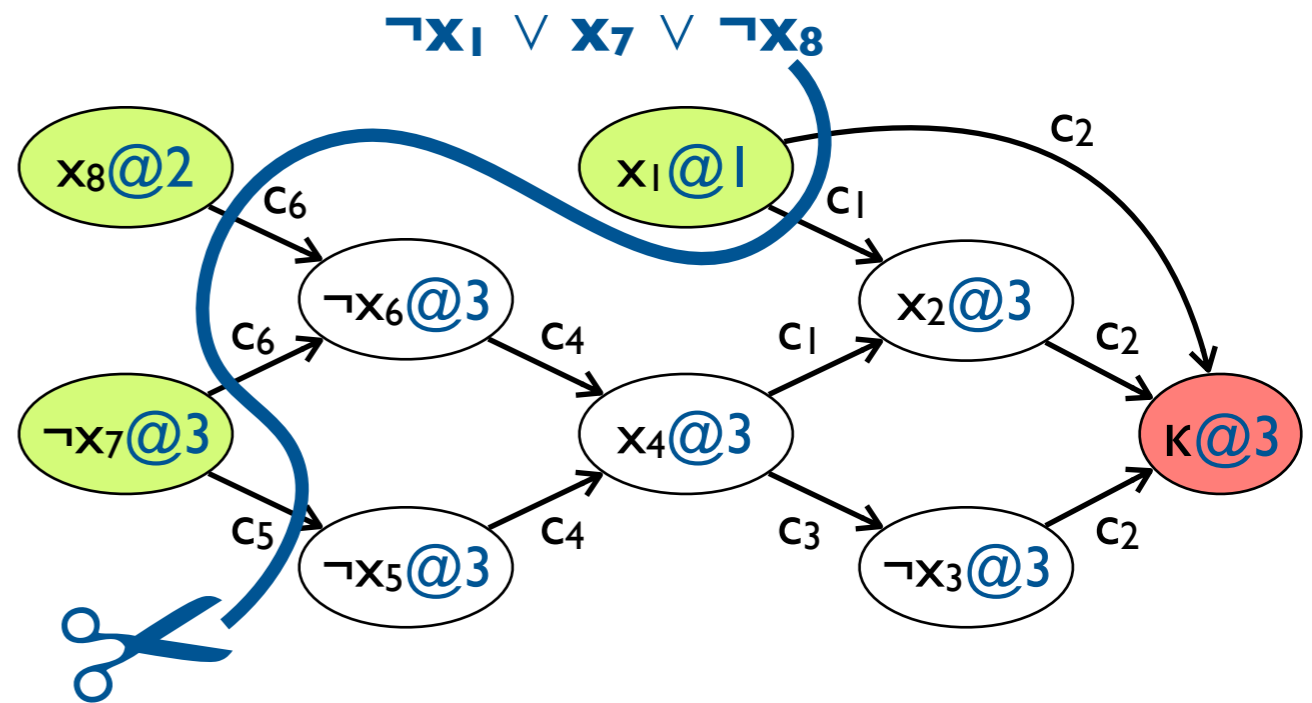
$F \leftarrow F \cup \{c\}$

if  $b < 0$  then return *false*

else **BACKTRACK**(F,A, b)

level  $\leftarrow b$

return *true*



A conflict clause is implied by F and it blocks partial assignments (PAs) that lead to the current conflict.

Every cut that separates sources from the sink defines a valid conflict clause.

# Using an implication graph to analyze a conflict

CDCL(F)

$A \leftarrow \{\}$

if **BCP**(F,A) = *conflict* then return *false*

level  $\leftarrow 0$

while hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

while **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

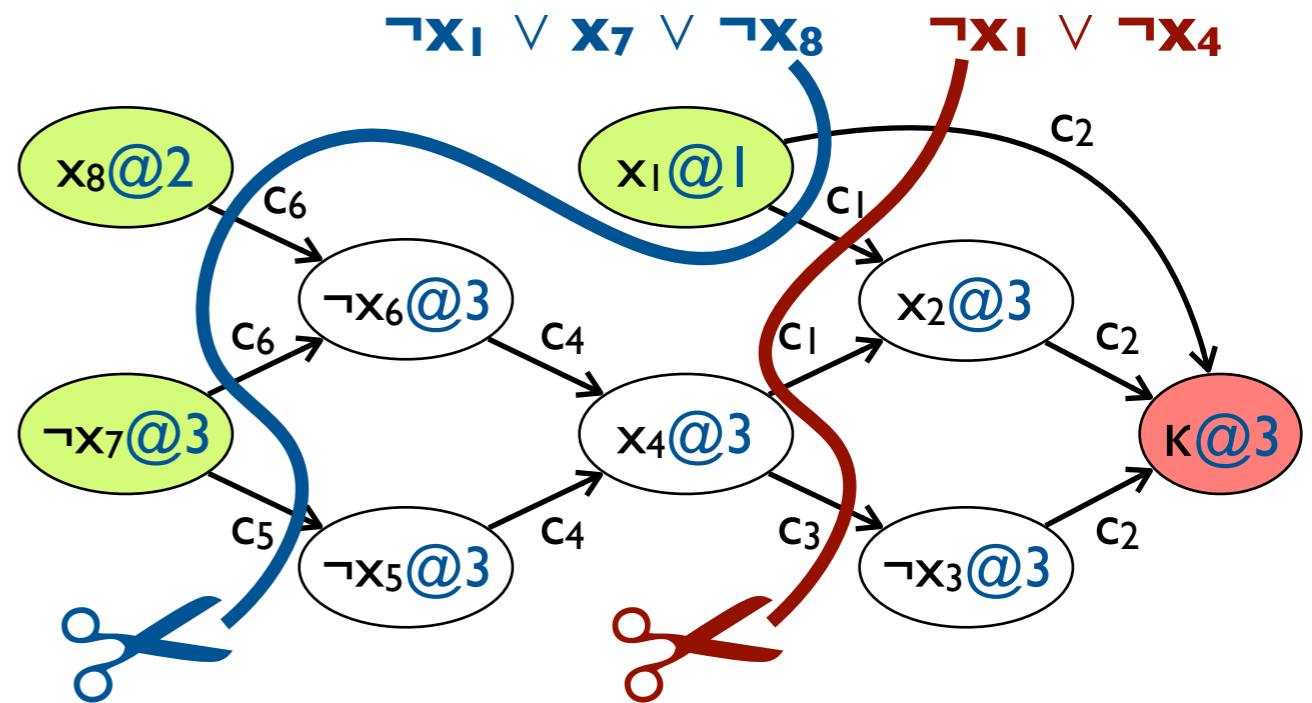
$F \leftarrow F \cup \{c\}$

if  $b < 0$  then return *false*

else **BACKTRACK**(F,A, b)

level  $\leftarrow b$

return *true*



A conflict clause is implied by F and it blocks partial assignments (PAs) that lead to the current conflict.

Every cut that separates sources from the sink defines a valid conflict clause.

# Using an implication graph to analyze a conflict

CDCL(F)

$A \leftarrow \{\}$

if **BCP**(F,A) = *conflict* then return *false*

level  $\leftarrow 0$

while hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

while **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

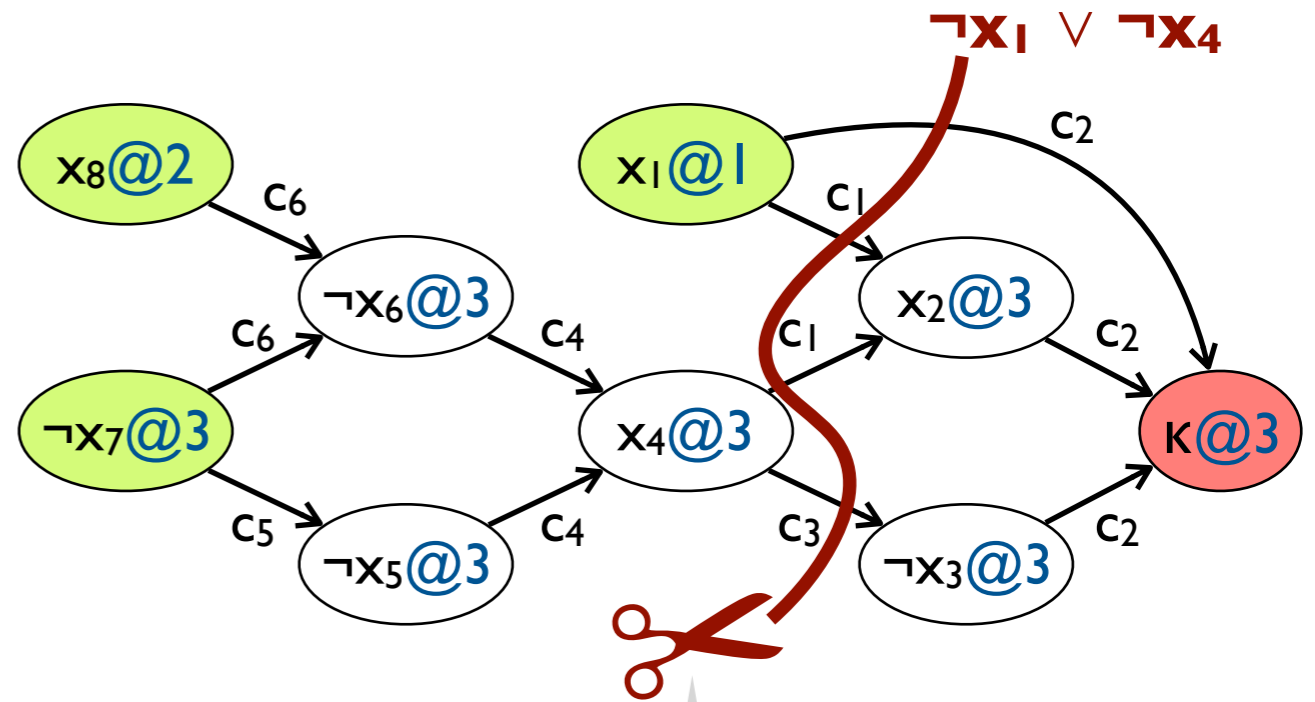
$F \leftarrow F \cup \{c\}$

if  $b < 0$  then return *false*

else **BACKTRACK**(F,A, b)

level  $\leftarrow b$

return *true*

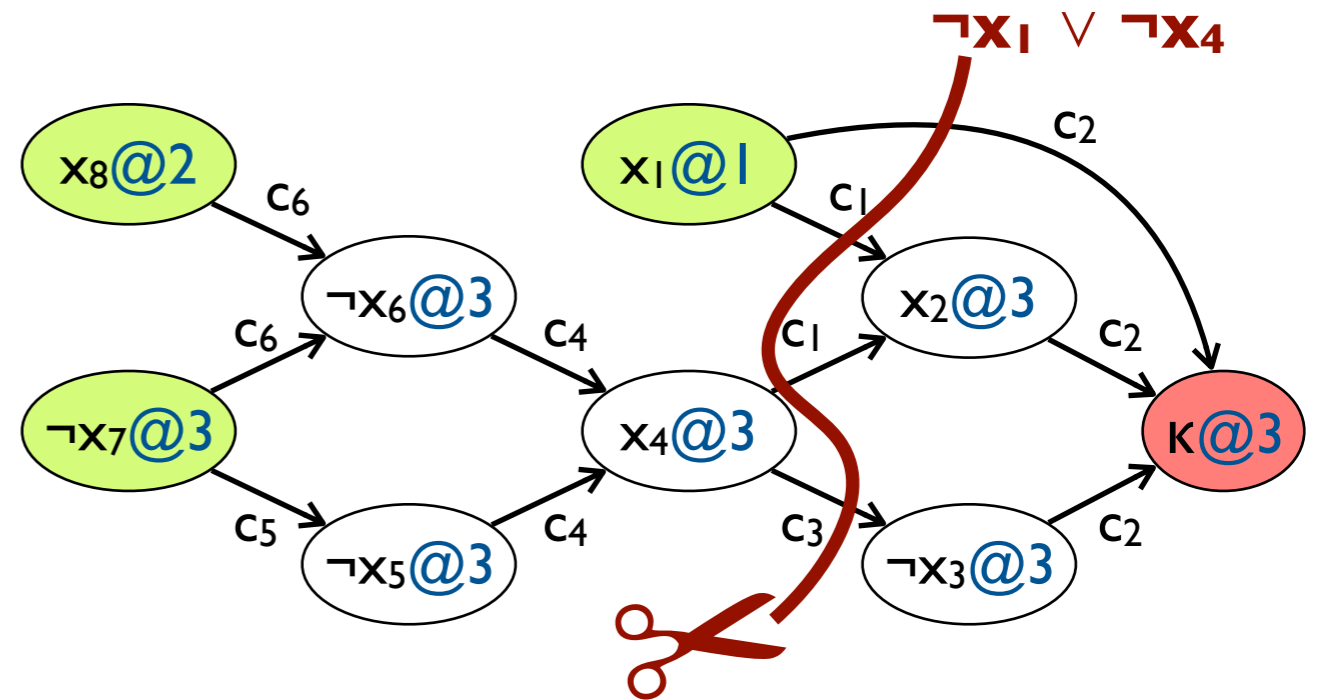


Cut after the **first unique implication point** to get the shortest conflict clause.

# Unique implication points (UIPs)

A unique implication point (UIP) is any node in the implication graph other than the conflict that is on all paths from the current decision literal ( $lit@d$ ) to the conflict ( $\kappa@d$ ).

A first UIP is the UIP that is closest to the conflict.

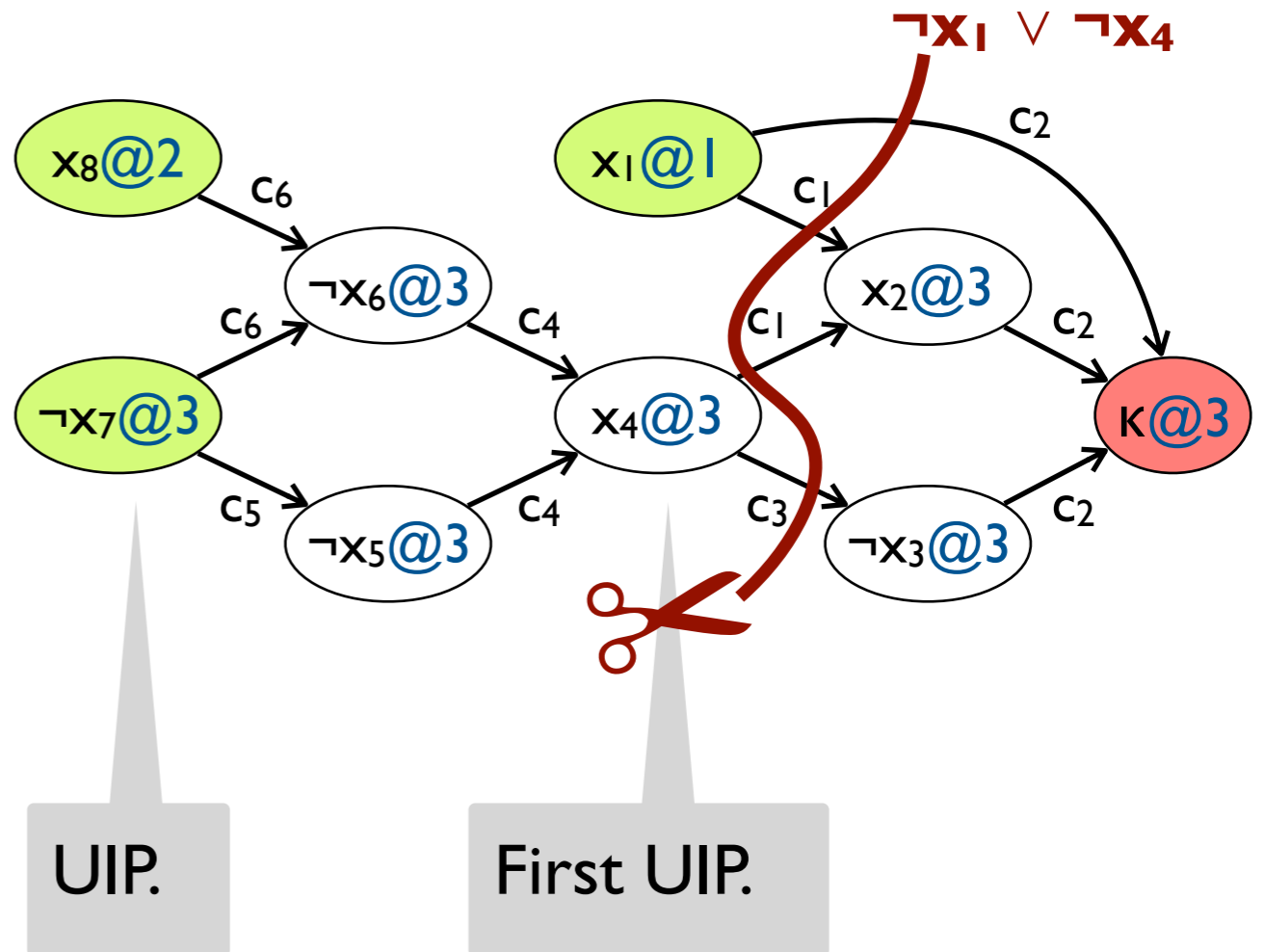




# Unique implication points (UIPs)

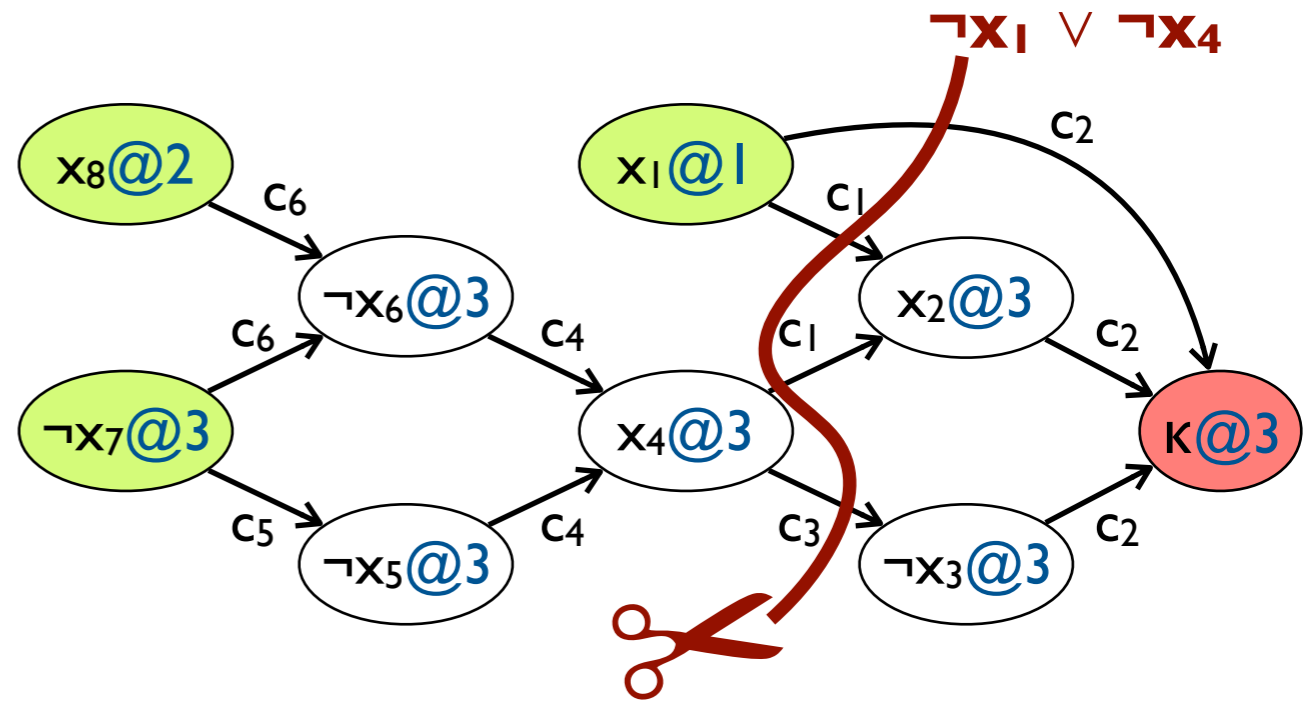
A unique implication point (UIP) is any node in the implication graph other than the conflict that is on all paths from the current decision literal (lit@d) to the conflict (κ@d).

A first UIP is the UIP that is closest to the conflict.



# ANALYZECONFLICT: computing the conflict clause

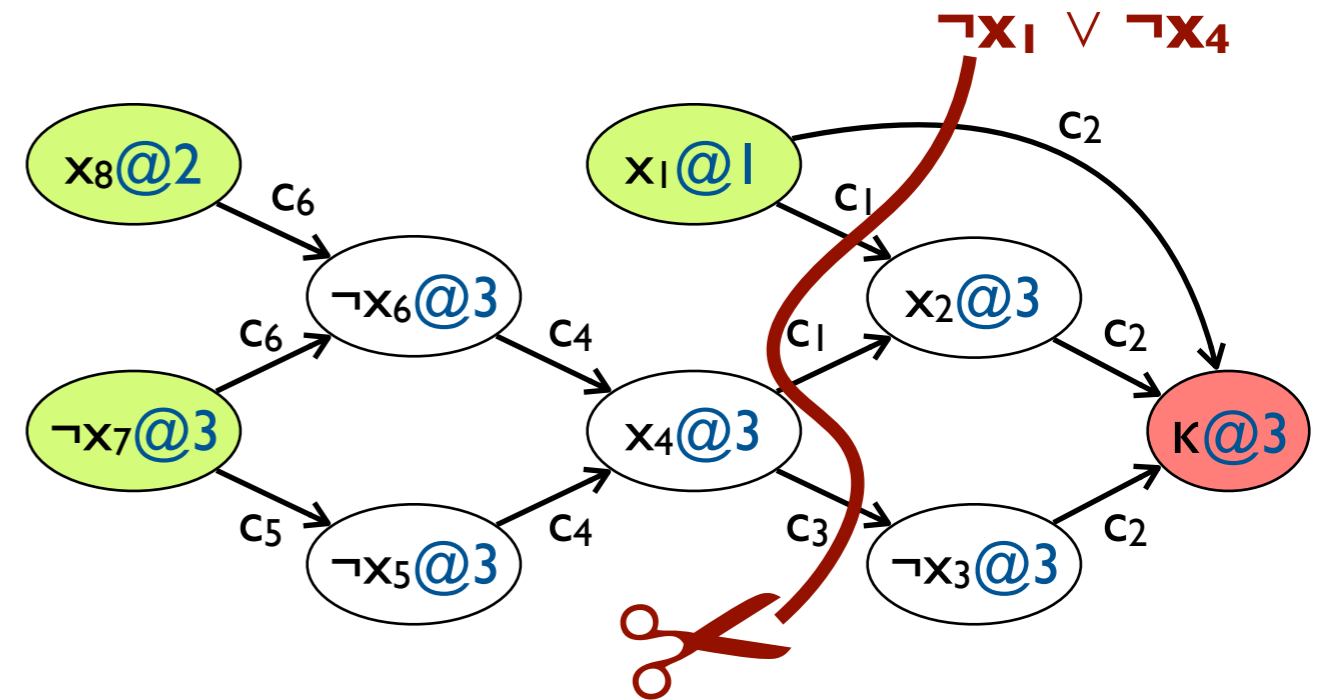
```
ANALYZECONFLICT()
  d ← level(conflict)
  if d = 0 then return -1
  c ← antecedent(conflict)
  while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
  b ← ...
  return ⟨b, c⟩
```



# ANALYZECONFLICT: computing the conflict clause

```

ANALYZECONFLICT()
  d ← level(conflict)
  if d = 0 then return -1
  c ← antecedent(conflict)
  while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
  b ← ...
  return ⟨b, c⟩
    
```



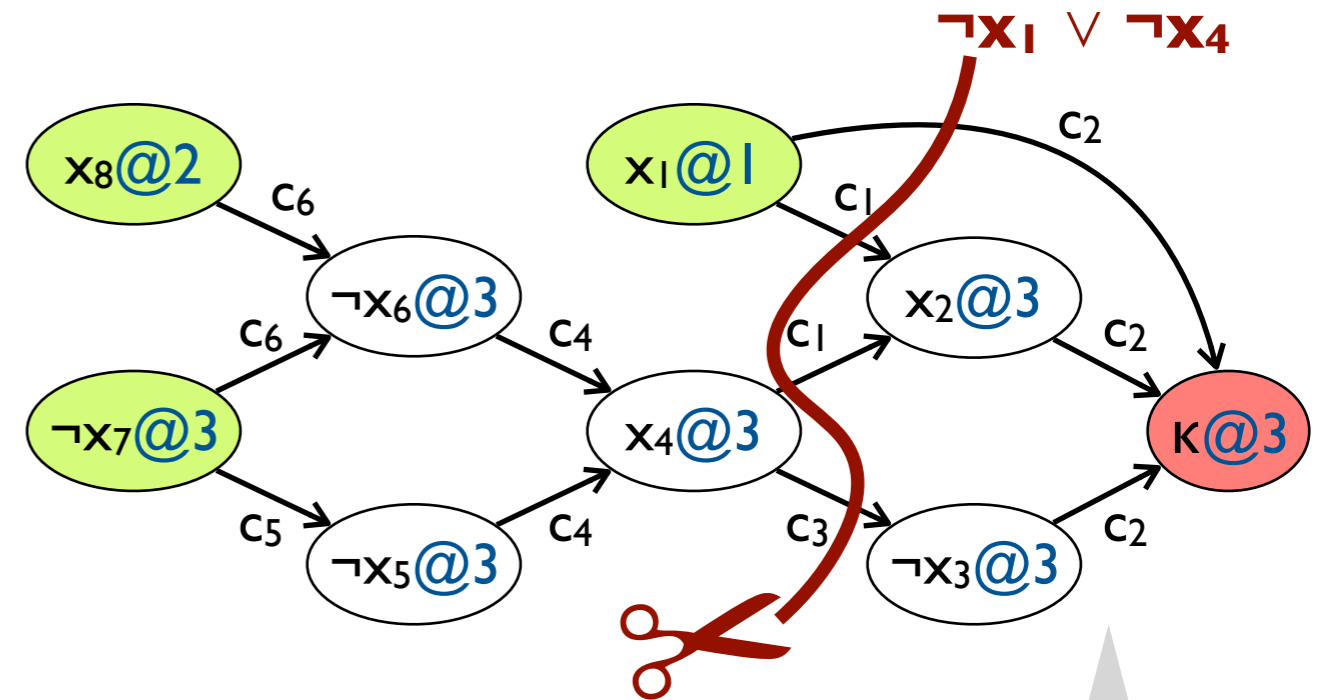
## Binary resolution rule

$$\frac{(a_1 \vee \dots \vee a_n \vee \beta) \quad (b_1 \vee \dots \vee b_m \vee \neg\beta)}{(a_1 \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_m)}$$

# ANALYZECONFLICT: computing the conflict clause

```

ANALYZECONFLICT()
  d ← level(conflict)
  if d = 0 then return -1
  c ← antecedent(conflict)
  while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
  b ← ...
  return ⟨b, c⟩
    
```



**Example:**

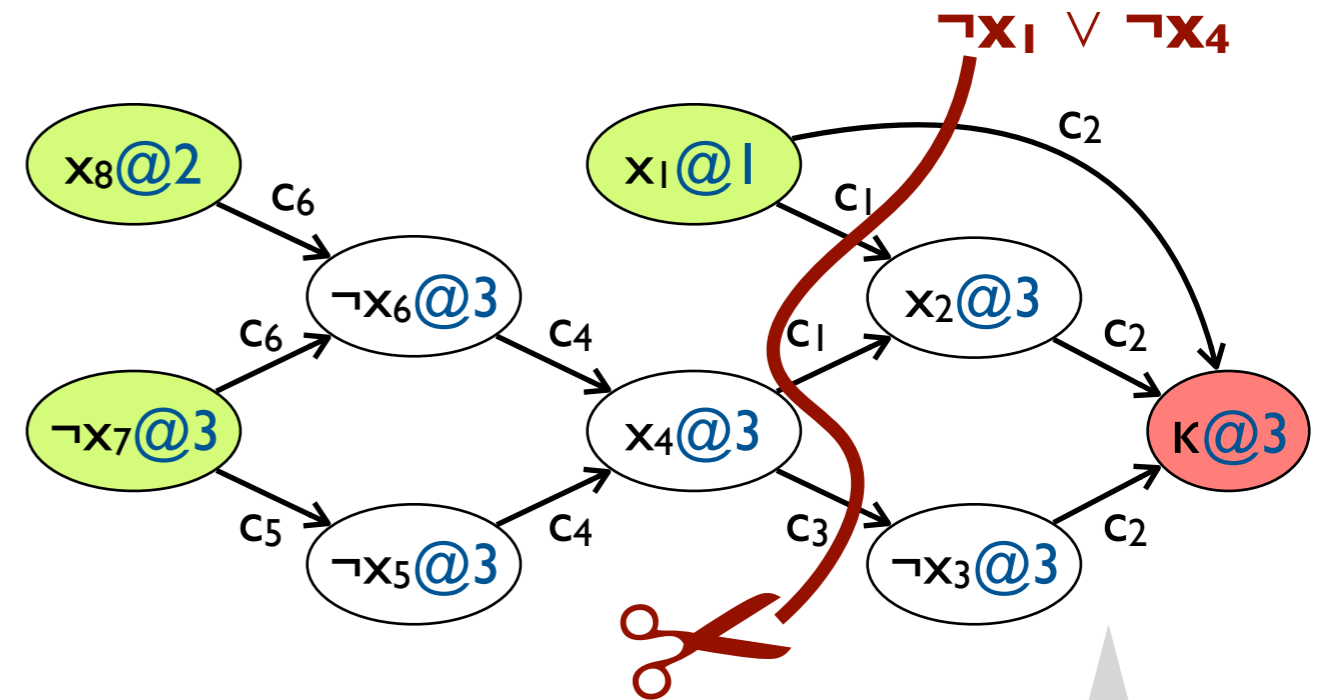
## Binary resolution rule

$$\frac{(a_1 \vee \dots \vee a_n \vee \beta) \quad (b_1 \vee \dots \vee b_m \vee \neg\beta)}{(a_1 \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_m)}$$

# ANALYZECONFLICT: computing the conflict clause

```

ANALYZECONFLICT()
  d ← level(conflict)
  if d = 0 then return -1
  c ← antecedent(conflict)
  while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
  b ← ...
  return ⟨b, c⟩
    
```



## Example:

- $c = c_2, t = x_2, v = x_2, a = c_1$

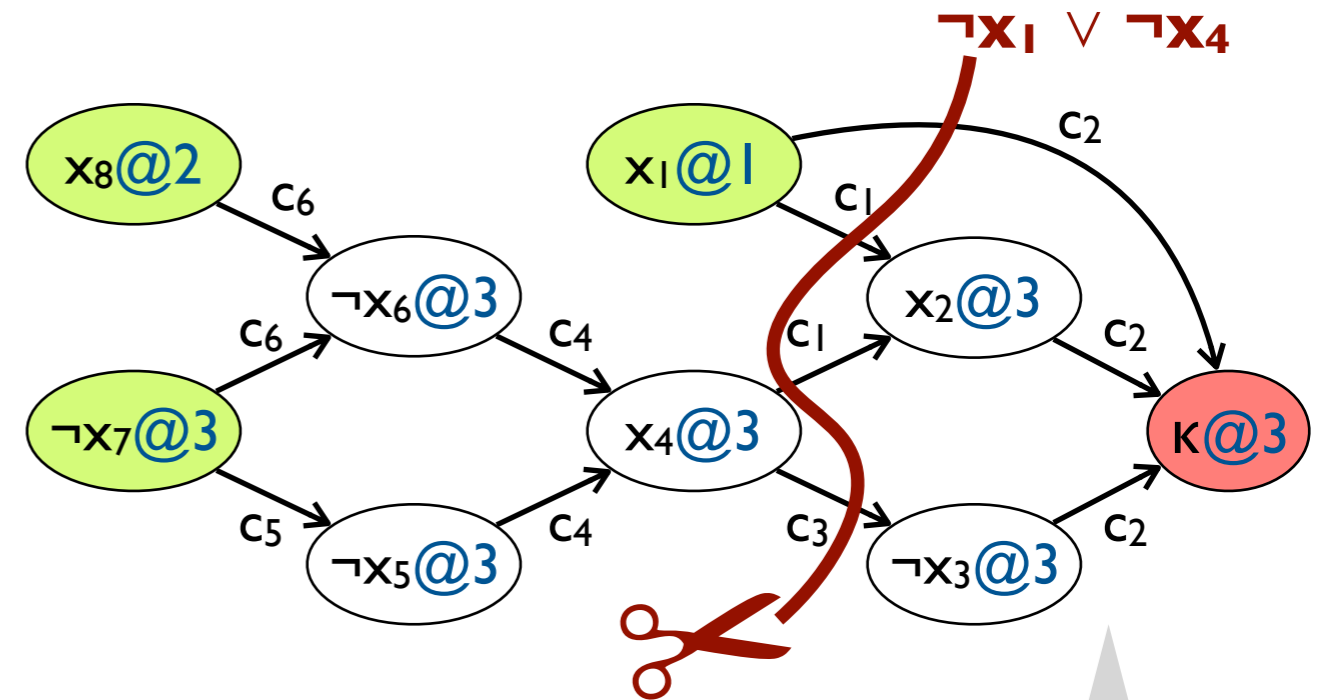
## Binary resolution rule

$$\frac{(a_1 \vee \dots \vee a_n \vee \beta) \quad (b_1 \vee \dots \vee b_m \vee \neg\beta)}{(a_1 \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_m)}$$

# ANALYZECONFLICT: computing the conflict clause

```

ANALYZECONFLICT()
  d ← level(conflict)
  if d = 0 then return -1
  c ← antecedent(conflict)
  while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
  b ← ...
  return ⟨b, c⟩
    
```



## Example:

- $c = c_2, t = x_2, v = x_2, a = c_1$
- $c = \neg x_1 \vee x_3 \vee \neg x_4, t = x_3, v = x_3, a = c_3$

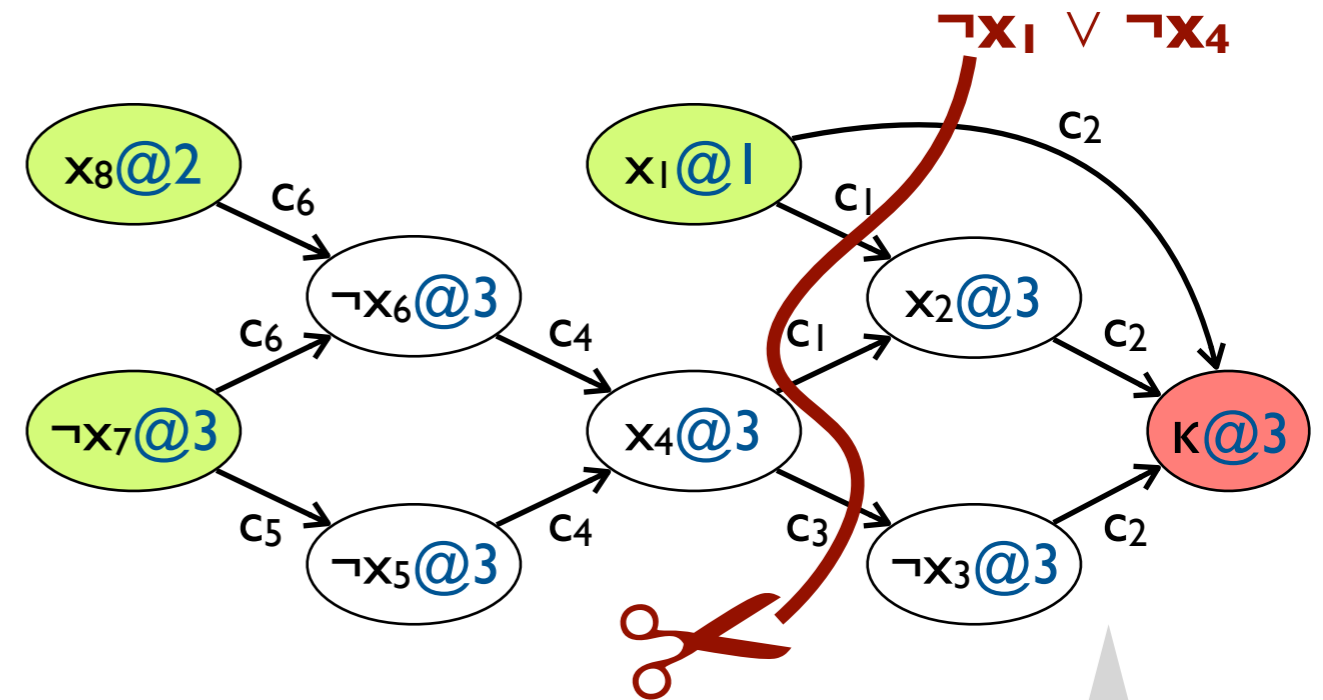
## Binary resolution rule

$$\frac{(a_1 \vee \dots \vee a_n \vee \beta) \quad (b_1 \vee \dots \vee b_m \vee \neg\beta)}{(a_1 \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_m)}$$

# ANALYZECONFLICT: computing the conflict clause

```

ANALYZECONFLICT()
  d ← level(conflict)
  if d = 0 then return -1
  c ← antecedent(conflict)
  while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
  b ← ...
  return ⟨b, c⟩
    
```



## Example:

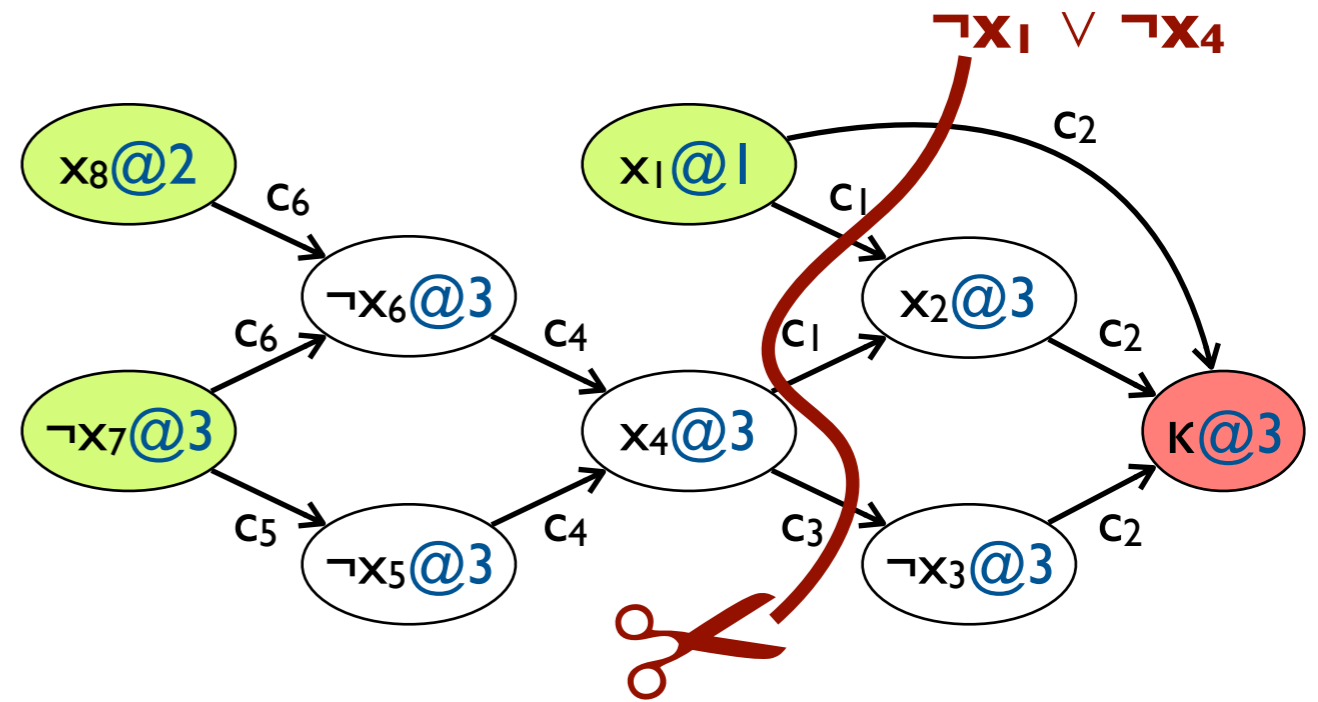
- $c = c_2, t = x_2, v = x_2, a = c_1$
- $c = \neg x_1 \vee x_3 \vee \neg x_4, t = x_3, v = x_3, a = c_3$
- $c = \neg x_1 \vee \neg x_4, \text{ done!}$

## Binary resolution rule

$$\frac{(a_1 \vee \dots \vee a_n \vee \beta) \quad (b_1 \vee \dots \vee b_m \vee \neg\beta)}{(a_1 \vee \dots \vee a_n \vee b_1 \vee \dots \vee b_m)}$$

# ANALYZECONFLICT: computing backtracking level

```
ANALYZECONFLICT()  
  d ← level(conflict)  
  if d = 0 then return -1  
  c ← antecedent(conflict)  
  while !oneLitAtLevel(c, d)  
    t ← lastAssignedLitAtLevel(c, d)  
    v ← varOfLit(t)  
    a ← antecedent(t)  
    c ← resolve(a, c, v)  
  b ← ...  
  return ⟨b, c⟩
```



To what level should we backtrack?

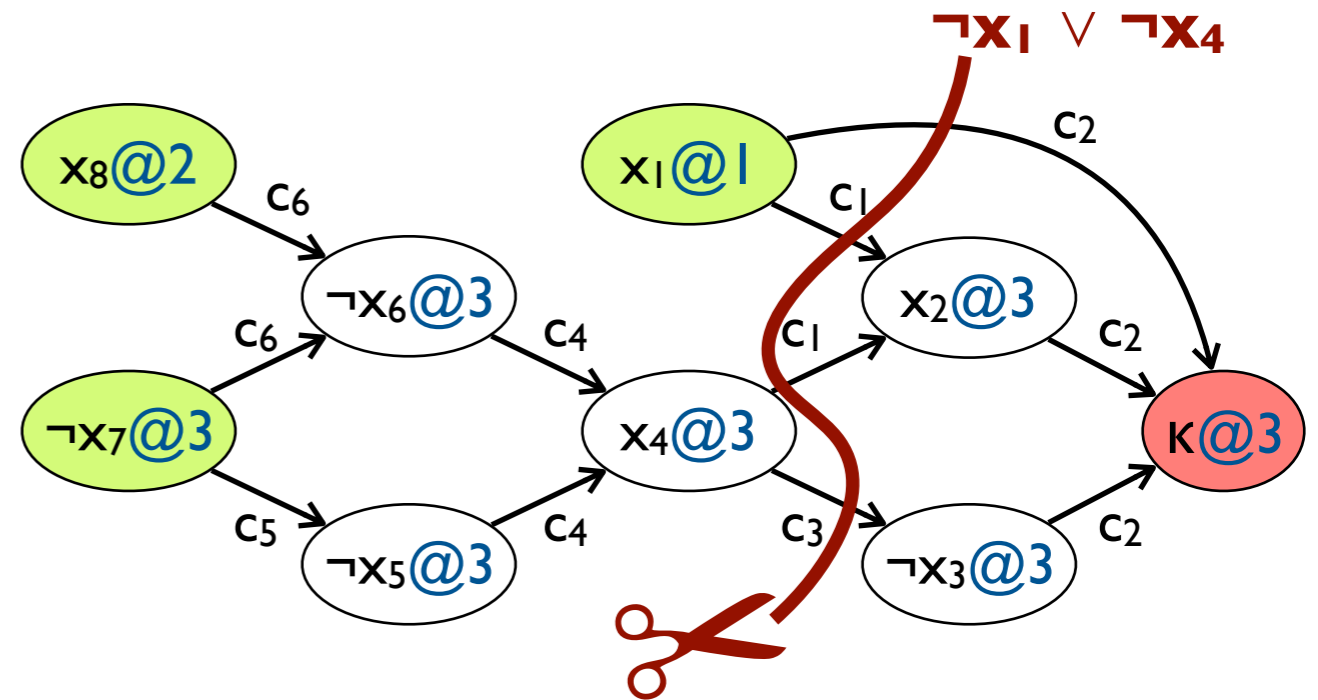


# ANALYZECONFLICT: computing backtracking level

```

ANALYZECONFLICT()
  d ← level(conflict)
  if d = 0 then return -1
  c ← antecedent(conflict)
  while !oneLitAtLevel(c, d)
    t ← lastAssignedLitAtLevel(c, d)
    v ← varOfLit(t)
    a ← antecedent(t)
    c ← resolve(a, c, v)
    b ← assertingLevel(c)
  return ⟨b, c⟩
    
```

By construction,  $c$  is unit at  $b$  (since it has only one literal at the current level  $d$ ).



Second highest decision level for any literal in  $c$ , unless  $c$  is unary. In that case, its asserting level is zero.

# Decision heuristics

CDCL(F)

$A \leftarrow \{\}$

if **BCP**(F,A) = *conflict* then return *false*

level  $\leftarrow 0$

while hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

while **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

$F \leftarrow F \cup \{c\}$

if  $b < 0$  then return *false*

else **BACKTRACK**(F,A, b)

level  $\leftarrow b$

return *true*

## Example heuristics:

- Dynamic Largest Individual Sum (DLIS)
- Variable State Independent Decaying Sum (VSIDS)

# Decision heuristics: DLIS

CDCL(F)

$A \leftarrow \{\}$

if **BCP**(F,A) = *conflict* then return *false*

level  $\leftarrow 0$

while hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

while **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

$F \leftarrow F \cup \{c\}$

if  $b < 0$  then return *false*

else **BACKTRACK**(F,A, b)

level  $\leftarrow b$

return *true*

- Choose the literal that satisfies the most unresolved clauses.
- Simple and intuitive.
- But expensive: complexity of making a decision proportional to the number of clauses.

# Decision heuristics: VSIDS (zChaff)

CDCL(F)

$A \leftarrow \{\}$

**if** **BCP**(F,A) = *conflict* **then return** *false*

level  $\leftarrow 0$

**while** hasUnassignedVars(F)

level  $\leftarrow$  level + 1

$A \leftarrow A \cup \{ \mathbf{DECIDE}(F,A) \}$

**while** **BCP**(F,A) = *conflict*

$\langle b, c \rangle \leftarrow \mathbf{ANALYZECONFLICT}()$

$F \leftarrow F \cup \{c\}$

**if**  $b < 0$  **then return** *false*

**else** **BACKTRACK**(F,A, b)

level  $\leftarrow b$

**return** *true*

- Count the number of *all* clauses in which a literal appears, and periodically divide all scores by a constant (e.g., 2).
- Variables involved in more recent conflicts get higher scores.
- Constant decision time when literals kept in a sorted list.

# Engineering matters (a lot)

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

Solvers spend most of their time in BCP, so this must be efficient. Naive implementation won't work on large problems.

# Engineering matters (a lot)

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

Solvers spend most of their time in BCP, so this must be efficient. Naive implementation won't work on large problems.

Most solvers heuristically discard conflict clauses that are old, long, irrelevant, etc. (Why won't this cause the solver to run forever?)

# BCP with watched literals (zChaff)

```
CDCL(F)
A ← {}
if BCP(F,A) = conflict then return false
level ← 0
while hasUnassignedVars(F)
  level ← level + 1
  A ← A ∪ { DECIDE(F,A) }
  while BCP(F,A) = conflict
    ⟨b, c⟩ ← ANALYZECONFLICT()
    F ← F ∪ {c}
    if b < 0 then return false
    else BACKTRACK(F,A, b)
      level ← b
return true
```

Based on the observation that a clause can't imply a new assignment if it has more than 2 unassigned literals left.

So, pick two unassigned literals per clause to watch.

If a watched literal is assigned, pick another unassigned literal to watch in its place.

If there is only one unassigned literal, it is implied by BCP.

# Summary

## Today

- The CDCL algorithm extends DPLL with
  - Non-chronological backtracking
  - Learning
  - Decision heuristics
  - Engineering matters

## Next lecture

- Practical applications of SAT solving