CS 292C Computer-Aided Reasoning for Software

Lecture 5: A Modern SAT Solver

Inspired by CSE507 from Emina Torlak and CS389L from Isil Dillig

Yu Feng
Fall 2019

Summary of previous lecture

- 2nd paper review was out
- 1st homework will be due on Wednesday
- Review of propositional logic
- Normal forms (NNF, DNF, CNF)
- A basic SAT solver (DPLL algorithm)

Outline of this lecture

- The CDCL algorithm
- Three important extensions of DPLL
    - Decision
    - Backtrack
    - Learning

A basic SAT solver (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 = $\bot$ then return false
  p ← choose(vars(G))
  return DPLL(G{p $\mapsto$ $\top$}) ||
               DPLL(G{p $\mapsto$ $\bot$})
```

Unit resolution rule

$$\frac{\beta \qquad b1 \vee \ldots \vee bm \vee \neg\beta}{b1 \vee \ldots \vee bm}$$

Davis-Putnam-Logemann-Loveland (1962)

A basic SAT solver (DPLL)

```
// Returns true if the CNF formula F is
// satisfiable; otherwise returns false.
DPLL(F)
  G ← BCP(F)
  if G = ⊤ then return true
    if G = ⊥ then return false
  p ← choose(vars(G))
  return DPLL(G{p ↦ ⊤}) ||
              DPLL(G{p ↦ ⊥})
```

Boolean constraint propagation applies unit resolution until fixed point.

If BCP cannot reduce F to a constant, we choose an unassigned variable and recurse assuming that the variable is either true or false.

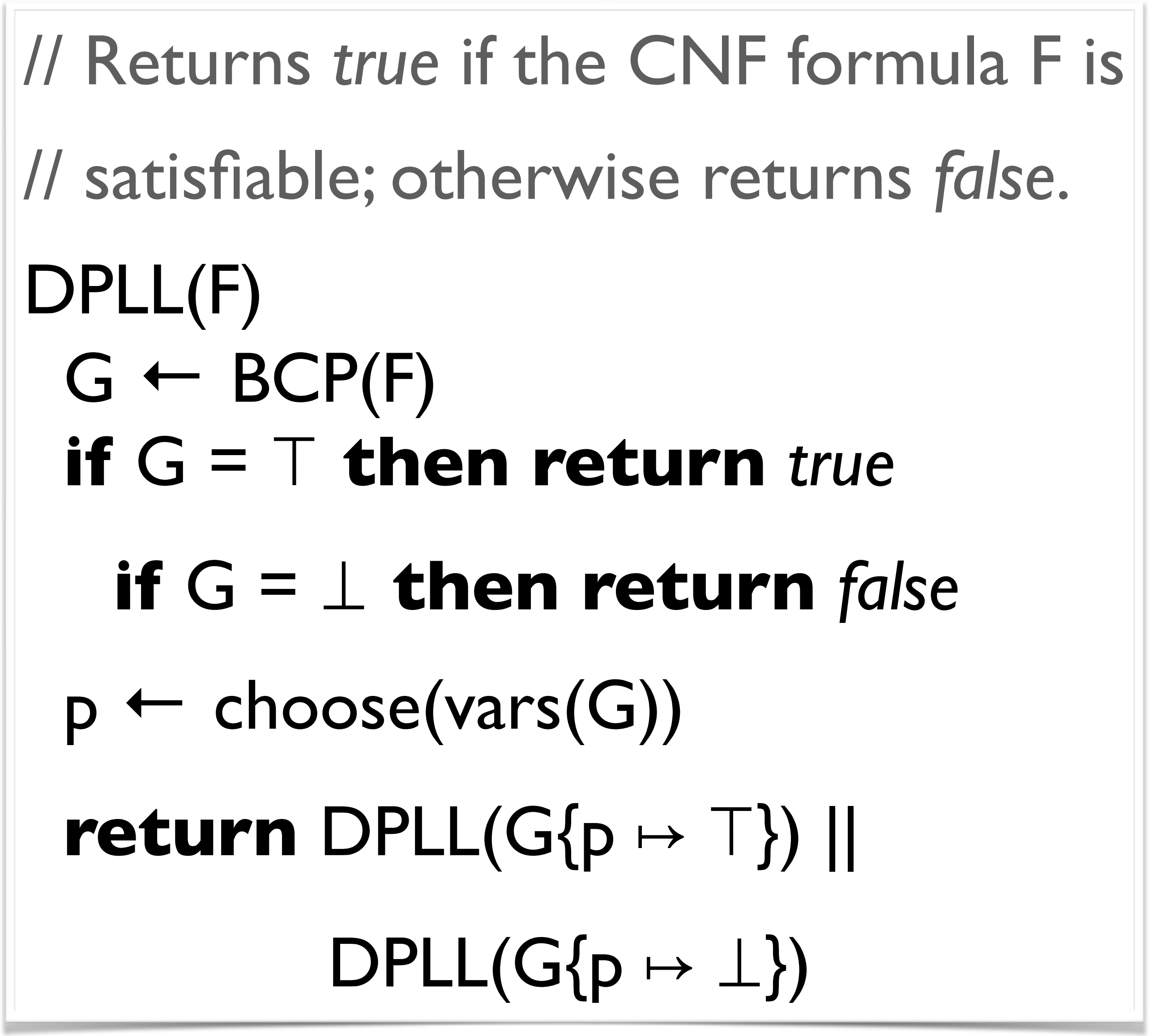
If the formula is satisfiable under either assumption, then we know that it has a satisfying assignment (expressed in the assumptions). Otherwise, the formula is unsatisfiable.

Unit resolution rule

$$\frac{\beta \qquad b_1 \vee \ldots \vee b_m \vee \neg\beta}{b_1 \vee \ldots \vee b_m}$$

Davis-Putnam-Logemann-Loveland (1962)

A basic SAT solver (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 = $\bot$ then return false
  p ← choose(vars(G))
  return DPLL(G{p $\mapsto$ $\top$}) ||
               DPLL(G{p $\mapsto$ $\bot$})
```

A basic SAT solver (DPLL)

```
// Returns true if the CNF formula F is
// satisfiable; otherwise returns false.
DPLL(F)
  $G \leftarrow$ BCP(F)
  if $G = \top$ then return true
    if $G = \bot$ then return false
  $p \leftarrow$ choose(vars(G))
  return DPLL($G\{p \mapsto \top\}$) ||
              DPLL($G\{p \mapsto \bot\}$)
```

No learning: throw away all the work to conclude the current partial assignment is bad. May get to conflict due to the same cause.

A basic SAT solver (DPLL)

```
// Returns *true* if the CNF formula F is
// satisfiable; otherwise returns *false*.
DPLL(F)
 G ← BCP(F)
 if G = ⊤ then return true
   if G = ⊥ then return false
 p ← choose(vars(G))
 return DPLL(G{p ↦ ⊤}) ||
            DPLL(G{p ↦ ⊥})
```

No learning: throw away all the work to conclude the current partial assignment is bad. May get to conflict due to the same cause.

Naive decision: The variable to branch on will significantly affect the performance.

A basic SAT solver (DPLL)

```
// Returns true if the CNF formula F is
// satisfiable; otherwise returns false.
DPLL(F)
 G ← BCP(F)
 if G = ⊤ then return true
   if G = ⊥ then return false
 p ← choose(vars(G))
 return DPLL(G{p ↦ ⊤}) ||
             DPLL(G{p ↦ ⊥})
```

No learning: throw away all the work to conclude the current partial assignment is bad. May get to conflict due to the same cause.

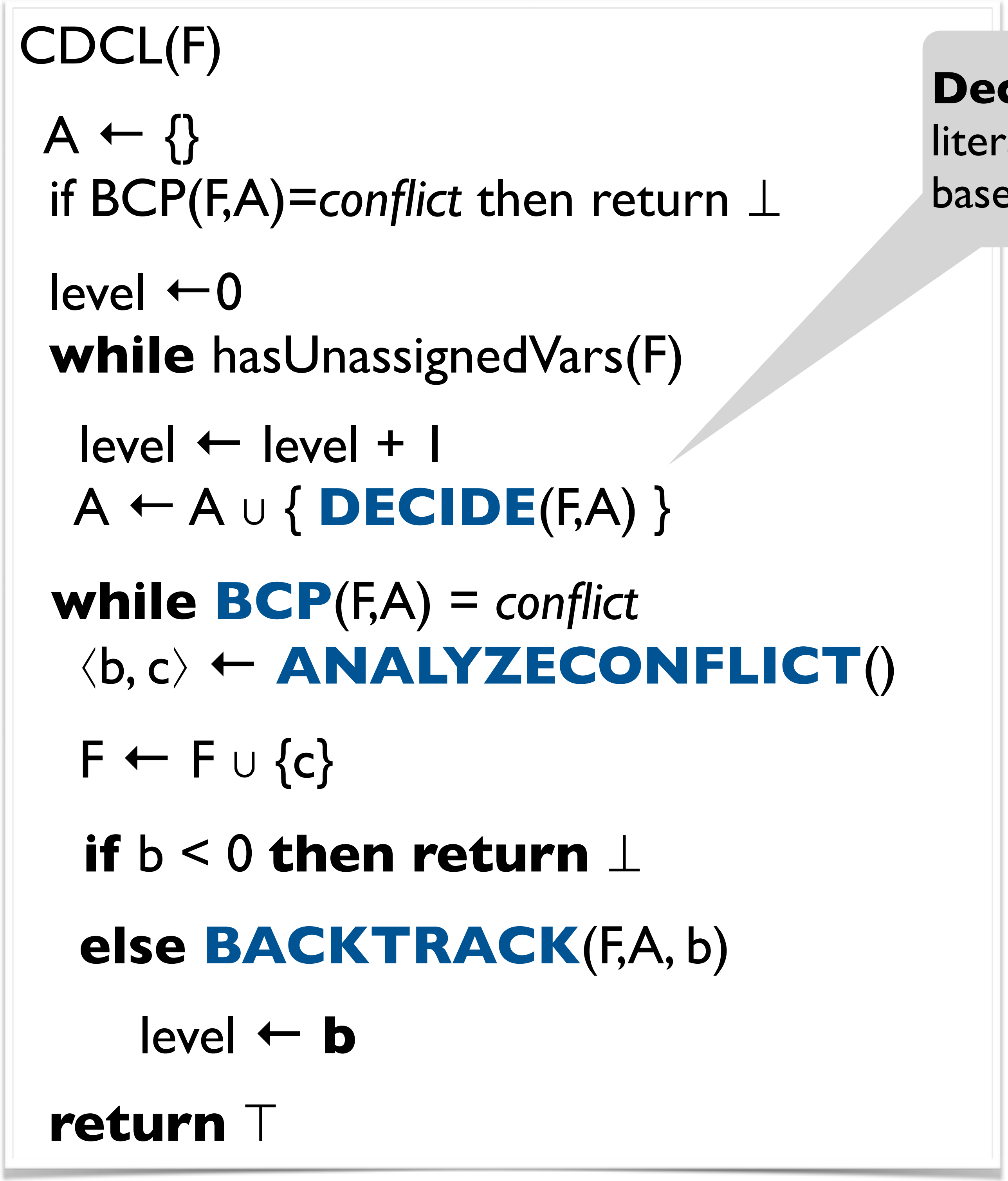
Naive decision: The variable to branch on will significantly affect the performance.

Chronological backtracking: backtrack on one level at a time, even if the root cause is at an earlier decision level.

A basic SAT solver (DPLL)

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
   else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

A basic SAT solver (DPLL)



```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
   else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

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

A basic SAT solver (DPLL)

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
    else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

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

Learning from mistakes: F augmented with a conflict clause that summarizes the root cause of the conflict

A basic SAT solver (DPLL)

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

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

Learning from mistakes: F augmented with a conflict clause that summarizes the root cause of the conflict

Non-chronological backtracking: backtrack b levels, based on the cause of the conflict

CDCL in a nutshell

```
CDCL(F)
  $A \leftarrow \{\}$
  if BCP(F,A)=*conflict* then return $\bot$
  level $\leftarrow 0$
  **while** hasUnassignedVars(F)
    level $\leftarrow$ level + 1
    $A \leftarrow A \cup \{$ **DECIDE**(F,A) $\}$
    **while** **BCP**(F,A) = *conflict*
      $\langle b, c \rangle \leftarrow$ **ANALYZECONFLICT**()
      $F \leftarrow F \cup \{c\}$
      **if** $b < 0$ **then return** $\bot$
      **else** **BACKTRACK**(F,A, b)
        level $\leftarrow$ **b**
  **return** $\top$
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

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

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

$c_4: x_4 \vee x_5 \vee x_6$

$c_5: x_7 \vee \neg x_5$

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

CDCL in a nutshell

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return $\perp$
 level ← 0
 while hasUnassignedVars(F)
   level ← level + 1
   A ← A $\cup$ { DECIDE(F,A) }
   while BCP(F,A) = conflict
     ⟨b, c⟩ ← ANALYZECONFLICT()
     F ← F $\cup$ {c}
     if b < 0 then return $\perp$
     else BACKTRACK(F,A, b)
       level ← b
 return $\top$
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

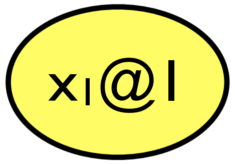
c_2: $\neg x_1 \lor \neg x_2 \lor x_3$

c_3: $\neg x_3 \lor \neg x_4$

c_4: $x_4 \lor x_5 \lor x_6$

c_5 : $x_7 \lor \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
 $A \leftarrow \{\}$
 if BCP(F,A)=conflict then return $\perp$
 level $\leftarrow 0$
 while hasUnassignedVars(F)
   level $\leftarrow$ level + 1
   $A \leftarrow A \cup \{$ DECIDE(F,A) $\}$
 while BCP(F,A) = conflict
   $\langle b, c \rangle \leftarrow$ ANALYZECONFLICT()
   $F \leftarrow F \cup \{c\}$
   if $b < 0$ then return $\perp$
   else BACKTRACK(F,A, b)
       level $\leftarrow$ b
 return $\top$
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

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

$c_3: \neg x_3 \lor \neg x_4$

$c_4: x_4 \lor x_5 \lor x_6$

$c_5 : x_7 \lor \neg x_5$

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

CDCL in a nutshell

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
   else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

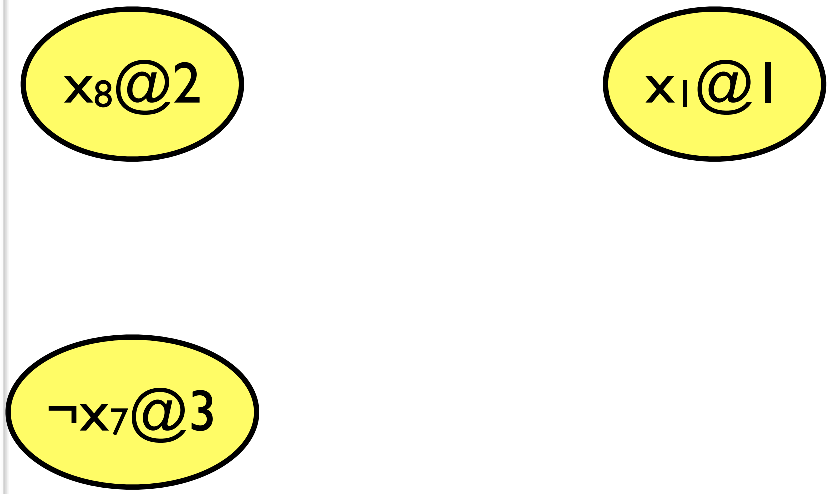
$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

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

$c_4: x_4 \vee x_5 \vee x_6$

$c_5: x_7 \vee \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
     else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

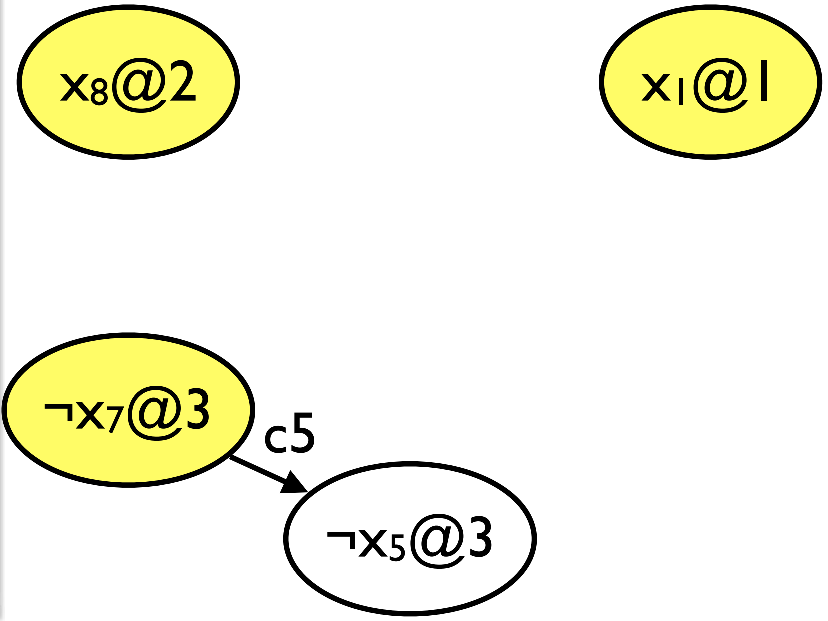
$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

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

$c_4: x_4 \vee x_5 \vee x_6$

$c_5: x_7 \vee \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

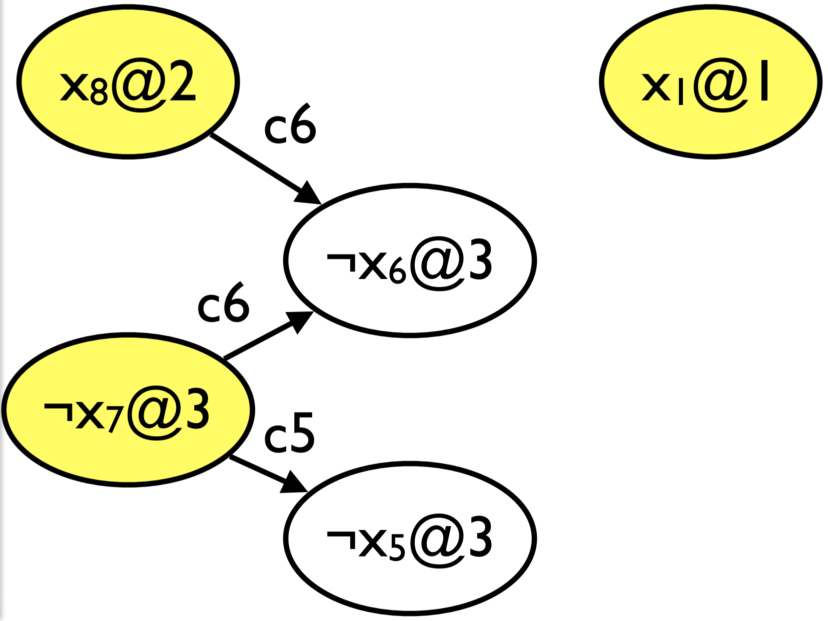
$c_2: \neg x_1 \lor \neg x_2 \lor x_3$

$c_3: \neg x_3 \lor \neg x_4$

$c_4: x_4 \lor x_5 \lor x_6$

$c_5 : x_7 \lor \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
     else BACKTRACK(F,A, b)
        level ← b
 return ⊤
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

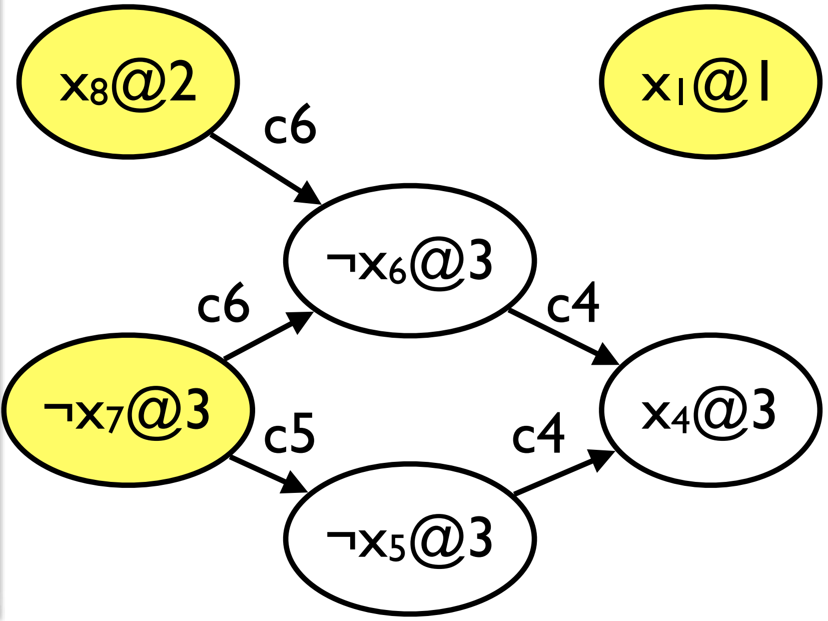
$c_2: \neg x_1 \vee \neg x_2 \vee x_3$

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

$c_4: x_4 \vee x_5 \vee x_6$

$c_5 : x_7 \vee \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

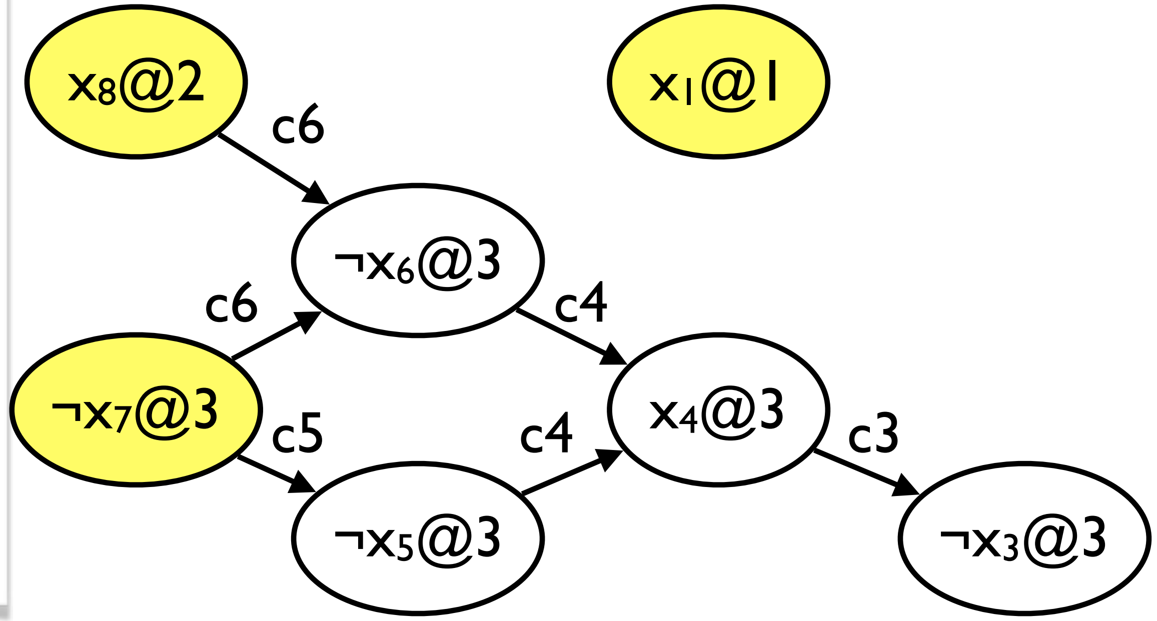
$c_2: \neg x_1 \lor \neg x_2 \lor x_3$

$c_3: \neg x_3 \lor \neg x_4$

$c_4: x_4 \lor x_5 \lor x_6$

$c_5 : x_7 \lor \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
    else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

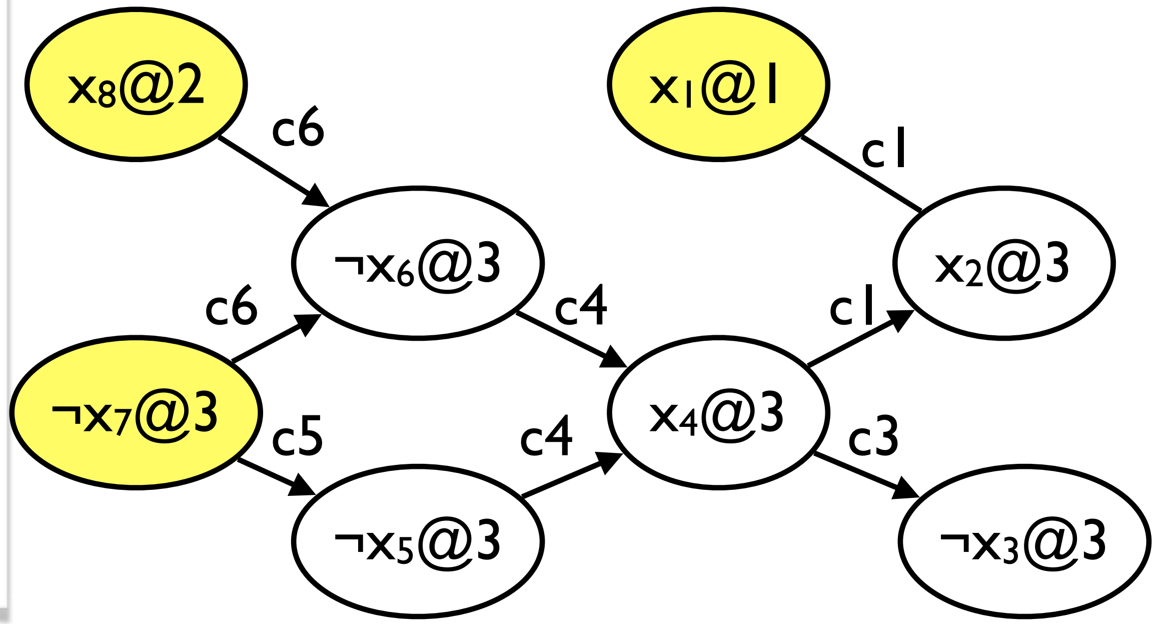
$c_2: \neg x_1 \lor \neg x_2 \lor x_3$

$c_3: \neg x_3 \lor \neg x_4$

$c_4: x_4 \lor x_5 \lor x_6$

$c_5 : x_7 \lor \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
     else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

$F = \{ c_1,c_2,c_3,c_4,\ldots,c_9 \}$

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

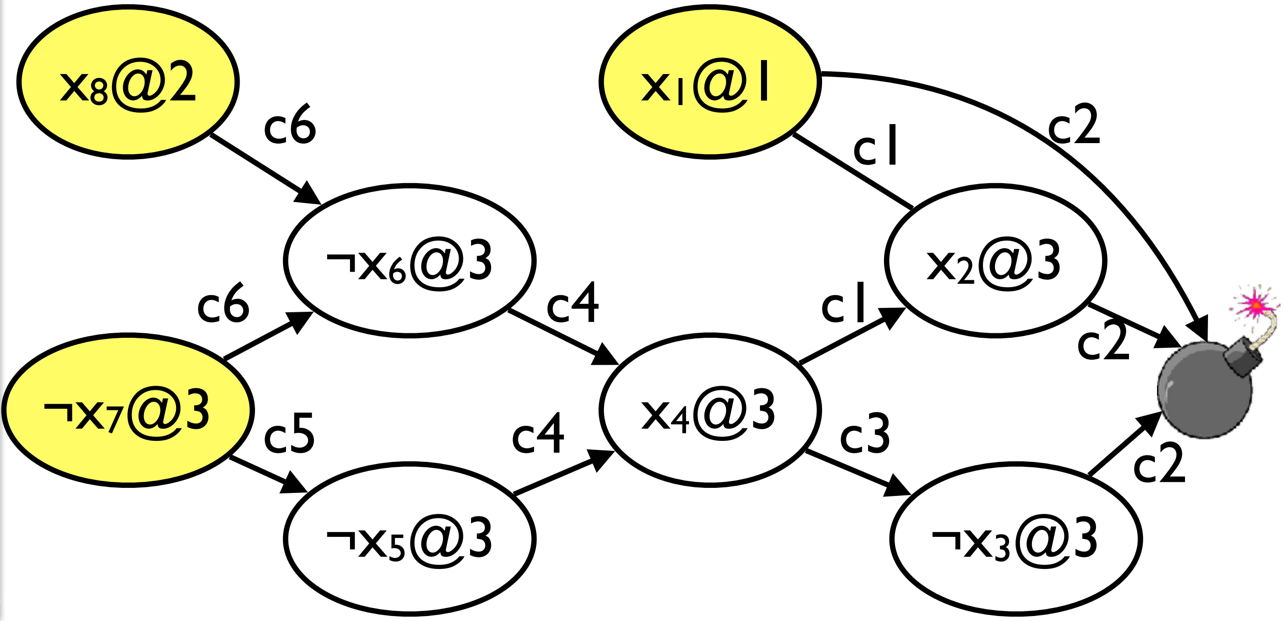
$c_2: \neg x_1 \lor \neg x_2 \lor x_3$

$c_3: \neg x_3 \lor \neg x_4$

$c_4: x_4 \lor x_5 \lor x_6$

$c_5 : x_7 \lor \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
  $A \leftarrow \{\}$
  if BCP(F,A)=conflict then return $\bot$
  level $\leftarrow$ 0
  while hasUnassignedVars(F)
    level $\leftarrow$ level + 1
    $A \leftarrow A \cup$ { DECIDE(F,A) }
    while BCP(F,A) = conflict
      $\langle b, c \rangle \leftarrow$ ANALYZECONFLICT()
      $F \leftarrow F \cup \{c\}$
      if b < 0 then return $\bot$
      else BACKTRACK(F,A, b)
        level $\leftarrow$ b
  return $\top$
```

Conflict clause
$\neg x_1 \lor \neg x_4$

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

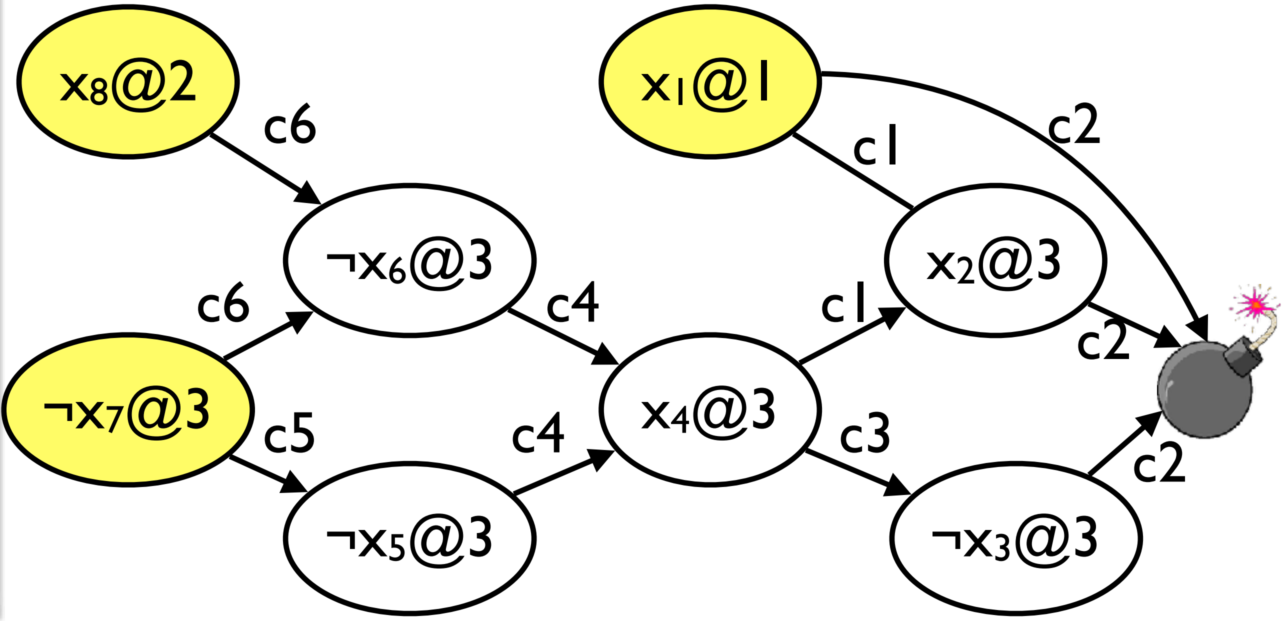
c_2: $\neg x_1 \lor \neg x_2 \lor x_3$

c_3: $\neg x_3 \lor \neg x_4$

c_4: $x_4 \lor x_5 \lor x_6$

c_5 : $x_7 \lor \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
     else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

Conflict clause
$\neg x_1 \lor \neg x_4$

Backtrack to
$x_1@1$

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

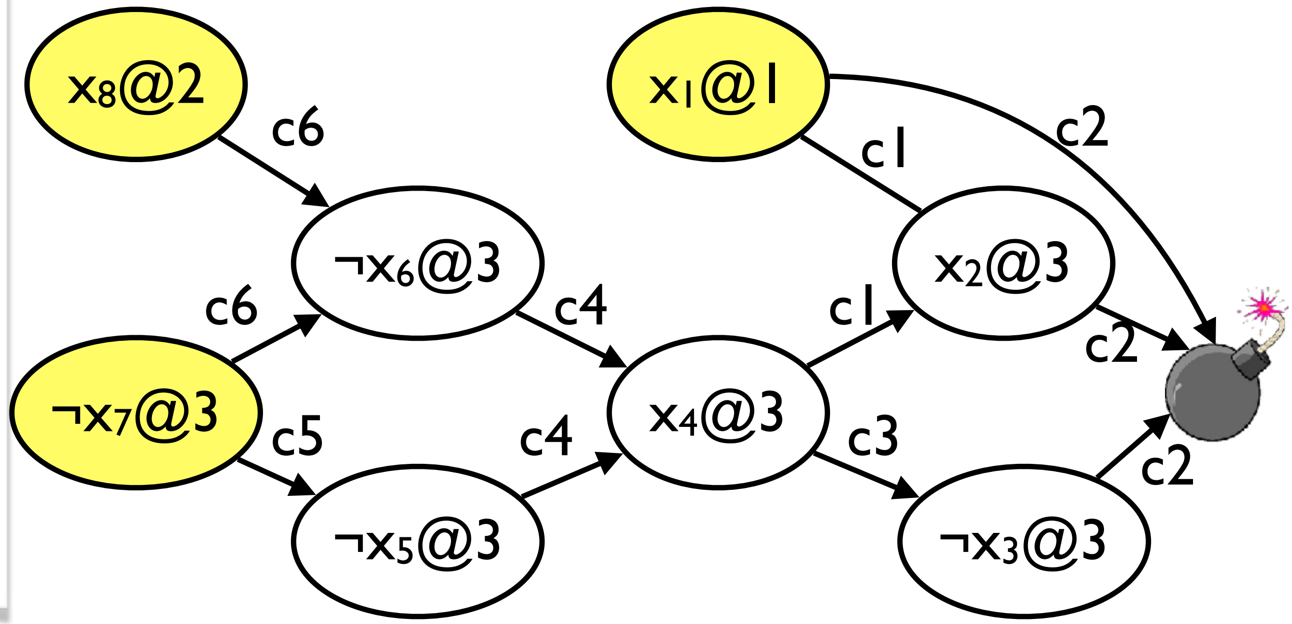
$c_2: \neg x_1 \lor \neg x_2 \lor x_3$

$c_3: \neg x_3 \lor \neg x_4$

$c_4: x_4 \lor x_5 \lor x_6$

$c_5: x_7 \lor \neg x_5$

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



CDCL in a nutshell

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
     else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

Conflict clause $\neg x_1 \lor \neg x_4$

Backtrack to $x_1@1$

$F = \{ c_1,c_2,c_3,c_4,\ldots,c_9 \}$

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

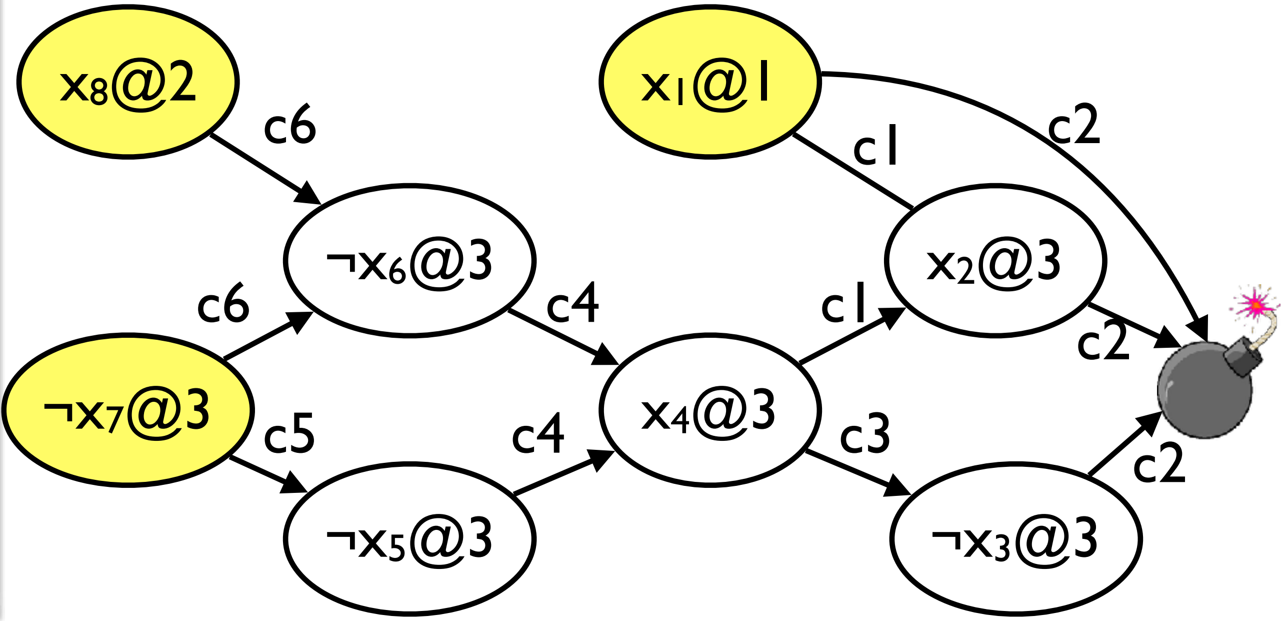
c_2: $\neg x_1 \lor \neg x_2 \lor x_3$

c_3: $\neg x_3 \lor \neg x_4$

c_4: $x_4 \lor x_5 \lor x_6$

c_5 : $x_7 \lor \neg x_5$

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



CDCL in action

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

- Definition
- Analyze conflict
- Decide heuristics
- Engineering tricks

Basic concepts in CDCL

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

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

A clause may be

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

$F = \{ c_1, c_2, c_3, c_4, \ldots, c_9 \}$

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

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

...

$c_8 : x_9 \lor \neg x_2$

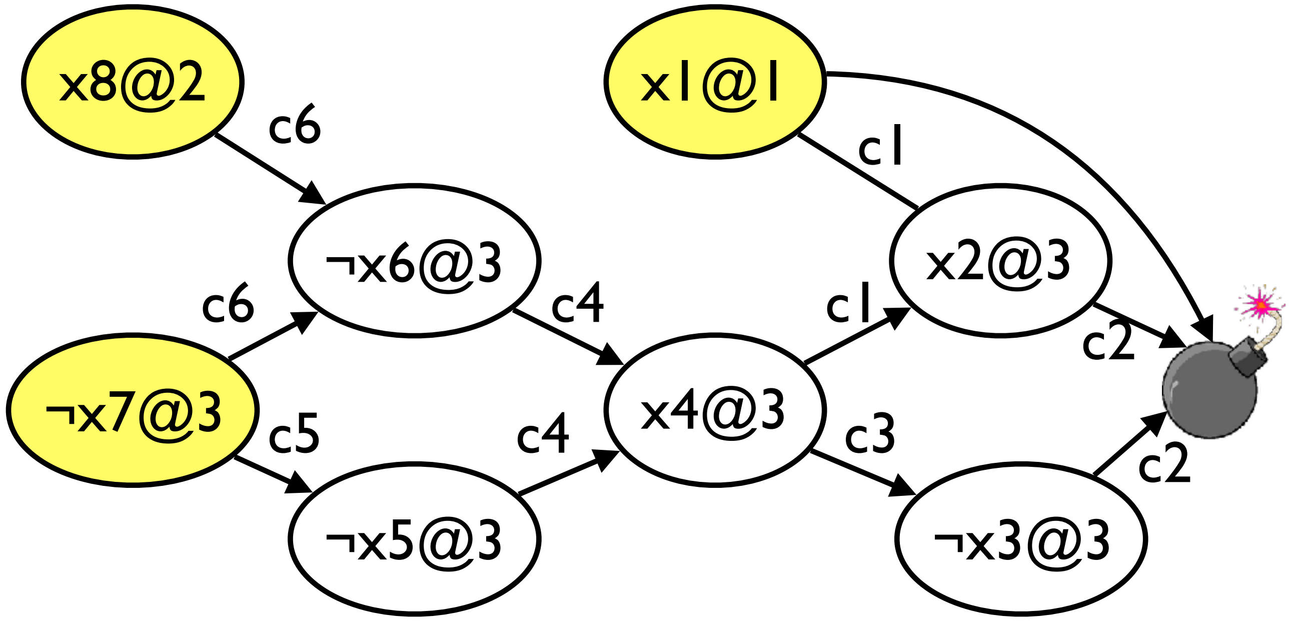
$c_9: x_9 \lor x_{10} \lor x_3$

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∈V is a literal (or κ) and the decision level at which it entered the current PA.
- ⟨v, w⟩ ∈ E iff v ≠ w, ¬v ∈ antecedent(w), and ⟨v, w⟩ is labeled with antecedent(w)

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

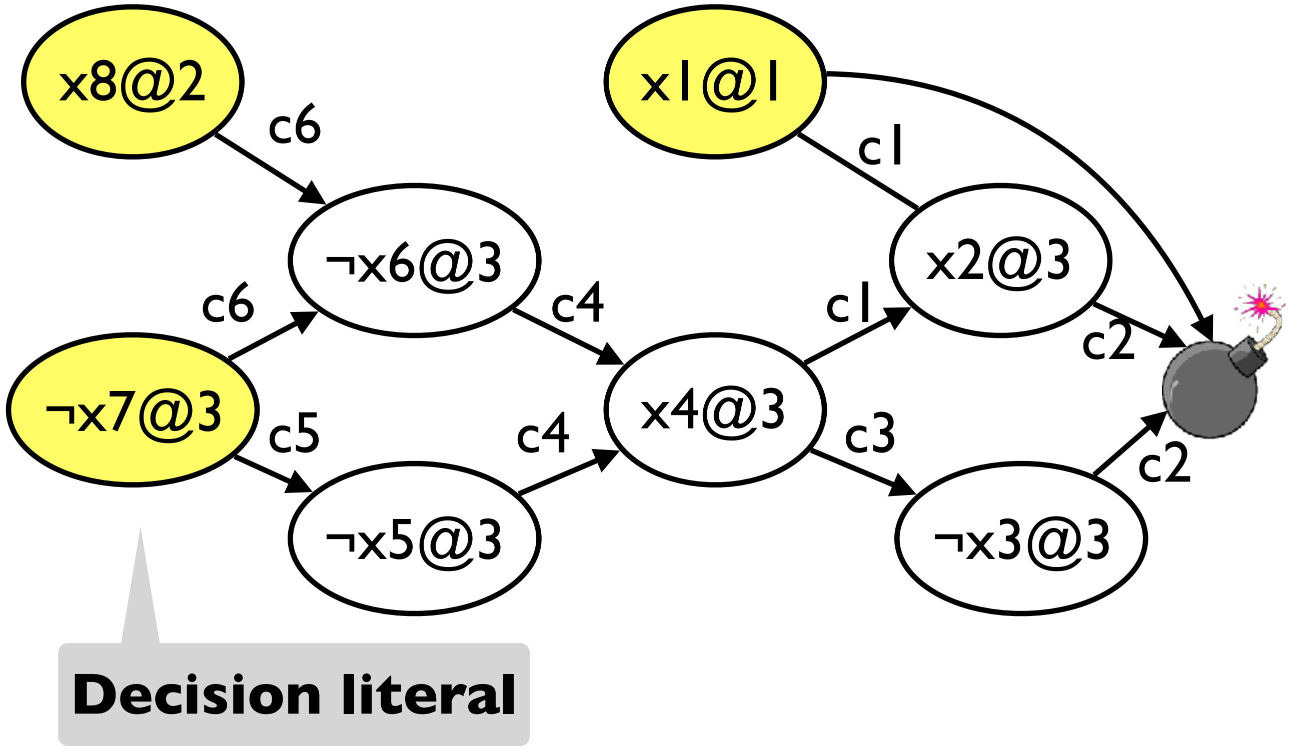


Implication graph

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

- v∈V is a literal (or κ) and the decision level at which it entered the current PA.
- ⟨v, w⟩ ∈ E iff v ≠ w, ¬v ∈ antecedent(w), and ⟨v, w⟩ is labeled with antecedent(w)

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



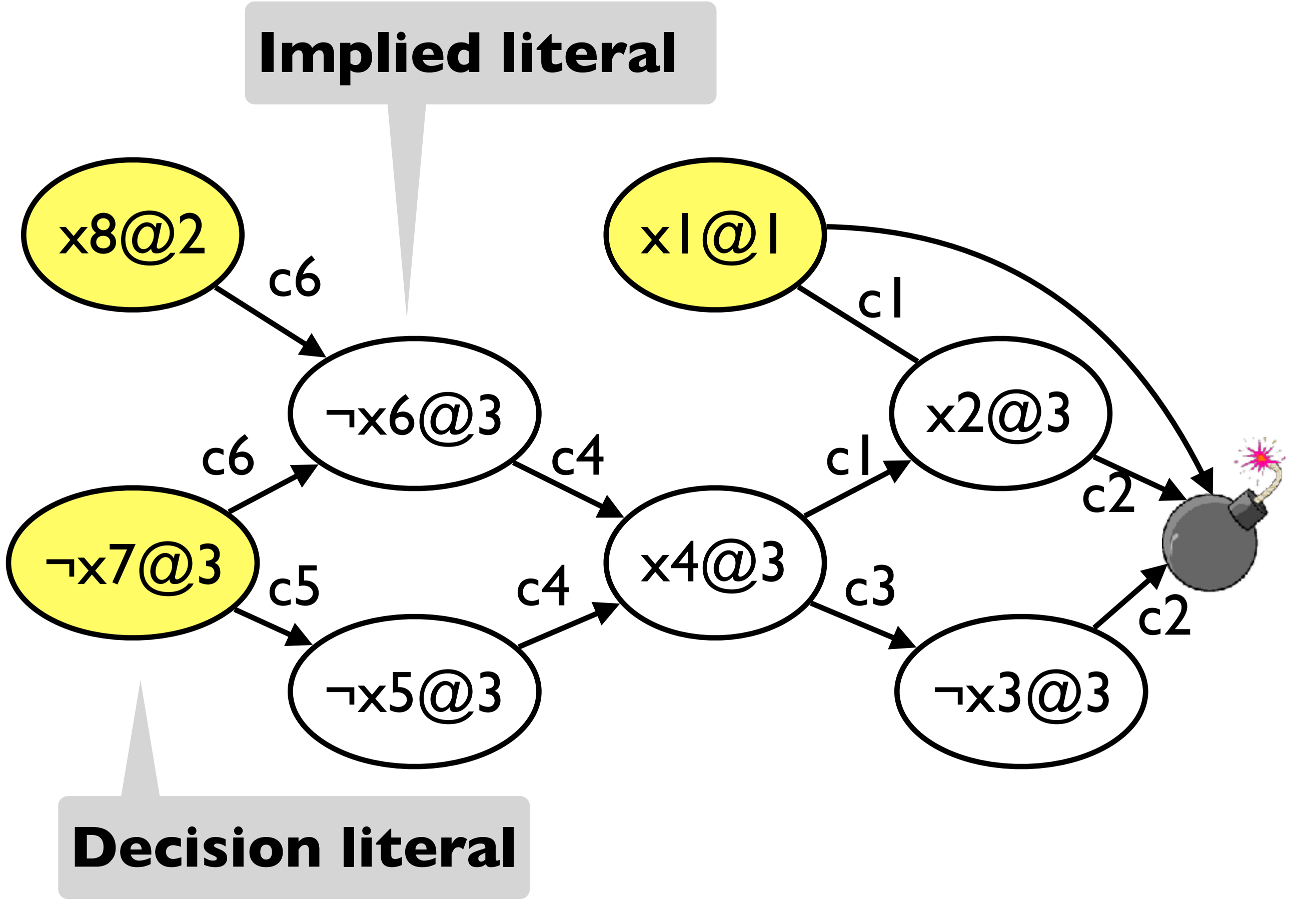
Decision literal

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∈V is a literal (or κ) and the decision level at which it entered the current PA.
- ⟨v, w⟩ ∈ E iff v ≠ w, ¬v ∈ antecedent(w), and ⟨v, w⟩ is labeled with antecedent(w)

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

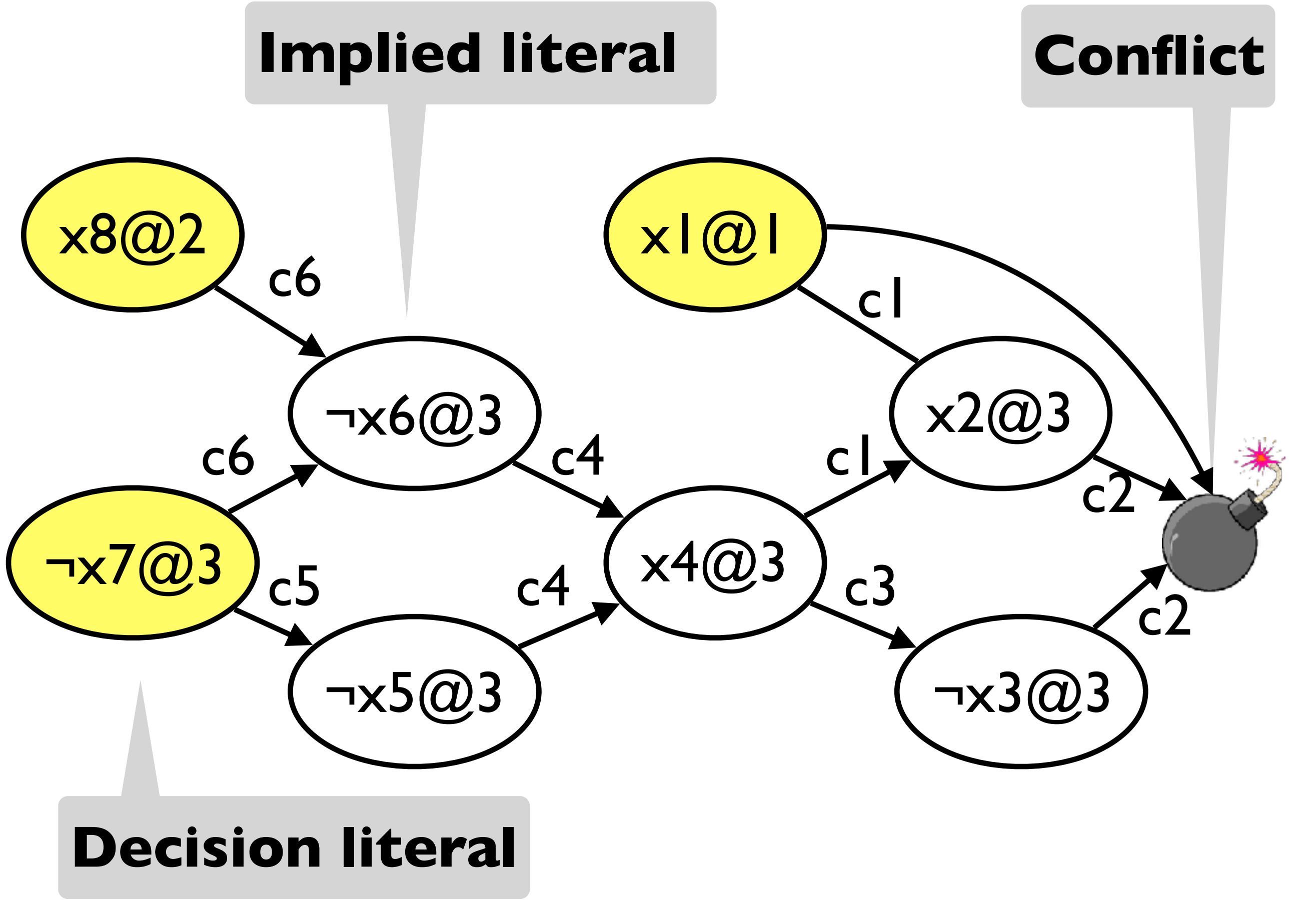


Implication graph

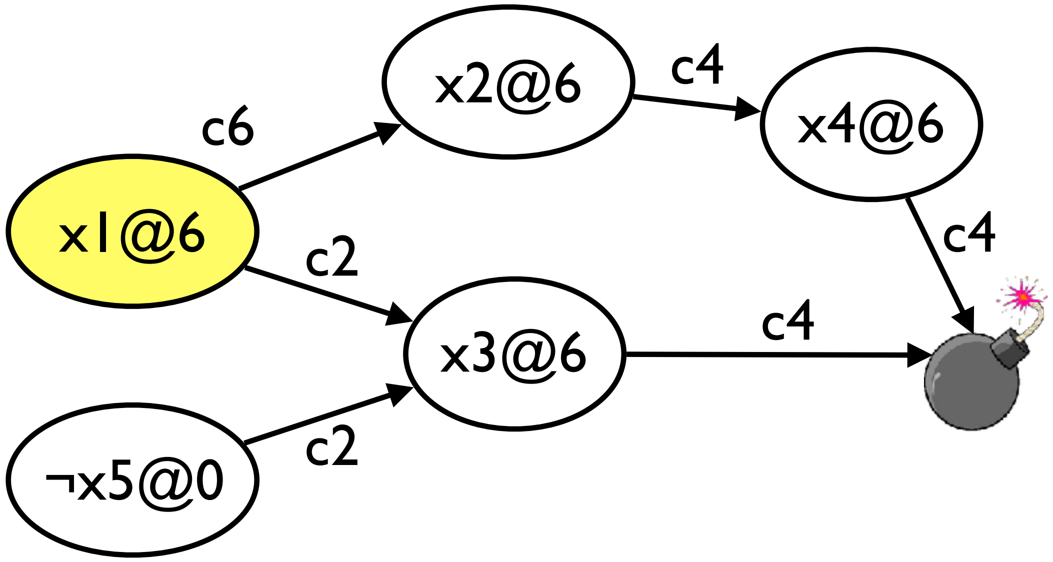
An **implication graph** G = (V, E) is a DAG that records the history of decisions and the resulting deductions derived with BCP.

- v∈V is a literal (or κ) and the decision level at which it entered the current PA.
- ⟨v, w⟩ ∈ E iff v ≠ w, ¬v ∈ antecedent(w), and ⟨v, w⟩ is labeled with antecedent(w)

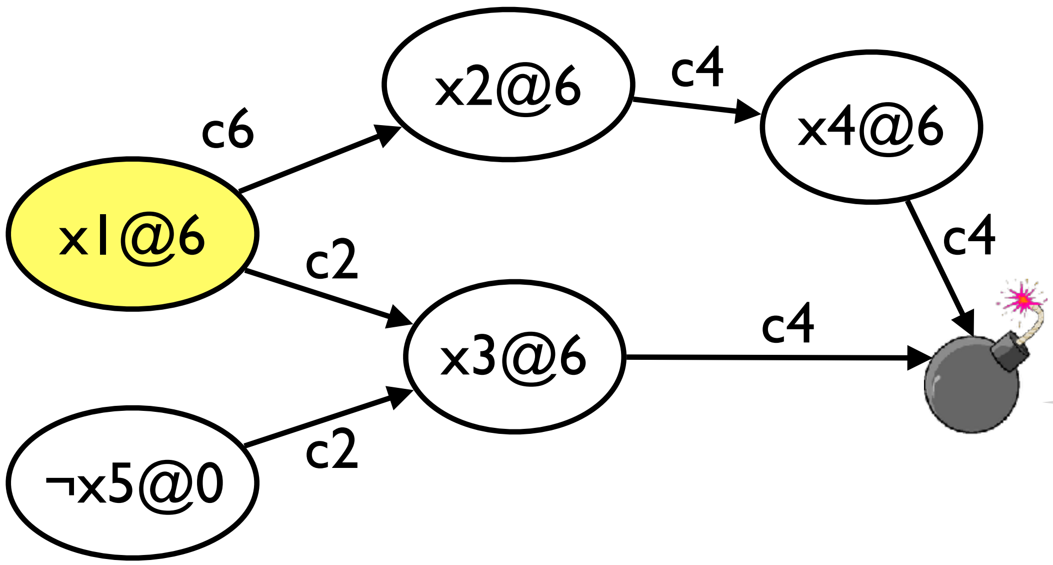
A unit clause c is the antecedent of its sole unassigned literal.



Implication graph by example



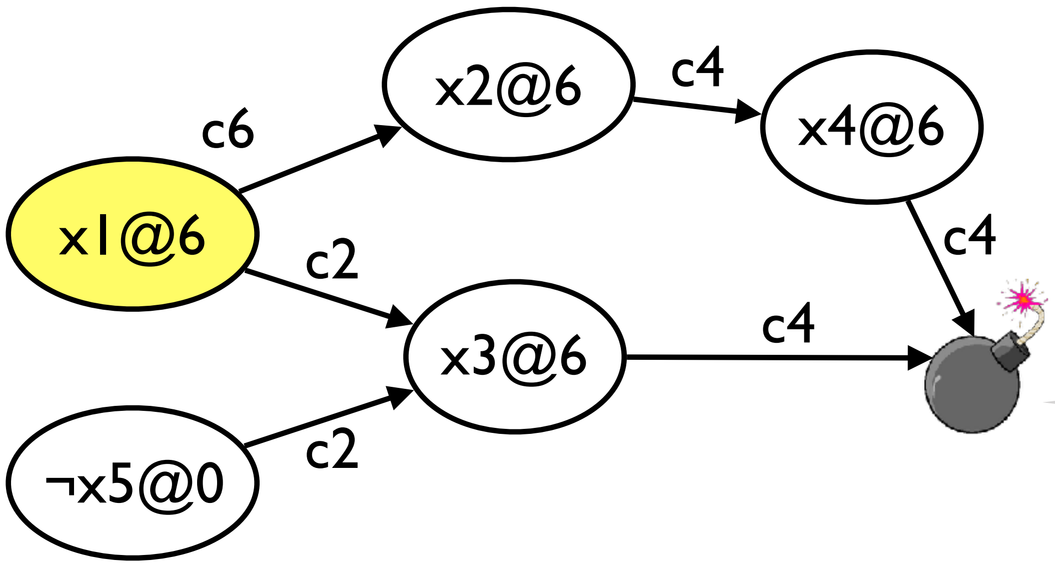
Implication graph by example



What clauses gave rise to this implication graph?

- $c_1 : \neg x_1 \vee x_2$
- $c_2 : \neg x_1 \vee x_3 \vee x_5$
- $c_3 : \neg x_2 \vee x_4$
- $c_4 : \neg x_3 \vee \neg x_4$
- $c_k : \neg x_5$

Implication graph by example



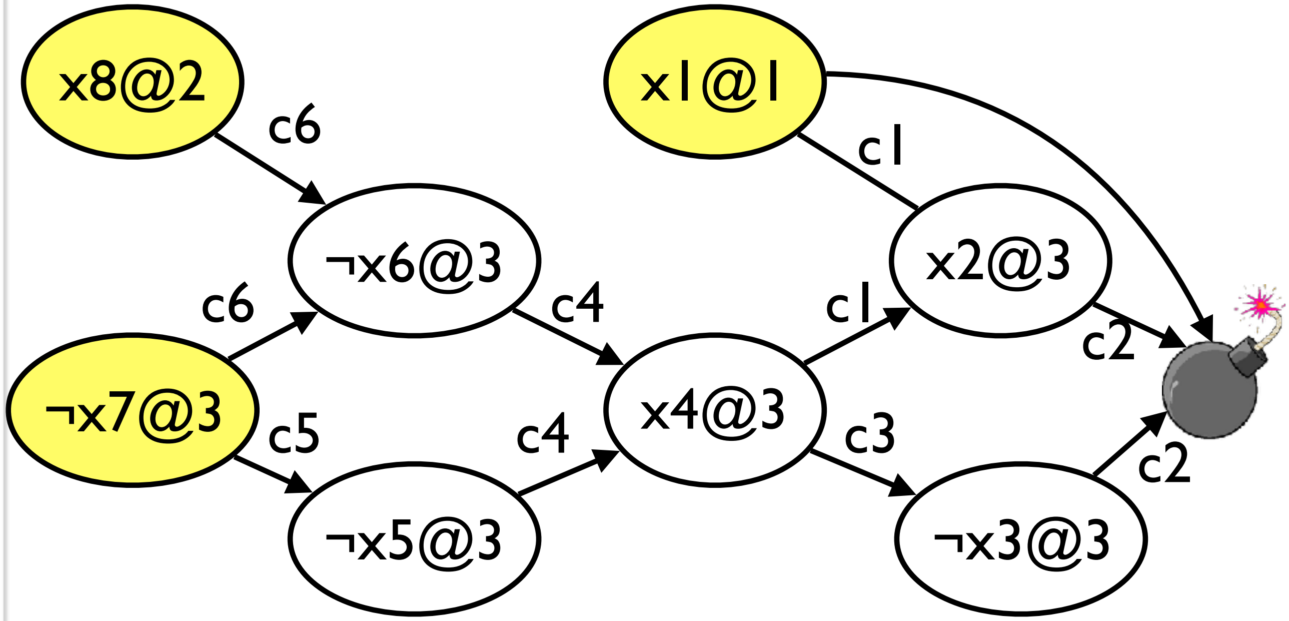
What clauses gave rise to this implication graph?

- $c_1 : \neg x_1 \lor x_2$
- $c_2 : \neg x_1 \lor x_3 \lor x_5$
- $c_3 : \neg x_2 \lor x_4$
- $c_4 : \neg x_3 \lor \neg x_4$
- $c_k : \neg x_5$

Implied by unary clauses

Implication graph for conflict analysis

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
   else BACKTRACK(F,A, b)
        level ← b
 return ⊤
```

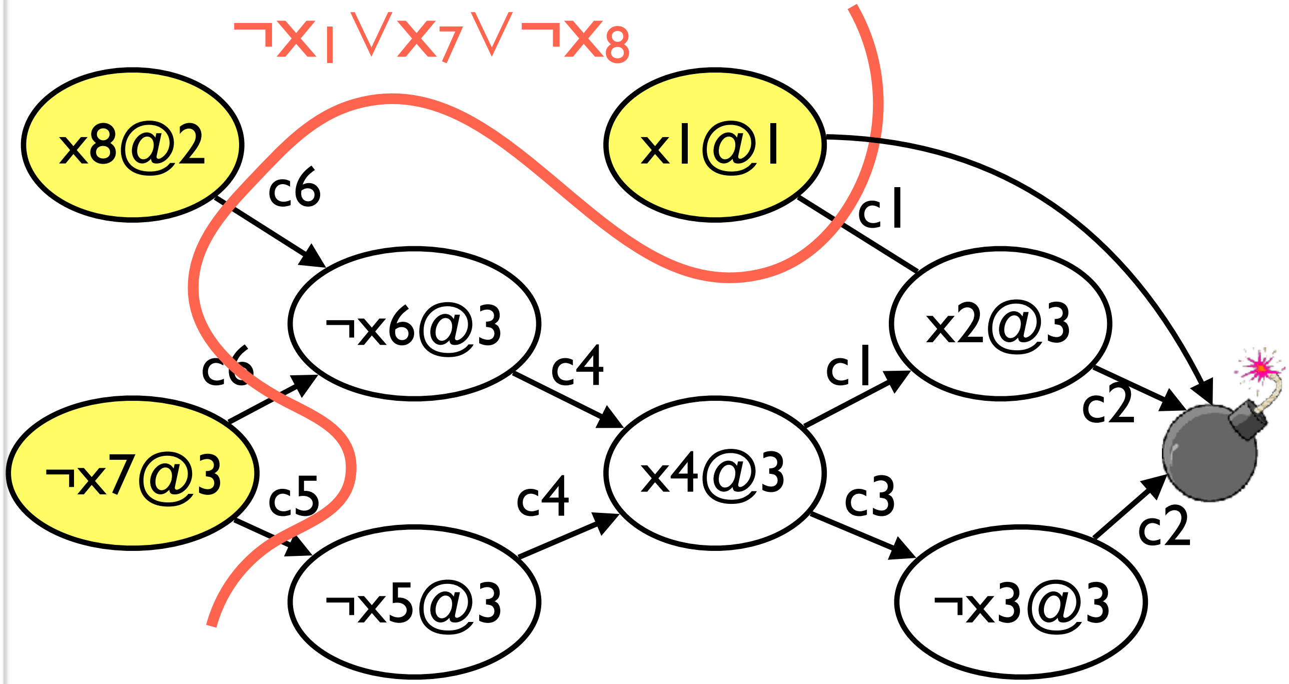


A **conflict clause** is implied by F and it blocks partial assignments (PAs) that lead to the current conflict.

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

Implication graph for conflict analysis

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
     else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```

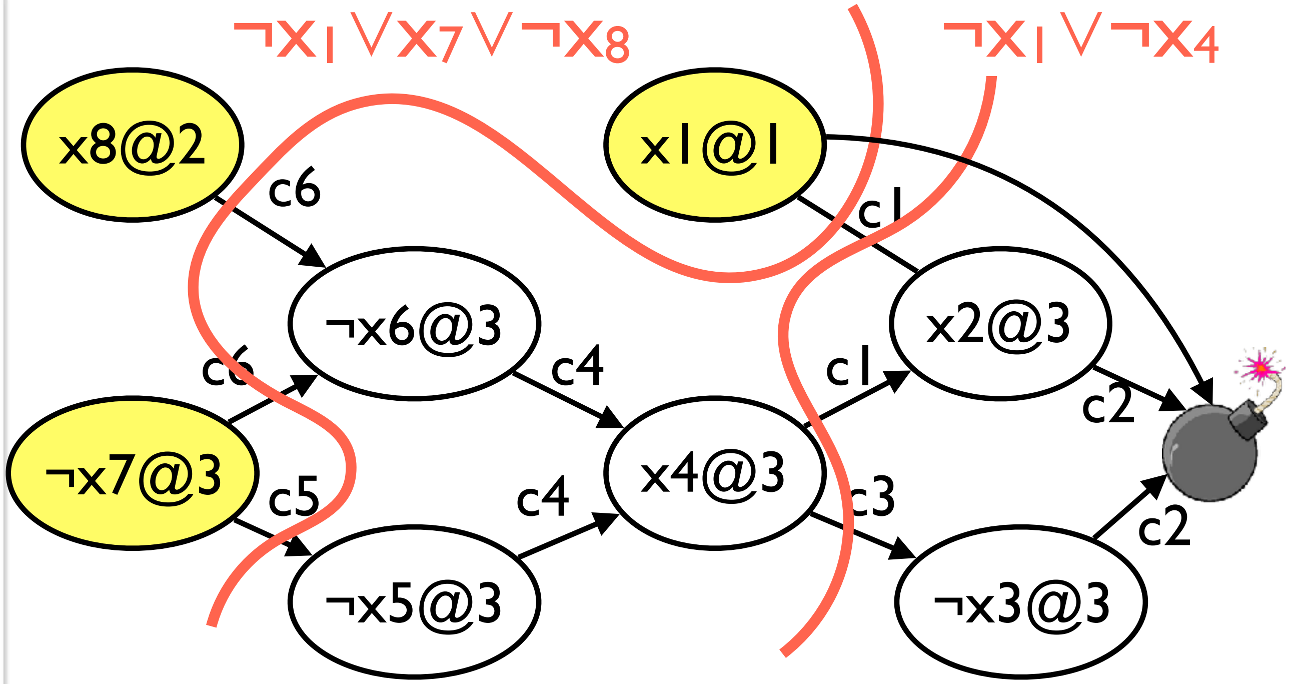


A **conflict clause** is implied by F and it blocks partial assignments (PAs) that lead to the current conflict.

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

Implication graph for conflict analysis

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

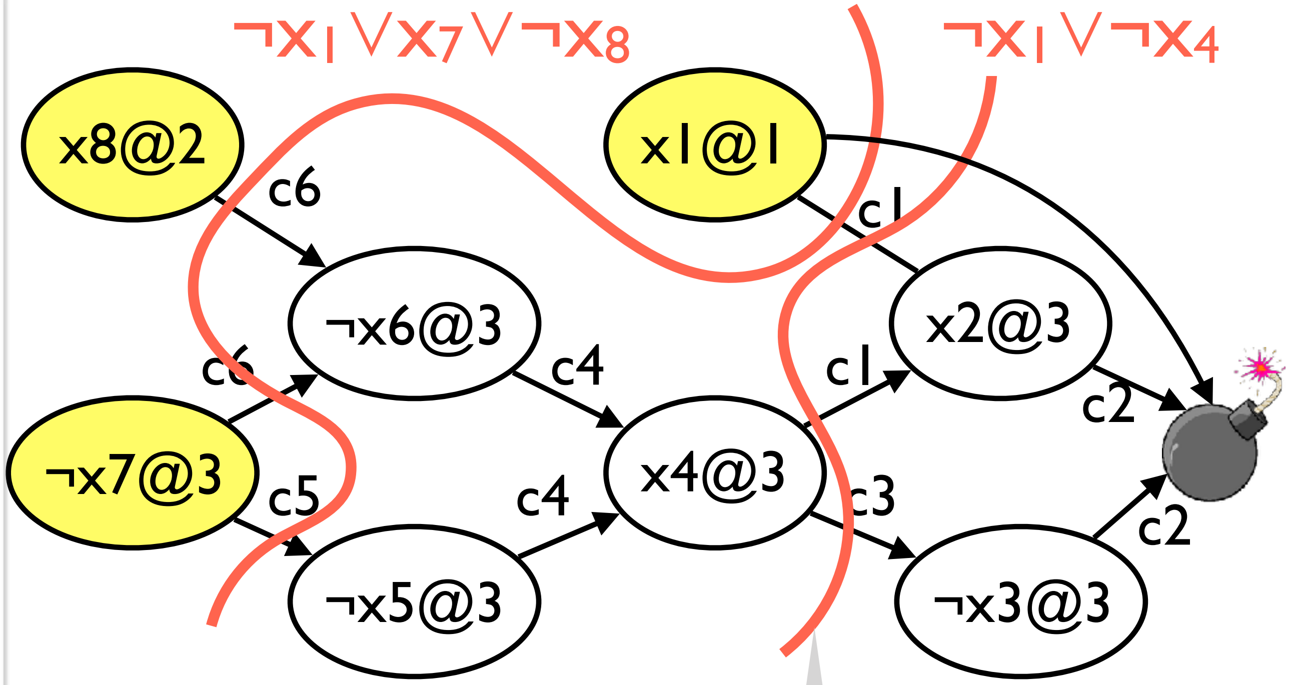


A **conflict clause** is implied by F and it blocks partial assignments (PAs) that lead to the current conflict.

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

Implication graph for conflict analysis

```
CDCL(F)
 A ← {}
 if BCP(F,A)=conflict then return ⊥
 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 ⊥
   else BACKTRACK(F,A, b)
       level ← b
 return ⊤
```



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

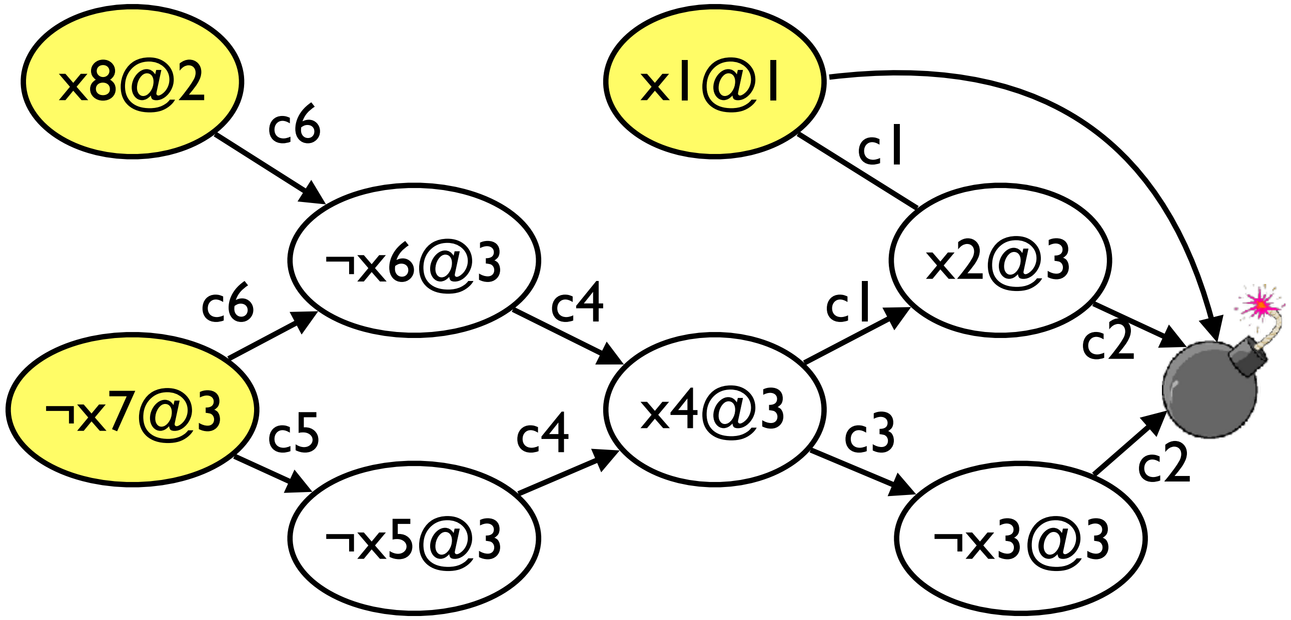
A **conflict clause** is implied by F and it blocks partial assignments (PAs) that lead to the current conflict.

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

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).

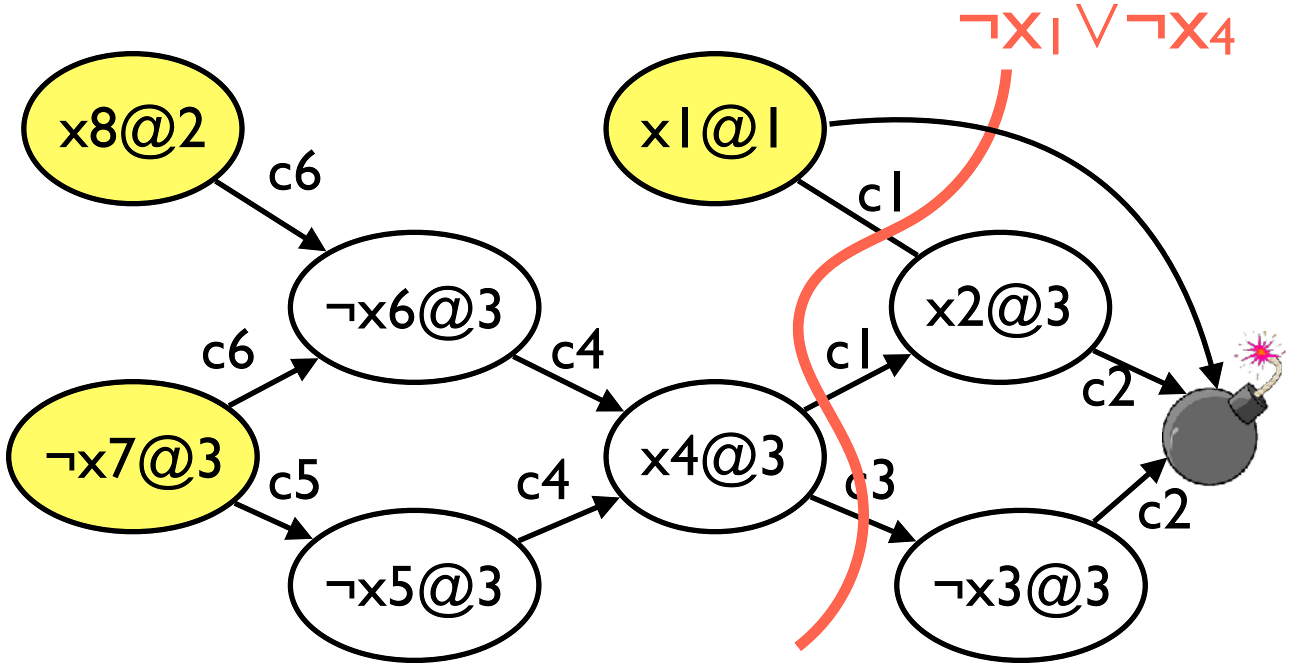
A **first UIP** is the UIP that is closest to the conflict.



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).

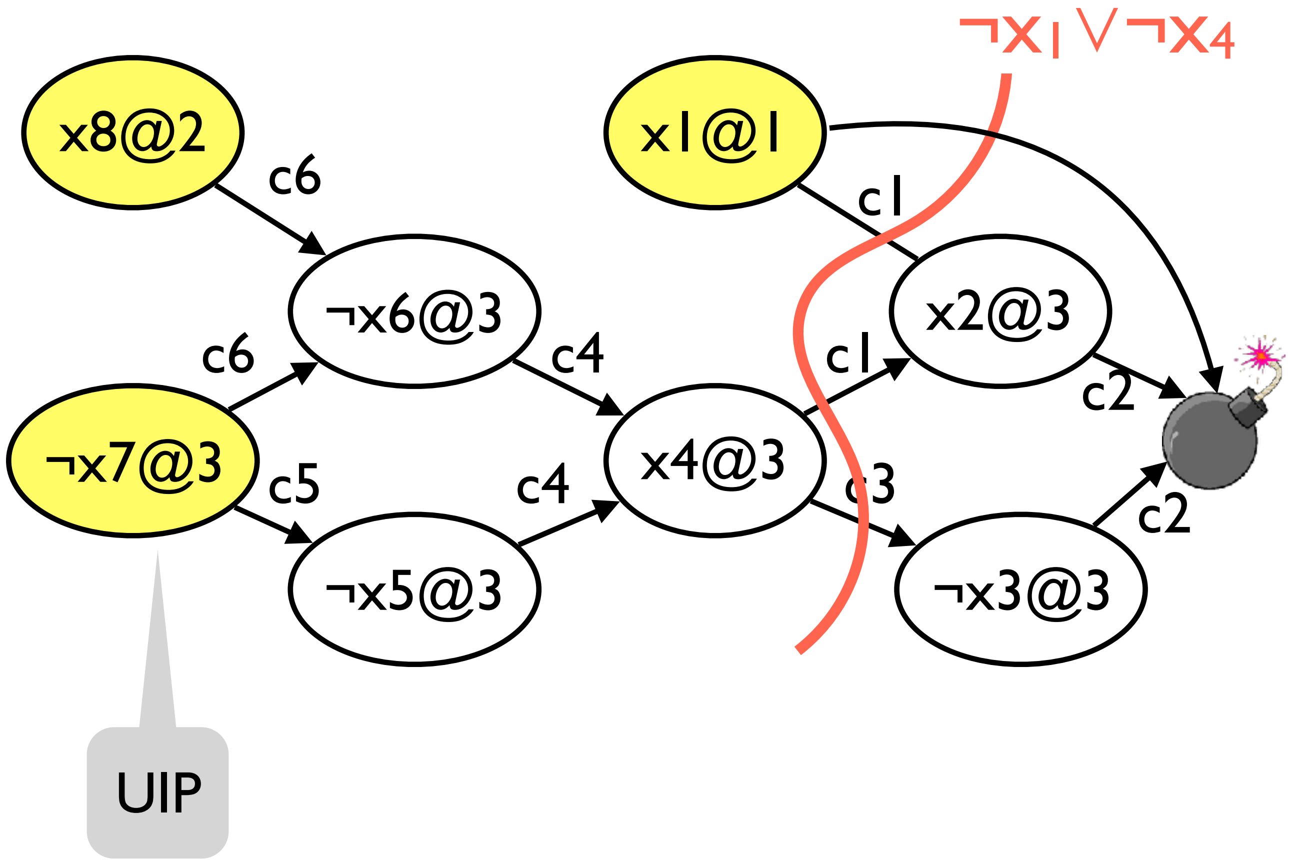
A **first UIP** is the UIP that is closest to the conflict.



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).

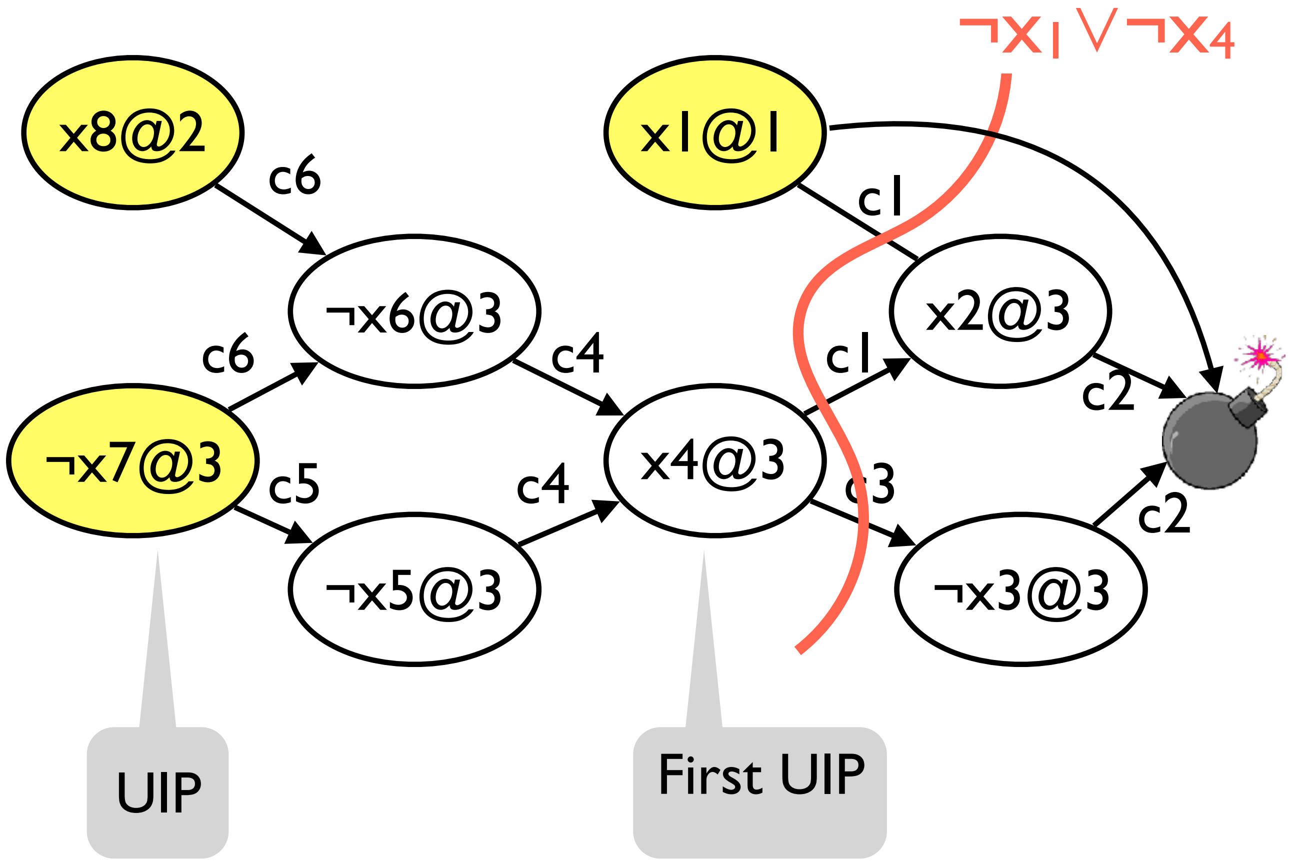
A **first UIP** is the UIP that is closest to the conflict.



Unique implication points (UIPs)

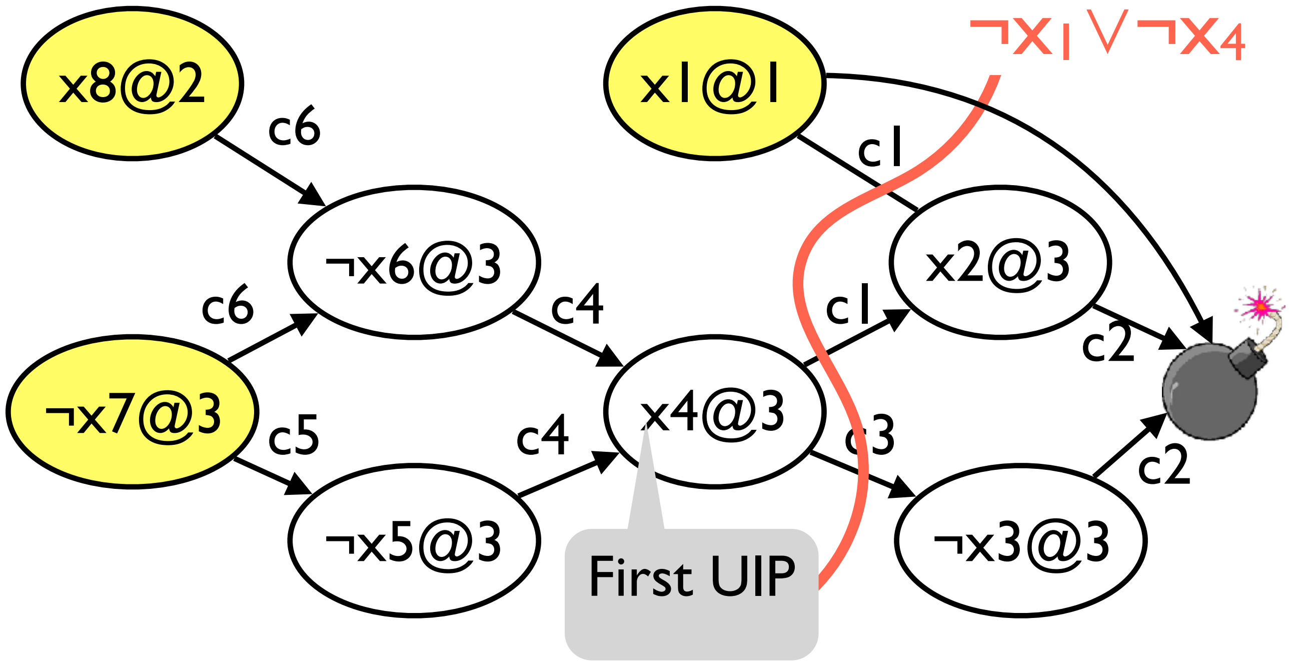
A **unique implication point** (UIP) is any node in the implication graph other than the conflict that is on all paths from the current decision literal (lit@d) to the conflict (κ@d).

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



Conflict analysis via resolution

- Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c′.
- Resolve c and c′.
- Set current clause be resolvent of c and c′.
- Repeat until current clause contains negation of the first UIP literal (as the **single literal** at **current** decision level)

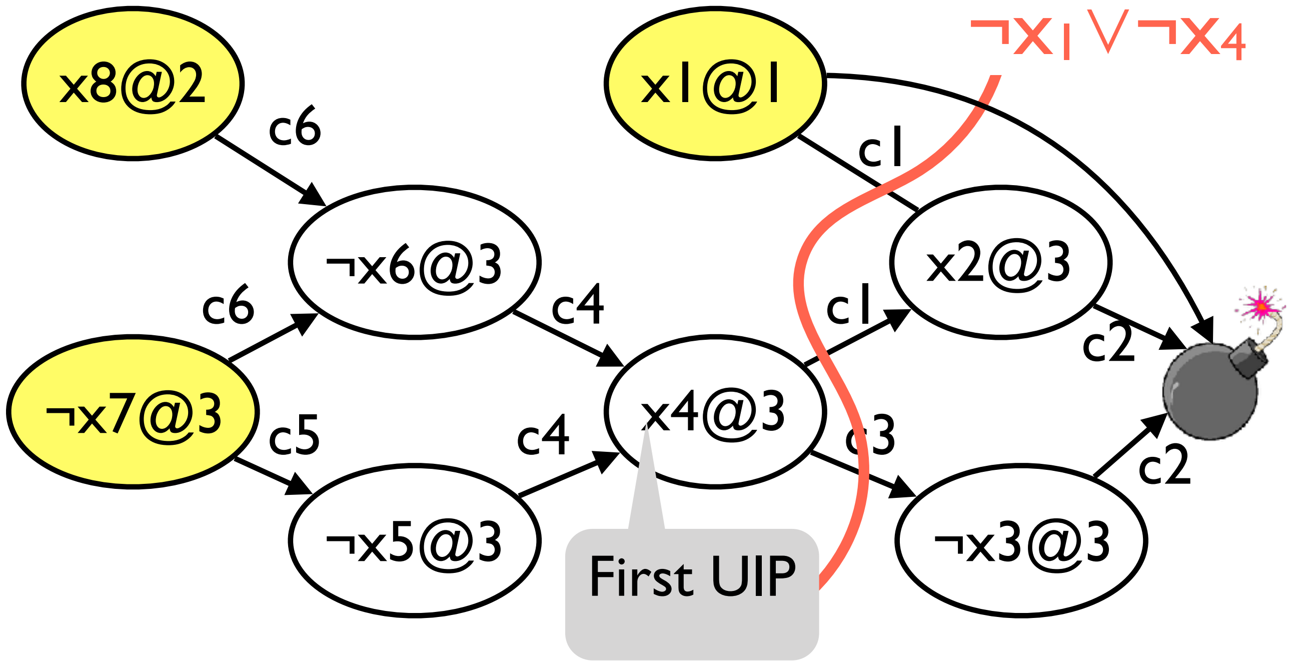


Unit resolution rule

$$\frac{a_1 \lor \ldots\ \beta \qquad b_1 \lor \ldots \lor b_m \lor \neg\beta}{a_1 \lor \ldots\ b_1 \lor \ldots \lor b_m}$$

Conflict analysis via resolution

- Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c′.
- Resolve c and c′.
- Set current clause be resolvent of c and c′.
- Repeat until current clause contains negation of the first UIP literal (as the **single literal** at **current** decision level)



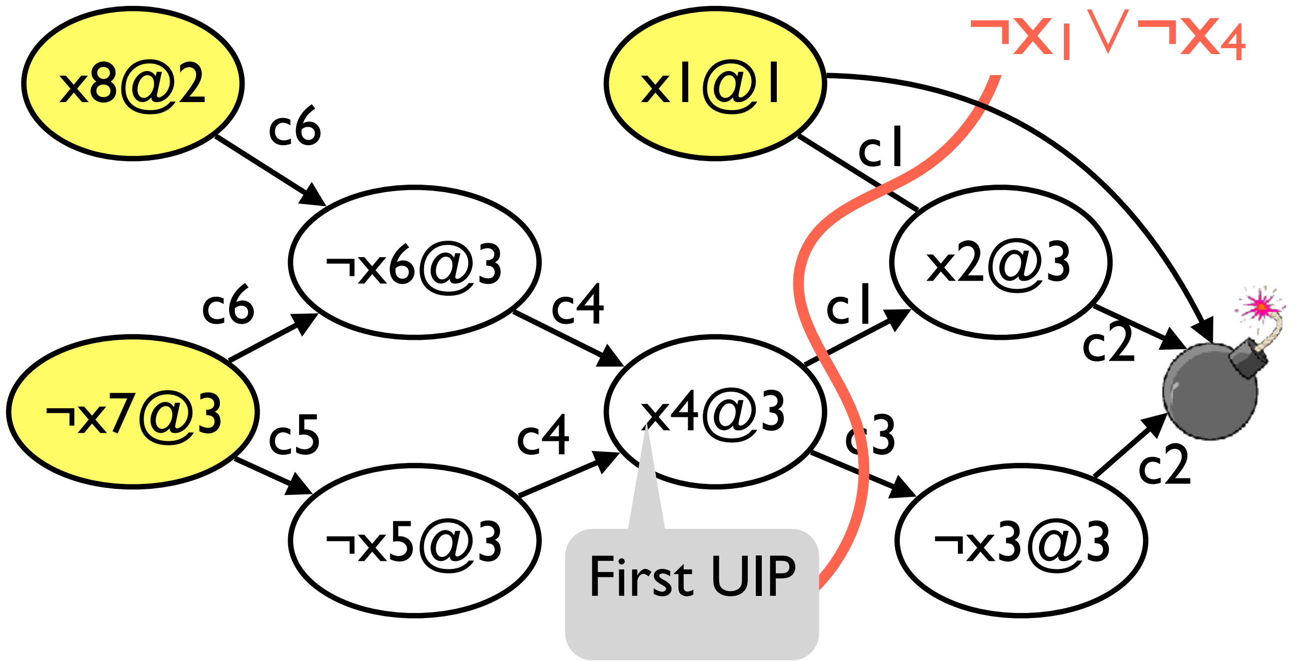
What is c? **c_2: $\neg x_1 \vee \neg x_2 \vee x_3$**

Unit resolution rule

$$\frac{a_1 \vee \ldots \beta \qquad b_1 \vee \ldots \vee b_m \vee \neg\beta}{a_1 \vee \ldots b_1 \vee \ldots \vee b_m}$$

Conflict analysis via resolution

- Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c′.
- Resolve c and c′.
- Set current clause be resolvent of c and c′.
- Repeat until current clause contains negation of the first UIP literal (as the **single literal** at **current** decision level)



What is c? **c_2: $\neg x_1 \vee \neg x_2 \vee x_3$**

Last assigned literal in c? **x_2**

Unit resolution rule

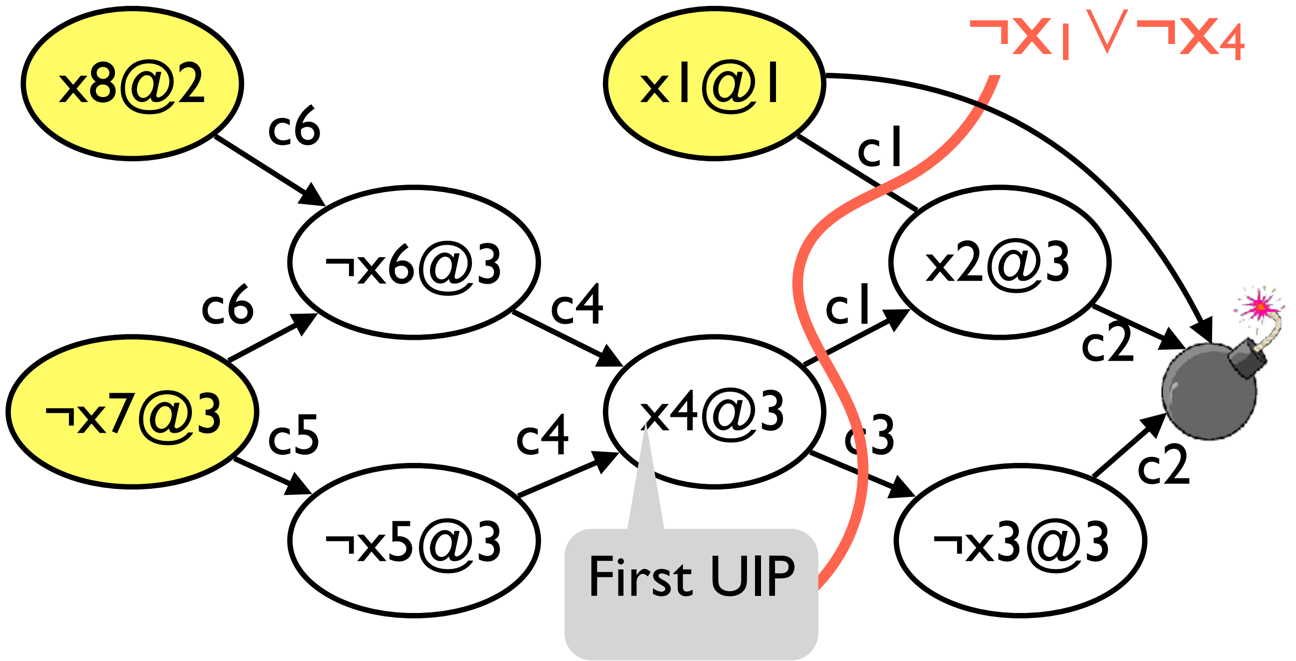
$$\frac{a_1 \vee \ldots\ \beta \qquad b_1 \vee \ldots \vee b_m \vee \neg\beta}{a_1 \vee \ldots\ b_1 \vee \ldots \vee b_m}$$

Conflict analysis via resolution

- Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c′.
- Resolve c and c′.
- Set current clause be resolvent of c and c′.
- Repeat until current clause contains negation of the first UIP literal (as the **single literal** at **current** decision level)

Unit resolution rule

$$\frac{a_1 \vee \ldots\ \beta \qquad b_1 \vee \ldots \vee b_m \vee \neg\beta}{a_1 \vee \ldots\ b_1 \vee \ldots \vee b_m}$$



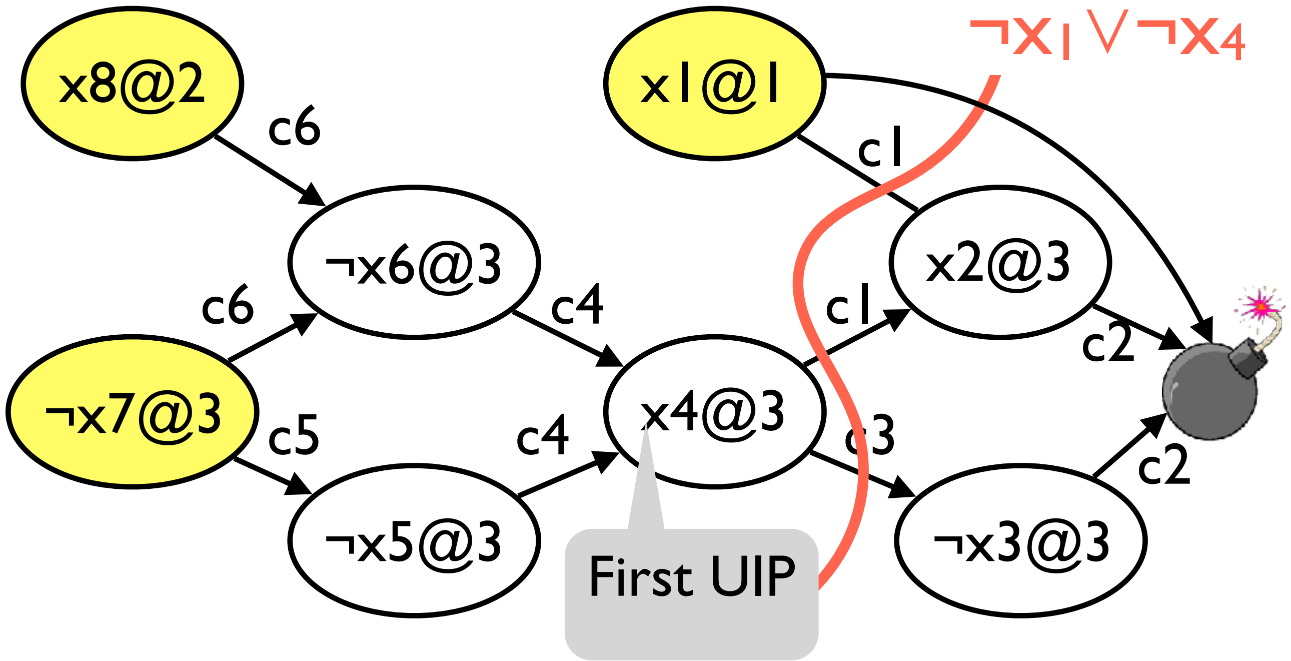
What is c? **c_2: $\neg x_1 \vee \neg x_2 \vee x_3$**

Last assigned literal in c? **x_2**

Incoming edge/clause c’ to **x_2**? **c_1: $\neg x_1 \vee x_2 \vee \neg x_4$**

Conflict analysis via resolution

- Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c′.
- Resolve c and c′.
- Set current clause be resolvent of c and c′.
- Repeat until current clause contains negation of the first UIP literal (as the **single literal** at **current** decision level)



What is c? **c_2: $\neg x_1 \vee \neg x_2 \vee x_3$**

Last assigned literal in c? **x_2**

Incoming edge/clause c’ to **x_2**? **c_1: $\neg x_1 \vee x_2 \vee \neg x_4$**

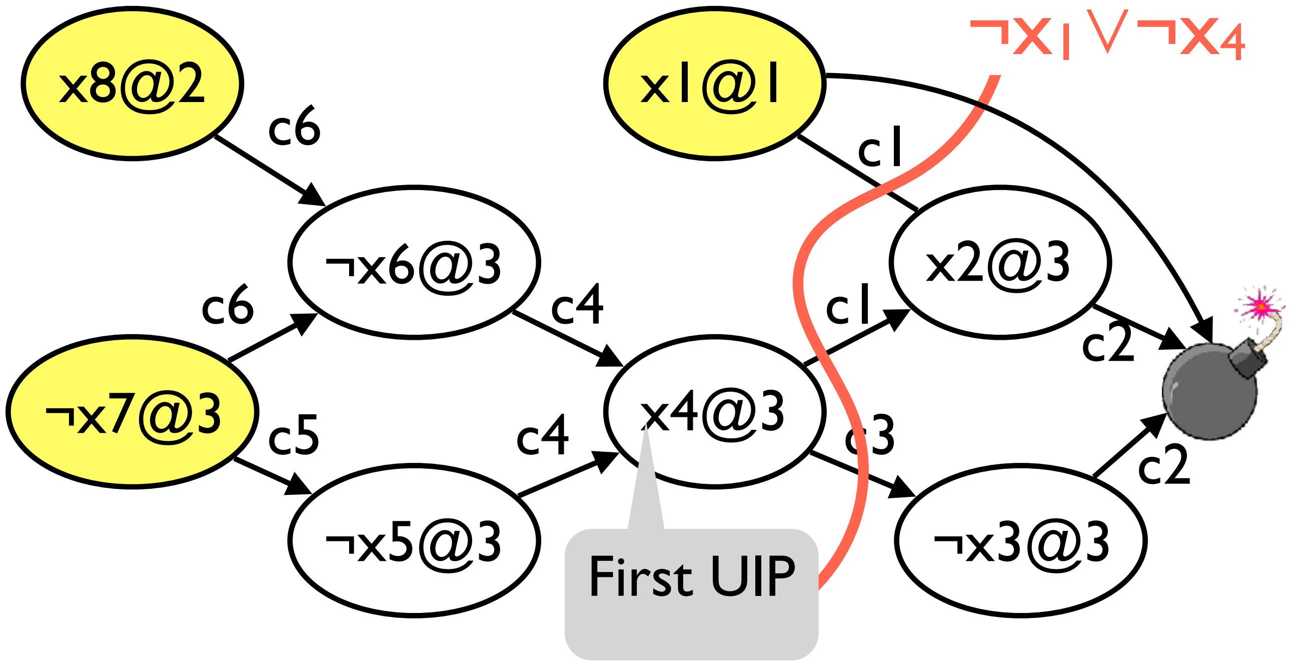
Resolve c and c’? **$\neg x_1 \vee x_3 \vee \neg x_4$**

Unit resolution rule

$$\frac{a_1 \vee \ldots\ \beta \qquad b_1 \vee \ldots \vee b_m \vee \neg\beta}{a_1 \vee \ldots\ b_1 \vee \ldots \vee b_m}$$

Conflict analysis via resolution

- Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c′.
- Resolve c and c′.
- Set current clause be resolvent of c and c′.
- Repeat until current clause contains negation of the first UIP literal (as the **single literal** at **current** decision level)



What is c? **c_2: $\neg x_1 \lor \neg x_2 \lor x_3$**

Last assigned literal in c? **x_2**

Incoming edge/clause c’ to **x_2**? **c_1: $\neg x_1 \lor x_2 \lor \neg x_4$**

Resolve c and c’? **$\neg x_1 \lor x_3 \lor \neg x_4$**

...

Unit resolution rule

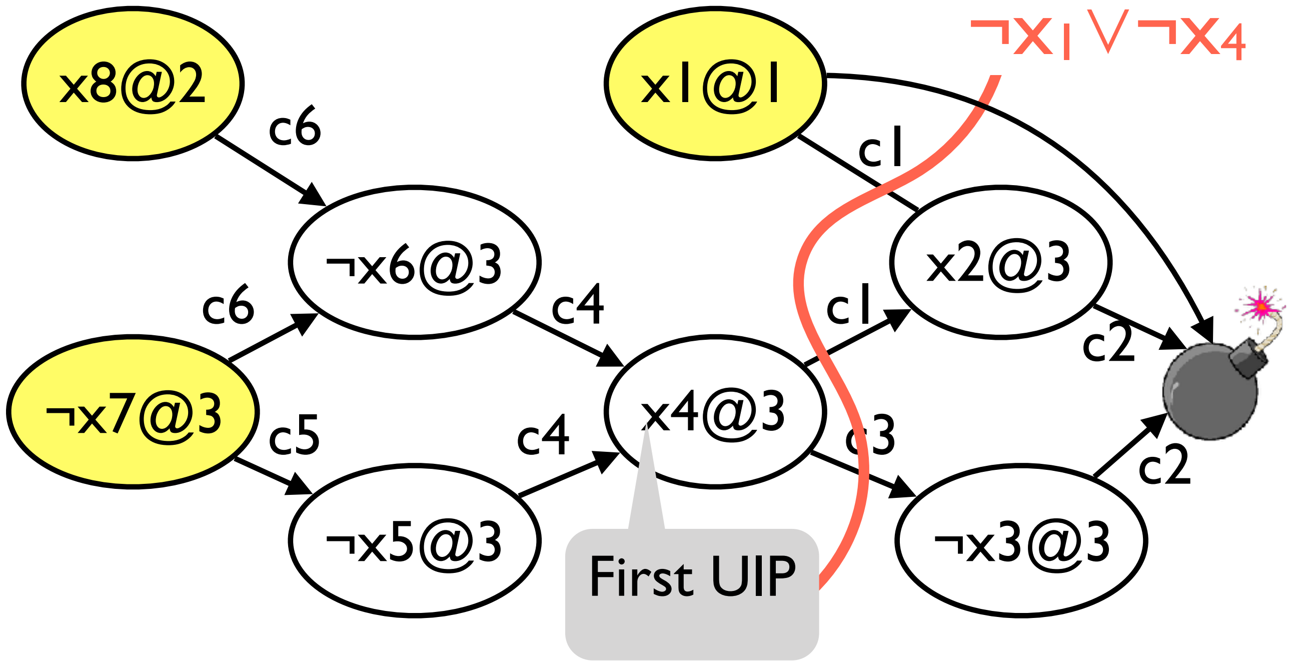
$$\frac{a_1 \lor \ldots\ \beta \qquad b_1 \lor \ldots \lor b_m \lor \neg\beta}{a_1 \lor \ldots\ b_1 \lor \ldots \lor b_m}$$

Conflict analysis via resolution

- Start with clause labeling incoming edge to conflict node, derive new clauses via resolution until we find literal in first UIP
- In current clause c, find last assigned literal l in c.
- Pick any incoming edge to l labeled with clause c′.
- Resolve c and c′.
- Set current clause be resolvent of c and c′.
- Repeat until current clause contains negation of the first UIP literal (as the **single literal** at **current** decision level)

Unit resolution rule

$$\frac{a_1 \vee \ldots \beta \qquad b_1 \vee \ldots \vee b_m \vee \neg\beta}{a_1 \vee \ldots b_1 \vee \ldots \vee b_m}$$



What is c? **c_2: $\neg x_1 \vee \neg x_2 \vee x_3$**

Last assigned literal in c? **x_2**

Incoming edge/clause c’ to **x_2**? **c_1: $\neg x_1 \vee x_2 \vee \neg x_4$**

Resolve c and c’? **$\neg x_1 \vee x_3 \vee \neg x_4$**

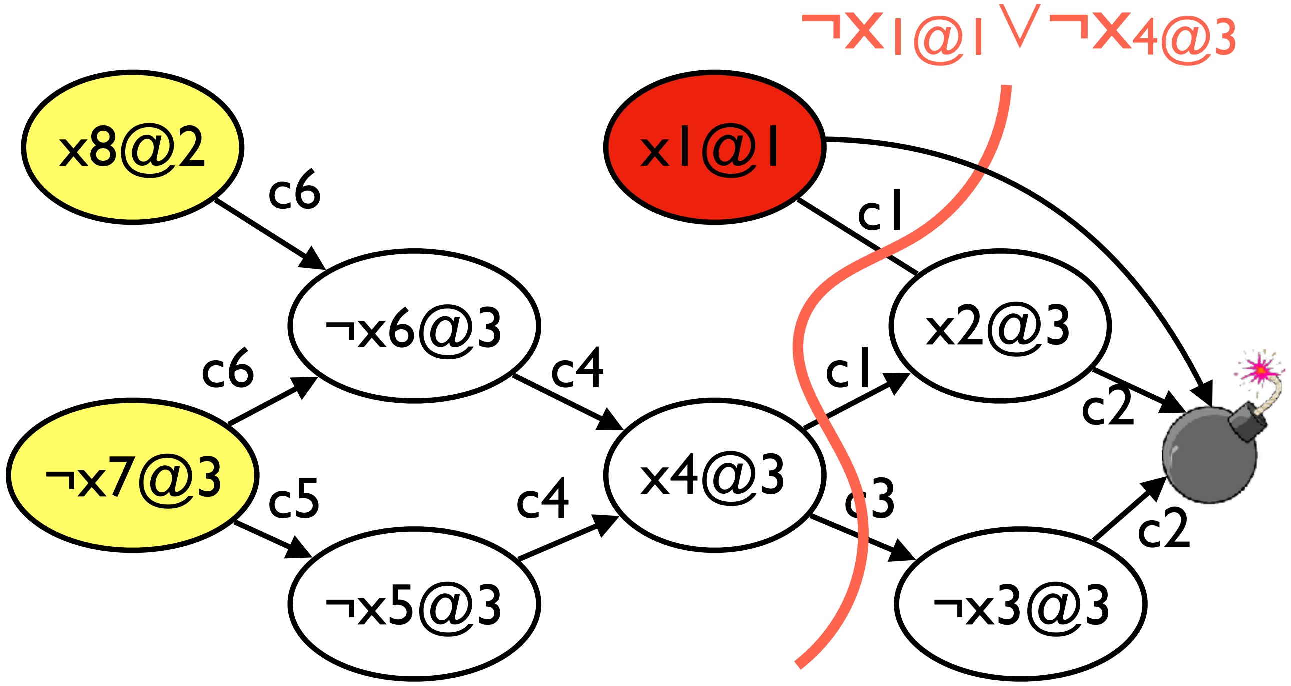
…

Conflict clause? **$\neg x_1 \vee \neg x_4$**

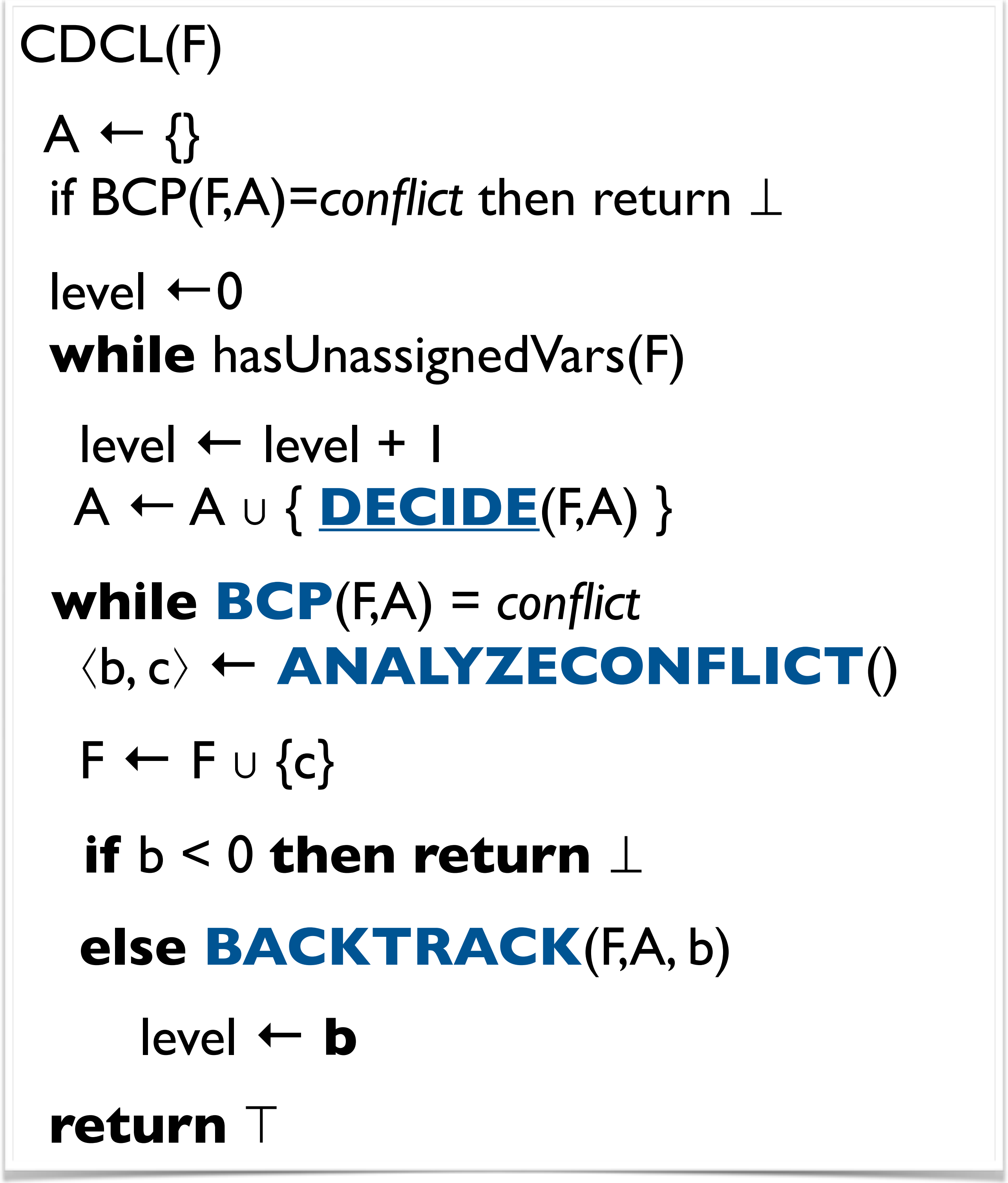
Conflict analysis: backtracking

Backtrack rule:

Second highest decision level for any literal in c



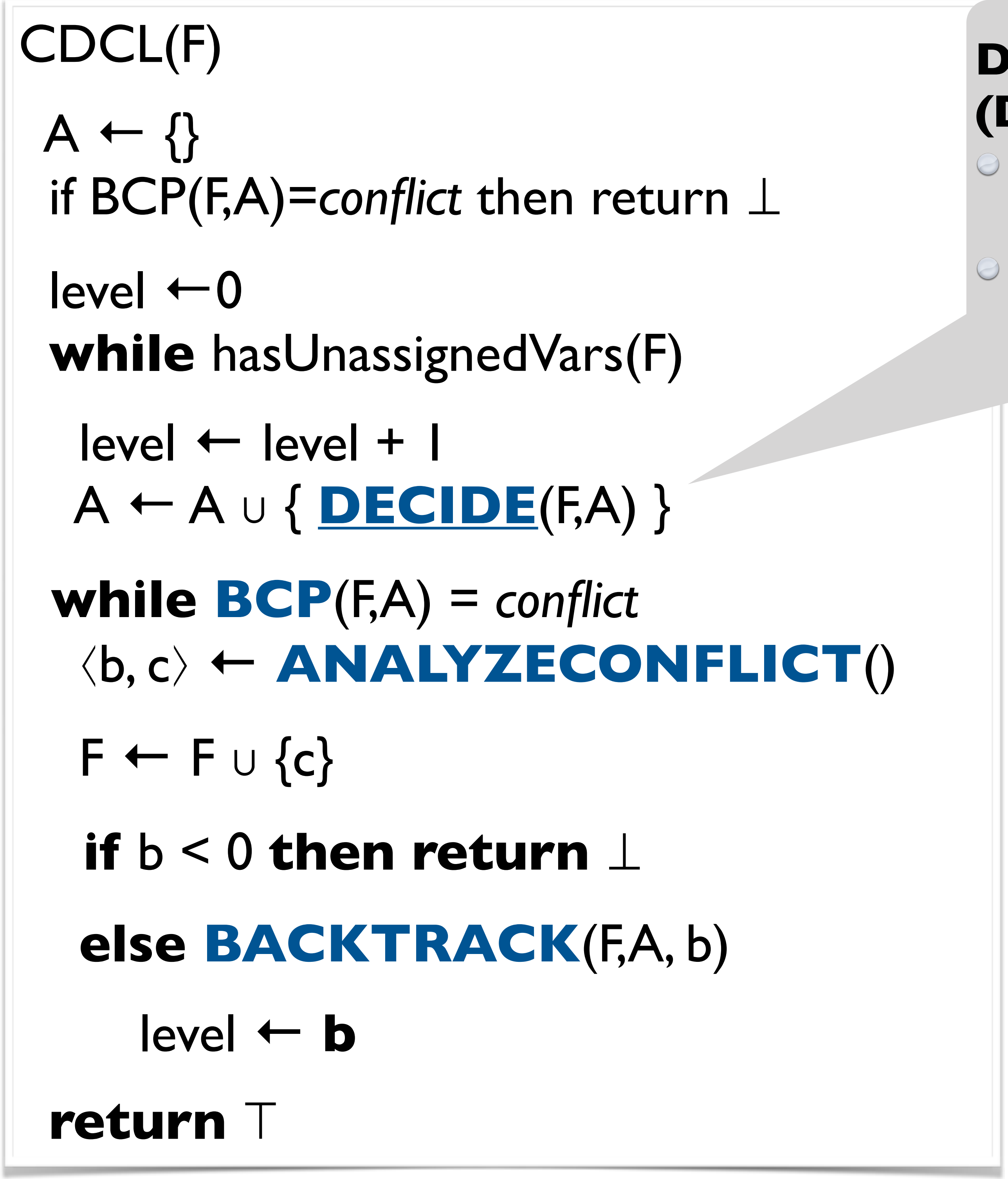
Decision heuristics



```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

Learning a SAT Solver from Single-Bit Supervision, ICLR'19

Decision heuristics



```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

Dynamic Largest Individual Sum (DLIS):

- Choose the literal that satisfies the most unresolved clauses
- expensive: complexity of making a decision proportional to the number of clauses

Learning a SAT Solver from Single-Bit Supervision, ICLR'19

Decision heuristics

```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

Dynamic Largest Individual Sum (DLIS):

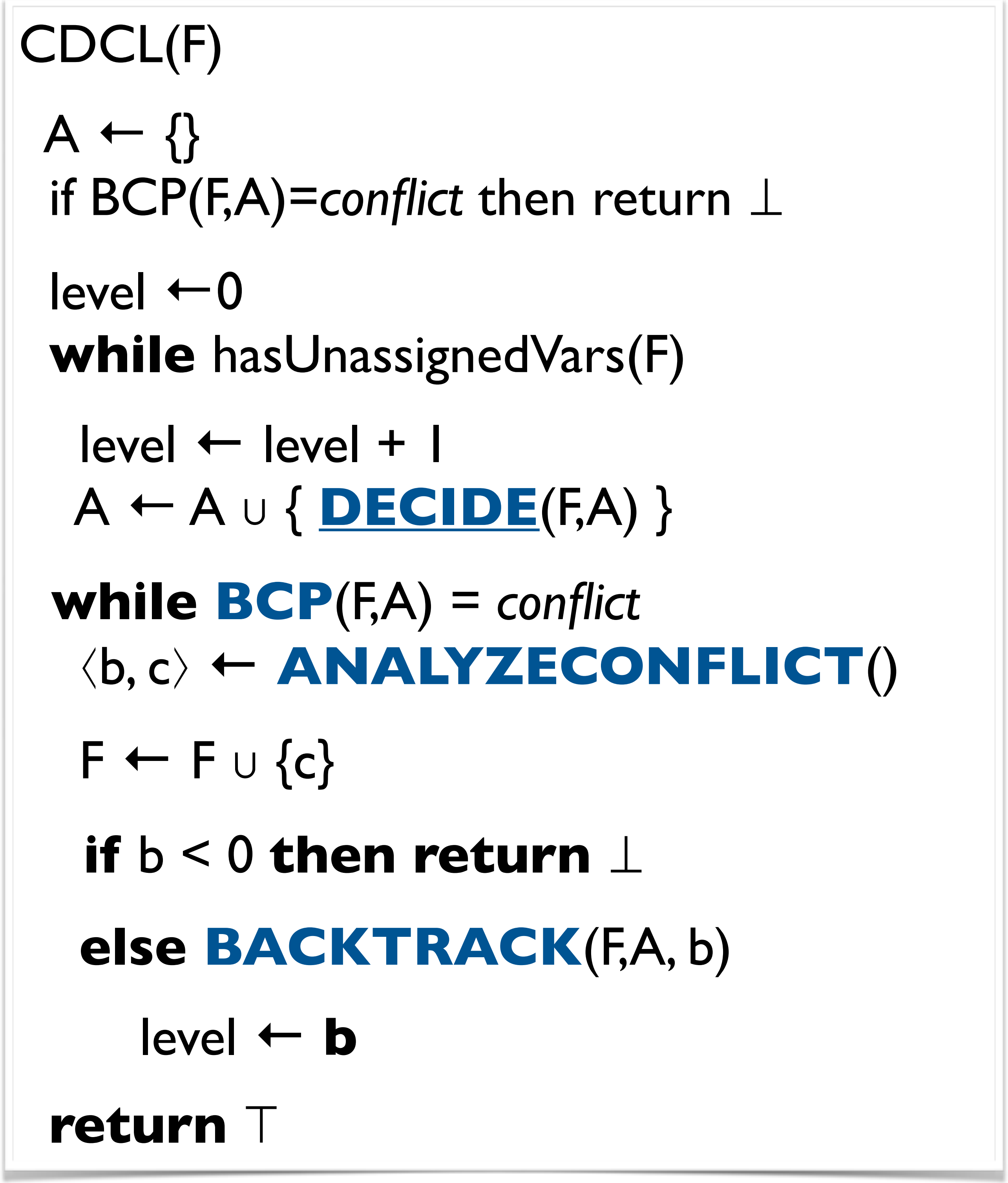
- Choose the literal that satisfies the most unresolved clauses
- expensive: complexity of making a decision proportional to the number of clauses

Variable State Independent Decaying Sum (VSIDS):

- 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 (zChaff)

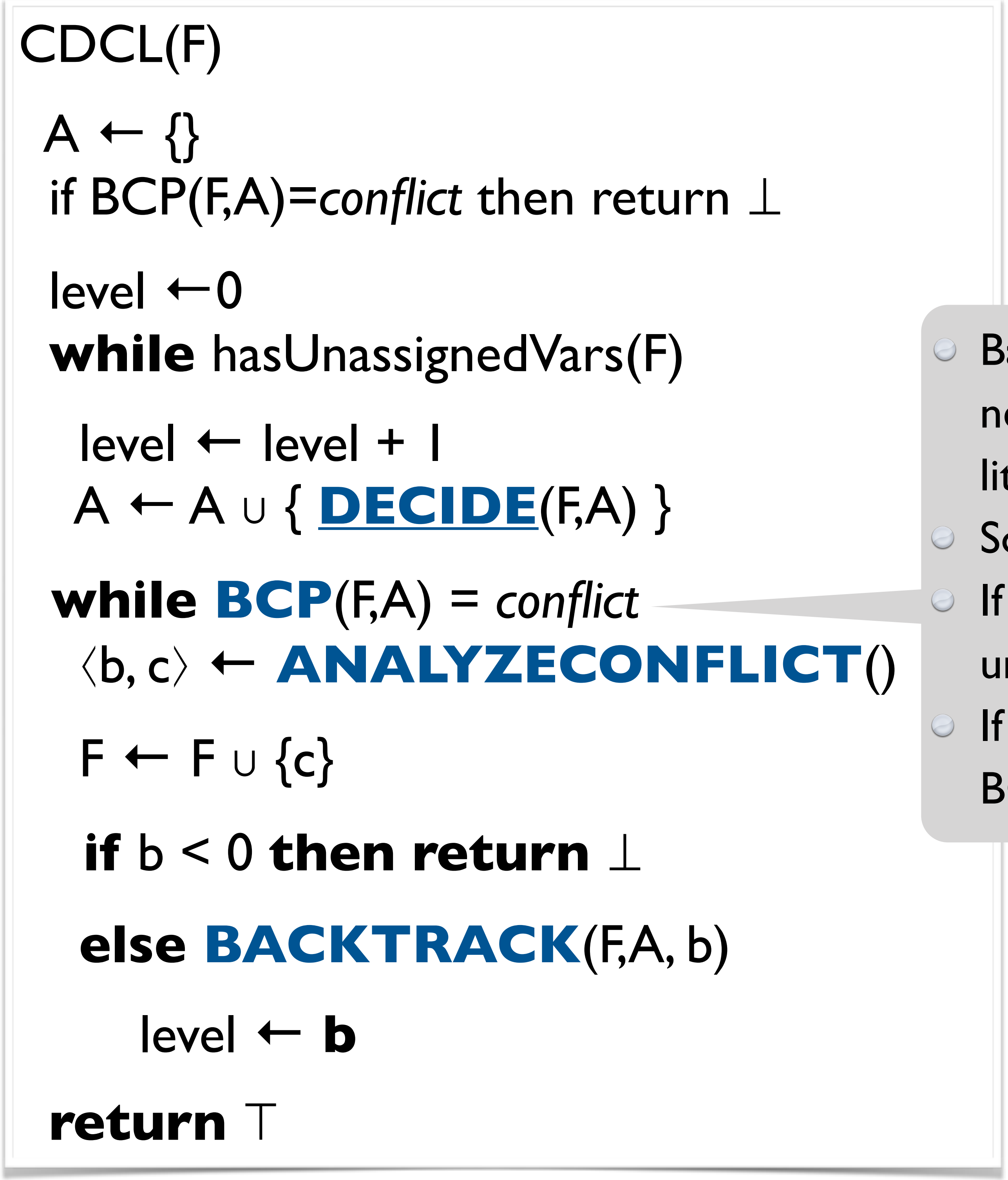
Learning a SAT Solver from Single-Bit Supervision, ICLR'19

BCP with watched literals (zChaff)



```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

BCP with watched literals (zChaff)



```
CDCL(F)
  A ← {}
  if BCP(F,A)=conflict then return ⊥
  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 ⊥
      else BACKTRACK(F,A, b)
        level ← b
  return ⊤
```

- 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.

TODOs by next lecture

- Work on the 2nd reading assignment
- Submit the 1st homework
- Start working your proposal
- Discuss your final project during office hour!