Lecture 4: Boolean Satisfiability Problem

Shuai Li

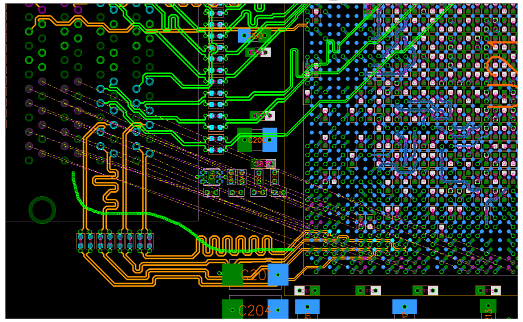
John Hopcroft Center, Shanghai Jiao Tong University

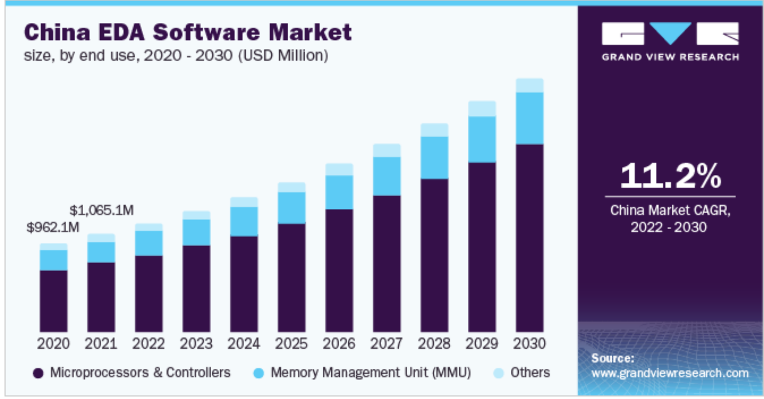
https://shuaili8.github.io

https://shuaili8.github.io/Teaching/CS3317/index.html

Part of slide credits: UW

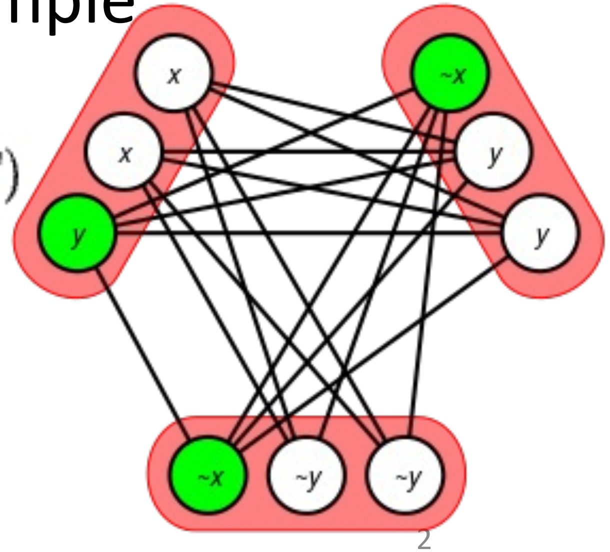
SAT





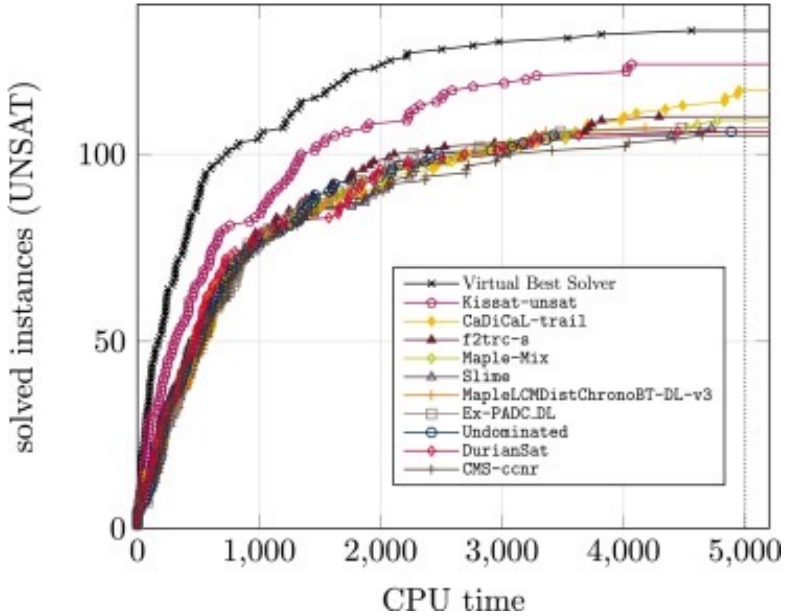
- Electronic design automation (EDA) is increasingly important
- EDA problems can be transformed into combinatorial optimization problems

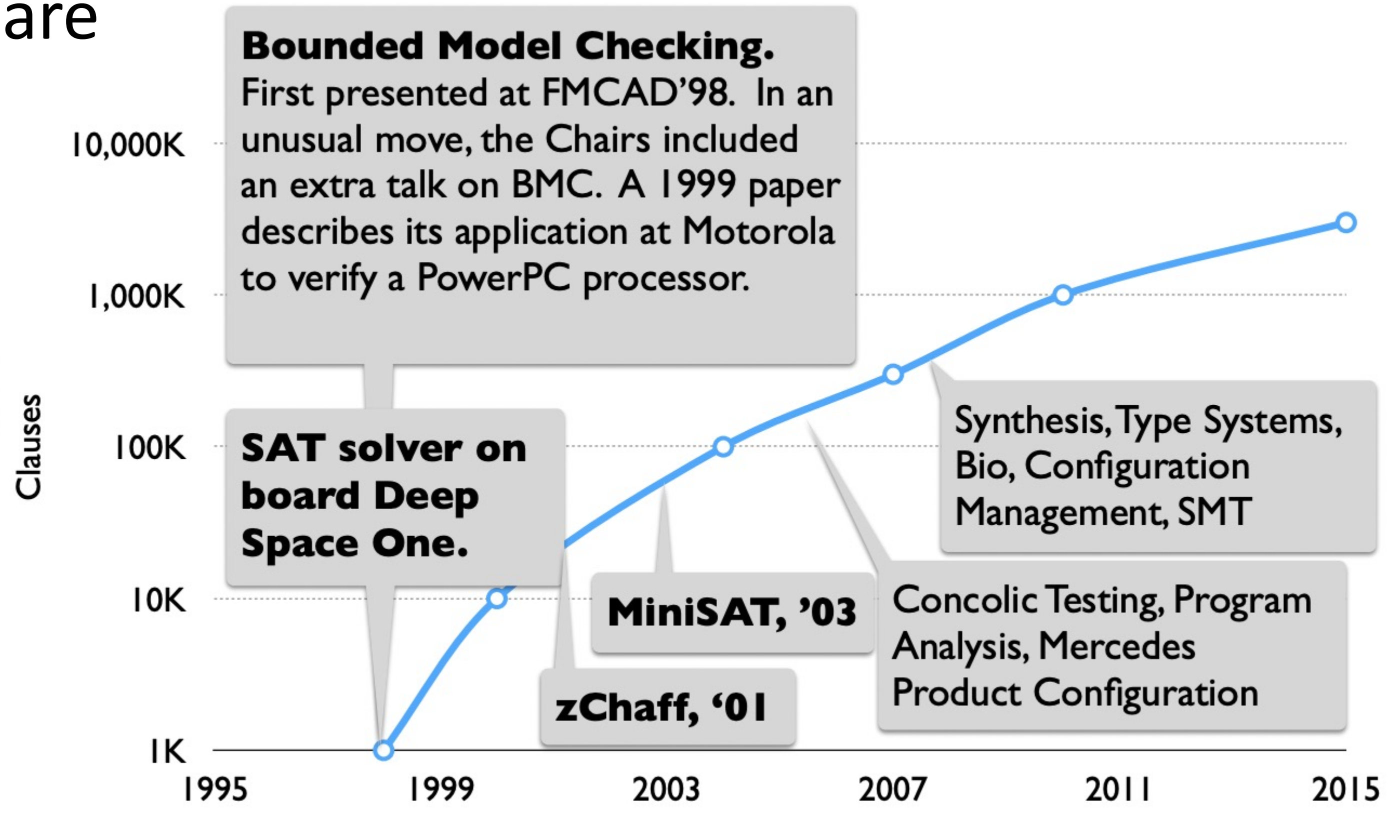
- The Boolean Satisfiability Problem is a fundamental example
  - Boolean formulas with Boolean variables
  - Conjunctive Normal Form (CNF): $(A \vee \neg B \vee \neg C) \wedge (\neg D \vee E \vee F)$
  - Satisfiable: The formula has an assignment under which the formula evaluates to True
  - Unsatisfiable: No such assignment exists for the formula
  - Reduce to find a clique in c-partite graph



SAT Solver

- SAT problem is NP-Complete with 2^n possible assignments
- In real world problems, there are often logical structures in the problem that an algorithm can utilize to search better





Based on a slide from Vijay Ganesh

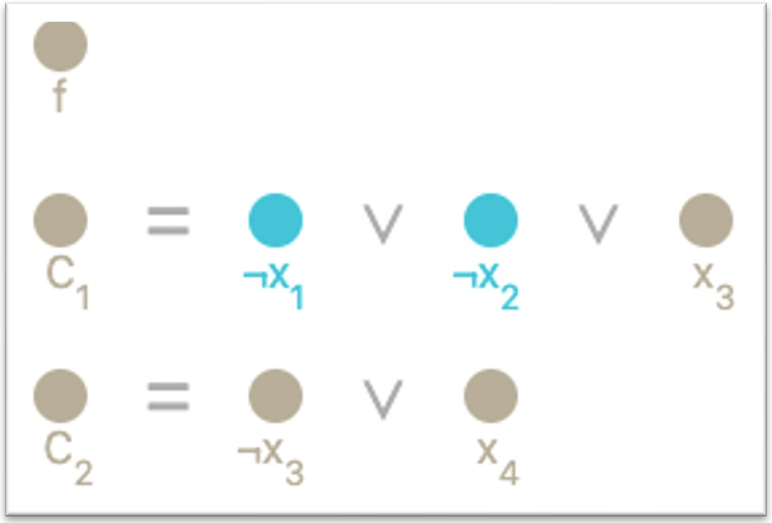
3-SAT

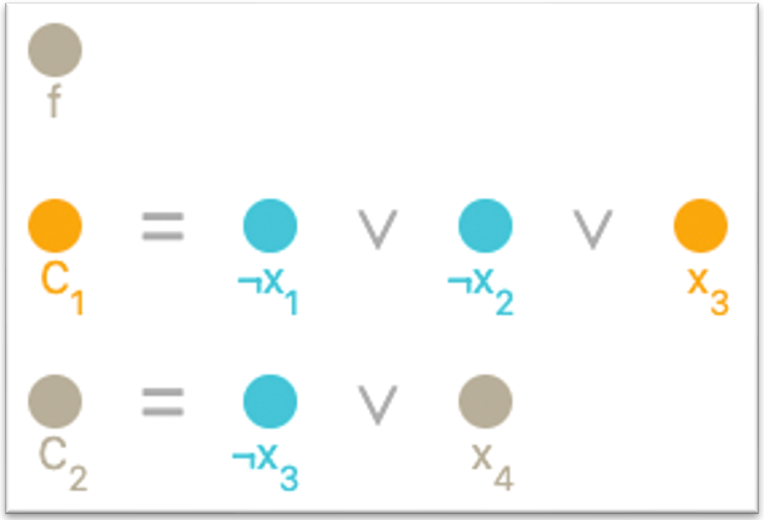
- To reduce the unrestricted SAT problem to 3-SAT, transform each clause $l_1 \lor \cdots \lor l_n$ to a conjunction of $n - 2$ clauses
  - not logically equivalent, but equisatisfiable
- Literal: either a variable or the negation of a variable
- Clause: a disjunction of literals (or a single literal)
- 3-SAT is one of Karp's 21 NP-complete problems, and it is used as a starting point for proving that other problems are also NP-hard
- Example: f = (¬x1 ∨ ¬x2 ∨ x3) ∧ (¬x3 ∨ x4)

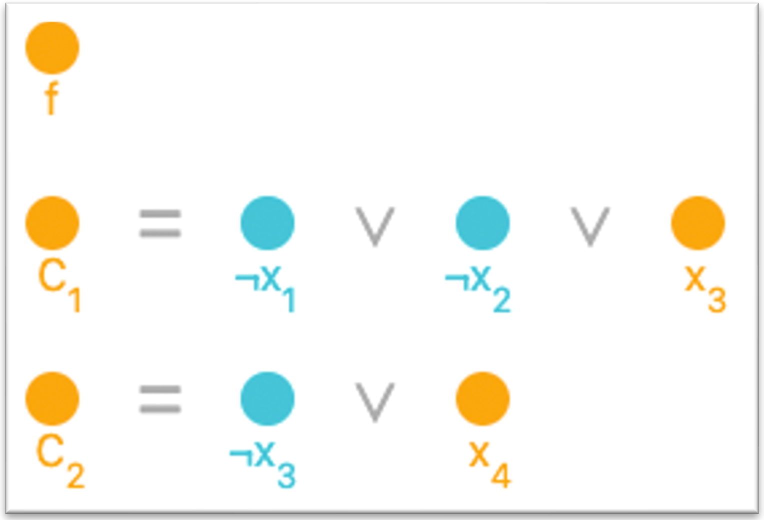
$$
\begin{aligned}
&(l_1 \lor l_2 \lor x_2) \land \\
&(\neg x_2 \lor l_3 \lor x_3) \land \\
&(\neg x_3 \lor l_4 \lor x_4) \land \cdots \land \\
&(\neg x_{n-3} \lor l_{n-2} \lor x_{n-2}) \land \\
&(\neg x_{n-2} \lor l_{n-1} \lor l_n)
\end{aligned}
$$

Boolean Constraint Propagation (BCP)

- Unit clause: A clause is unit under a partial assignment when that assignment makes every literal in the clause unsatisfied but leaves a single literal undecided
- Example: guess x1 and x2 are both true







Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

- a SAT solver: recursive backtracking + BCP

- DPLL:
  - Run BCP on the formula
  - If the formula evaluates to True, return True
  - If the formula evaluates to False, return False
  - If the formula is still Undecided:
    - Choose the next unassigned variable
    - Return (DPLL with that variable True) || (DPLL with that variable False)

- Demo

Shortcomings of DPLL

- DPLL:
  - Run BCP on the formula
  - If the formula evaluates to True, return True
  - If the formula evaluates to False, return False
  - If the formula is still Undecided:
    - Choose the next unassigned variable
    - Return (DPLL with that variable True) || (DPLL with that variable False)

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 partial assignment became doomed at a lower level

Conflict Driven Clause Learning (CDCL)

- CDCL improves on all three aspects!

- 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

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

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

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

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

$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…

CDCL by example 2

- 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

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

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

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

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

$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

...

...

$x_1@1$

CDCL by example 3

- 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

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

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

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

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

$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…

x_1@1

CDCL by example 4

- 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

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

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

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

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

$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…

$x_8 @ 2$ $x_1 @ 1$

CDCL by example 5

- 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

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

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

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

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

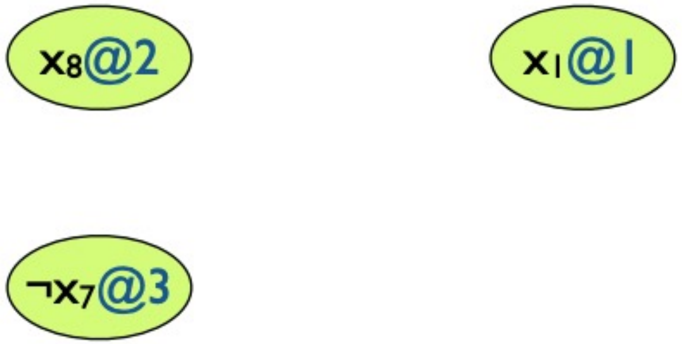
$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…



CDCL by example 6

- 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

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

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

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

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

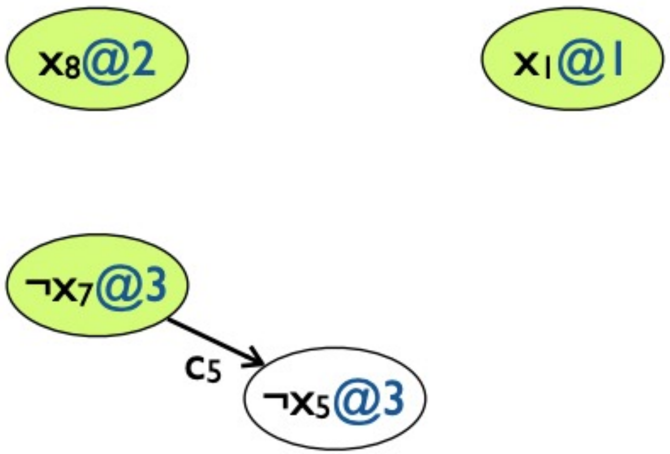
$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…



CDCL by example 7

- 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

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

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

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

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

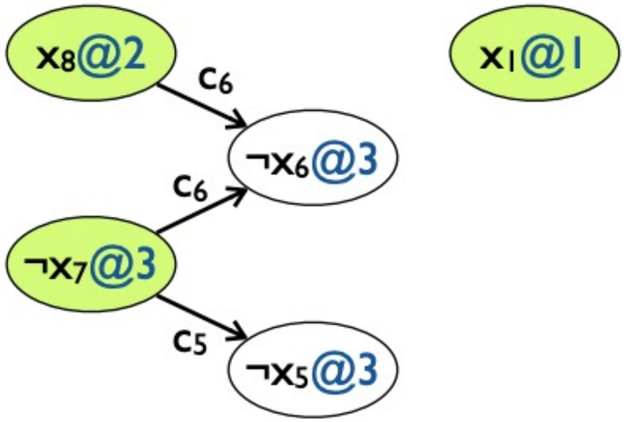
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

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

…

…



CDCL by example 8

- 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

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

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

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

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

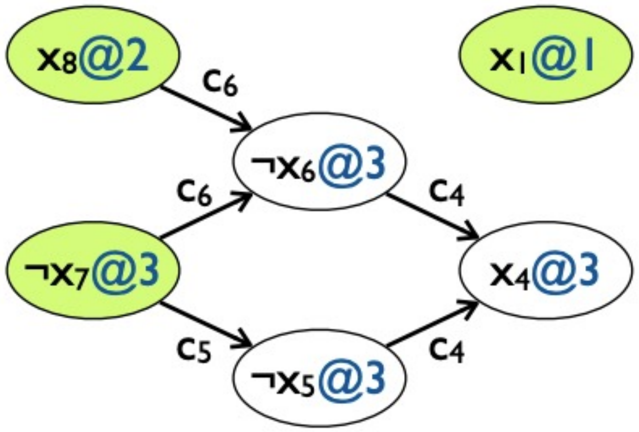
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

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

…

…



CDCL by example 9

- 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

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

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

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

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

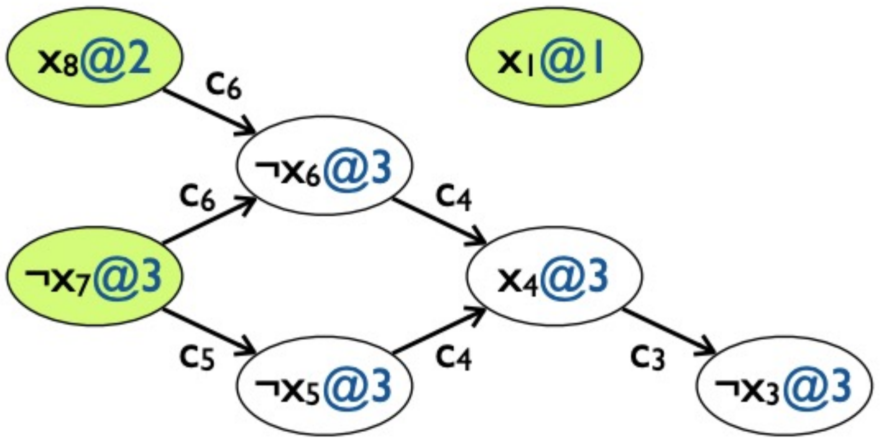
$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…



CDCL by example 10

- 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

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

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

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

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

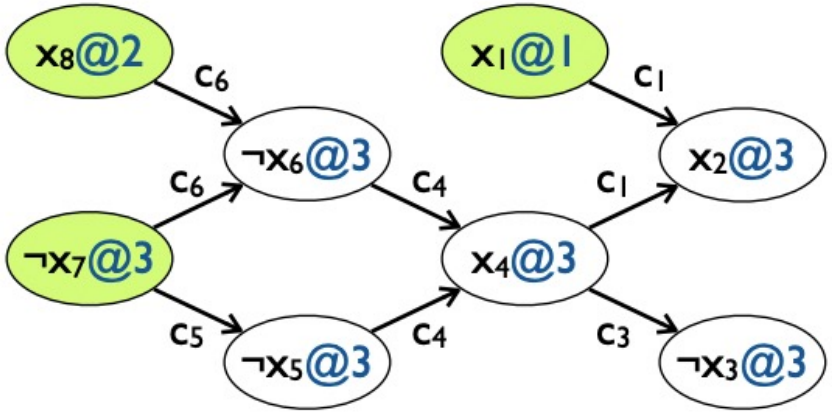
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

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

…

…



CDCL by example 11

- 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

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

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

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

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

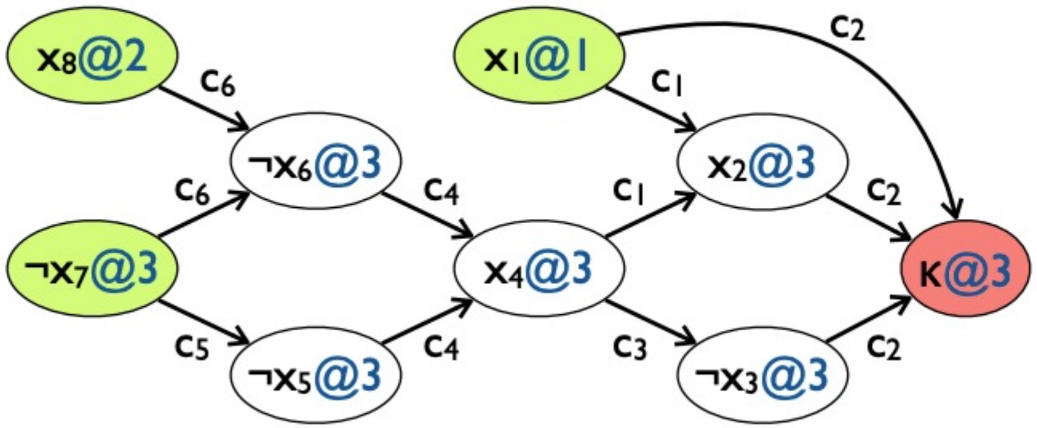
$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…



CDCL by example 12

- 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,-x1 v -x4)

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

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

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

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

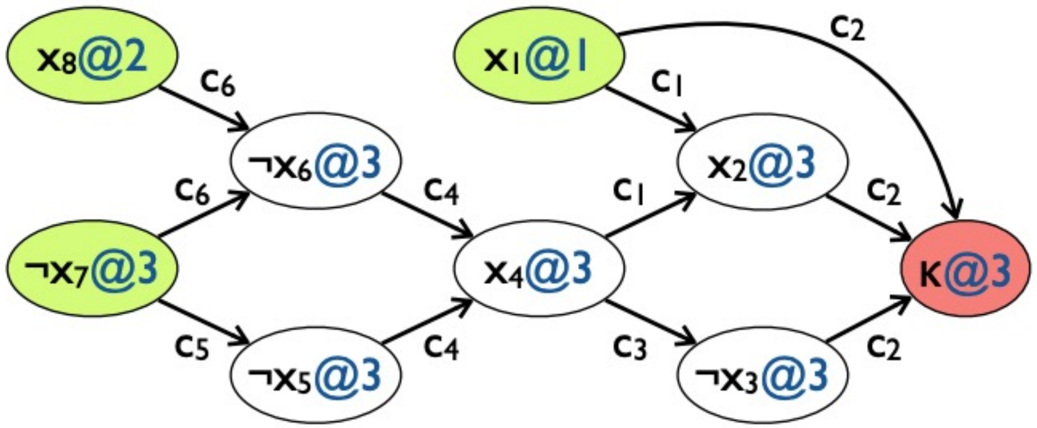
$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

…



CDCL by example 13

- 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,-x1 v -x4)

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

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

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

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

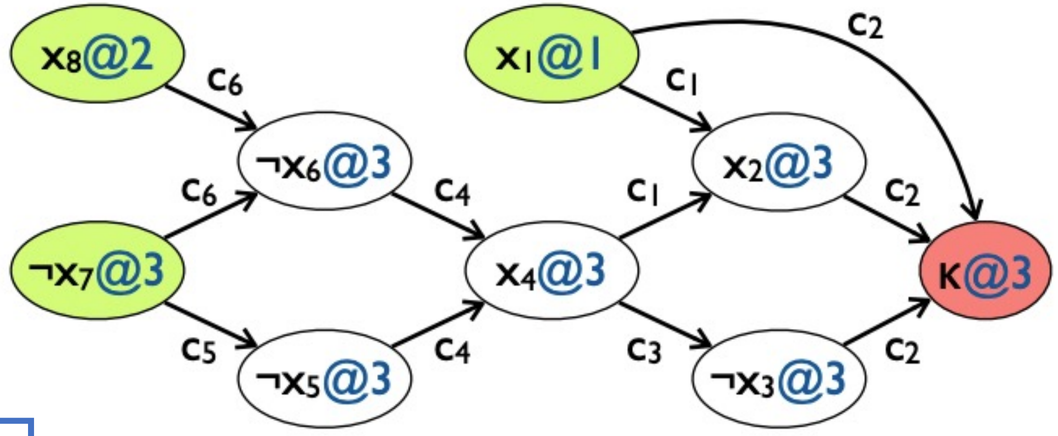
$c_4 : x_4 \vee x_5 \vee x_6$

$c_5 : \neg x_5 \vee x_7$

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

…

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



CDCL by example 14

- 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,-x1 v -x4)

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

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

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

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

$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

$c : \neg x_1 \lor \neg x_4$

CDCL by example 14

- 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,-x1 v -x4)

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

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

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

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

$c_4 : x_4 \lor x_5 \lor x_6$

$c_5 : \neg x_5 \lor x_7$

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

…

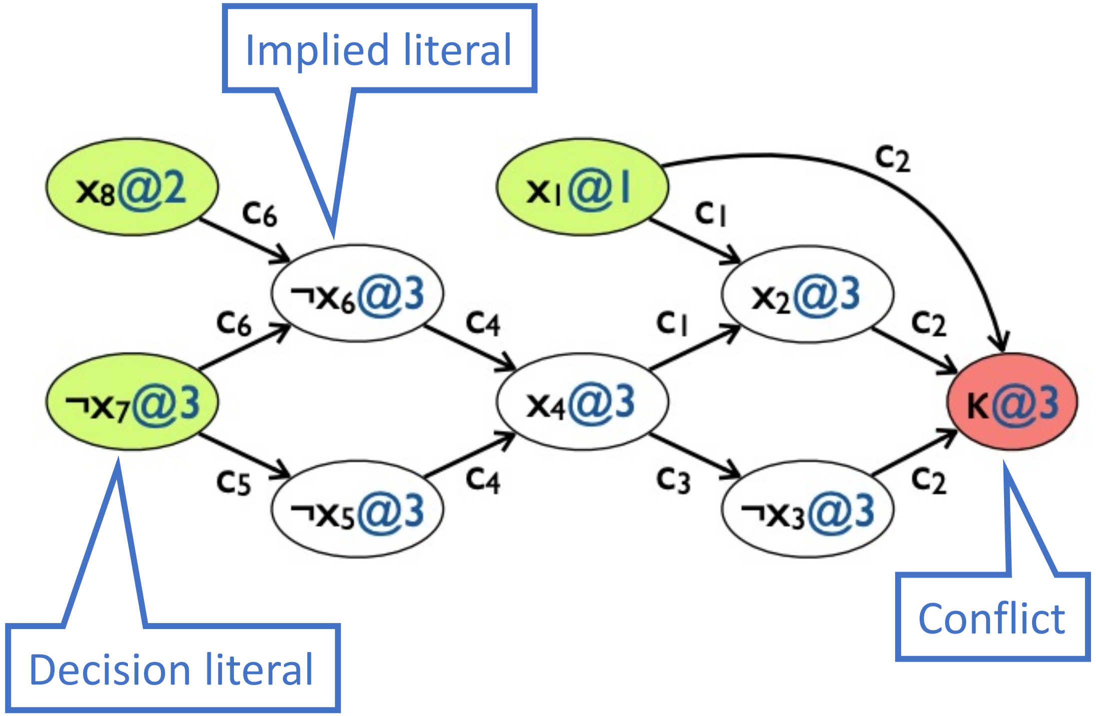
$c \;: \neg x_1 \lor \neg x_4$

Conflict clause is unit after backtracking

$x_1@1$

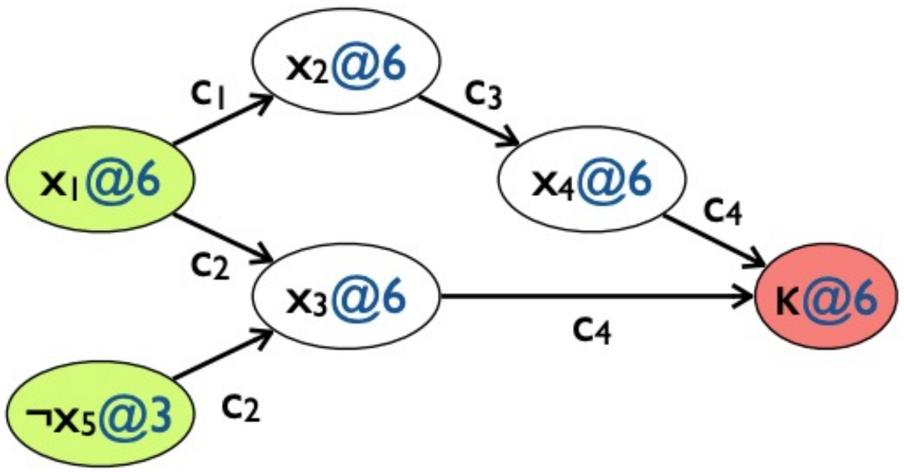
Implication graph

- 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 partial assignment (PA)
  - ⟨v, w⟩ ∈ E iff v ≠ w, ¬v ∈ antecedent(w), and ⟨v, w⟩ is labeled with antecedent(w)
- A unit clause c is an antecedent of its sole unassigned literal



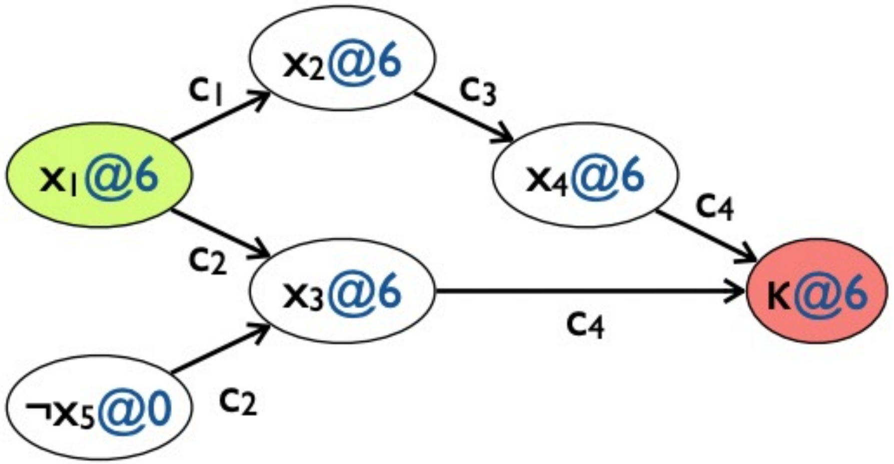
Quiz a

- What clauses gave rise to this implication graph?
- c1 :
- c2 :
- c3 :
- c4 :



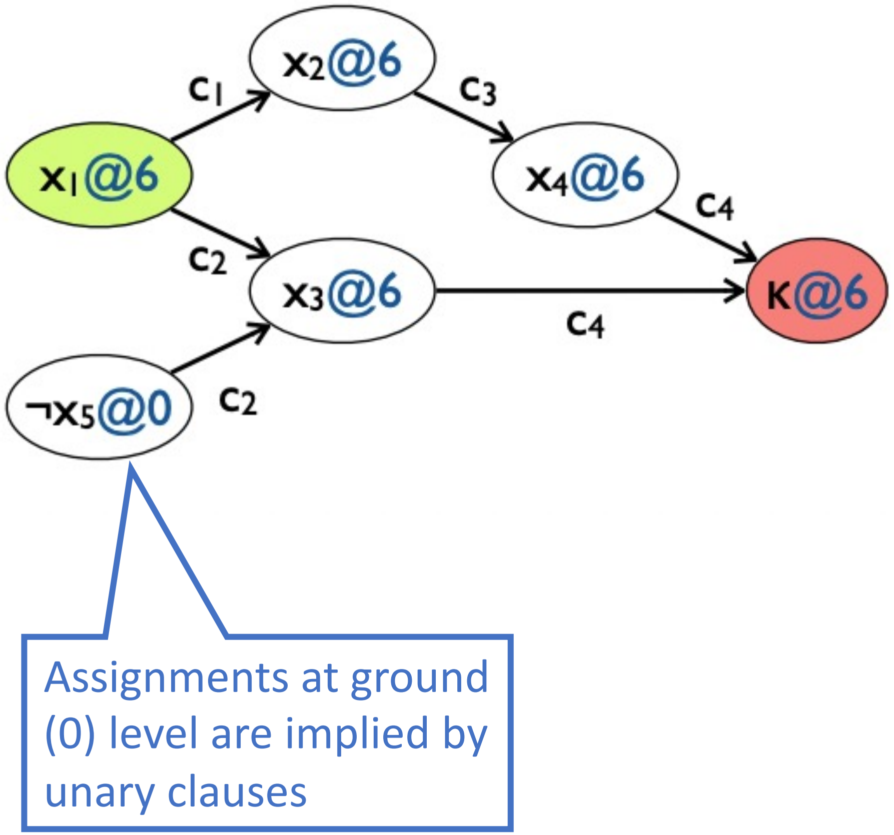
Quiz b

- What clauses gave rise to this implication graph?
- c1 :
- c2 :
- c3 :
- c4 :



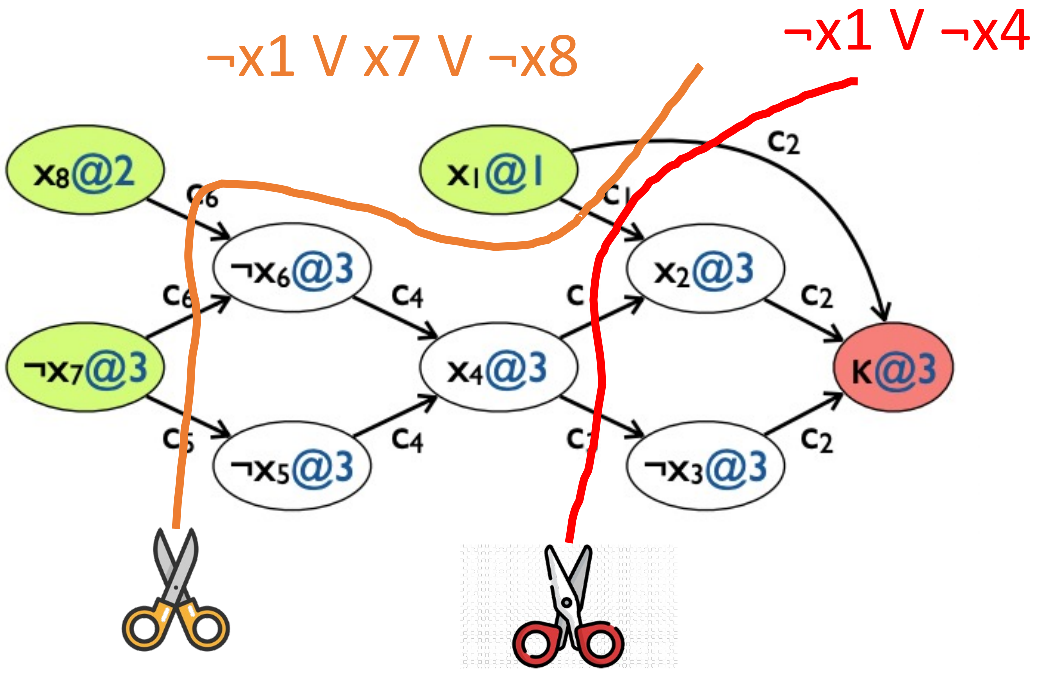
Quiz b-2

- What clauses gave rise to this implication graph?
- c1 :
- c2 :
- c3 :
- c4 :
- c5: ¬x5



Using an implication graph to analyze a conflict

```
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
```

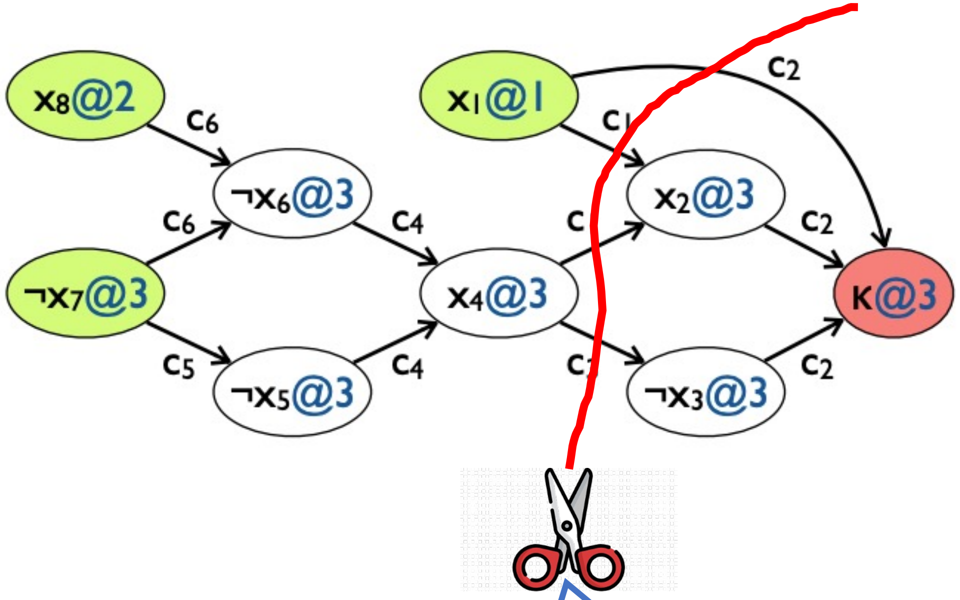


- A conflict clause is implied by F and it blocks PAs that lead to the current conflict
- Every cut that separates sources from the sink defines a valid conflict clause

Using an implication graph to analyze a conflict 2

```
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
```

¬x1 V ¬x4

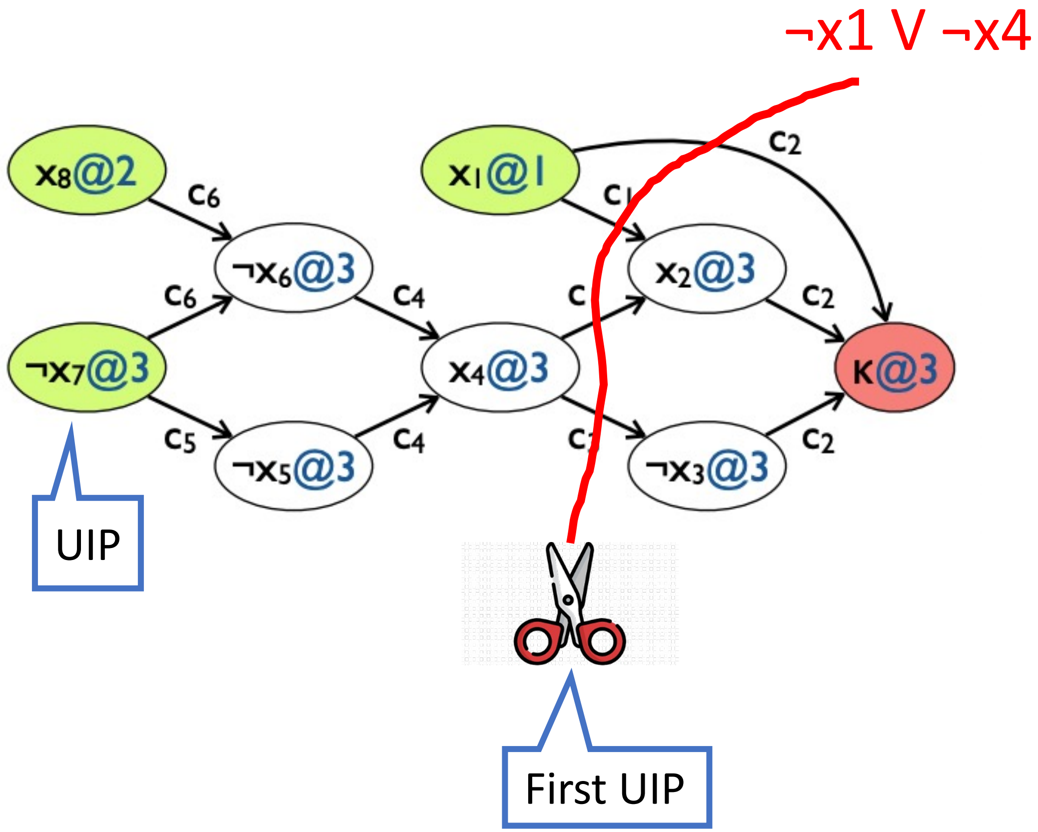


Cut after the first unique implication point to get the shortest conflict clause

Unique implication points (UIPs)

- A 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

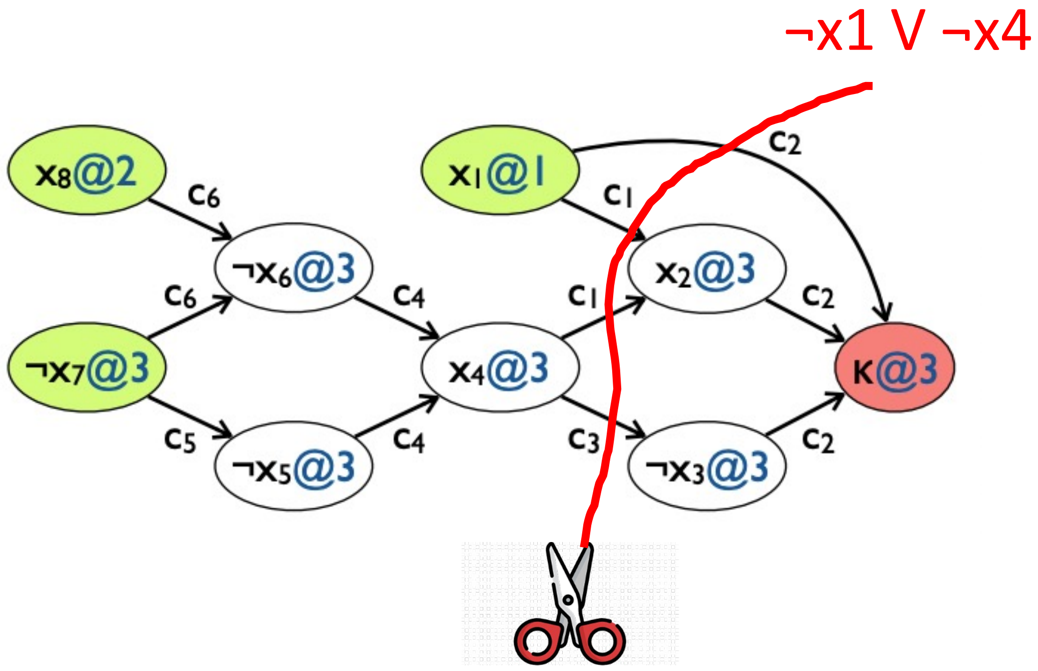


ANALYZECONFLICT: Computing the conflict clause

- ANALYZECONFLICT()
  - d ← level(conflict)
  - if d = 0 then return -1
  - c ← antecedent(conflict)
  - repeat
    - t ← lastAssignedLitAtLevel(c, d)
    - v ← varOfLit(t)
    - ante ← antecedent(t)
    - c ← resolve(ante, c, v)
  - until oneLitAtLevel(c, d)
  - b ←…
  - return ⟨b, c⟩

Binary resolution rule

$$\frac{A \vee B,\ \neg B \vee C}{A \vee C}$$

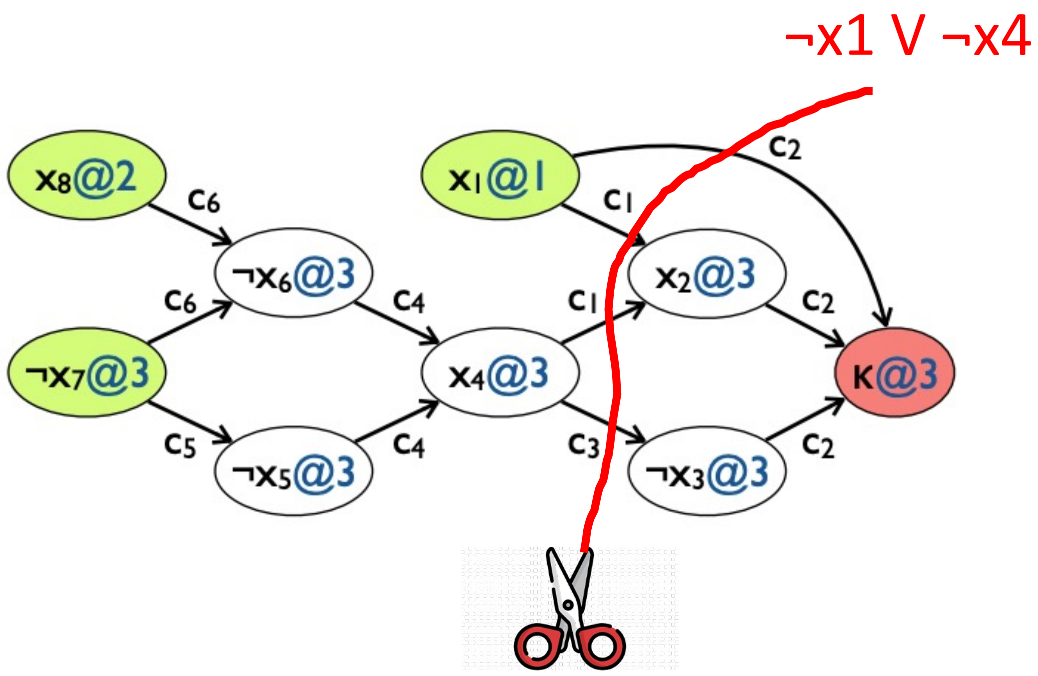


Example:

- c = c2, t = x2, v = x2, ante = c1
- c = ¬x1 V x3 V ¬x4, t = x3, v = x3, ante = c3
- c = ¬x1 V ¬x4, done!

ANALYZECONFLICT: Computing the conflict clause 2

- ANALYZECONFLICT()
  - d ← level(conflict)
  - if d = 0 then return -1
  - c ← antecedent(conflict)
  - repeat
    - t ← lastAssignedLitAtLevel(c, d)
    - v ← varOfLit(t)
    - ante ← antecedent(t)
    - c ← resolve(ante, c, v)
  - until oneLitAtLevel(c, d)
  - b ← assertingLevel(c)
  - return ⟨b, c⟩



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

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

Decision heuristics

- CDCL(F):
  - A ← {}
  - if BCP(F, A) = conflict then return false
  - level ← 0
  - while hasUnassignedVars(F)
    - level ← level + 1
    - A ← A ∪ { $\boxed{\text{DECIDE(F, A)}}$ }
    - 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

Dynamic Largest Individual Sum (DLIS)

- Choose the literal that satisfies the most unresolved clauses
  - Let cnt(l) = number of occurrences of literal l in unsatisfied clauses
  - Set the l with highest cnt(l)
- Simple and intuitive
- But expensive:
  - complexity of making a decision proportional to the number of clauses

Decision heuristics 2

- CDCL(F):
  - A ← {}
  - if BCP(F, A) = conflict then return false
  - level ← 0
  - while hasUnassignedVars(F)
    - level ← level + 1
    - A ← A ∪ { $\boxed{\text{DECIDE(F, A)}}$ }
    - 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

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)
  - For each literal l, maintain a VSIDS score
  - Initially: set to cnt(l)
  - Increment score by 1 each time it appears in an added (conflict) clause
  - Divide all scores by a constant (say 2) periodically (say every N backtracks)
- 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

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

- 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

Shuai Li

https://shuaili8.github.io

- SAT
    - CHF, 3-SAT
- Boolean Constraint Propagation (BCP)
    - unit clause
    - DPLL algorithm: backtracking + BCP
- Conflict Driven Clause Learning (CDCL)
    - Conflict clause learning, first UIP
    - Non-chronological backtracking
    - Decision heuristics
        - Dynamic Largest Individual Sum (DLIS)
        - Variable State Independent Decaying Sum (VSIDS)
    - Engineering matters

Questions?

References

- Tutorial SAT Solvers I: Introduction and applications by BOREALIS AI link
- Tutorial SAT Solvers II: Algorithms https://www.borealisai.com/research-blogs/tutorial-10-sat-solvers-ii-algorithms/
- Exponential Recency Weighted Average Branching Heuristic for SAT Solvers (AAAI 2016)
- Combining VSIDS and CHB Using Restarts in SAT (CP 2021)