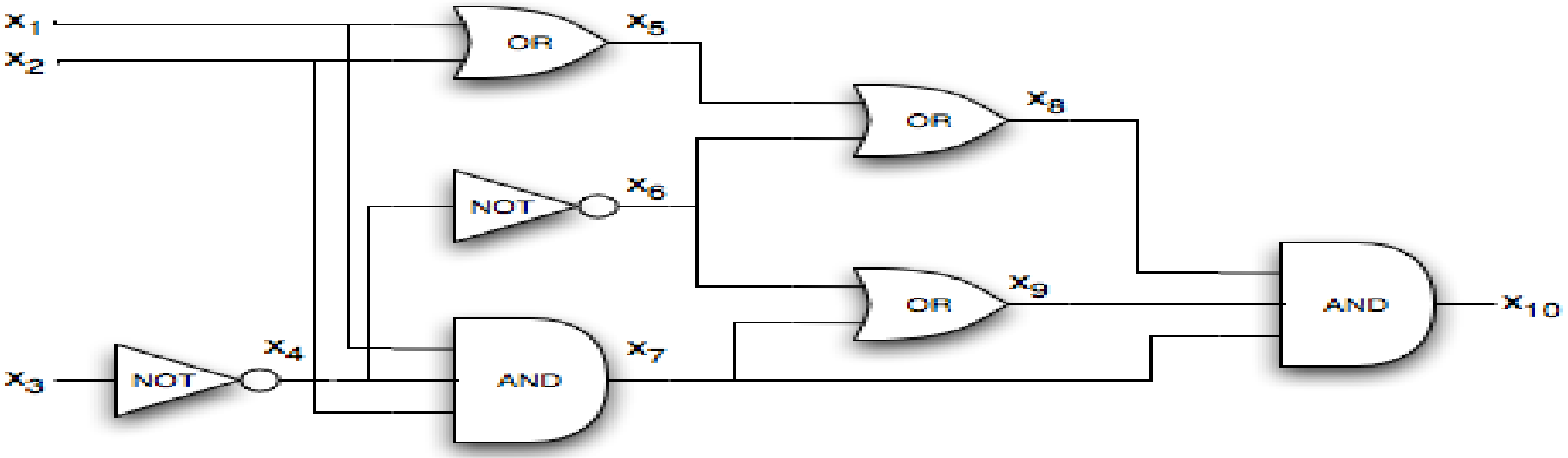
Karlsruhe Institute of Technology

Practical SAT Solving

Lecture 7

Carsten Sinz, Tomáš Balyo | June 4, 2019

INSTITUTE FOR THEORETICAL COMPUTER SCIENCE



Lecture Outline

- CDCL Algorithm
- Clause Learning

Reminder – Saturation Algorithm

Saturation Algorithm

- INPUT: CNF formula F
- OUTPUT: $\{SAT, UNSAT\}$

while (true) **do**
  $R = \text{resolveAll}(F)$
  if $(R \cap F \neq R)$ **then** $F = F \cup R$
  else break
if $(\bot \in F)$ **then return** *UNSAT* **else return** *SAT*

Properties of the saturation algorithm:

- it is sound and complete – always terminates and answers correctly
- has exponential time and space complexity (always for Pigeons)

Reminder – DPLL Algorithm

```
boolean DPLL(ClauseSet S)
{
  while ( S contains a unit clause {L} ) {
    delete from S clauses containing L;  // unit-subsumption
    delete ¬L from all clauses in S;     // unit-resolution
  }
  if ( ⊥ ∈ S ) return false;             // empty clause?
  while ( S contains a pure literal L )
    delete from S all clauses containing L;
  if ( S = ∅ ) return true;              // no clauses?
  choose a literal L occurring in S;     // case-splitting
  if ( DPLL(S ∪ {{L}} ) return true;     // first branch
  else if ( DPLL(S ∪ {{¬L}} ) return true;  // second branch
  else return false;
}
```

Reminder – “Modern” DPLL Algorithm with “Trail”

```
boolean mDPLL(ClauseSet S, PartialAssignment α)
{
  while ( (S, α) contains a unit clause {L} ) {
    add {L = 1} to α
  }
  if ( a literal is assigned both 0 and 1 in α ) return false;
  if ( all literals assigned ) return true;
  choose a literal L not assigned in α occurring in S;
  if ( mDPLL(S, α ∪ {L = 1} ) return true;
  else if ( mDPLL(S, α ∪ {L = 0} ) return true;
  else return false;
}
```

(S, α): clause set S as “seen” under partial assignment α

DPLL à la CDCL

```
boolean DPLL(ClauseSet S) {
  α = ∅, Trail = new Stack()
  while (not all variables assigned in α) {
    if (unitPropagation(S, α) = CONFLICT) {
      L = the last literal in Trail not tried both True and False
      if (no such L) return UNSATISFIABLE
      α = unassign all literals after L in Trail
      pop all literals after L in Trail
      α = (α \ {L}) ∪ {¬L}
    } else {
      L = pick an unassigned literal
      add {L = 1} to α
      Trail.push(L)
    }
  }
  return SATISFIABLE
}
```

CDCL Basic Ideas

- CDCL = Conflict Driven Clause Learning
- CDCL can be understood as a combination of DPLL and the Resolution Saturation Algorithm
- CDCL is like “DPLL à la CDCL” with the difference that Conflicts are handled differently
- Everytime a conflict is reached a new clause is “learned”, i.e., added to the clause set
- The learned clause is obtained by resolution from clauses already in the clause set (like in the Saturation algorithm)

CDCL sketch

```
boolean CDCL(ClauseSet S) {
  α = ∅, Trail = new Stack()
  while (not all variables assigned in α) {
    if (unitPropagation(S, α) = CONFLICT) {
      analyze the conflict which gives us:
      - a new learned clause C, S = S ∪ {C}
      - if C = ∅ return UNSATISFIABLE
      - a literal L in the Trail to which we backtrack
      update α and Trail according to L
    } else {
      L = pick an unassigned literal
      add {L = 1} to α
      Trail.push(L)
    }
  }
  return SATISFIABLE
}
```

Terminology

Trail

The partial assignment that represents the current path in the search tree

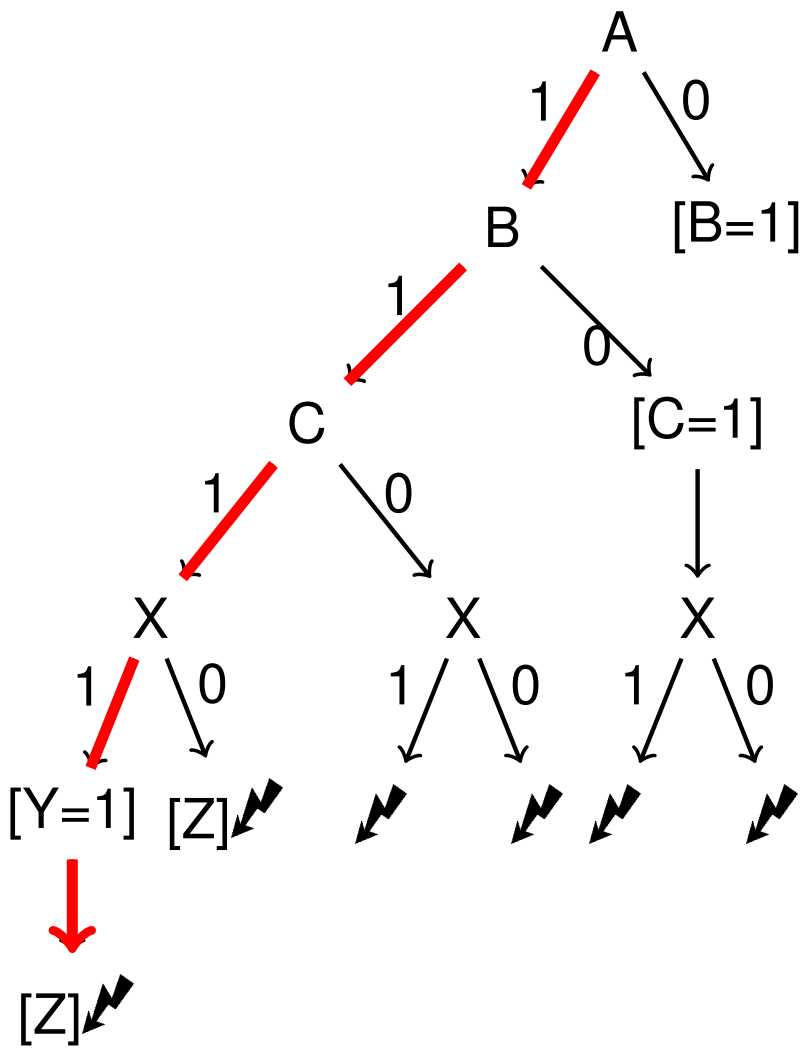
Decision Level

Starting with decision level 0 every branching step increases the decision level by 1. Note: Unit-propagation does not change the decision level.

Conflicting Clause

Clause with all literals assigned to 0 under the current trail (empty under unit resolution)

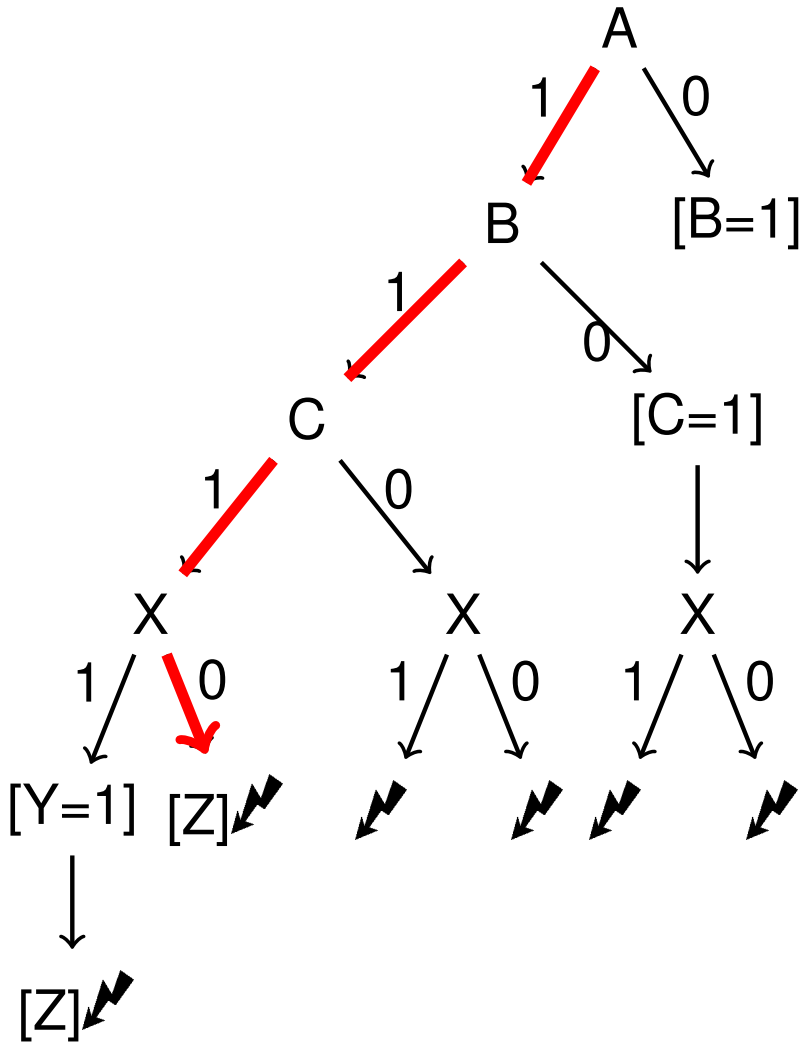
DPLL: Chronological Backtracking



$$F = \{\{A, B\},$$
$$\{B, C\},$$
$$\{\neg A, \neg X, Y\},$$
$$\{\neg A, X, Z\},$$
$$\{\neg A, \neg Y, Z\},$$
$$\{\neg A, X, \neg Z\},$$
$$\{\neg A, \neg Y, \neg Z\}\}$$

- Trail: A, B, C, X, Y, Z
- Conflicting Clause: $\{\neg A, \neg Y, \neg Z\}$

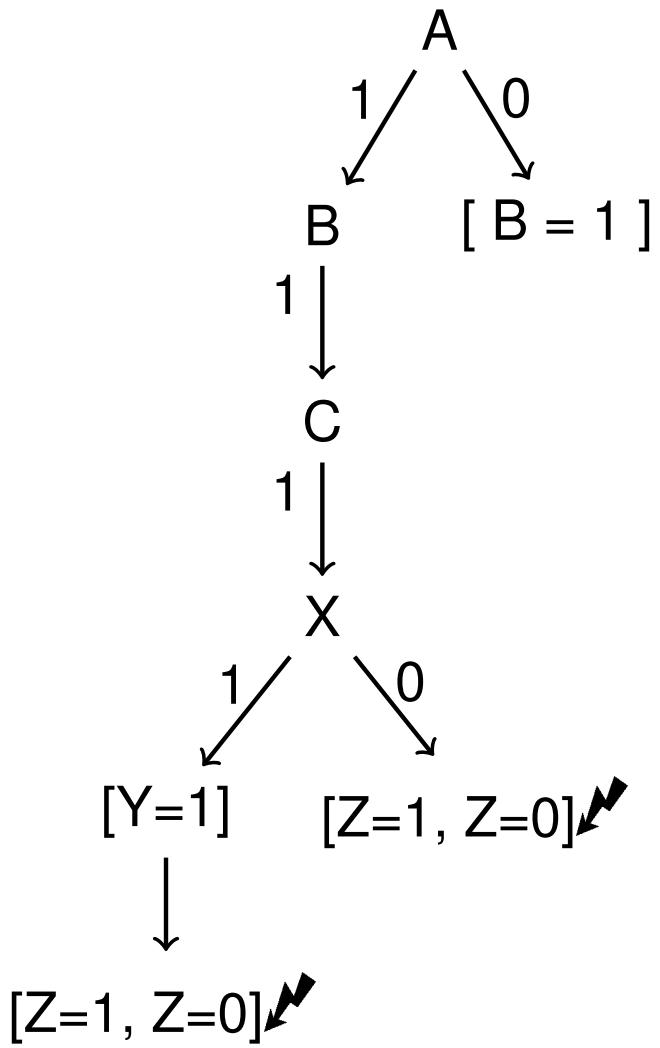
DPLL: Chronological Backtracking



$$F = \{\{A, B\},$$
$$\{B, C\},$$
$$\{\neg A, \neg X, Y\},$$
$$\{\neg A, X, Z\},$$
$$\{\neg A, \neg Y, Z\},$$
$$\{\neg A, X, \neg Z\},$$
$$\{\neg A, \neg Y, \neg Z\}\}$$

- Trail: $A, B, C, \neg X, Z$
- Conflicting Clause: $\{\neg A, X, \neg Z\}$

Non-Chronological Backtracking – Backjumping



- The first two conflicting clauses $\{\neg A, \neg Y, \neg Z\}$, $\{\neg A, X, \neg Z\}$ contain only a fraction of the assignments on the trail
- Assignments to B and C obviously play no role in the present conflicting state and we could immediately backtrack to flip the assignment to A
- How to find out which assignments on the trail are relevant for the actual conflict?

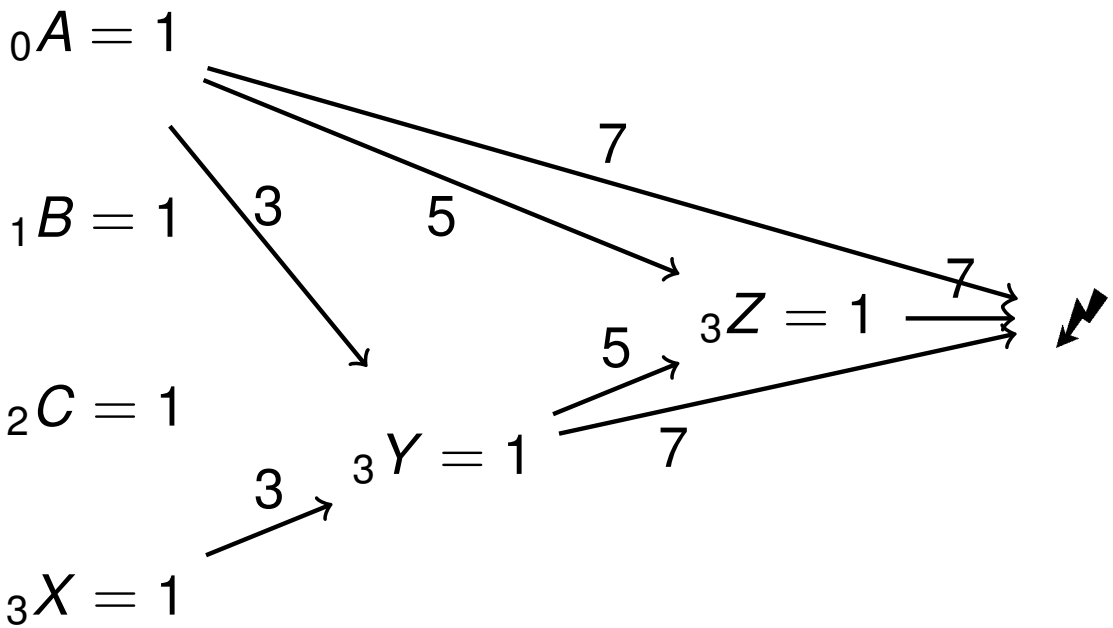
Implication Graph

Implication Graph

The Implication Graph $G_I = (V \cup \{\lightning\}, E)$ (for a given formula F and a trail T ending in a conflict) is a directed acyclic graph (DAG) where each vertex $v \in V$ is a tuple $(x = b, d)$ consisting of an assignment $x = b$ on the trail and its decision level d. There is an additional vertex $\lightning$ (with no assignment and decision level) indicating a conflicting assignment. For each clause $C = (l_1, \ldots, l_k, u)$ that became unit and caused a unit propagation of u there are edges $e \in E$ connecting the assignment of literal l_i to the assignment of unit u. Similarly, for a clause C, in which all literals are assigned false, there are edged from each literal in C to $\lightning$.

Example: Implication Graph

The graph shows the implication graph for the conflicting state under the trail A, B, C, X, Y, Z. The edge labels denote clauses, node labels indicate an assignment and its decision level.

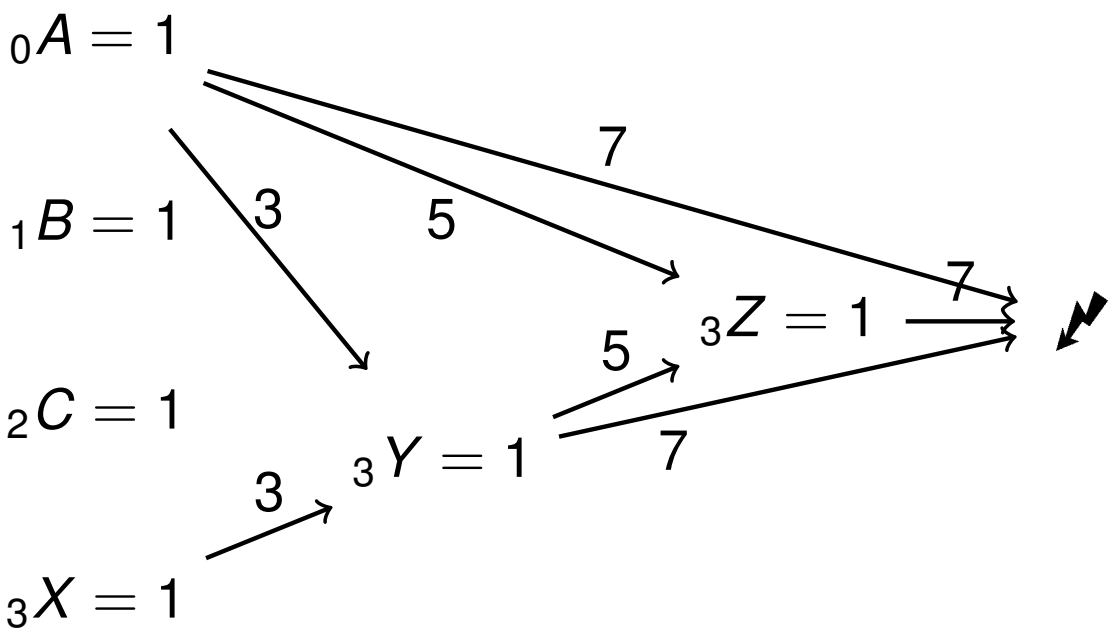


$$
\begin{aligned}
F = \{&\{A, B\}, && =: 1\\
&\{B, C\}, && =: 2\\
&\{\neg A, \neg X, Y\}, && =: 3\\
&\{\neg A, X, Z\}, && =: 4\\
&\{\neg A, \neg Y, Z\}, && =: 5\\
&\{\neg A, X, \neg Z\}, && =: 6\\
&\{\neg A, \neg Y, \neg Z\}\} && =: 7
\end{aligned}
$$

Implication Graph

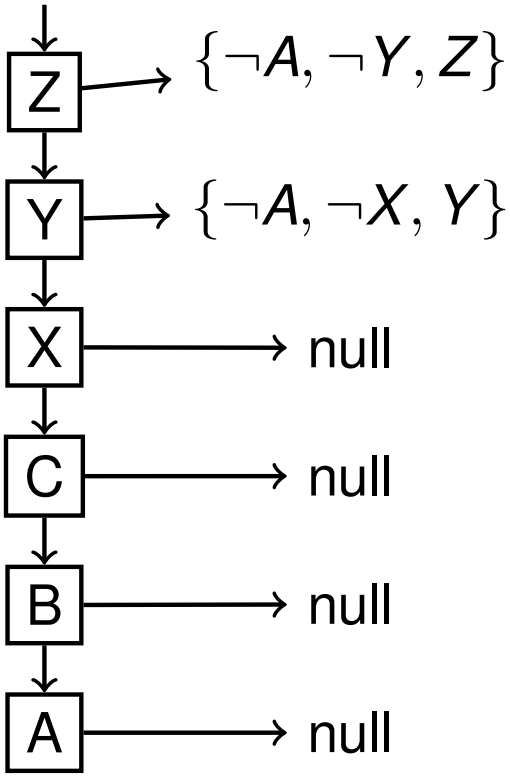
In the implication graph

- the sink is always the conflicting assignment
- the sources are the desicion literals that take part in the conflict
- we can use it to detect the *reasons* for a conflict



Implication Graph: Implementation

Trail with conflicting clause $\{\neg A, \neg Y, \neg Z\}$:



- for each assignment store a pointer to the reason clause, the one that has become unit
- decision literals store a null pointer
- with the clause pointers and the trail we can trace all the implications back to their sources

Review: Resolution

- General deduction rule for clauses: $(A \vee l), (B \vee \bar{l}) \implies (A \vee B)$
- Can be used to extend DPLL Search by adding resolved clauses to the input formula
- Special Case: Unit Resolution (central part of DPLL Search)
- Resolution Operator: $C \circ_l D$ denotes the resolvent of clauses C and D on literal l

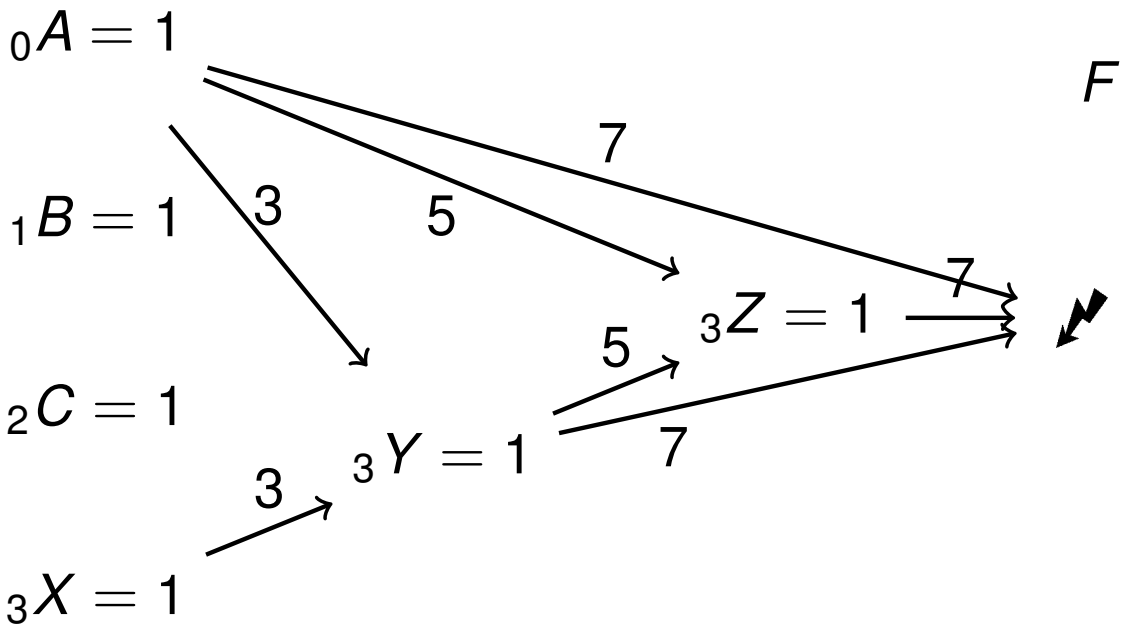
Clause Learing

Back-jumping does not fully avoid the occurrence of the same conflict, the same (partial) assignments may generate the same conflict in another part of the search tree.

- Idea: store no-good constraint like in CSP (No-Good-Learning)
- Great in SAT: no-good is clause!
- generate conflict clauses and add them to CNF
- the literals contributing to a conflict form a partial assignment and its negation is a clause (implied by the original CNF)
- adding this clause avoids the conflicting partial assignment

Conflict Analysis: Example

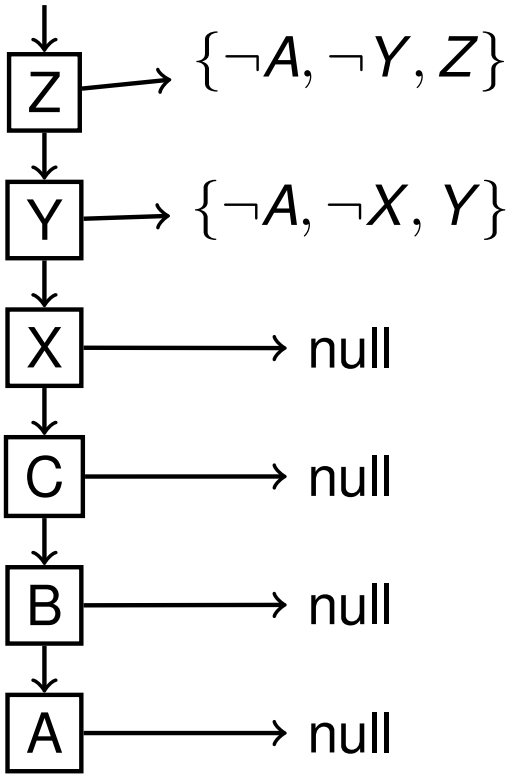
In the first conflict of the previous example we can learn the clause $\{\neg A, \neg X\}$ in order to prevent the solver to pick the same partial assignment again. This can also be expressed as the following sequence of resolution steps along the edges in the implication graph:
$C = (7 \circ_Z 5) \circ_Y 3$



$$
\begin{aligned}
F = \{&\{A, B\}, && =: 1\\
&\{B, C\}, && =: 2\\
&\{\neg A, \neg X, Y\}, && =: 3\\
&\{\neg A, X, Z\}, && =: 4\\
&\{\neg A, \neg Y, Z\}, && =: 5\\
&\{\neg A, X, \neg Z\}, && =: 6\\
&\{\neg A, \neg Y, \neg Z\}\} && =: 7
\end{aligned}
$$

Confict Clause

Trail with conflicting clause $\{\neg A, \neg Y, \neg Z\}$:



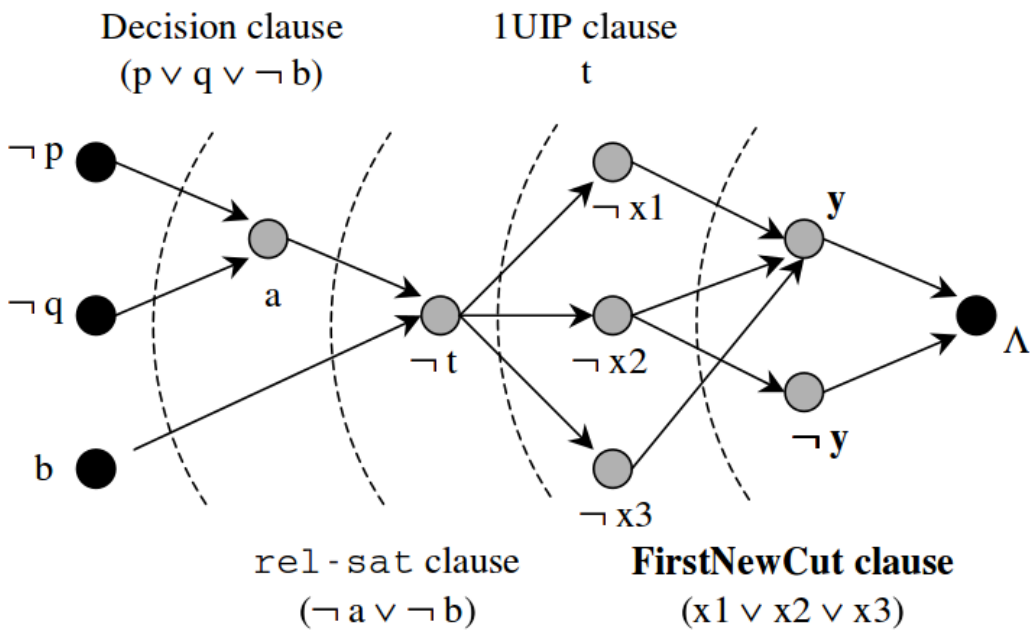
Given the trail the conflict clause (learned clause) can be generated by subsequent resolution, starting with the conflict clause $\{\neg A, \neg Y, \neg Z\}$ and the first reason clause on the trail $\{\neg A, \neg Y, Z\}$ we can resolve the intermediate clause $\{\neg A, \neg Y\}$. An additional resolution step results in the conflict clause $C = \{\neg A, \neg X\}$.

Properties of conflict clause C:

- $F \models C$
- $F \cup \neg C \vdash_{UP} \bot$
- $D \notin F, \forall D \subseteq C$

Unique Implication Points (UIP)

In general: several possibilities to learn a clause from an implication graph. A learned clause can be resolved for every cut in the the implication graph.



Graphic from Beame et al.: “Understanding the Power of Clause Learning”

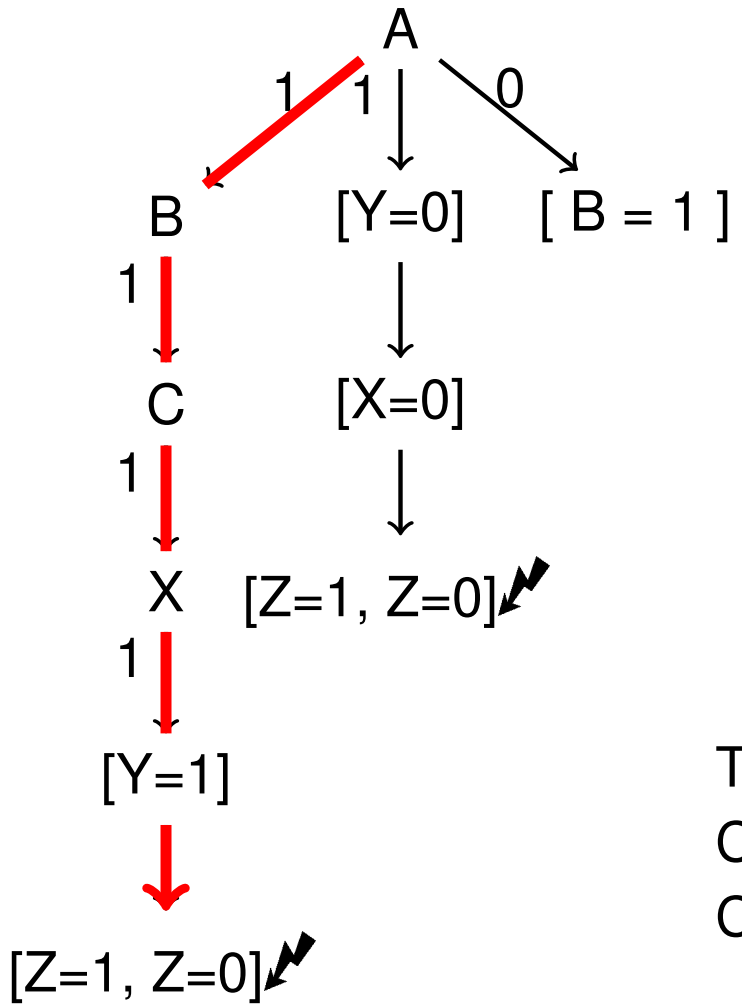
- UIP is a dominator in the implication graph
- A node v is a dominator (for ↯), if all paths from the last decision to ↯ contain v
- FirstUIP: “first” dominator (seen from conflict side)

Common Implementation (1UIP Learning Scheme)

- FirstUIP-clause: resolve conflicting clause and reason clauses until only a single literal of the current decision level remains
- Advantage: Stopping at a UIP always leads to an asserting clause. Algorithm becomes easier: backtrack until clause becomes asserting
- Undo all decisions until that level where the learned clause becomes asserting. The assertion level is the second highest level in a conflict clause.

Backtracking with Clause Learning

In our previous example the conflict clause (1UIP) with backtracking to assertion level changes the decision tree like this:



$$F = \{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\}\}$$

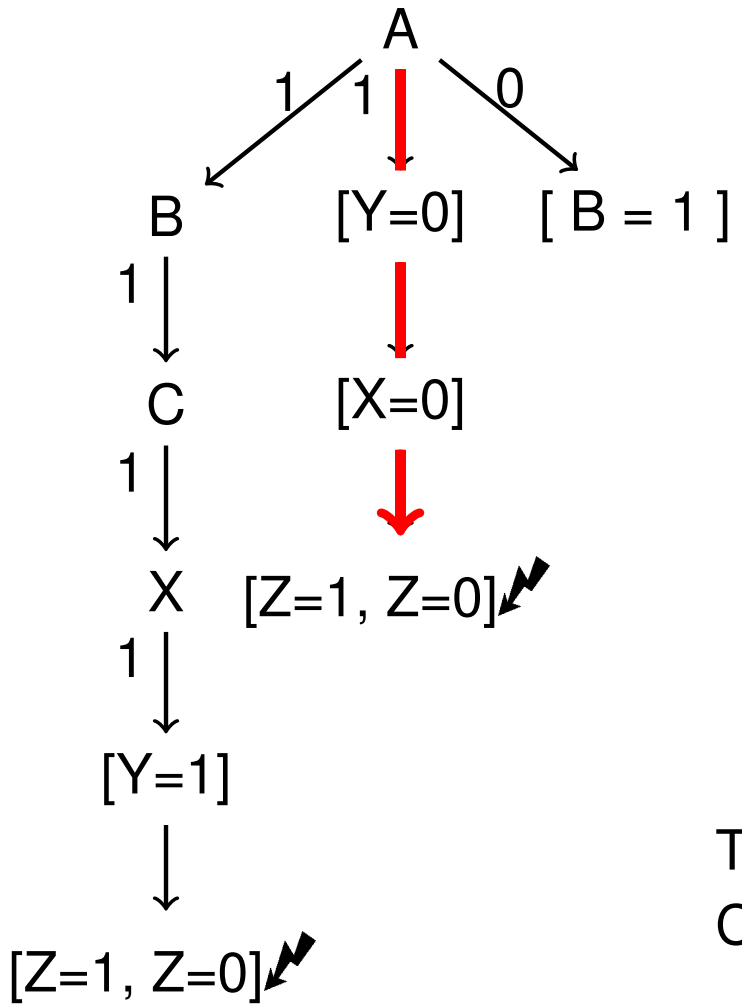
Trail: A, B, C, X, Y, Z

Conflicting Clause: $\{\neg A, \neg Y, \neg Z\}$

Conflict Clause (1UIP): $\{\neg A, \neg Y\}$

Backtracking with Clause Learning

In our previous example the conflict clause (1UIP) with backtracking to assertion level changes the decision tree like this:



$$F = \{\{A, B\}, \{B, C\}, \{\neg A, \neg X, Y\}, \{\neg A, X, Z\}, \{\neg A, \neg Y, Z\}, \{\neg A, X, \neg Z\}, \{\neg A, \neg Y, \neg Z\} \{\neg A, \neg Y\}\}$$

Trail: $A, \neg Y, \neg X, Z$

Conflicting Clause: $\{\neg A, X, \neg Z\}$

Clause Database Maintenance

Without restrictions the clause database grows exponentially. Besides the risk of running out of memory this also slows down unit propagation. Clause database is regularly cleaned up. Quality measures for clauses (e.g. clause size, clause activity, LBD value)

CDCL Algorithm

```
Data: Formula F
Result: SAT / UNSAT
1  dl ← 0
2  V ← ∅
3  if unit propagation (F, V) == CONFLICT then
4      return UNSAT
5  while not all variables assigned do
6      (x, l) ← pick branching literal
7      dl ← dl + 1
8      V ← ∪(x, l)
9      if unit propagation (F, V) == CONFLICT then
10         bl ← analyze conflict
11         if bl < 0 then
12             return UNSAT
13         else
14             backtrack to bl
15             dl ← bl
16 return SAT
```

Resolution Proof

With Clause Learning a complete resolution refutation can be derived with CDCL

- proof can serve as a certificate for validating the correctness of the SAT solver
- resolution refutations based on clause learning find key practical applications (e.g. model checking)
- can help to determine minimally unsatisfiable subsets in an unsatisfiable formula