Recording agreement

- The lectures will be recorded.
- You are not obligated neither to enable video or audio nor to share other personal data as real names or photos of yourself.
- The transmission of video and audio is disabled by default when joining the lecture. If you do not want to be part of the recording, simply do not activate the transmission.

- If you agree to be part of the recording, read the conditions provided in Moodle. When activating video or audio, you agree with the terms.

- Note that the breakout rooms are not recorded.

Some additions to the last lecture

- The “big” Boolean notation builds propositional expressions parametrically, e.g.

$$\bigwedge_{i=1}^{5} x_i \qquad \textit{is defined as} \qquad x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \ .$$

- Tranformation to NNF can be done with an effort (time and space) that is linear in the size of the formula.
  Idea: Number of transformation steps $\leq$ number of operands in the formula.

Satisfiability Checking

05 SAT solving

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 21/22

Given:

- Propositional logic formula φ in CNF.

Question:

- Is φ satisfiable?

  (Is there a model for φ?)

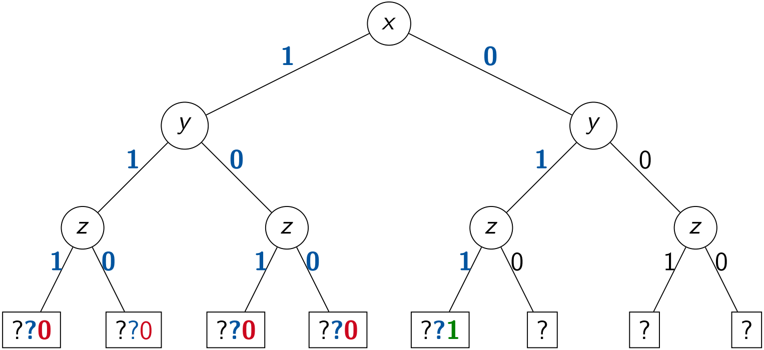
Enumeration algorithm

```
bool Enumeration(CNF_Formula φ){
    trail.clear(); //stack of entries (x, v, b) assigning value v to proposition x
                   //and a flag b stating whether ¬v has already been processed for x
    while (true) {
        if (!decide()) {
            if all clauses of φ are satisfied by the assignment in trail then return SAT;
            else if (!backtrack()) then return UNSAT
        }
    }
}
bool decide() {
    if (all variables are assigned) then return false;
    choose unassigned variable x not yet in trail;
    choose value v ∈ {0, 1};
    trail.push(x, v, false);
    return true
}
bool backtrack() {
    while (true){
        if (trail.empty()) then return false;
        (x, v, b):=trail.pop()
        if (!b) then { trail.push((x, ¬v, true)); return true }
    }
}
```

Static decision heuristics example

$$\underbrace{(\neg x \vee y \vee z)}_{c_1} \wedge \underbrace{(y \vee \neg z)}_{c_2} \wedge \underbrace{(\neg x \vee \neg y)}_{c_3}$$

Static variable order $x < y < z$, sign: try positive first

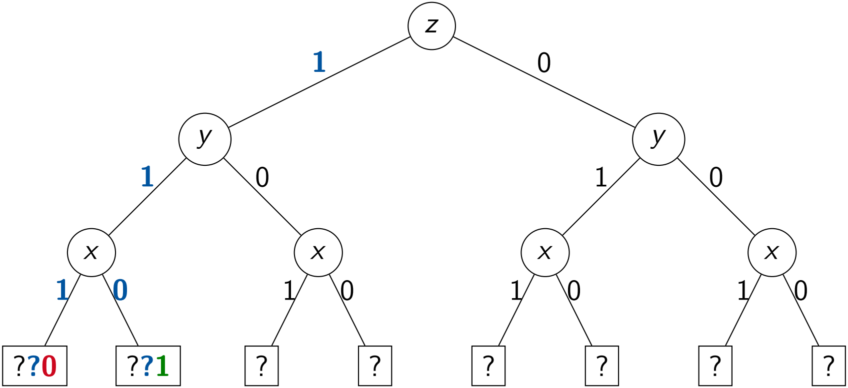


For unsatisfiable problems, all assignments need to be checked.
For satisfiable problems, variable and sign ordering might strongly influence the running time.

Static decision heuristics example

$$\underbrace{(\neg x \vee y \vee z)}_{c_1} \wedge \underbrace{(y \vee \neg z)}_{c_2} \wedge \underbrace{(\neg x \vee \neg y)}_{c_3}$$

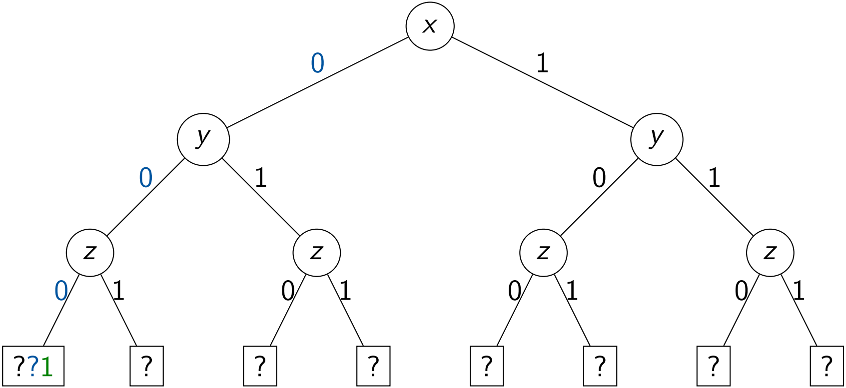
Static variable order $z < y < x$, sign: try positive first



Static decision heuristics example

$$\underbrace{(\neg x \vee y \vee z)}_{c_1} \wedge \underbrace{(y \vee \neg z)}_{c_2} \wedge \underbrace{(\neg x \vee \neg y)}_{c_3}$$

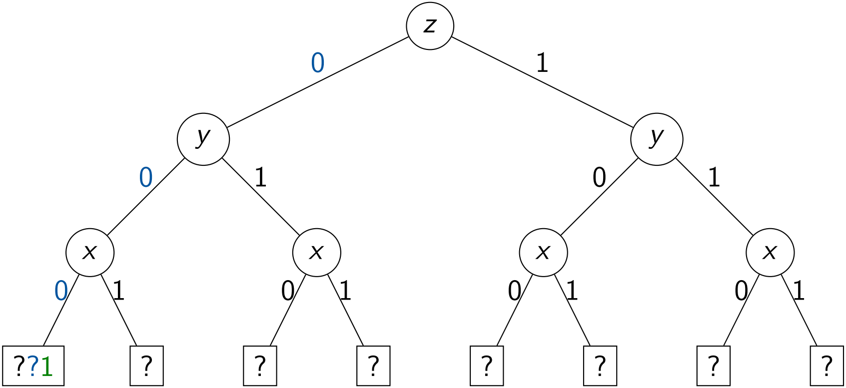
Static variable order $x < y < z$, sign: try negative first



Static decision heuristics example

$$\underbrace{(\neg x \vee y \vee z)}_{c_1} \wedge \underbrace{(y \vee \neg z)}_{c_2} \wedge \underbrace{(\neg x \vee \neg y)}_{c_3}$$

Static variable order $z < y < x$, sign: try negative first



Dynamic decision heuristics example: DLIS

Dynamic Largest Individual Sum (DLIS): Choose an assignment that increases the most the number of satisfied clauses.

- For each literal l, let C_l be the number of unresolved clauses in which l appears.
- Let l be a literal for which C_l is maximal ($C_{l'} \leq C_l$ for all literals l').
- If l is a variable x then assign *true* to x.
- Otherwise if l is a negated variable $\neg x$ then assign *false* to x.

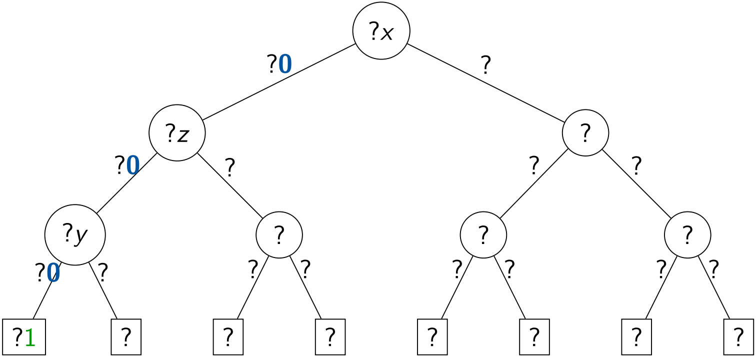
- Requires $\mathcal{O}(\#\mathit{literals})$ queries for each decision.

Dynamic decision heuristics example: DLIS

$$\underbrace{(\neg x \lor y \lor z)}_{c_1} \land \underbrace{(y \lor \neg z)}_{c_2} \land \underbrace{(\neg x \lor \neg y)}_{c_3}$$

$$\begin{array}{lll} C_x = 0 & C_y = 210 & C_z = \\ C_{\neg x} = 20 & C_{\neg y} = 10 & C_{\neg z} = \end{array}$$

Fallback literal order (in case of equal values): $\neg x < x < \neg z < z < \neg y < y$



Static decision heuristics example: Jeroslow-Wang method

Jeroslow-Wang method

Compute for every literal l the following static value:

$$J(l) : \sum_{\textit{clause } c \textit{ in the CNF containing } l} 2^{-|c|}$$

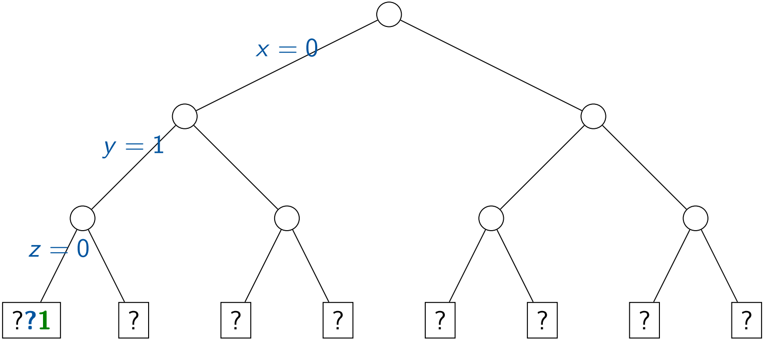
- Choose a literal l that maximizes $J(l)$.
- This gives an exponentially higher weight to literals in shorter clauses.

Jeroslow-Wang method: Example

$$\underbrace{(\neg x \vee y \vee z)}_{c_1} \wedge \underbrace{(y \vee \neg z)}_{c_2} \wedge \underbrace{(\neg x \vee \neg y)}_{c_3}$$

Static Jeroslow-Wang method

$J(x) = 0,\ J(\neg x) = \frac{1}{8}+\frac{1}{4},\ J(y) = \frac{1}{8}+\frac{1}{4},\ J(\neg y) = \frac{1}{4},\ J(z) = \frac{1}{8},\ J(\neg z) = \frac{1}{4}$



Decision heuristics

- We will see other (more advanced) decision heuristics later.

1 Enumeration (decision)

2 Boolean constraint propagation (BCP)

3 Conflict resolution and backtracking

4 Enumeration (decision) revisited

Status of a clause

- Given a (partial) assignment, a clause can be

satisfied:	at least one literal is satisfied
unsatisfied:	all literals are assigned but none are statisfied
unit:	all but one literals are assigned but none are satisfied
unresolved:	all other cases

- Example: $c = (x_1 \vee x_2 \vee x_3)$

x_1	x_2	x_3	c
1	0		satisfied
0	0	0	unsatisfied
0	0		unit
	0		unresolved

BCP: Unit clauses are used to imply consequences of decisions.

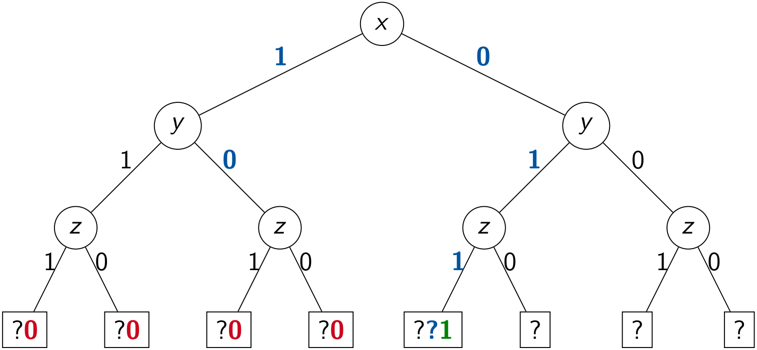
Some notations:

- Decision Level (DL) is a counter for decisions
- Antecedent(l): unit clause implying the value of the literal l (nil if decision)

Boolean constraint propagation: Example

$$\underbrace{(\neg x \vee y \vee \mathbf{z})}_{c_1} \wedge \underbrace{(y \vee \neg \mathbf{z})}_{c_2} \wedge \underbrace{(\neg x \vee \neg \mathbf{y})}_{c_3}$$

Static variable order $x < y < z$, sign: try positive first



The DPLL algorithm: Enumeration + propagation

```
bool DPLL(CNF_Formula φ){
  trail.clear(); //trail is a global stack of assignments
  if (!BCP()) then return UNSAT;
  while (true) {
    if (!decide()) then return SAT;
    while (!BCP())
      if (!backtrack()) then return UNSAT;
  }
}
bool BCP() { //more advanced implementation: return false as soon as an unsatisfied clause is detected
  while (there is a unit clause implying that a variable x must be set to a value v)
    trail.push(x, v, true);
  if (there is an unsatisfied clause) then return false;
  return true;
}
```

The DPLL algorithm: Enumeration + propagation

```
bool decide() {
  if (all variables are assigned) then return false;
  choose unassigned variable x;
  choose value v ∈ {0,1};
  trail.push(x, v, false);
  return true
}
bool backtrack() {
  while (true){
    if (trail.empty()) then return false;
    (x,v,b)=trail.pop()
    if (!b) then {
      trail.push(x, ¬v, true);
      return true
    }
  }
}
```

Watched literals

- For BCP, it would be a large effort to check for each propagation the value of each literal in each clause.
- Idea: in each clause watch two different literals such that either one of them is *true* or both are unassigned
  $\rightarrow$ clause is neither unit nor unsatisfied.

  If a literal l gets *true*, we check each clause in which $\neg l$ is a watched literal (which is now *false*).

  - If the other watched literal is *true*, the clause is satisfied.
  - Else, if we find a new literal to watch, we are done.
  - Else, if the other watched literal is unassigned, the clause is unit.
  - Else, if the other watched literal is *false*, the clause is conflicting.

Bonus exercise 7

Assume the following clause:

$$(a \vee b \vee \neg c \vee d)$$

Assume that a is *false*, b is *true*, whereas c and d are not assigned.

Which literal pairs are suited to be watched after this assignment?

- $a,\ b$
- $a,\ \neg c$
- $a,\ d$
- $b,\ \neg c$
- $b,\ d$
- $\neg c,\ d$

Implication graph

We represent (partial) variable assignments in the form of an implication graph.

Definition

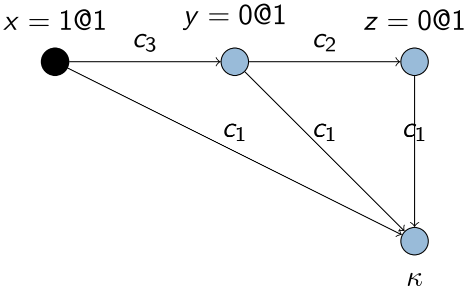
An implication graph is a labeled directed acyclic graph $G = (V, E, L)$, where

- V is a set of nodes, one for each currently assigned variable and an additional conflict node κ if there is a currently conflicting clause c_{confl}.
- L is a labeling function assigning a lable to each node. The conflict node (if any) is labelled by $L(\kappa) = \kappa$. Each other node n, representing that x is assigned $v \in \{0, 1\}$ at decision level d, is labeled with $L(n) = (x = v@d)$; we define $literal(n) = x$ if $v = 1$ and $literal(n) = \neg x$ if $v = 0$.
- $E = \{(n_i, n_j) | n_i, n_j \in V, n_i \neq n_j, \neg literal(n_i) \in \text{Antecedent}(literal(n_j))\} \cup \{(n, \kappa) | n, \kappa \in V, \ \neg literal(n) \in c_{confl}\}$ is the set of directed edges where each edge (n_i, n_j) is labeled with $\text{Antecedent}(literal(n_j))$ if $n_j \neq \kappa$ and with c_{confl} otherwise.

Implication graph: Example

$$\underbrace{(\neg x \lor y \lor z)}_{c_1} \land \underbrace{(y \lor \neg z)}_{c_2} \land \underbrace{(\neg x \lor \neg y)}_{c_3}$$

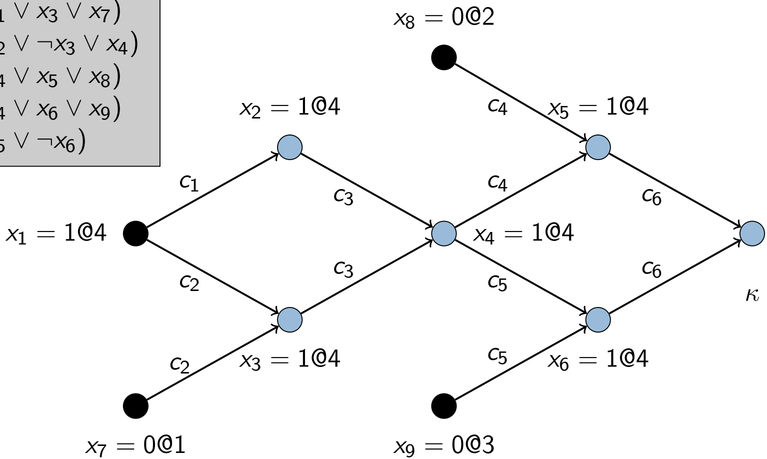
Static variable order $x < y < z$, sign: try positive first



Implication graph: Example

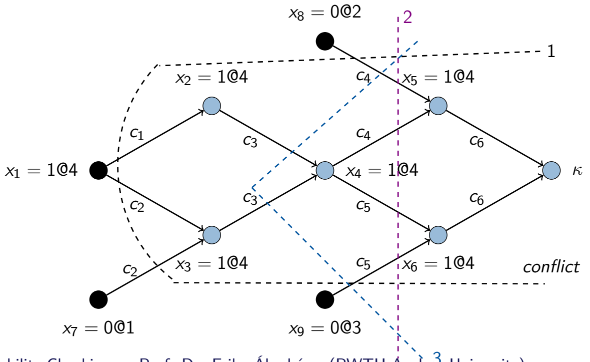
Decisions: $\{x_7 = 0@1,\ x_8 = 0@2,\ x_9 = 0@3,\ x_1 = 1@4\}$

$c_1 = (\neg x_1 \vee x_2)$
$c_2 = (\neg x_1 \vee x_3 \vee x_7)$
$c_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$
$c_4 = (\neg x_4 \vee x_5 \vee x_8)$
$c_5 = (\neg x_4 \vee x_6 \vee x_9)$
$c_6 = (\neg x_5 \vee \neg x_6)$



Conflict resolution

- Assume that the current (partial) assignment doesn't satisfy our formula.
- Let L be a set of literals labeling nodes that form a cut in the implication graph, seperating a conflict node from the roots.
- $\bigvee_{l \in L} \neg l$ is called a conflict clause: it is false under the current assignment but its satisfaction is necessary for the satisfaction of the formula.



1.$(x_8 \vee \neg x_1 \vee x_7 \vee x_9)$

2.$(x_8 \vee \neg x_4 \vee x_9)$

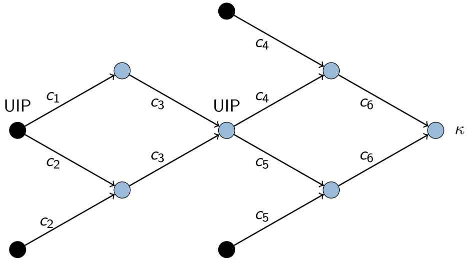
3.$(x_8 \vee \neg x_2 \vee \neg x_3 \vee x_9)$

$\vdots$

$\vdots$

Conflict resolution

- Which conflict clauses should we consider?
- An asserting clause is a conflict clause with a single literal from the current decision level.
  Backtracking (to the right level) makes it a unit clause.
- Modern solvers consider only asserting clauses.
- Assume an implication graph G with a conflict node κ. A unique implication point (UIP) for κ in G is a node $n \neq \kappa$ in G such that all paths from the last decision to κ go through n.
- The first UIP is the UIP closest to the conflict node.



Conflict-driven backtracking

- Usually, the asserting conflict clause is learnt by adding it to the clause set. However, this is not necessary for completeness.
- Backtrack to the second highest decision level *dl* in the asserting conflict clause (but do not erase it).
- This way the literal with the currently highest decision level will be implied at decision level *dl*.
- Propagate all new assignments.

Q: What happens if the asserting conflict clause has a single literal?
For example, from $(x \vee \neg y) \wedge (x \vee y)$ and decision $x = 0$, we get (x).
A: Backtrack to DL0.

Q: What happens if the conflict appears at decision level 0?
A: The formula is unsatisfiable.

The CDCL algorithm

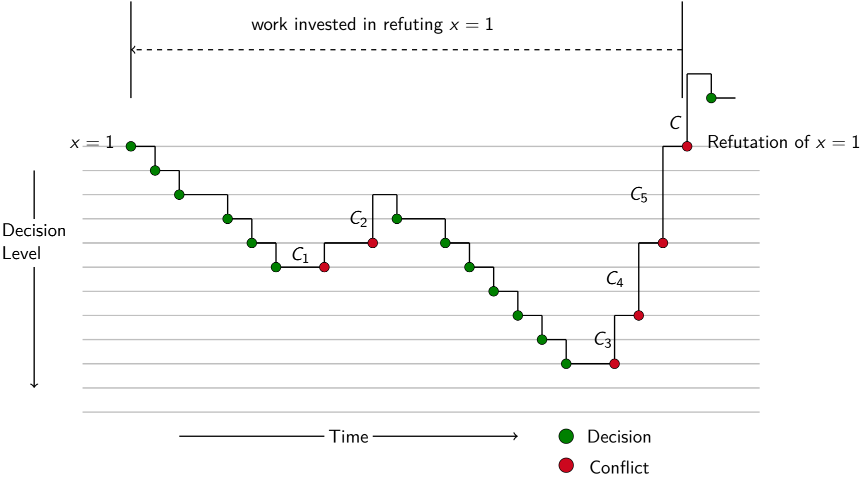
Choose the next variable and value.
Return false if all variables are assigned.

```
if (!BCP()) return UNSAT;
while (true)
{
    if (!decide()) return SAT;
    while (!BCP())
        if (!resolve_conflict()) return UNSAT;
}
```

Boolean constraint propagation.
Return false if reached a conflict.

Conflict resolution and backtracking. Return false if impossible.

Progress of a DPLL+CDCL-based SAT solver



Conflict clauses and (binary) resolution

- The (binary) resolution is a sound (and complete) inference rule:

$$\frac{(\beta \vee a_1 \vee \ldots \vee a_n) \qquad (\neg\beta \vee b_1 \vee \ldots \vee b_m)}{(a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)}\text{(Binary Resolution)}$$

- Example:

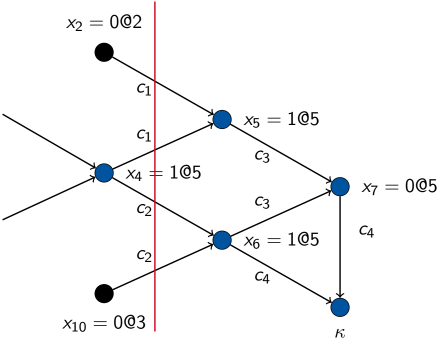
$$\frac{(x_1 \vee x_2) \qquad (\neg x_1 \vee x_3 \vee x_4)}{(x_2 \vee x_3 \vee x_4)}$$

What is the relation of binary resolution and conflict clauses?

Conflict clauses and (binary) resolution

- Consider the following example:

$c_1 = (\neg x_4 \vee x_2 \vee x_5)$
$c_2 = (\neg x_4 \vee x_{10} \vee x_6)$
$c_3 = (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
$c_4 = (\neg x_6 \vee x_7)$
$\vdots \qquad \vdots$

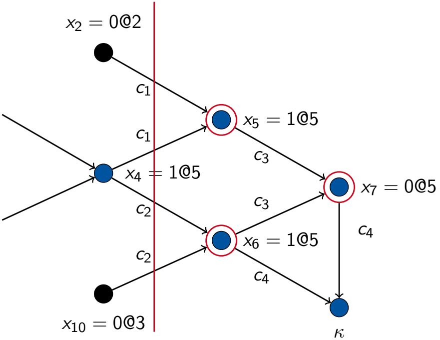


- Asserting conflict clause: $c_5 : (x_2 \vee \neg x_4 \vee x_{10})$

Conflict clauses and (binary) resolution

- Assigment order: x_4, x_5, x_6, x_7 Conflict clause: $c_5 : (x_2 \vee \neg x_4 \vee x_{10})$

$c_1 = (\neg x_4 \vee x_2 \vee x_5)$
$c_2 = (\neg x_4 \vee x_{10} \vee x_6)$
$c_3 = (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
$c_4 = (\neg x_6 \vee x_7)$
$\vdots \qquad \vdots$



- Starting with the conflicting clause, apply resolution with the antecedent of the last assigned literal, until we get an asserting clause:
  - $\text{T1} = \text{Res}(c_4, c_3, x_7) = (\neg x_5 \vee \neg x_6)$
  - $\text{T2} = \text{Res}(\text{T1}, c_2, x_6) = (\neg x_4 \vee \neg x_5 \vee x_{10})$
  - $\text{T3} = \text{Res}(\text{T2}, c_1, x_5) = (x_2 \vee \neg x_4 \vee x_{10})$

Finding the asserting conflict clause

```
bool analyze_conflict() {
    if (current_decision_level = 0) then return false;
    cl := current_conflicting_clause;
    while (cl is not asserting) do {
        lit := last_assigned_literal(cl);
        var := variable_of_literal(lit);
        ante := antecedent(var);
        cl := resolve(cl, ante, var);
    }
    add_clause_to_database(cl);
    return true;
}
```

Applied to our example:

name	*cl*	*lit*	*var*	*ante*
c_4	$(\neg x_6 \vee x_7)$	x_7	x_7	c_3
	$(\neg x_5 \vee \neg x_6)$	$\neg x_6$	x_6	c_2
	$(\neg x_4 \vee x_{10} \vee \neg x_5)$	$\neg x_5$	x_5	c_1
c_5	$(\neg x_4 \vee x_2 \vee x_{10})$			

Unsatisfiable core

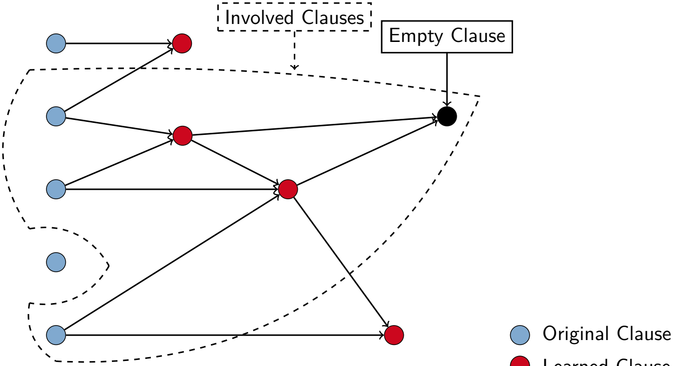
Definition

An unsatisfiable core of an unsatisfiable CNF formula is an unsatisfiable subset of the original set of clauses.

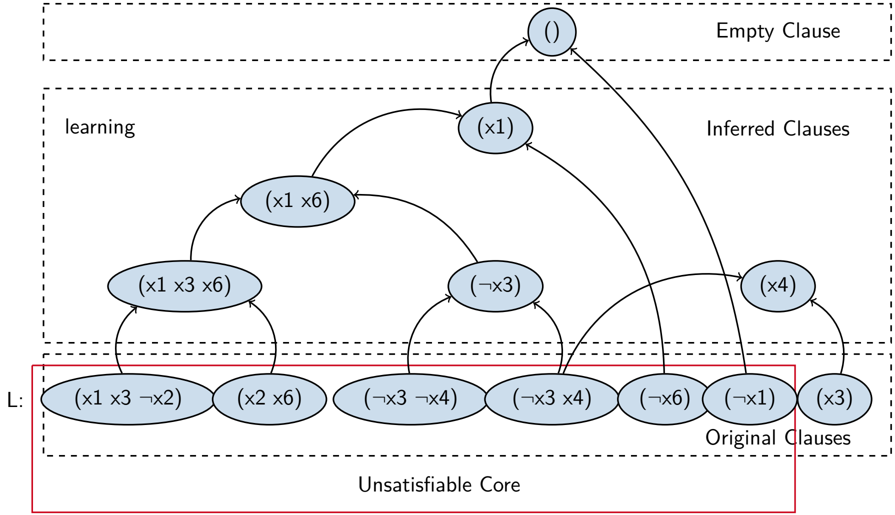
- The set of all original clauses is an unsatisfiable core.
- The set of those original clauses that were used for resolution in conflict analysis during SAT-solving (inclusively the last conflict at decision level 0) gives us an unsatisfiable core which is in general much smaller.
- However, this unsatifiable core is still not always minimal (i.e., we can remove clauses from it still having an unsatisfiable core).

The resolution graph

A resolution graph gives us more information to get a minimal unsatisfiable core.



Resolution graph: Example



Termination

Theorem

It is never the case that the solver enters decision level dl again with the same partial assignment.

Proof.

Define a partial order on partial assignments: $\alpha < \beta$ iff either α is an extension of β or α has more assignments at the smallest decision level at that α and β do not agree.
BCP decreases the order, conflict-driven backtracking also. Since the order always decreases during the search, the theorem holds. $\square$

Bonus exercise 8

Assume the following CNF:

$$(\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg b \vee \neg c)$$

Assigning $a := 1$ leads to a conflict.
What is the result of conflict resolution?

- (a)
- $(\neg a)$
- $(\neg b)$
- $(\neg a \vee b)$
- $(a \vee c)$

Decision heuristics: VSIDS

- VSIDS (variable state independent decaying sum)
- Gives priority to variables involved in recent conflicts.
- "Involved" can have different definitions. We take those variables that occur in clauses used for conflict resolution.

1. Each variable in each polarity has a counter initialized to 0.
2. We define an increment value (e.g., 1).
3. When a conflict occurs, we increase the counter of each variable, that occurs in at least one clause used for conflict resolution, by the increment value.
   Afterwards we increase the increment value (e.g., by 1).
4. For decisions, the unassigned variable with the highest counter is chosen.
5. Periodically, all the counters and the increment value are divided by a constant.

Decision heuristics: VSIDS

- Chaff holds a list of unassigned variables sorted by the counter value.
- Updates are needed only when adding new conflict causes.
- Thus - decision is made in constant time.

Decision heuristics: VSIDS

VSIDS is a 'quasi-static' strategy:

- static because it doesn't depend on current assignment
- dynamic because it gradually changes. Variables that appear in recent conflicts have higher priority.

  This strategy is a conflict-driven decision strategy.

"...employing this strategy dramatically (i.e., an order of magnitude) improved performance..."

Learning target

- Enumeration:
  What kind of (static and dynamic) variable ordering heuristics can be used?
- DPLL SAT solving:
  How does propagation work with enumeration?
  What are watched literals?
- DPLL+CDCL SAT solving:
  How can resolution be used for conflict resolution?
  How to formalize and execute the resulting DPLL+CDCL SAT solving algorithm?
  How to construct unsatisfiable cores?