Satisfiability Checking

SAT Solving

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 16/17

Given:

- Propositional logic formula φ in CNF.

Question:

- Is φ satisfiable?

  (Is there a model for φ?)

SAT-solving: Components

- Decision (enumeration)
- Boolean constraint propagation (BCP)
- Conflict resolution and backtracking

SAT-solving: Components

- Decision (enumeration)
- Boolean constraint propagation (BCP)
- Conflict resolution and backtracking

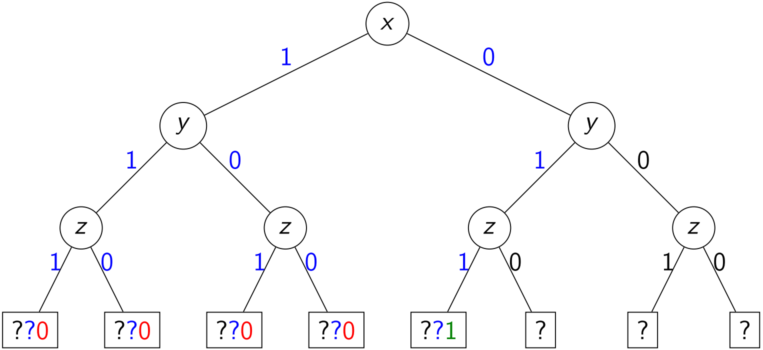
Enumeration algorithm

```
bool Enumeration(CNF_Formula φ){
  trail.clear(); //trail is a stack
  while (true) {
    if there are unassigned variables then {
      choose unassigned variable x
      choose value v ∈ {0,1}
      trail.push(x, v, false)
    } else {
      if all clauses of φ are satisfied then return SAT
      while (true){
        if (!trail.empty()) then (x,v,b)=trail.pop()
        else return UNSAT;
        if (!b) {
          trail.push(x, ¬v, true)
          break
        }
      }
    }
  }
}
```

Example CNF: Decision heuristics

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

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



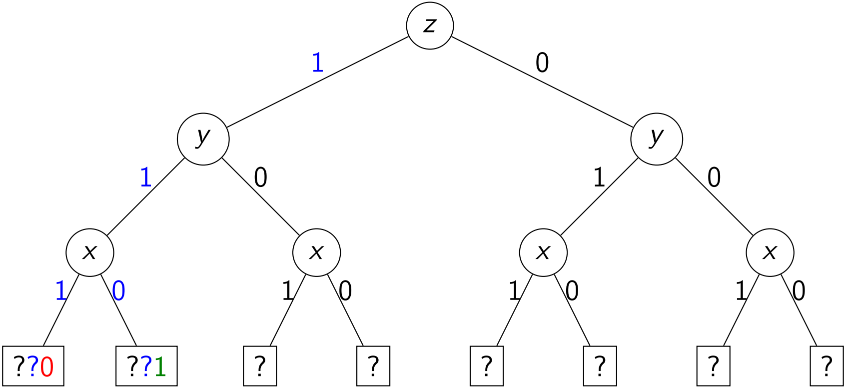
For unsatisfiable problems, all assignments need to be checked.

For satisfiable problems, variable and sign ordering might strongly influence the running time.

Example CNF: Decision heuristics

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

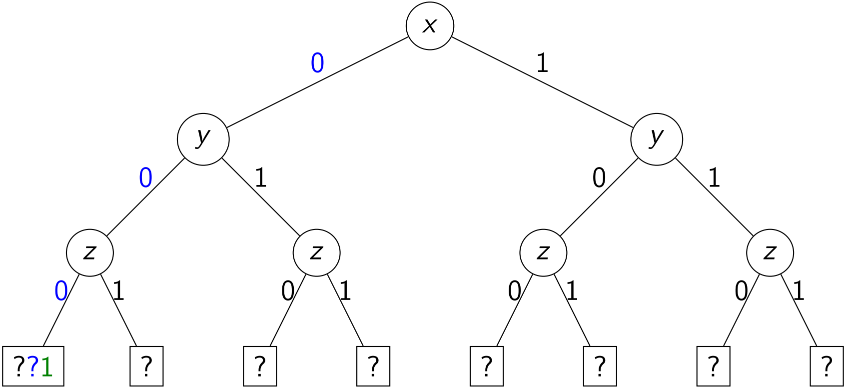
Static variable order $z < y < x$, sign: try positive first



Example CNF: Decision heuristics

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

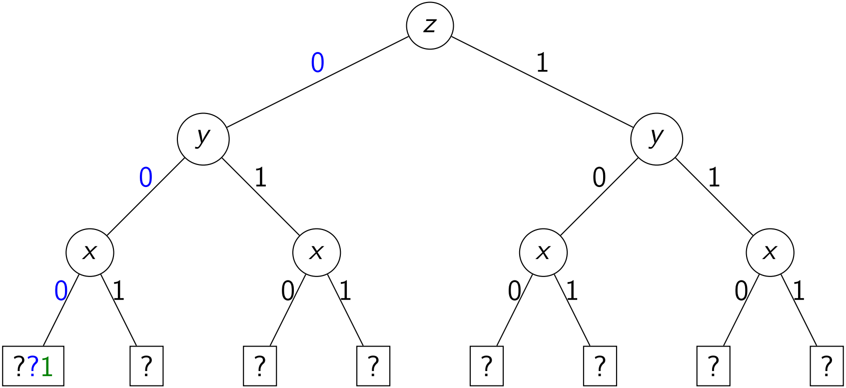
Static variable order $x < y < z$, sign: try negative first



Example CNF: Decision heuristics

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

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



Decision heuristics

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

- For each variable x, let C_x be the number of unresolved clauses in which x appears positively.
- For each variable x, let $C_{\neg x}$ be the number unresolved clauses in which x appears negatively.
- Let x be a variable for which C_x is maximal ($C_x \geq C_z$ for all variables z).
- Let y be a variable for which $C_{\neg y}$ is maximal ($C_{\neg y} \geq C_{\neg z}$ for all variables z).
- If $C_x > C_{\neg y}$ choose x and assign it TRUE.
- Otherwise choose y and assign it FALSE.

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

Example CNF: Decision heuristics DLIS

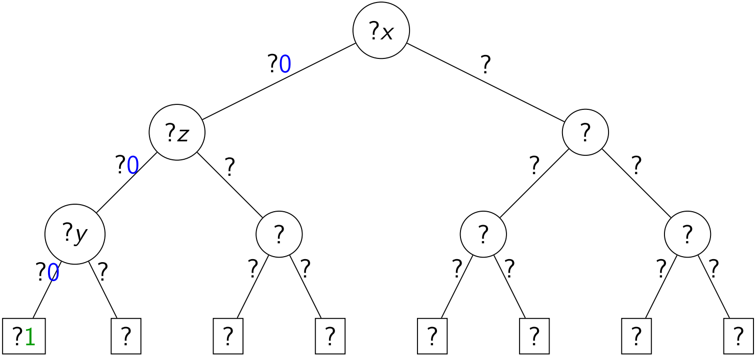
$$\underbrace{(\neg x \vee y \vee z)}_{c_1} \wedge \underbrace{(y \vee \neg z)}_{c_2} \wedge \underbrace{(\neg x \vee \neg y)}_{c_3}$$

$C_x = 0$ $\quad C_y = 210$ $\quad C_z =$

$C_{\neg x} = 20$ $\quad C_{\neg y} = 10$ $\quad C_{\neg z} =$

Dynamic Largest Individual Sum (DLIS) literal order

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



Decision heuristics

Jersolow-Wang method

Compute for every literal l the following static value:

$$J(l) : \sum_{l \in c, c \in \phi} 2^{-|c|}$$

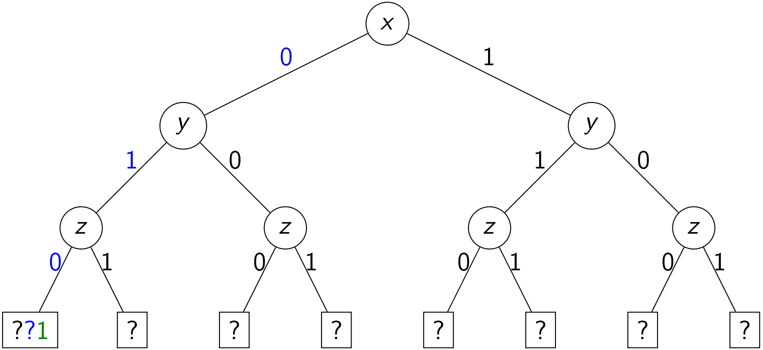
- Choose a literal l that maximizes $J(l)$.
- This gives an exponentially higher weight to literals in shorter clauses

Example CNF: Jersolow-Wang method

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

Static Jersolow-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.

SAT-solving: Components

- Decision (enumeration)
- Boolean constraint propagation (BCP)
- Conflict resolution and backtracking

Status of 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.

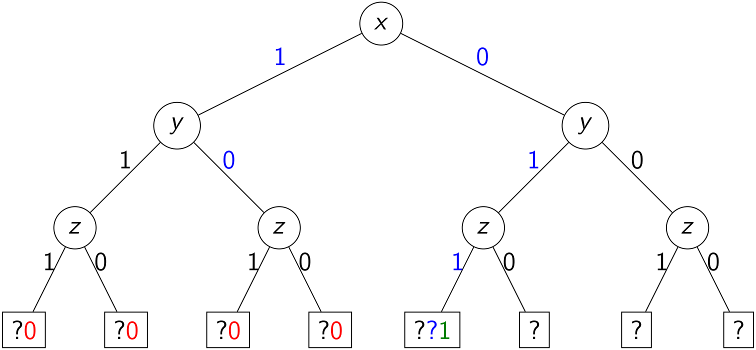
Some notations:

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

Example CNF: Boolean constraint propagation

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

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 (cont)

```
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) {
      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.
- One could keep for each literal a list of clauses in which it occurs.
- It is even enough to watch two literals in each clause such that either one of them is true or both are unassigned.
  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.

SAT-solving: Components

- Decision (enumeration)
- Boolean Constraint Propagation (BCP)
- Conflict resolution and backtracking

Implication graph

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

Definition

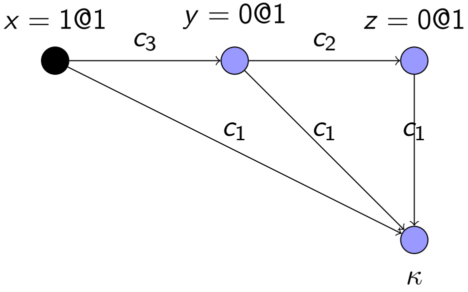
An implication graph is a labeled directed acyclic graph $G = (V, E, L)$, where

- 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

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

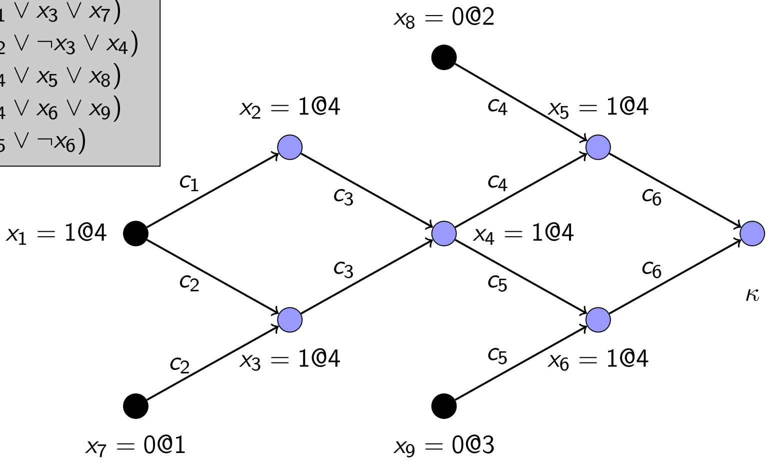
Static variable order $x < y < z$, sign: try positive first



Implication graph: Example

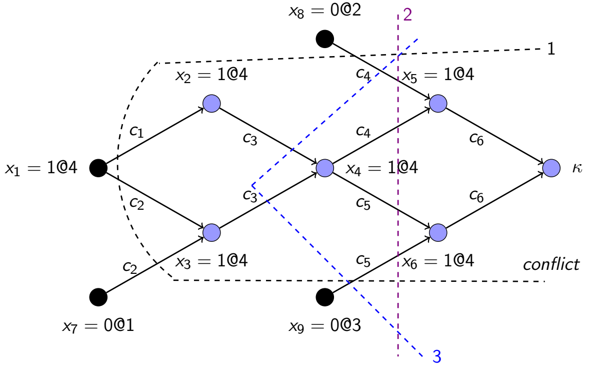
Decisions: $\{x_7 = 0@1,\ x_8 = 0@2,\ x_9 = 0@3,\ x_1 = 1@4\}$

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



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: its satisfaction is necessary for the satisfaction of the formula.



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

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

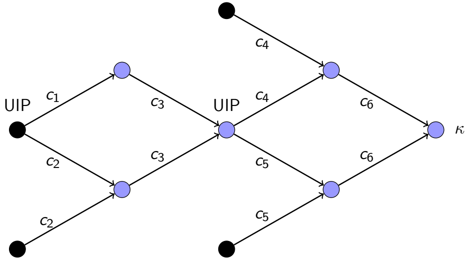
3.$(x_8 \vee \neg x_2 \vee \neg x_3 \vee x_9)$

$\vdots$

$\vdots$

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.
- A unique implication point (UIP) is an internal node in the implication graph such that all paths from the last decision to the conflict node go through it.
- The first UIP is the UIP closest to the conflict.



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 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

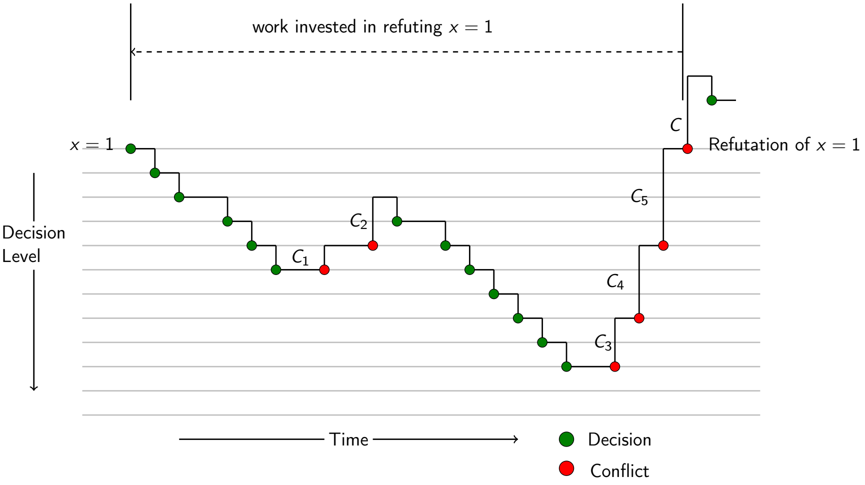
```
if (!BCP()) return UNSAT;
while (true)
{
      if (!decide()) return SAT;
      while (!BCP())
            if (!resolve_conflict()) return UNSAT;
}
```

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

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 resolution

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

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

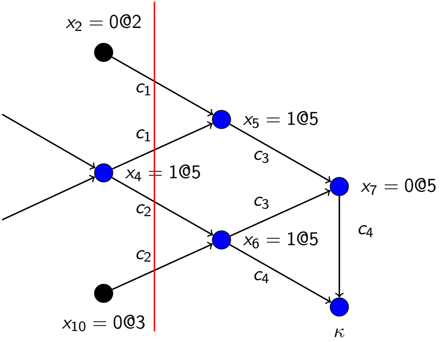
- Example:

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

What is the relation of resolution and conflict clauses?

- Consider the following example:

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

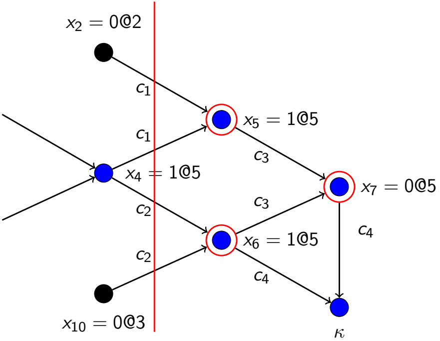


- Conflict clause: $c_5 : (x_2 \vee \neg x_4 \vee x_{10})$

Conflict clauses and resolution

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

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



- Assigment order: x_4, x_5, x_6, x_7
  - $\mathrm{T1} = \mathrm{Res}(c_4, c_3, x_7) = (\neg x_5 \vee \neg x_6)$
  - $\mathrm{T2} = \mathrm{Res}(\mathrm{T1}, c_2, x_6) = (\neg x_4 \vee \neg x_5 \vee x_{10})$
  - $\mathrm{T3} = \mathrm{Res}(\mathrm{T2}, c_1, x_5) = (x_2 \vee \neg x_4 \vee x_{10})$

Finding the conflict clause

```
procedure analyze_conflict() {
   if (current_decision_level = 0) return false;
   cl := current_conflicting_clause;
   while (not stop_criterion_met(cl)) 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

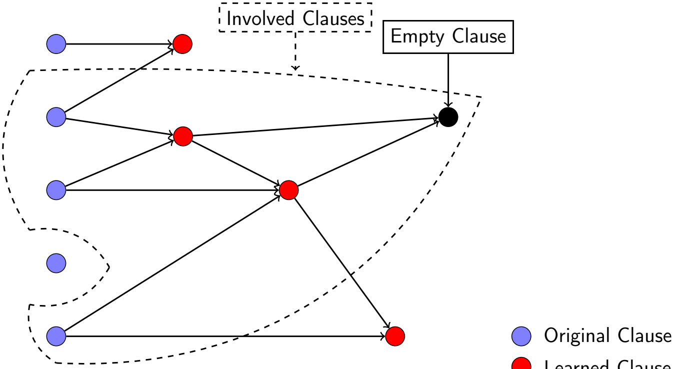
Definition

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

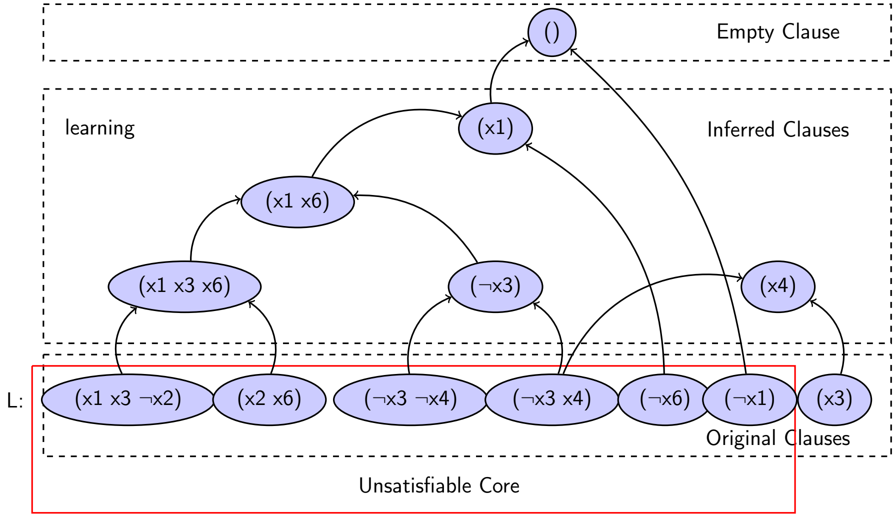
- The set of all original clauses is an unsatisfiable core.
- The set of those original clauses that were used for resolution in conflict analysis during SAT-solving (inclusively the last conflict at decision level 0) gives us an unsatisfiable core which is in general much smaller.
- However, this unsatifiable 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. □

SAT-solving: Components

Back to decision heuristics...

- Decision (enumeration)
- Boolean Constraint Propagation (BCP)
- Conflict resolution and backtracking

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 (cont'd)

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

Decision heuristics

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..."