Algorithms (IV)

DPLL Algorithm

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Why SAT Solving

The First Example

Let $S = \{s_1, \ldots, s_n\}$ be a set of radio stations, each of which has to be allocated one of k transmission frequencies, for some $k < n$. Two stations that are too close to each other cannot have the same frequency. The set of pairs having this constraint is denoted by E. Satisfying

- Every station is assigned at least one frequency.
- Every station is assigned not more than one frequency.
- Close stations are not assigned the same frequency.

Give solution to work out that whether k is enough for a given situation.

The Solution

Define a set of propositional variables

$$\{x_{ij} | i \in \{1, \ldots, n\}, j \in \{1, \ldots, k\}\}$$

Intuitively, variable x_{ij} is set to true if and only if station i is assigned the frequency j.

The Solution

Every station is assigned at least one frequency:

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k} x_{ij}$$

Every station is assigned not more than one frequency:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k-1} (x_{ij} \to \wedge_{j<t\leq k} \neg x_{it})$$

Close stations are not assigned the same frequency. For each $(i,j) \in E$,

$$\bigwedge_{t=1}^{k} x_{it} \to \neg x_{jt}$$

The Second Example

Consider the two code fragments. The fragment on the right-hand side might have been generated from the fragment on the left-hand side by an optimizing compiler. We would like to check if the two programs are equivalent.

```
if(!a && !b) h();
else
    if(!a) g();
    else f();
```

```
if(a) f();
else
    if(b) g();
    else h();
```

The Solution

$$(\text{if } x \text{ then } y \text{ else } z) \equiv (x \wedge y) \vee (\neg x \wedge z)$$

$$(\neg a \wedge \neg b) \wedge h \vee \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \vee a \wedge f)$$

$$\leftrightarrow a \wedge f \vee \neg a \wedge (b \wedge g \vee \neg b \wedge h)$$

Before Beginning

Q: Can a proportional formula be transformed into an equivalent CNF formula effectively?

A: It can, however, while potentially increasing the size of the formula exponentially.

Yet, any propositional formula can also be transformed into an equisatisfiable CNF formula with only a linear increase in the size of the formula. The price to be paid is n new Boolean variables, known as Tseitin's encoding.

Two Usual Ways to Implement

- Exhaustive Search (DPLL Algorithm): traversing and backtracking on a binary tree.
- Stochastic Search: guessing a full assignment, and flipping values of variables according to some heuristic.

Backtracking

Backtracking

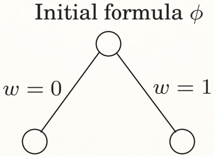
It is often possible to reject a solution by looking at just a small portion of it.

An Solution of SAT

- For example, if an `instance` of SAT contains the clause $(x_1 \vee x_2)$, then all assignments with $x_1 = x_2 = \texttt{false}$ can be instantly eliminated.
- To put it differently, by quickly checking and discrediting this partial assignment, we are able to prune a quarter of the entire search space.
- A promising direction, but can it be systematically exploited?

An Example

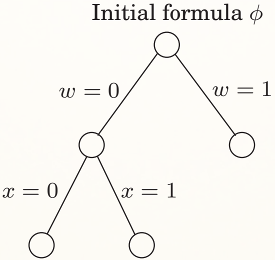
$$(w \lor x \lor y \lor z)(w \lor \overline{x})(x \lor \overline{y})(y \lor \overline{z})(z \lor \overline{w})(\overline{w} \lor \overline{z})$$



Plugging $w = 0$ and $w = 1$ into Φ, we find that no clause is immediately violated and thus neither of these two partial assignments can be eliminated outright.

An Example

$$\Phi = (w \vee x \vee y \vee z)(w \vee \overline{x})(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{w})(\overline{w} \vee \overline{z})$$



The partial assignment $w = 0, x = 1$ violates the clause $(w \vee \overline{x})$ and can be terminated, thereby pruning a good chunk of the search space.

An Example

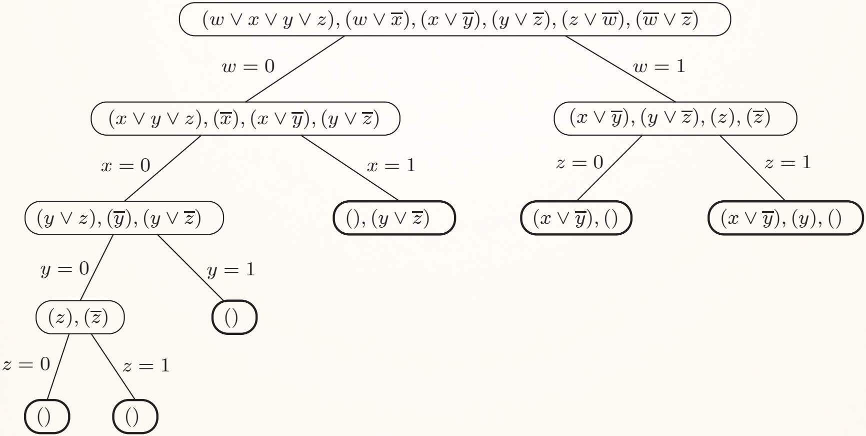
$$\Phi = (w \vee x \vee y \vee z)(w \vee \overline{x})(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{w})(\overline{w} \vee \overline{z})$$

- Backtracking explores the space of assignments, only growing the tree only at nodes where there is uncertainty.
- Each node of the search tree can be described either by a partial assignment or by the clauses that remain.
- If $w = 0$ and $x = 0$ then any clause with w or x is instantly satisfied and any literal $\overline{w}$ or $\overline{x}$ is not satisfied and can be removed.
- What's left is

$$(y \vee z)(\overline{y})(y \vee \overline{z})$$

- Thus the nodes of the search tree, representing partial assignments, are themselves SAT subproblems.

An Example


$(w \vee x \vee y \vee z), (w \vee \overline{x}), (x \vee \overline{y}), (y \vee \overline{z}), (z \vee \overline{w}), (\overline{w} \vee \overline{z})$
$w = 0$
$w = 1$
$(x \vee y \vee z), (\overline{x}), (x \vee \overline{y}), (y \vee \overline{z})$
$(x \vee \overline{y}), (y \vee \overline{z}), (z), (\overline{z})$
$x = 0$
$x = 1$
$z = 0$
$z = 1$
$(y \vee z), (\overline{y}), (y \vee \overline{z})$
$(), (y \vee \overline{z})$
$(x \vee \overline{y}), ()$
$(x \vee \overline{y}), (y), ()$
$y = 0$
$y = 1$
$(z), (\overline{z})$
$()$
$z = 0$
$z = 1$
$()$
$()$

Basic Functions

`Decide()`: Choose the next variable and value. Return False if all variables are assigned.

`BCP()`: Apply repeatedly the unit clause rule. Return False if reached a conflict.

`Resolve-conflict()`: Backtrack until no conflict. Return False if impossible.

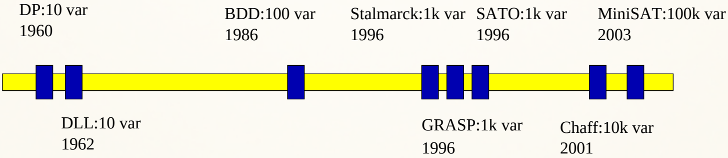
Algorithm

```
SAT()

while true do
    if ¬ Decide () then
        return true ;
    end
    else
        while ¬ BCP () do
            if ¬ Resolve-conflict () then return false;
        end
    end
end
```

A Brief History

- Originally, DPLL was incomplete method for SAT in FO logic
- First paper (Davis and Putnam) in 1960: memory problems
- Second paper (Davis, Logemann and Loveland) in 1962: Depth-first-search with backtracking
- Late 90's and early 00's improvements make DPLL efficient:
- Break-through systems: GRASP, SATO, zChaff, MiniSAT, Z3



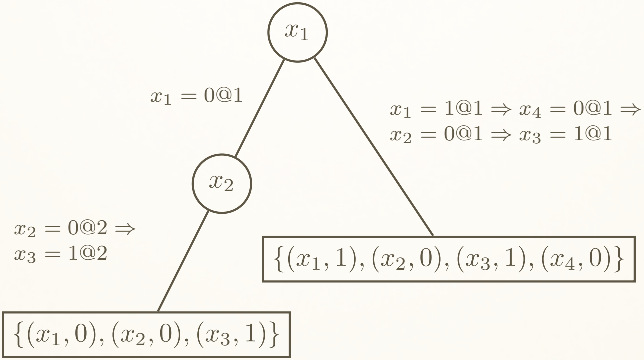
Basic Backtracking Search

Organize the search in the form of a decision tree

- Each node corresponds to a decision.
- Definition: Decision Level (DL) is the depth of the node in the decision tree.
- Notation: $x = v@d$, where $x \in \{0, 1\}$ is assigned to v at decision level d.

Backtracking Search in Action

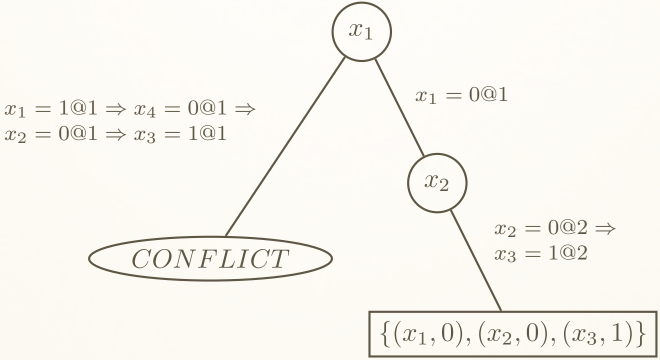
$$(x_2 \vee x_3), (\neg x_1 \vee, \neg x_4), (\neg x_2 \vee x_4)$$



No backtrack in this example, regardless of the decision!

Backtracking Search in Action

$(x_2 \vee x_3), (\neg x_1 \vee, \neg x_4), (\neg x_2 \vee x_4), (\neg x_1 \vee x_2 \vee \neg x_3)$



Status of a Clause

A clause can be

- Satisfied: at least one literal is satisfied
- Unsatisfied: all literals are assigned but non are satisfied
- 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

Decision heuristics - DLIS

DLIS (Dynamic Largest Individual Sum)

Choose the assignment that increases the most the number of satisfied clauses.

For a given variable x:

- C_{xp}: $\sharp$ unresolved clauses in which x appears positively
- C_{xn}: $\sharp$ unresolved clauses in which x appears negatively
- Let x be the literal for which C_{xp} is maximal
- Let y be the literal for which C_{yn} is maximal
- If $C_{xp} > C_{yn}$ choose x and assign it TRUE
- Otherwise choose y and assign it FALSE

Requires l ($\sharp$ literals) queries for each decision.

Decision heuristics - JW

Jeroslow-Wang

Compute for every clause w and every literal l in each phase

$$J(l) = \sum_{l \in w, w \in \varphi} 2^{-|w|}$$

where $|w|$ the length.

Choose the literal l that maximizes $J(l)$.

This gives an exponentially higher weight to literals in shorter clauses.

Next

We will see other (more advanced) decision Heuristics soon.

These heuristics are integrated with a mechanism called Learning with Conflict-Clauses, which we will learn next.

Learning New Clause

Implication graphs and Learning

Current truth assignment

$\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2\}$

Current decision assignment $\{x_1 = 1@6\}$

$w_1 = \neg x_1 \vee x_2$

$w_2 = \neg x_1 \vee x_3 \vee x_9$

$w_3 = \neg x_2 \vee \neg x_3 \vee x_4$

$w_4 = \neg x_4 \vee x_5 \vee x_{10}$

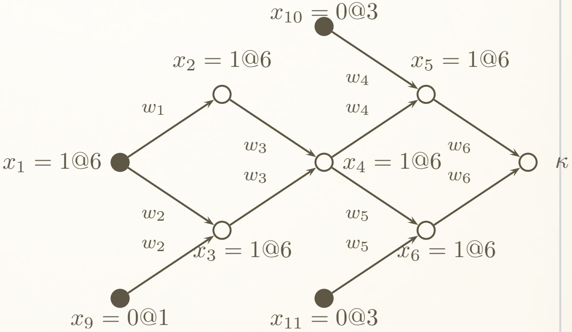
$w_5 = \neg x_4 \vee x_6 \vee x_{11}$

$w_6 = \neg x_5 \vee \neg x_6$

$w_7 = x_1 \vee x_7 \vee \neg x_{12}$

$w_8 = x_1 \vee x_8$

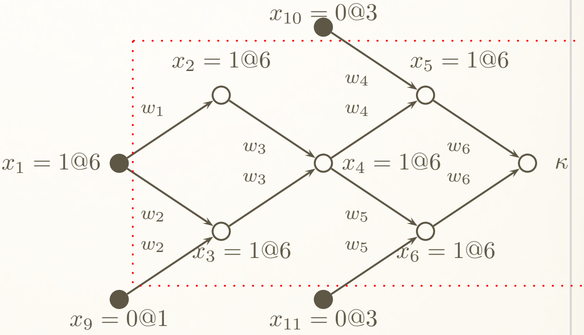
$w_9 = \neg x_7 \vee \neg x_8 \vee \neg x_{13}$



Implication Graphs and Learning

Current truth assignment
$\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2\}$
Current decision assignment $\{x_1 = 1@6\}$

$w_1 = \neg x_1 \vee x_2$
$w_2 = \neg x_1 \vee x_3 \vee x_9$
$w_3 = \neg x_2 \vee \neg x_3 \vee x_4$
$w_4 = \neg x_4 \vee x_5 \vee x_{10}$
$w_5 = \neg x_4 \vee x_6 \vee x_{11}$
$w_6 = \neg x_5 \vee \neg x_6$
$w_7 = x_1 \vee x_7 \vee \neg x_{12}$
$w_8 = x_1 \vee x_8$
$w_9 = \neg x_7 \vee \neg x_8 \vee \neg x_{13}$



We learn the conflict clause $w_{10} : (\neg x_1 \vee x_9 \vee x_{11} \vee x_{10})$

Flipped Assignment

Current truth assignment

$\{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2\}$

Current flipped assignment $\{x_1 = 0@6\}$

$w_1 = \neg x_1 \vee x_2$

$w_2 = \neg x_1 \vee x_3 \vee x_9$

$w_3 = \neg x_2 \vee \neg x_3 \vee x_4$

$w_4 = \neg x_4 \vee x_5 \vee x_{10}$

$w_5 = \neg x_4 \vee x_6 \vee x_{11}$

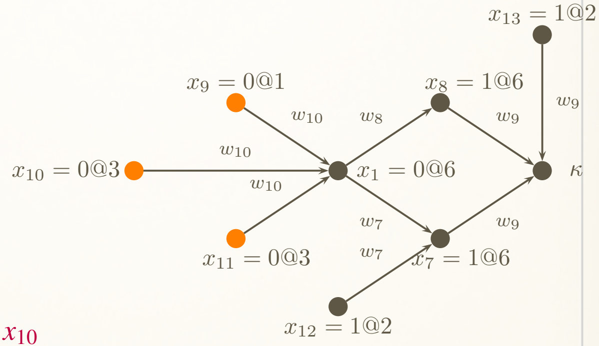
$w_6 = \neg x_5 \vee \neg x_6$

$w_7 = x_1 \vee x_7 \vee \neg x_{12}$

$w_8 = x_1 \vee x_8$

$w_9 = \neg x_7 \vee \neg x_8 \vee \neg x_{13}$

$w_{10} = \neg x_1 \vee x_9 \vee x_{11} \vee x_{10}$



Another conflict clause: $w_{11} : (\neg x_{13} \vee \neg x_{12} \vee x_{11} \vee x_{10} \vee x_9)$

Where should we backtrack to now?

Non-Chronological Backtracking

Which assignments caused the conflicts?

- $x_9 = 0@1$
- $x_{10} = 0@3$
- $x_{11} = 0@3$
- $x_{12} = 1@2$
- $x_{13} = 1@2$

These assignments are sufficient for causing a conflict.

Backtrack to $DL = 3$

Non-Chronological Backtracking

So the rule is: backtrack to the largest decision level in the conflict clause.

This works for both the initial conflict and the conflict after the flip.

Q: What if the flipped assignment works?
A: Change the decision retroactively.

Non-Chronological Backtracking

$$x_1 = 0$$
$$x_2 = 0$$
$$x_3 = 1$$
$$x_4 = 0$$
$$x_5 = 0$$

Non-Chronological Backtracking

$$
\begin{aligned}
x_1 &= 0 \\
x_2 &= 0 \\
x_3 &= 1 \\
x_4 &= 0 \\
x_5 &= 0
\end{aligned}
$$

Non-Chronological Backtracking

$x_1 = 0$

$x_2 = 0$

$x_3 = 1$

$x_4 = 0$

$x_5 = 0$ $\quad\quad$ $x_5 = 1$

$x_7 = 0$

$x_9 = 1$

Non-Chronological Backtracking

$x_1 = 0$		
$x_2 = 0$		
$x_3 = 1$		
$x_4 = 0$		
$x_5 = 0$	$x_5 = 1$	
	$x_7 = 0$	
	$x_9 = 1$	$x_9 = 0$

Non-Chronological Backtracking

$x_1 = 0$		
$x_2 = 0$		
$x_3 = 1$		$x_3 = 0$
$x_4 = 0$		
$x_5 = 0$	$x_5 = 1$	
	$x_7 = 0$	
	$x_9 = 1$	$x_9 = 0$

Non-Chronological Backtracking

$x_1 = 0$
$x_2 = 0$

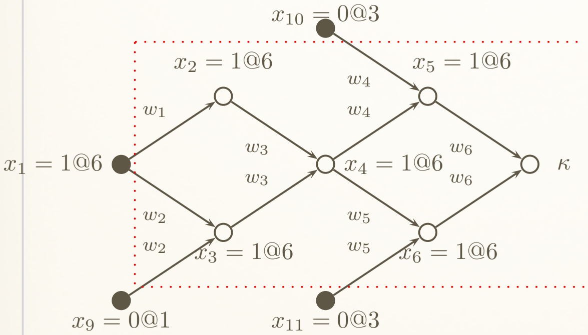
$x_3 = 0$
$x_6 = 0$
$\ldots$
$\ldots$

More Conflict Clauses

Def: A Conflict Clause is any clause implied by the formula

Let L be a set of literals labeling nodes that form a cut in the implication graph, separating the conflict node from the roots.

Claim: $\bigvee_{l \in L} \neg l$ is a Conflict Clause.



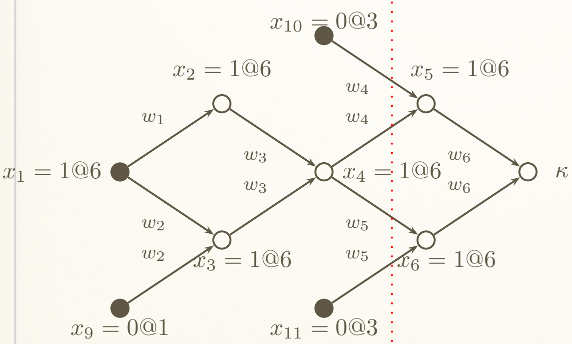
$x_{10} \vee \neg x_1 \vee x_9 \vee x_{11}$

More Conflict Clauses

Def: A Conflict Clause is any clause implied by the formula

Let L be a set of literals labeling nodes that form a cut in the implication graph, separating the conflict node from the roots.

Claim: $\bigvee_{l \in L} \neg l$ is a Conflict Clause.



$x_{10} \vee \neg x_1 \vee x_9 \vee x_{11}$

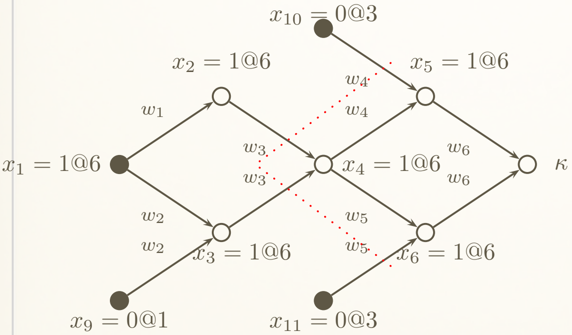
$x_{10} \vee \neg x_4 \vee x_{11}$

More Conflict Clauses

Def: A Conflict Clause is any clause implied by the formula

Let L be a set of literals labeling nodes that form a cut in the implication graph, separating the conflict node from the roots.

Claim: $\bigvee_{l \in L} \neg l$ is a Conflict Clause.



$x_{10} \vee \neg x_1 \vee x_9 \vee x_{11}$
$x_{10} \vee \neg x_4 \vee x_{11}$
$x_{10} \vee \neg x_2 \vee \neg x_3 \vee x_{11}$

Conflict Clause

How many clauses should we add?

If not all, then which ones?

- Shorter ones?
- Check their influence on the backtracking level?
- The most “influential”?

Conflict Clause

Def: 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.

Asserting clauses are those that force an immediate change in the search path.

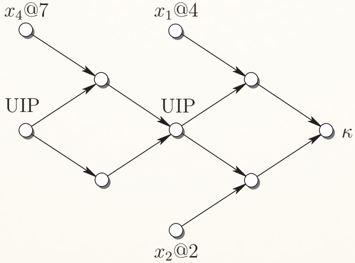
Modern solvers only consider Asserting Clauses.

Unique Implication Points (UIPs)

Unique Implication Point (UIP)

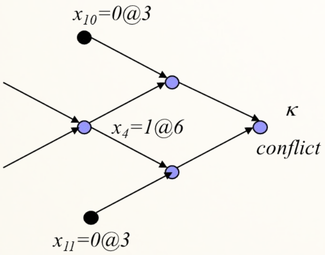
A Unique Implication Point (UIP) is an internal node in the Implication Graph that all paths from the decision to the conflict node go through it.

The First-UIP is the closest UIP to the conflict.



Alternative Backtracking

Conflict-Driven Backtracking



Conflict clause: $(x_{10} \vee \neg x_4 \vee \neg x_{11})$

With standard Non-Chronological Backtracking we backtracked to $DL = 6$.

Conflict-driven Backtrack: backtrack to the second highest decision level in the clause (without erasing it).

In this case, to $DL = 3$.

Q: why?

Conflict-Driven Backtracking

$$
\begin{aligned}
x_1 &= 0 \\
x_2 &= 0 \\
x_3 &= 1 \\
x_4 &= 0 \\
x_5 &= 0
\end{aligned}
$$

Conflict-Driven Backtracking

$$
\begin{aligned}
x_1 &= 0 \\
x_2 &= 0 \\
x_3 &= 1 \\
x_4 &= 0 \\
x_5 &= 0
\end{aligned}
$$

Conflict-Driven Backtracking

$x_1 = 0$
$x_2 = 0$
$x_3 = 1$
$x_4 = 0$
$x_5 = 0$

Conflict-Driven Backtracking

$x_1 = 0$
$x_2 = 0$

$x_5 = 1$
$x_7 = 0$
$x_9 = 1$

Conflict-Driven Backtracking

$x_1 = 0$
$x_2 = 0$

$x_5 = 1$
$x_7 = 0$
$x_9 = 1$

Conflict-Driven Backtracking

$x_1 = 0$

$x_2 = 0$

$x_5 = 1$

$x_7 = 0$

$x_9 = 1$

Conflict-Driven Backtracking

$x_1 = 0$
$x_2 = 0$

$x_5 = 1$

$x_9 = 1$
$x_6 = 0$

Conflict-Driven Backtracking

So the rule is: backtrack to the second highest decision level dl, but do not erase it.

This way the literal with the currently highest decision level will be implied in $DL = dl$.

Q: what if the conflict clause has a single literal?

For example, from $(x \vee \neg y) \wedge (x \vee y)$ and decision $x = 0$, we learn the conflict clause (x).

Resolution

The binary resolution is a sound inference rule:

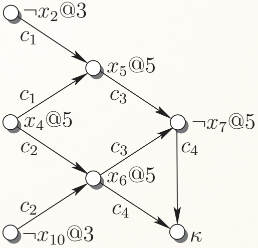
$$\frac{(a_1 \vee \ldots \vee a_n \vee \beta) \quad (b_1 \vee \ldots \vee b_m \vee \neg\beta)}{(a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)} \text{ Binary Resolution}$$

Example

$$\frac{x_1 \vee x_2 \quad \neg x_1 \vee x_3 \vee x_4}{x_2 \vee x_3 \vee x_4}$$

Example

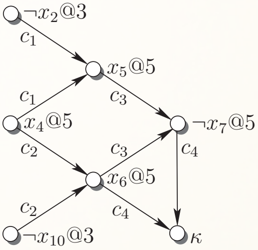
$c_1 = (\neg x_4 \vee x_2 \vee x_5)$
$c_2 = (\neg x_4 \vee x_{10} \vee x_6)$
$c_3 = (\neg x_5 \vee \neg x_6 \vee \neg x_7)$
$c_4 = (\neg x_6 \vee x_7)$



Conflict Clause : $c_5 = (\neg x_4 \vee x_2 \vee x_{10})$

Example

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



Assume that the implication order in the BCP was x_4, x_5, x_6, x_7.

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	$(\neg x_4 \vee x_2 \vee x_{10})$			

The Algorithm

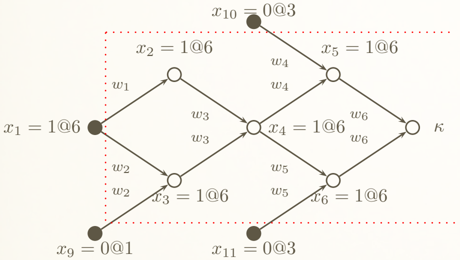
```
ANALYZE-CONFLICT()
```

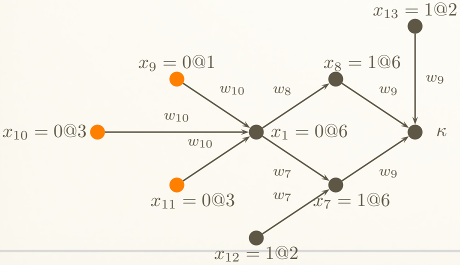
if *current_desicion_level* $= 0$ **then** `return` False;
while $\neg$ `STOP-CRITERION-MET` *(cl)* **do**
    lit := `LAST-ASSIGNED-LITERAL` (*cl*);
    var := `VARIABLE-OF-LITERAL` (*lit*);
    ante := `Antecedent`(*lit*);
    cl := `RESOLVE` (*cl*, *ante*, *var*);
end
`ADD-CLAUSE-TO-DATABASE` (*cl*);

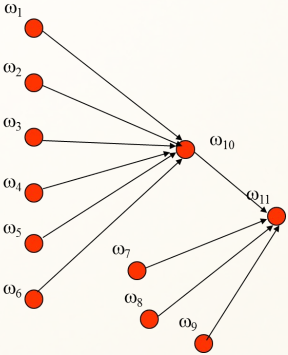
name	*cl*	*lit*	*var*	*ante*
c_4	$(\neg x_6 \vee x_7)$	x_7	x_7	c_3
	$(\neg x_5 \vee \neg x_6)$	$\neg x_6$	x_6	c_2
	$(\neg x_4 \vee x_{10} \vee \neg x_5)$	$\neg x_5$	x_5	c_1
c_5	$(\neg x_4 \vee x_2 \vee x_{10})$			

Resolution Graph

The resolution graph keeps track of the inference relation.

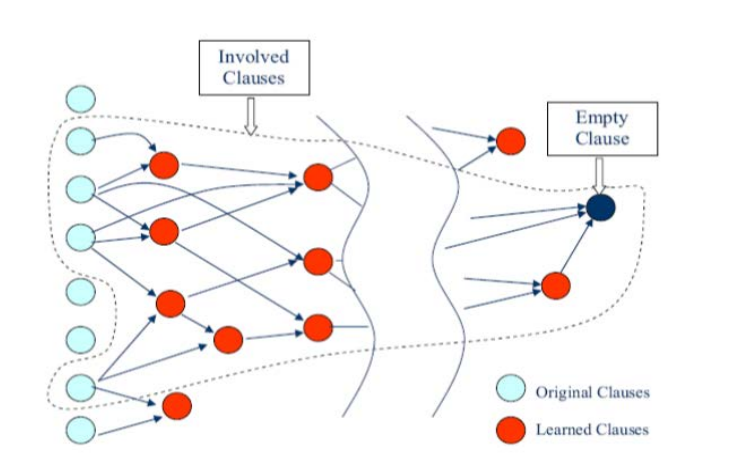






Resolution Graph

What is it good for?
Example: for computing an unsatisfiable core



from SAT'03

Decision Heuristics - VSIDS

VSIDS (Variable State Independent Decaying Sum)

Each literal has a counter initialized to 0.

When a clause is added, the counters are updated.

The unassigned variable with the highest counter is chosen.

Periodically, all the counters are divided by a constant.

firstly implemented in **Chaff**

Decision Heuristics - VSIDS

Chaff holds a list of unassigned variables sorted by the counter value.

Updates are needed only when adding conflict clauses.

Thus, decision is made in constant time.

Decision Heuristics - VSIDS

VSIDS is a quasi-static strategy:

- static because it does not 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, which dramatically improves performance.

Decision Heuristics - Berkmin

Keep conflict clauses in a stack

Choose the first unresolved clause in the stack (If there is no such clause, use VSIDS)

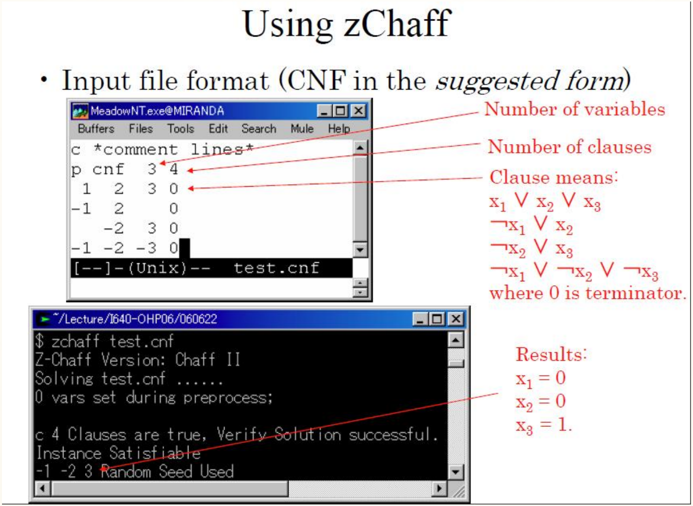
Choose from this clause a variable + value accprding to some scoring (e.g. VSIDS)

This gives absolute priority to conflicts.

SAT Solver

- SAT solver is to be said as the ”most successful formal tools”.
- There are a SAT Competitions every one or two years.
  - http://www.satcompetition.org/
- Zchaff(The champion of 2004) can handle 100,000 variables with millions of clauses (Experiments: 800 variables with 9,000 clauses in 0.0sec).

Zchaff


Using zChaff
• Input file format (CNF in the *suggested form*)
MeadowNT.exe@MIRANDA
Buffers Files Tools Edit Search Mule Help
c *comment lines*
p cnf 3 4
1 2 3 0
-1 2 0
-2 3 0
-1 -2 -3 0
[--]-(Unix)-- test.cnf
Number of variables
Number of clauses
Clause means:
$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee x_2$
$\neg x_2 \vee x_3$
$\neg x_1 \vee \neg x_2 \vee \neg x_3$
where 0 is terminator.
~/Lecture/I640-OHP06/060622
$ zchaff test.cnf
Z-Chaff Version: Chaff II
Solving test.cnf ......
0 vars set during preprocess;
c 4 Clauses are true, Verify Solution successful.
Instance Satisfiable
-1 -2 3 Random Seed Used
Results:
$x_1 = 0$
$x_2 = 0$
$x_3 = 1.$