Decision Procedures and Verification

Martin Blicha

Charles University

5.2.2018
Algorithms for Propositional Satisfiability
Why study propositional satisfiability?

- Interesting from both theoretical and practical perspective
- First problem to be proven NP-complete [Cook ’71, Levin ’73]
- Many industrial problems encoded as SAT
  - Hardware and software verification
  - Automated planning - Planning as Satisfiability
  - Product configuration
  - ...

Progress in SAT solving

Image source: Decision Procedures. Kroening D., Strichman O.
Progress in SAT solving

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

Image source: Decision Procedures. Kroening D., Strichman O.
Approaches to SAT solving

DPLL framework
- complete procedure

Stochastic search
- incomplete procedure
Approaches to SAT solving

DPLL framework
➤ complete procedure
➤ very efficient for instance with structure

Stochastic search
➤ incomplete procedure
➤ better at solving random satisfiable instances
## Approaches to SAT solving

<table>
<thead>
<tr>
<th>DPLL framework</th>
<th>Stochastic search</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete procedure</td>
<td>incomplete procedure</td>
</tr>
<tr>
<td>very efficient for instance</td>
<td>better at solving random satisfiable</td>
</tr>
<tr>
<td>with structure</td>
<td>instances</td>
</tr>
<tr>
<td>important when proof of unsatisfiability required</td>
<td>can be faster to obtain satisfying</td>
</tr>
<tr>
<td>(e.g. verification)</td>
<td>assignment</td>
</tr>
</tbody>
</table>
Problems of naive satisfiability algorithm

Naive algorithm

Enumerate all assignments. Check if formula is satisfied under any of them.
Problems of naive satisfiability algorithm

Naive algorithm
Enumerate all assignments. Check if formula is satisfied under any of them.

- Unnecessary repetition of partial assignment leading to conflict.
- No information preserved between tries of different assignments
- Lots of unnecessary work being done over and over again.
DPLL algorithm - overview

- *DPLL* algorithm (Davis-Putnam-Loveland-Logemann, 1962)
- Input formula assumed to be in CNF
- *Search* in a tree of partial assignments
- *Backtracking* on conflict
- *Unit propagation* prunes the tree
Basic concepts

Definition (state of a clause)

Let $\alpha : V \to \{ \text{True}, \text{False} \}$ be an assignment of variables from $V$. Then generalization of $\alpha$ on a clauses of set of variables $V' \supseteq V$ is $\alpha^* : \{ c \mid c \text{ is a clause over } V' \} \to \{ \text{True}, \text{False}, \text{Undef} \}$.

- $c$ is satisfied, $\alpha^*(c) = \text{True}$, if at least one literal in $c$ is satisfied by $\alpha$
- $c$ is conflicting, $\alpha^*(c) = \text{False}$, if all literals are falsified by $\alpha$
- $c$ is unresolved, $\alpha^*(c) = \text{Undef}$, otherwise.
Basic concepts

Definition (state of a clause)
Let $\alpha : V \to \{True, False\}$ be an assignment of variables from $V$. Then generalization of $\alpha$ on a clauses of set of variables $V' \supseteq V$ is $\alpha^* : \{c \mid c \text{ is a clause over } V'\} \to \{True, False, Undef\}$.

- $c$ is satisfied, $\alpha^*(c) = True$, if at least one literal in $c$ is satisfied by $\alpha$
- $c$ is conflicting, $\alpha^*(c) = False$, if all literals are falsified by $\alpha$
- $c$ is unresolved, $\alpha^*(c) = Undef$, otherwise.

Definition (unit clause)
A clause $c$ is unit under assignment $\alpha$ if it is not satisfied and all but one literals are falsified by $\alpha$. 

Basic concepts

Example
Let $\alpha$ be $\{x_1 \mapsto 1, x_2 \mapsto 0, x_4 \mapsto 1\}$. Then

- $x_1 \lor x_3 \lor \neg x_4$ is satisfied,
- $\neg x_1 \lor x_2$ is conflicting,
- $\neg x_1 \lor \neg x_4 \lor x_3$ is unit,
- $\neg x_1 \lor x_3 \lor x_5$ is unresolved.
Basic concepts

Example
Let \(\alpha\) be \(\{x_1 \mapsto 1, x_2 \mapsto 0, x_4 \mapsto 1\}\). Then

- \(x_1 \lor x_3 \lor \neg x_4\) is satisfied,
- \(\neg x_1 \lor x_2\) is conflicting,
- \(\neg x_1 \lor \neg x_4 \lor x_3\) is unit,
- \(\neg x_1 \lor x_3 \lor x_5\) is unresolved.

Definition (unit clause rule, antecedent clause)
Given a partial assignment \(\alpha\) and a clause \(c\) that is unit under \(\alpha\), \(\alpha\) must be extended so that it satisfies the last unassigned literal \(l\). We say that \(l\) is implied by \(c\) (under \(\alpha\)) and we call \(c\) the antecedent of \(l\).
Basic concepts

Example
Let $\alpha$ be \{ $x_1 \mapsto 1$, $x_2 \mapsto 0$, $x_4 \mapsto 1$ \}. Then

- $x_1 \lor x_3 \lor \neg x_4$ is satisfied,
- $\neg x_1 \lor x_2$ is conflicting,
- $\neg x_1 \lor \neg x_4 \lor x_3$ is unit,
- $\neg x_1 \lor x_3 \lor x_5$ is unresolved.

Definition (unit clause rule, antecedent clause)
Given a partial assignment $\alpha$ and a clause $c$ that is unit under $\alpha$, $\alpha$ must be extended so that it satisfies the last unassigned literal $l$. We say that $l$ is implied by $c$ (under $\alpha$) and we call $c$ the \textit{antecedent} of $l$.

Example
The clause $c = \neg x_1 \lor \neg x_4 \lor x_3$ and the partial assignment \{ $x_1 \mapsto 1$, $x_4 \mapsto 1$ \} imply $x_3 \mapsto 1$ and Antecedent$(x_3) = c$. 
The power of unit propagation

- The goal is to satisfy a CNF formula.

\[
(\neg u \lor w) \land (u \lor v) \land (u) \land (\neg w \lor z); \quad \alpha = \{\}
\]

- \((u)\) is unit under \(\alpha\)
The power of unit propagation

- The goal is to satisfy a CNF formula.

- \((\neg u \lor w) \land (u \lor v) \land (u) \land (\neg w \lor z); \; \alpha = \{u\}\)
  - \((\neg u \lor w)\) is unit under \(\alpha\)
The power of unit propagation

- The goal is to satisfy a CNF formula.

\[(\neg u \lor w) \land (u \lor v) \land (u) \land (\neg w \lor z)\];
\[\alpha = \{u, w\}\]

- \((\neg w \lor z)\) is unit under \(\alpha\)
The power of unit propagation

The goal is to satisfy a CNF formula.

\((\neg u \lor w) \land (u \lor v) \land (u) \land (\neg w \lor z)\); 
\(\alpha = \{u, w, z\}\)

- All clauses are satisfied by \(\alpha\).

Solved by unit propagation. No decisions needed.
DPLL algorithm

1: procedure DPLL(\(\varphi, \alpha\))
2: if \(\forall c \in \varphi\) then \(c\) is satisfied by \(\alpha\) return TRUE
3: if \(\exists c \in \varphi\) then \(c\) is conflicting under \(\alpha\) return FALSE
4: \(\alpha \leftarrow \alpha \cup UNIT\text{-}\text{PROPAGATION()}\)
5: \(x \leftarrow SELECT\text{-}\text{VAR()}\)
6: if DPLL(\(\alpha \cup \{x \mapsto 1\}\)) then return TRUE
7: if DPLL(\(\alpha \cup \{x \mapsto 0\}\)) then return TRUE
8: return FALSE
DPLL algorithm

Notes

- UNIT-PROPAGATION applies unit clause rule until no clauses are unit.
- After unit propagation, assignment can be extended with pure literals.
  - But this is not used in practice (too costly).
- The phase of unit propagation possibly with pure literals is often referred to as BCP - Boolean constraint propagation.
- SELECT-VAR selects an unassigned variable. Both values of the variable are tried.
DPLL - running example

$$\varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$

1. Decide $$\alpha(x_1) = 1$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
   \text{unit clause}

2. Derive $$\alpha(x_3) = 0$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
   \text{unit clause}
   $$(\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
   \text{unit clause}

3. Derive $$\alpha(x_2) = 1, \alpha(x_4) = 1$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
   \text{conflict clause}

4. Derive $$\alpha(x_3) = 1$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
DPLL - running example

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

   unit clause

2. Derive \( \alpha(x_3) = 0 \):
   
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

   unit clause
   
   \[ (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

   unit clause

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):
   
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

   conflict clause

4. Derive \( \alpha(x_3) = 1 \):
   
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

   unit clause

\[ \alpha = \{ x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0 \} \] is a satisfying assignment of \( \varphi \)
DPLL - running example

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):

\[
(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
(\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
\]

unit clause

2. Derive \( \alpha(x_3) = 0 \):

\[
(x_1 \lor x_3 \lor x_4) \land \\
(\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land \\
(\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
\]

unit clause

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):

\[
(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
(\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
\]

conflict clause

4. Derive \( \alpha(x_3) = 1 \):

\[
(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
(\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
\]

\[ \alpha = \{ x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0 \} \] is a satisfying assignment of \( \varphi \)
DPLL - running example

$$\varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$

1. Decide $$\alpha(x_1) = 1$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
   unit clause

2. Derive $$\alpha(x_3) = 0$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3)$$
   unit clause
   $$(\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
   unit clause

3. Derive $$\alpha(x_2) = 1, \alpha(x_4) = 1$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$
   conflict clause

4. Derive $$\alpha(x_3) = 1$$:
   $$(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)$$

$$\alpha = \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0\}$$ is a satisfying assignment of $$\varphi$$
\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   - unit clause

2. Derive \( \alpha(x_3) = 0 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   - unit clause

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   - conflict clause

4. Derive \( \alpha(x_3) = 1 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

\[ \alpha = \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0\} \] is a satisfying assignment of \( \varphi \)
DPLL - running example

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   unit clause

2. Derive \( \alpha(x_3) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   unit clause

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   conflict clause

4. Derive \( \alpha(x_3) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   unit clause

1. Decide \( \alpha(x_1) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   unit clause

2. Decide \( \alpha(x_2) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   unit clause

3. Derive \( \alpha(x_4) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   unit clause

4. Derive \( \alpha(x_3) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]  
   unit clause

\[ \alpha = \{ x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0 \} \]

is a satisfying assignment of \( \varphi \)
DPLL - running example

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   \[ \text{unit clause} \]

2. Derive \( \alpha(x_3) = 0 \):
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   \[ \text{unit clause} \]

3. Derive \( \alpha(x_2) = 1 \), \( \alpha(x_4) = 1 \):
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   \[ \text{conflict clause} \]

4. Derive \( \alpha(x_3) = 1 \):
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

\[ \alpha = \{ x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0 \} \]

is a satisfying assignment of \( \varphi \)
DPLL - running example

\( \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \)

1. Decide \( \alpha(x_1) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
   (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   unit clause

2. Derive \( \alpha(x_3) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land \\
   (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land \\
   (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   unit clause

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
   (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   conflict clause

4. Derive \( \alpha(x_3) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
   (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   unit clause

5. Derive \( \alpha(x_1) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
   (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   unit clause

6. Derive \( \alpha(x_4) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
   (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   unit clause

\( \alpha = \{ x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0 \} \) is a satisfying assignment of \( \varphi \).
DPLL - running example

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   unit clause

2. Derive \( \alpha(x_3) = 0 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   unit clause
   \[(\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   unit clause

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   conflict clause

4. Derive \( \alpha(x_3) = 1 \):
   \[(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   unit clause

\[ \alpha = \{ x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0 \} \] is a satisfying assignment of \( \varphi \)
Deficiencies of DPLL

\[ \psi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \land \\
(\neg y_1 \lor y_3 \lor y_4) \land (\neg y_2 \lor y_3 \lor y_4) \land (\neg y_3 \lor \neg y_4) \land (\neg y_3 \lor y_4) \land (y_3 \lor \neg y_4) \]

- fixed variable ordering: \[ y_1, x_1, x_2, x_3, x_4, y_2, y_3, y_4 \]
- no pure literal propagation
Deficiencies of DPLL

\[ \psi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \land \\
(\neg y_1 \lor y_3 \lor y_4) \land (\neg y_2 \lor y_3 \lor y_4) \land (\neg y_3 \lor \neg y_4) \land (\neg y_3 \lor y_4) \land (y_3 \lor \neg y_4) \]

- fixed variable ordering: \( y_1, x_1, x_2, x_3, x_4, y_2, y_3, y_4 \)
- no pure literal propagation
- algorithm tries \( x_1 \mapsto 1 \) for both branches \( y_1 \mapsto 1 \) and \( y_1 \mapsto 0 \)
Deficiencies of DPLL

\[ \psi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \land \\
(\neg y_1 \lor y_3 \lor y_4) \land (\neg y_2 \lor y_3 \lor y_4) \land (\neg y_3 \lor \neg y_4) \land (\neg y_3 \lor y_4) \land (y_3 \lor \neg y_4) \]

- fixed variable ordering: \( y_1, x_1, x_2, x_3, x_4, y_2, y_3, y_4 \)
- no pure literal propagation
- algorithm tries \( x_1 \mapsto 1 \) for both branches \( y_1 \mapsto 1 \) and \( y_1 \mapsto 0 \)
- repeats *the same conflict* on \( x \) variables in both of these branches
Deficiencies of DPLL

\[ \psi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \land \\
(\neg y_1 \lor y_3 \lor y_4) \land (\neg y_2 \lor y_3 \lor y_4) \land (\neg y_3 \lor \neg y_4) \land (\neg y_3 \lor y_4) \land (y_3 \lor \neg y_4) \]

- fixed variable ordering: \( y_1, x_1, x_2, x_3, x_4, y_2, y_3, y_4 \)
- no pure literal propagation
- algorithm tries \( x_1 \mapsto 1 \) for both branches \( y_1 \mapsto 1 \) and \( y_1 \mapsto 0 \)
- repeats the same conflict on \( x \) variables in both of these branches
- DPLL can repeat the same mistake over and over again
Deficiencies of DPLL

\[ \psi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \land (\neg y_1 \lor y_3 \lor y_4) \land (\neg y_2 \lor y_3 \lor y_4) \land (\neg y_3 \lor \neg y_4) \land (\neg y_3 \lor y_4) \land (y_3 \lor \neg y_4) \]

- fixed variable ordering: \( y_1, x_1, x_2, x_3, x_4, y_2, y_3, y_4 \)
- no pure literal propagation
- algorithm tries \( x_1 \mapsto 1 \) for both branches \( y_1 \mapsto 1 \) and \( y_1 \mapsto 0 \)
- repeats the same conflict on \( x \) variables in both of these branches
- DPLL can repeat the same mistake over and over again
- SAT solver should learn from past mistakes
Implication graph

Definition (Implication graph)

Implication graph for $\alpha$ is an acyclic labeled directed graph $G = (V \cup \{K\}, E)$ where:

- Vertices $V$ correspond to variables.
  - labeled by current assignment and decision level $x@N (\neg x@N)$: $x$ is assigned True (False) at decision level $N$.
- Edges $E$ represents reasons for assigning a value.
  - $(x, y) \in E$, if $\neg x \in \text{Antecedent}(y)$ with $\alpha(x) = 1$ or $x \in \text{Antecedent}(y)$ with $\alpha(x) = 0$
  - $(x, y)$ is labeled with $\text{Antecedent}(y)$.
- Vertex $K$ represents a conflict
  - $(x, K) \in E$, if $\neg x \in c$ with $\alpha(x) = 1$ or $x \in c$ with $\alpha(x) = 0$
    where $c$ is a conflicting clause under $\alpha$.
  - $(x, K)$ is labeled the corresponding conflict clause.
Implication graph

Definition (Implication graph)

Implication graph for \( \alpha \) is an acyclic labeled directed graph \( G = (V \cup \{K\}, E) \) where:

- Vertices \( V \) correspond to variables.
  - labeled by current assignment and decision level
  - \( x@N (\neg x@N) \): \( x \) is assigned True (False) at decision level \( N \).
- Edges \( E \) represents reasons for assigning a value.
  - \( (x, y) \in E \), if \( \neg x \in \text{Antecedent}(y) \) with \( \alpha(x) = 1 \) or \( x \in \text{Antecedent}(y) \) with \( \alpha(x) = 0 \)
  - \( (x, y) \) is labeled with \( \text{Antecedent}(y) \).
- Vertex \( K \) represents a conflict
  - \( (x, K) \in E \), if \( \neg x \in c \) with \( \alpha(x) = 1 \) or \( x \in c \) with \( \alpha(x) = 0 \)
    where \( c \) is a conflicting clause under \( \alpha \).
  - \( (x, K) \) is labeled the corresponding conflict clause.

- Roots (no incoming edges) correspond to decisions, inner nodes (except \( K \)) to unit propagation. If a there is a path from roots to \( K \) we call the implication graph the conflict graph.
Example of implication graph

\[ \varphi = \left( x_1 \lor x_3 \lor x_4 \right) \land \left( \neg x_1 \lor x_2 \lor x_3 \right) \land \left( \neg x_1 \lor \neg x_3 \right) \land \left( \neg x_2 \lor \neg x_4 \right) \land \left( x_3 \lor x_4 \right) \]
Example of implication graph

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):

\[
\begin{align*}
(x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land \\
(\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
\end{align*}
\]

- unit clause

- \( x_1 \oplus 1 \)
Example of implication graph

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   unit clause

2. Derive \( \alpha(x_3) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)
   \]
   unit clause

unit clause
Example of implication graph

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land
   \underbrace{(\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)}_{\text{unit clause}}
   \]

2. Derive \( \alpha(x_3) = 0 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land
   \underbrace{(\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)}_{\text{unit clause}}
   \]

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):
   \[
   (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land
   \underbrace{(\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4)}_{\text{conflict clause}}
   \]
Example of implication graph

\[ \varphi = (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]

1. Decide \( \alpha(x_1) = 1 \):
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   unit clause

2. Derive \( \alpha(x_3) = 0 \):
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   unit clause

3. Derive \( \alpha(x_2) = 1, \alpha(x_4) = 1 \):
   \[ (x_1 \lor x_3 \lor x_4) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_4) \land (x_3 \lor x_4) \]
   conflict clause
Definition (separating cut)
A separating cut in a conflict graph is a minimal set of edges whose removal breaks all paths from the root nodes to the conflict node.

- Each cut splits the graph to *reason* side and *conflict* side.
- The set of nodes on reason side with an edge to conflict side constitutes a sufficient condition for the conflict.
- Its negation is a conflict clause.
Definition (separating cut)
A separating cut in a conflict graph is a minimal set of edges whose removal breaks all paths from the root nodes to the conflict node.

- Each cut splits the graph to reason side and conflict side.
- The set of nodes on reason side with an edge to conflict side constitutes a sufficient condition for the conflict.
- Its negation is a conflict clause.
Clause learning

Observation
Every separating cut in conflict graph determines a conflict clause $c$ such that $\varphi \rightarrow c$, where $\varphi$ is the input formula.

- The conflict clause can be added to the input formula without effecting satisfiability.
- It prunes the search tree.
- This process is referred to as learning.
  - SAT solver is "learning" from its past mistakes.
Clause learning

Observation
Every separating cut in conflict graph determines a conflict clause $c$ such that $\varphi \rightarrow c$, where $\varphi$ is the input formula.

- The conflict clause can be added to the input formula without effecting satisfiability.
- It prunes the search tree.
- This process is referred to as learning.
  - SAT solver is "learning" from its past mistakes.

- First cut $\Rightarrow$ conflict clause $\neg x_1$. 
Clause learning

Observation
Every separating cut in conflict graph determines a conflict clause \( c \) such that \( \varphi \rightarrow c \), where \( \varphi \) is the input formula.

- The conflict clause can be added to the input formula without effecting satisfiability.
- It prunes the search tree.
- This process is referred to as learning.
  - SAT solver is “learning” from its past mistakes.
- First cut \( \Rightarrow \) conflict clause \( \neg x_1 \).
- Second cut \( \Rightarrow \) conflict clause \( \neg x_2 \lor x_3 \).
Clause learning strategies

- Different cuts correspond to different conflict clause.
- Impossible to predict if a clause will be more useful than other.
- In general smaller clauses are more desirable.
  - Less storage space
  - Earlier unit propagation
- Any number of conflict clauses could be learnt.
- Many SAT solvers learn a single clause with a special property, an *asserting* clause.
Asserting clause and UIP

Definition (asserting clause)
Asserting clause is a conflict clause that contains exactly one literal from the current decision level.

Definition
(unique implication point) Unique implication point (UIP) is any vertex other than $K$ that is on all paths from the current decision level vertex to $K$.

- UIP always exists (at least the decision vertex itself)
- there may be more UIPs

Definition (first UIP)
First UIP is the UIP that is closest to $K$
Clause learning and backtracking

- Find the conflict clause containing the negation of first UIP as its single literal from current decision level.
  - asserting
- Backtracking with asserting clause
  - Backtrack to the *second highest decision level* from levels of literals in the conflict clause.
  - Equivalently (for asserting clause) to the highest decision level of its literals, excluding the UIP.
  - The newly learnt clause is unit at this decision level \( \Rightarrow \) Unit propagation is immediately triggered.
- Notes:
  - If a conflict clause contains only literals from decision level 0, then the input formula is unsatisfiable.
  - If a conflict clause contains a single literal, the backtrack level is 0.
CDCL Algorithm

1: procedure CDCL(\(\varphi\))
2: \(\alpha \leftarrow \emptyset\)
3: if \(BCP(\varphi, \alpha) = NULL\) then return FALSE
4: while TRUE do
5: \((x, v) \leftarrow SELECT(\varphi, \alpha)\)
6: if \((x, v) = NULL\) then return TRUE
7: \(\alpha \leftarrow \alpha \cup \{x \leftarrow v\}\)
8: \((result, \alpha) \leftarrow BCP(\varphi, \alpha)\)
9: while not result do
10: \((level, \varphi) \leftarrow ANALYZE-CONFLICT(\varphi, \alpha)\)
11: if level < 0 then return FALSE
12: BACKTRACK(\(\varphi, level\))
13: \((result, \alpha) \leftarrow BCP(\varphi, \alpha)\)
CDCL Algorithm

- **BCP.** Performs unit propagation iteratively. Returns updated assignment and conflict indicator.
- **SELECT.** Selects unassigned variable and its polarity. Returns NULL if all variables are assigned.
- **ANALYZE-CONFLICT.** Determines backtrack level and extends $\varphi$ with learned clause(s).
- **BACKTRACK.** Backtracks to the given decision level. Erases all assignments made after this level.
CDCL Algorithm

Notes

- Algorithm always terminates.
  - Idea of a proof: The algorithm never enters the same decision level with the same partial assignment twice.
Algorithm always terminates.
  - Idea of a proof: The algorithm never enters the same decision level with the same partial assignment twice.

Learned clauses can be pruned.
  - Too many learned clauses slow down the solver too much.
  - Many of them are not used more than once.
CDCL Algorithm

Notes

- Algorithm always terminates.
  - Idea of a proof: The algorithm never enters the same decision level with the same partial assignment twice.
- Learned clauses can be pruned.
  - Too many learned clauses slow down the solver too much.
  - Many of them are not used more than once.
- Implication graph can be represented implicitly (decision trail with decision levels and polarity for variables, map of literals to antecedents).
Computing asserting clause

1: procedure ANALYZE-CONFLICT(ϕ, α)
2: if decision-level = 0 then return (−1, ϕ)
3: c ← unsatisfied clause w.r.t. α
4: while c is not asserting do
5: l ← most recently assigned literal in c
6: c ← RESOLVE(c, Antecedent(l), Var(l))
7: ϕ ← ϕ ∪ c
8: return (LEVEL(c), ϕ)
Computing asserting clause

1: procedure ANALYZE-CONFLICT(ϕ, α)
2: if decision-level = 0 then return (−1, ϕ)
3: c ← unsatisfied clause w.r.t. α
4: while c is not asserting do
5: l ← most recently assigned literal in c
6: c ← RESOLVE(c, Antecedent(l), Var(l))
7: ϕ ← ϕ ∪ c
8: return (LEVEL(c), ϕ)

- RESOLVE(c₁, c₂, v) returns resolvent of c₁ and c₂ where x is the resolution variable.
- LEVEL(c) returns the second highest decision level of literals in c. (Returns 0 is c has only one literal.)
Decision heuristic

Strategy by which the variables and the value given to them are chosen

- Variable State Independent Decaying Sum (VSIDS)
  - SAT solver \textsc{Chaff}, 2001
  - \textit{conflict driven} heuristic, tries to solve recent conflicts first
  - Every literal has a score (based on how many occurrences there are). Score of literals in newly learned clauses increases. The score is periodically divided by $d > 1$. 

- Berkmin
  - member of a family of clause-based heuristics
  - SAT solver \textsc{BerkMin}, 2002
  - Score for every variable (divided) and for every literal (not divided). Stack of conflict clauses. Pick a variable from unresolved clause with highest score. Assign a polarity based on literal score.

- Other heuristics: Jeroslow-Wang, Dynamic Largest Individual Sum, Clause-Move-To-Front, \ldots
Decision heuristic

Strategy by which the variables and the value given to them are chosen

- Variable State Independent Decaying Sum (VSIDS)
  - SAT solver CHAFF, 2001
  - *conflict driven* heuristic, tries to solve recent conflicts first
  - Every literal has a score (based on how many occurrences there are). Score of literals in newly learned clauses increases. The score is periodically divided by $d > 1$.

- Berkmin
  - member of a family of *clause-based* heuristics
  - SAT solver BERKMIN, 2002
  - Score for every variable (divided) and for every literal (not divided). Stack of conflict clauses. Pick a variable from unresolved clause with highest score. Assign a polarity based on literal score.
Decision heuristic

Strategy by which the variables and the value given to them are chosen

- **Variable State Independent Decaying Sum (VSIDS)**
  - SAT solver **CHAFF**, 2001
  - *conflict driven* heuristic, tries to solve recent conflicts first
  - Every literal has a score (based on how many occurrences there are). Score of literals in newly learned clauses increases. The score is periodically divided by \( d > 1 \).

- **Berkmin**
  - member of a family of *clause-based* heuristics
  - SAT solver **BerkMin**, 2002
  - Score for every variable (divided) and for every literal (not divided). Stack of conflict clauses. Pick a variable from unresolved clause with highest score. Assign a polarity based on literal score.

- **Other heuristics**: Jeroslow-Wang, Dynamic Largest Individual Sum, Clause-Move-To-Front, ...
Watched literals

- Data structure for efficient unit propagation and backtracking.
- Goal: to quickly determine *unit clauses*, to avoid checking all the clauses

**Two watched literals**

- Two unassigned literals are marked in every unresolved clause.
- Watched occurrences are known for every variable.

\[
\begin{align*}
\text{c}_1 &= (\downarrow x_1 \lor x_2 \lor \downarrow x_3) \\
\text{c}_2 &= (\downarrow \neg x_1 \lor x_2 \lor \downarrow \neg x_4) \\
\text{c}_3 &= (\downarrow \neg x_1 \lor x_3 \lor \downarrow \neg x_4)
\end{align*}
\]

Watched occurrences
\[
\begin{align*}
x_1: \{c_1 : x_1, c_2 : \neg x_1, c_3 : x_1\} \\
x_2: \{} \\
x_3: \{c_1 : x_3\} \\
x_4: \{c_2 : \neg x_4, c_3 : \neg x_4\}
\end{align*}
\]
Watched literals

Easy updates

- Assign \( x_1 = \text{False} \)

\[
\begin{align*}
    c_1 &= (x_1 \lor x_2 \lor x_3) \\
    c_2 &= (\neg x_1 \lor x_2 \lor \neg x_4) \\
    c_3 &= (\neg x_1 \lor x_3 \lor \neg x_4)
\end{align*}
\]

- Only \( c_1 \) needs to be examined \( \Rightarrow \) find new unassigned literal and update occurrences

\[
\begin{align*}
    c_1 &= (x_1 \lor x_2 \lor x_3) \\
    c_2 &= (\neg x_1 \lor x_2 \lor \neg x_4) \\
    c_3 &= (\neg x_1 \lor x_3 \lor \neg x_4)
\end{align*}
\]

Watched occurrences

- \( x_1: \{c_1 : x_1, c_2 : \neg x_1, c_3 : x_1\} \)
- \( x_2: \{\} \)
- \( x_3: \{c_1 : x_3\} \)
- \( x_4: \{c_2 : \neg x_4, c_3 : \neg x_4\} \)
Watched literals

Easy updates

- Assign $x_3 = False$
- Only $c_1$ is examined!
- New unassigned literal cannot be found $\Rightarrow c_1$ is unit

$$c_1 = (x_1 \lor x_2 \lor x_3)$$
$$c_2 = (\neg x_1 \lor x_2 \lor \neg x_4)$$
$$c_3 = (\neg x_1 \lor x_3 \lor \neg x_4)$$

- $x_2 = True$ is derived.
- No clause needs to be examined, $\neg x_2$ is not watched anywhere.

$$c_1 = (x_1 \lor x_2 \lor x_3)$$
$$c_2 = (\neg x_1 \lor x_2 \lor \neg x_4)$$
$$c_3 = (\neg x_1 \lor x_3 \lor \neg x_4)$$

Watched occurrences

$x_1$: { $c_2 : \neg x_1$, $c_3 : x_1$ }
$x_2$: { $c_1 : x_2$ }
$x_3$: { $c_1 : x_3$ }
$x_4$: { $c_2 : \neg x_4$, $c_3 : \neg x_4$ }

$x_1$: { $c_2 : \neg x_1$, $c_3 : x_1$ }
$x_2$: { $c_1 : x_2$ }
$x_3$: { $c_1 : x_3$ }
$x_4$: { $c_2 : \neg x_4$, $c_3 : \neg x_4$ }
Watched literals
Easy backtracking
  ▶ Do nothing
Watched literals

Easy backtracking

- Do nothing

- Simply remove assignment, the invariants of the data structure are preserved
Watched literals

Easy backtracking

- Do nothing

- Simply remove assignment, the invariants of the data structure are preserved

- \( \alpha = \{x_1 = False, x_3 = False, x_2 = True\} \)

\[
\begin{align*}
c_1 &= (x_1 \lor x_2 \lor x_3) \\
c_2 &= (\neg x_1 \lor x_2 \lor \neg x_4) \\
c_3 &= (\neg x_1 \lor x_3 \lor \neg x_4)
\end{align*}
\]

Watched occurrences

\[
\begin{align*}
x_1: \{c_2 : \neg x_1, c_3 : x_1\} \\
x_2: \{c_1 : x_2\} \\
x_3: \{c_1 : x_3\} \\
x_4: \{c_2 : \neg x_4, c_3 : \neg x_4\}
\end{align*}
\]
Watched literals

Easy backtracking

- **Do nothing**

- Simply remove assignment, the invariants of the data structure are preserved

\[ \alpha = \{ x_1 = \text{False}, x_3 = \text{False}, x_2 = \text{True} \} \]

\[
\begin{align*}
c_1 &= (x_1 \lor \downarrow x_2 \lor \downarrow x_3) \\
c_2 &= (\downarrow \neg x_1 \lor x_2 \lor \downarrow \neg x_4) \\
c_3 &= (\downarrow \neg x_1 \lor x_3 \lor \downarrow \neg x_4)
\end{align*}
\]

\[ \alpha = \{ \} \]

\[
\begin{align*}
c_1 &= (x_1 \lor \downarrow x_2 \lor \downarrow x_3) \\
c_2 &= (\downarrow \neg x_1 \lor x_2 \lor \downarrow \neg x_4) \\
c_3 &= (\downarrow \neg x_1 \lor x_3 \lor \downarrow \neg x_4)
\end{align*}
\]

**Watched occurrences**

\[
\begin{align*}
x_1: \{ c_2: \neg x_1, c_3: x_1 \} \\
x_2: \{ c_1: x_2 \} \\
x_3: \{ c_1: x_3 \} \\
x_4: \{ c_2: \neg x_4, c_3: \neg x_4 \}
\end{align*}
\]
Summary

- DPLL
  - unit propagation
- CDCL
  - implication graph, asserting clause, clause learning
- decision heuristics
- two watched literals