Automated Program Verification and Testing
15414/15614 Fall 2016
Lecture 3:
Practical SAT Solving

Matt Fredrikson
mfredrik@cs.cmu.edu

October 17, 2016
**Goal**: Give meaning to propositional formulas

Assign Boolean truth values to (formula, interpretation) pairs

\[ F + \text{Interpretation } I = \text{Truth Value} \ (true, \ false) \]

Note: we often abbreviate *true* by 1 and *false* by 0

**Interpretation**

An interpretation \( I \) for propositional formula \( F \) maps every propositional variable appearing in \( F \) to a truth value, i.e.:

\[ I = \{ P \mapsto true, Q \mapsto false, R \mapsto false, \ldots \} \]
Satisfying Interpretation

$I$ is a *satisfying interpretation* of a propositional formula $F$ if $F$ is *true* under $I$. We denote this with the notation:

$$I \models F$$

Falsifying Interpretation

$I$ is a *falsifying interpretation* of a propositional formula $F$ if $F$ is *false* under $I$. We denote this with the notation:

$$I \not\models F$$
Review: Conjunctive Normal Form (CNF)

Take the form:
\[
\bigwedge_i \bigvee_j P_{ij}
\]

To convert to CNF:
1. Convert to NNF
2. Distribute \( \vee \) over \( \wedge \)

Naive approach has exponential blowup

Tseitin’s transformation: linear increase in formula size
Satisfiability Problem

SAT Problem

Given a propositional formula $F$, decide whether there exists an interpretation $I$ such that $I \models F$.

3SAT was the first established NP-Complete problem (Cook, 1971)

Most important logical problems can be reduced to SAT

- Validity
- Entailment
- Equivalence
All of the algorithms we talk about assume that formulas are in CNF

We’ll refer to a formula as a set of clauses $F = \{C_1, \ldots, C_n\}$

Likewise, clauses as sets of literals

$$(P \lor Q) \land (Q \rightarrow \neg P) \quad \{\{P, Q\}, \{-Q, \neg P\}\}$$

Some convenient notation:

- $C_i\{P \mapsto F\}$: $C_i$ with $F$ substituted for $P$
- $C_i[P]$: $P$ appears positive in $C_i$, i.e., $C_i = \{\ldots, P, \ldots\}$
- $C_i[\neg P]$: $P$ appears negated in $C_i$, i.e., $C_i = \{\ldots, \neg P, \ldots\}$
- $C_i \lor C_j$: union of $C_i$ and $C_j$, $C_i \cup C_j$
- $F_i \land F_j$: union of $F_i$ and $F_j$, $F_i \cup F_j$
Resolution

Single inference rule:

\[
\frac{C_1[P] \quad C_2[\neg P]}{C_1\{P \leftrightarrow \bot\} \lor C_2\{\neg P \leftrightarrow \bot\}}
\]

Given two clauses that share variable \( P \) but disagree on its value:

1. If \( P \) is \textit{true}, then some other literal in \( C_2 \) must be true
2. If \( P \) is \textit{false}, then some other literal in \( C_1 \) must be true
3. Therefore, resolve on \( P \) in both clauses by removing it
4. \( C_1\{P \leftrightarrow \bot\} \lor C_2\{\neg P \leftrightarrow \bot\} \) is called the \textit{resolvent}

If \( C_1\{P \leftrightarrow \bot\} \lor C_2\{\neg P \leftrightarrow \bot\} = \bot \lor \bot = \bot \):

1. Then \( C_1 \land C_2 \) is unsatisfiable
2. Any CNF containing \( \{C_1, C_2\} \) is unsatisfiable
function Resolution(F)
    \( F' = \emptyset \)
    repeat
        \( F \leftarrow F \cup F' \)
        for all \( C_i, C_j \in F \) do
            \( C' = \text{Resolve}(C_i, C_j) \)
            if \( C' = \bot \) then
                return unsat
            end if
            \( F' \leftarrow F' \cup \{C'\} \)
        end for
        until \( F' \subseteq F \)
    return sat
end function

1. For each round, compute all possible resolvents
2. \( F' \) holds set of all resolvents
3. At each round, update \( F \) to contain past resolvents
4. Repeat resolution on updated \( F \)
5. Terminate when:
   - Encounter \( \bot \) resolvent
   - Don’t find anything new to add to \( F \)
Resolution: Example

\[(P \lor Q) \land (P \rightarrow R) \land (Q \rightarrow R) \land \neg R\]

\[
\begin{align*}
(C_1) &\quad (P \lor Q) \\
(C_2) &\quad (\neg P \lor R) \\
(C_3) &\quad (\neg Q \lor R) \\
(C_4) &\quad (\neg R)
\end{align*}
\]

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>$P \lor Q$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>$\neg P \lor R$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>$\neg Q \lor R$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>$\neg R$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>$Q \lor R$</td>
<td>1 &amp; 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>$\neg P$</td>
<td>2 &amp; 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>$\neg Q$</td>
<td>3 &amp; 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$R$</td>
<td>3 &amp; 5</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$Q$</td>
<td>4 &amp; 5</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$P$</td>
<td>1 &amp; 8</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\bot$</td>
<td>4 &amp; 9</td>
</tr>
</tbody>
</table>

Matt Fredrikson

SAT Solving
Why is resolution particularly bad for large problems?

**Hint:** What does this technique build along the way?

Space complexity: \( \exp(O(N)) \)

Example: \( m \) pigeons won’t go into \( n \) holes when \( m > n \)

- \( p_{i,j} \): pigeon \( i \) goes in hole \( j \)
- \( p_{i,1} \lor p_{i,2} \lor \cdots \lor p_{i,n} \): every pigeon \( i \) gets a hole
- \( \neg p_{i,j} \lor \neg p_{i',j} \): no hole \( j \) gets two pigeons \( i \neq i' \)
- Resolution proof size: \( \exp(\Omega(N)) \)
Partial Interpretations

Starting from an empty interpretation:

- Extend for each variable
- No direct modifications to literals in formula

More flexibility in implementation strategy (more on this later)

If $I$ is a *partial* interpretation, literals $\ell$ can be *true*, *false*, *undef*:

- *true* (satisfied): $I \models \ell$
- *false* (conflicting): $I \not\models \ell$
- *undef*: $\text{var}(\ell) \not\in I$

Given a clause $C$ and interpretation $I$:

- $C$ is *true* under $I$ iff $I \models C$
- $C$ is *false* under $I$ iff $I \not\models C$
- $C$ is *unit* under $I$ iff $C = C' \lor \ell$, $I \not\models C$, $\ell$ is *undef*
- Otherwise it is *undef*
Example

$I = \{ P_1 \leftrightarrow 1, P_2 \leftrightarrow 0, P_4 \leftrightarrow 1 \}$

\[
\begin{align*}
P_1 \lor P_3 \lor \neg P_4 & \quad \text{satisfied} \\
\neg P_1 \lor P_2 & \quad \text{conflicting} \\
\neg P_1 \lor \neg P_4 \lor P_3 & \quad \text{unit} \\
\neg P_1 \lor P_3 \lor P_5 & \quad \text{undef}
\end{align*}
\]
Transition system is a binary relation over **states**

Transitions are induced by *guarded* transition rules

---

**Procedure State**

<table>
<thead>
<tr>
<th>The possible states are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ sat</td>
</tr>
<tr>
<td>▶ unsat</td>
</tr>
<tr>
<td>▶ ([I]) (\parallel) (F)</td>
</tr>
</tbody>
</table>

Where \([I]\) is an *ordered* interpretation, \(F\) is a CNF.

---

Initial state: \([\emptyset]\) \(\parallel\) \(F\)

Final states: *sat*, *unsat*

Ex. intermediate states:

- \([\emptyset]\) \(\parallel\) \(F_1, C\): empty interpretation, \(F = F_1 \land C\)

- \([I_1, \overline{P}, I_2]\) \(\parallel\) \(F\): interp. assigns \(I_1\) first, then \(P \leftrightarrow 0\), then \(I_2\)
Basic Search

**Decision Rule**

\[
[I] \parallel F \leftrightarrow [I, P^\circ] \parallel F \text{ if } \begin{cases} P \text{ occurs in } F \\ P \text{ unassigned in } I \end{cases}
\]

**Backtrack Rule**

\[
[I_1, P^\circ, I_2] \parallel F \leftrightarrow [I_1, \overline{P}] \parallel F \text{ if } \begin{cases} [I_1, P, I_2] \not\models F \\ P \text{ last decision in interp.} \end{cases}
\]

**Sat Rule**

\[
[I] \parallel F \leftrightarrow \text{sat if } [I] \models F
\]

**Unsat Rule**

\[
[I] \parallel F \leftrightarrow \text{unsat if } \begin{cases} [I] \not\models F \\ \text{No decisions in } I \end{cases}
\]
Example

\[ F := \begin{align*}
C_1 &= \neg P_1 \lor P_2 \\
C_2 &= \neg P_3 \lor P_4 \\
C_3 &= \neg P_6 \lor \neg P_5 \lor \neg P_2 \\
C_4 &= \neg P_5 \lor P_6 \\
C_5 &= P_5 \lor P_7 \\
C_6 &= \neg P_1 \lor P_5 \lor P_7
\end{align*} \]

<table>
<thead>
<tr>
<th>( I )</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_2^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( P_2^o, P_4^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( P_2^o, P_4^o, P_5^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( P_2^o, P_4^o, P_5^o, P_6^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( P_2^o, P_4^o, P_5^o, \overline{P_6} )</td>
<td>Backtrack</td>
</tr>
<tr>
<td>( P_2^o, P_4^o, \overline{P_5} )</td>
<td>Backtrack</td>
</tr>
<tr>
<td>( P_2^o, P_4^o, \overline{P_5}, P_7^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( P_2^o, P_4^o, \overline{P_5}, P_7^o )</td>
<td>Sat</td>
</tr>
</tbody>
</table>
Unit Propagation

Recall *unit* clauses. For an interpretation $I$ and clause $C$,

- $I$ does not satisfy $C$,
- All but one literals in $C$ are assigned

$I$ implies an assignment for the unassigned literal

**Unit Propagation Rule**

\[
[I] \parallel F, C \lor \neg P \leftrightarrow [I, P \lor \neg \overline{P}] \parallel F, C \lor \neg P \quad \text{if} \quad \begin{cases} [I] \not\models C \\ P \text{ undefined in } I \end{cases}
\]

This is a restricted form of resolution
Example Revisited

\[ F := \begin{align*}
C_1 &= \neg P_1 \lor P_2 \\
C_2 &= \neg P_3 \lor P_4 \\
C_3 &= \neg P_6 \lor \neg P_5 \lor \neg P_2 \\
C_4 &= \neg P_5 \lor P_6 \\
C_5 &= P_5 \lor P_7 \\
C_6 &= \neg P_1 \lor P_5 \lor \neg P_7
\end{align*} \]

<table>
<thead>
<tr>
<th>( I )</th>
<th>Rule</th>
<th>( I )</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1^o )</td>
<td>Decide</td>
<td>( P_1^o, P_2, \overline{P_3} )</td>
<td>Backtrack</td>
</tr>
<tr>
<td>( P_1^o, P_2 )</td>
<td>Propagate</td>
<td>( P_1^o, P_2, \overline{P_3}, P_5^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3^o )</td>
<td>Decide</td>
<td>( P_1^o, P_2, \overline{P_3}, P_5^o, \overline{P_6} )</td>
<td>Propagate</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3^o, P_4 )</td>
<td>Propagate</td>
<td>( P_1^o, P_2, \overline{P_3}, \overline{P_5} )</td>
<td>Backtrack</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3^o, P_4, P_5^o )</td>
<td>Decide</td>
<td>( P_1^o, P_2, \overline{P_3}, \overline{P_5}, P_7 )</td>
<td>Propagate</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3^o, P_4, \overline{P_5} )</td>
<td>Propagate</td>
<td>( \overline{P_1} )</td>
<td>Backtrack</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3^o, P_4, \overline{P_5}, P_7 )</td>
<td>Propagate</td>
<td>( \overline{P_1}, P_2^o, P_3^o, P_4, \overline{P_5}, P_7 )</td>
<td>Sat</td>
</tr>
</tbody>
</table>
Example

\[ F := \begin{align*}
C_1 &= \neg P_1 \lor P_2 \\
C_2 &= \neg P_2 \lor P_3 \\
C_3 &= \neg P_3 \lor P_4 \\
C_4 &= \neg P_4 \lor P_5 \\
C_5 &= \neg P_5 \lor \neg P_1 \\
C_6 &= P_1 \lor P_2 \lor P_3 \lor P_4 \lor \neg P_5
\end{align*} \]

<table>
<thead>
<tr>
<th>I</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( P_1^o, P_2 )</td>
<td>Propagate</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3 )</td>
<td>Propagate</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3, P_4 )</td>
<td>Propagate</td>
</tr>
<tr>
<td>( P_1^o, P_2, P_3, P_4, P_5 )</td>
<td>Propagate</td>
</tr>
<tr>
<td>( \overline{P_1} )</td>
<td>Backtrack</td>
</tr>
<tr>
<td>( \overline{P_1}, P_2^o )</td>
<td>Decide</td>
</tr>
<tr>
<td>( \overline{P_1}, P_2^o, P_3 )</td>
<td>Propagate</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>(Several propagations)</td>
</tr>
<tr>
<td>( \overline{P_1}, P_2^o, P_3, P_4, P_5 )</td>
<td>Sat</td>
</tr>
</tbody>
</table>
The backtracking rule seems short-sighted
- It always jumps to the most recent decision
- It does not keep information about the conflict

Backjump Rule

\[
[I_1, P^\circ, I_2] \parallel F \leftrightarrow [I_1, \ell] \parallel F, C \quad \text{if}
\]

\[
\begin{align*}
[I_1, P^\circ, I_2] &\neq F \\
\exists C \text{ s.t. } &
\begin{cases}
F \Rightarrow (C \rightarrow \ell) \\
I_1 \models C \\
\text{var(}\ell\text{) undefined in } I_1 \\
\text{var(}\ell\text{) appears in } F
\end{cases}
\end{align*}
\]

\(C\) is called a conflict clause

Will help us prevent similar conflicts in the future
Example Revisited (again)

\[ F := \begin{align*}
C_1 &= \neg P_1 \lor P_2 \\
C_2 &= \neg P_3 \lor P_4 \\
C_3 &= \neg P_6 \lor \neg P_5 \lor \neg P_2 \\
C_4 &= \neg P_5 \lor P_6 \\
C_5 &= P_5 \lor P_7 \\
C_6 &= \neg P_1 \lor P_5 \lor \neg P_7 \\
C_7 &= \neg P_1 \lor \neg P_5
\end{align*} \]

\[ I \quad \text{Rule} \]

\begin{align*}
P_1^o &\quad \text{Decide} \\
P_1^o, P_2 &\quad \text{Propagate} \\
P_1^o, P_2, P_3^o &\quad \text{Decide} \\
P_1^o, P_2, P_3^o, P_4 &\quad \text{Propagate} \\
P_1^o, P_2, P_3^o, P_4, P_5^o &\quad \text{Decide} \\
P_1^o, P_2, P_3^o, P_4, P_5^o, \overline{P_6} &\quad \text{Propagate} \\
P_1^o, P_2, \overline{P_5} &\quad \text{Backjump, } P_1 \rightarrow \neg P_5 \\
P_1^o, P_2, \overline{P_5}, P_7 &\quad \text{Propagate} \\
\overline{P_1} &\quad \text{Backjump, } \text{true} \rightarrow \neg P_1 \\
\ldots
\end{align*}
Finding a Conflict Clause

The Backjump rule requires a conflict clause

To find one, we construct an *implication graph* $G = (V, E)$

- $V$ has a node for each decision literal in $I$, labeled with the literal’s value and its decision level.
- For each clause $C = \ell_1 \lor \cdots \lor \ell_n \lor \ell$ where $\ell_1, \ldots, \ell_n$ are assigned false,
  1. Add a node for $\ell$ with the decision level in which it entered $I$
  2. Add edges $(\ell_i, \ell)$ for $1 \leq i \leq n$ to $E$
- Add a special *conflict node* $\Lambda$. For any conflict variable with nodes labeled $P$ and $\neg P$, add edges from these nodes to $\Lambda$ in $E$.
- Label each edge with the clause that caused the implication.

The implication graph contains sufficient information to generate a conflict clause
Implication Graph

\[ F := \begin{align*}
C_1 &= \neg P_1 \lor P_2 \\
C_2 &= \neg P_3 \lor P_4 \\
C_3 &= \neg P_6 \lor \neg P_5 \lor \neg P_2 \\
C_4 &= \neg P_5 \lor P_6 \\
C_5 &= P_5 \lor P_7 \\
C_6 &= \neg P_1 \lor P_5 \lor \neg P_7
\end{align*} \]

\[ I = [P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}] \]
Conflict Graph

Implication graph where:

- Exactly one conflict variable
- All nodes have a path to $\Lambda$

$$
\begin{align*}
C_1 &= \neg P_1 \lor P_2 \\
C_2 &= \neg P_3 \lor P_4 \\
C_3 &= \neg P_6 \lor \neg P_5 \lor \neg P_2 \\
C_4 &= \neg P_5 \lor P_6 \\
C_5 &= P_5 \lor P_7 \\
C_6 &= \neg P_1 \lor P_5 \lor \neg P_7
\end{align*}
$$

$$
I = [P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}]
$$
Generating Conflict Clauses

Consider a conflict graph $G$

1. Pick a cut in $G$ such that:
   - All of the decision nodes are on one side (the “reason” side)
   - At least one conflict literal is on the other (the “conflict” side)

2. Pick all nodes $K$ on the reason side with an edge crossing the cut

3. The nodes in $K$ form a cause of the conflict

4. The negations of the corresponding literal form the conflict clause
Generating Conflict Clauses

\[ C_1 = \neg P_1 \lor P_2 \quad C_2 = \neg P_3 \lor P_4 \]
\[ C_3 = \neg P_6 \lor \neg P_5 \lor \neg P_2 \]
\[ C_4 = \neg P_5 \lor P_6 \quad C_5 = P_5 \lor P_7 \]
\[ C_6 = \neg P_1 \lor P_5 \lor \neg P_7 \]

\[ I = [P_1^O, P_2, P_3^O, P_4, P_5^O, \overline{P_6}] \]

Conflict clause: \(\neg P_1 \lor \neg P_5\)
Generating Conflict Clauses

\[ C_1 = \neg P_1 \lor P_2 \quad C_2 = \neg P_2 \lor P_3 \]
\[ C_3 = \neg P_3 \lor P_4 \quad C_4 = \neg P_4 \lor P_5 \]
\[ C_5 = \neg P_5 \lor \neg P_1 \]
\[ C_6 = P_1 \lor P_2 \lor P_3 \lor P_4 \lor \neg P_5 \]

\[ I = [P_1^c, P_2, P_3, P_4, P_5] \]

Conflict clause: \( P_1 \rightarrow \neg P_2 \)

Any others?

Does order matter?
Generating Conflict Clauses

This corresponds to resolution:

1. Let \( C \) be the conflicted clause
2. Pick most recently implied literal in conflict graph \( G \)
3. Let \( C' \) be the clause that implied it
4. Let \( C \leftarrow \text{resolve}(C, C') \)
5. Repeat step 2 while applicable

\[
C_1 = \neg P_1 \lor P_2 \quad C_2 = \neg P_3 \lor P_4 \\
C_3 = \neg P_6 \lor \neg P_5 \lor \neg P_2 \\
C_4 = \neg P_5 \lor P_6 \quad C_5 = P_5 \lor P_7 \\
C_6 = \neg P_1 \lor P_5 \lor \neg P_7
\]

\[
I = [P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P}_6]
\]

1. \( C = \neg P_5 \lor P_6 \)
2. Pick \( \overline{P}_6 \)
3. \( C' = \neg P_6 \lor \neg P_5 \lor \neg P_2 \)
4. \( C = \neg P_5 \lor \neg P_2 \)
5. Pick \( P_2 \)
6. \( C' = \neg P_1 \lor P_2 \)
7. \( C = \neg P_1 \lor \neg P_5 \)
Generating Conflict Clauses

The textbook doesn’t cover this at all

For more information, see:


▶ *Decision Procedures* by Kroening and Strichman. Download a copy from the library by visiting: [http://vufind.library.cmu.edu/vufind/Record/1607216](http://vufind.library.cmu.edu/vufind/Record/1607216)
DPLL and CDCL

Original DPLL used:

Decide, Sat/Unsat, Propagate, Backtrack

Modern DPLL replaces:

Backtrack with Backjump

These are called Conflict Driven Clause Learning (CDCL) solvers

In addition, most use:

- “Forgetting”: periodically forget learned clauses
- Restart: reset interpretation, but keep learned clauses

```python
while(1) {
    while(exists_unit(I, F))
        I, F = propagate(I, F);
        I, F = decide(I, F);
    if(conflict(I, F)) {
        if(has_decision(I))
            I, F = backjump(I, F);
        else
            return unsat;
    } else if(sat(I, F))
        return sat;
}
```
Correctness of DPLL

Soundness
For every execution starting with $[\emptyset] \parallel F$ and ending with $[I] \parallel sat$ (resp. $[I] \parallel unsat$), $F$ is satisfiable (resp. unsatisfiable).

Completeness
If $F$ is satisfiable (resp. unsatisfiable), then every execution starting with $[\emptyset] \parallel F$ ends with $[I] \parallel sat$ (resp. $[I] \parallel unsat$).

Note: Termination not obvious with Backjump. Define a metric that decreases:
- When adding a decision level (Decide)
- When adding literal to the current decision level (Propagate)
- When adding literal to previous decision level (Backjump)
Practical Considerations

Conflict-Driven Clause Learning (CDCL) made large-scale SAT practical

- GRASP solver, 1996
- From hundreds and low-thousands to thousands and millions of variables
- Focus shifted towards better heuristics, implementation

Several considerations proved effective:

- Make resolution more efficient: keep # memory accesses per iteration low
- Simple, low-overhead decision guidance
- Strategies for forgetting learned clauses
**Watch Pointers**

**Idea:** Watch two unassigned literals in each non-satisfied clause. Ignore the rest.

Maintain two lists for each variable $P$
- The first, $L_P$, contains watching clauses with $P$
- The second, $L_{\overline{P}}$, contains watching clauses with $\overline{P}$

Each time an assignment to is made to $P$:
1. For clauses in $L_{\overline{P}}$, find another literal in the clause to watch
2. If (1) is not possible, the clause is unit

**Advantages:**
1. When $P$ assigned, only examine clauses in the appropriate list
2. No overhead when backtracking
Dynamic Largest Individual Sum (DLIS)

Decision heuristic: choose variable that satisfies the most clauses

How do we implement this?
- Maintain sat counters for every variable
- When clauses are satisfied, update counters
- Must touch every clause containing literal set to 1
- Need to reverse process when backtracking

More overhead than unit propagation...

Probably not worth it
Rank variables by literal count in the initial database
  ▶ Only increment when clauses are learned
  ▶ Periodically divide all counts by 2

Main idea: bias towards literals from recent conflicts
  ▶ Conflict adds 1 to each literal in conflict clause
  ▶ More time passed $\rightarrow$ more divisions by 2
  ▶ Effectively solves conflicts before moving onto new clauses

Use heap structure to find unassigned variable with the highest ranking
Other Approaches

There are other good SAT-solving approaches

Randomized approaches (GSAT, WSAT)
  - Hill-climbing, local search algorithms
  - State: full interpretation, Cost: # non-satisfied clauses
  - Move: flip one assignment

Binary decision diagrams
  - Efficiently represent formula as a DAG
  - Manipulate formula by changing graph structure

Stalmarck’s algorithm
  - Breadth-first search: try both branches at once
  - Also branch on variable relationships
Install Dafny on your machine

See the **Assignments** section on course webpage for a guide