Abstract DPLL

We now return to DPLL. To facilitate a deeper look at DPLL, we use a high-level framework called *Abstract DPLL*. 

Abstract DPLL uses states and transitions to model the progress of the algorithm. Most states are of the form $M || F$, where $M$ is a sequence of annotated literals denoting a partial truth assignment, and $F$ is the CNF formula being checked, represented as a set of clauses. The initial state is $\emptyset || F$, where $F$ is to be checked for satisfiability. Transitions between states are defined by a set of conditional transition rules.
Abstract DPLL

We now return to DPLL. To facilitate a deeper look at DPLL, we use a high-level framework called *Abstract DPLL*.

- Abstract DPLL uses *states* and *transitions* to model the progress of the algorithm.
Abstract DPLL

We now return to DPLL. To facilitate a deeper look at DPLL, we use a high-level framework called Abstract DPLL.

- Abstract DPLL uses states and transitions to model the progress of the algorithm.
- Most states are of the form $M \parallel F$, where
  - $M$ is a sequence of annotated literals denoting a partial truth assignment, and
  - $F$ is the CNF formula being checked, represented as a set of clauses.
Abstract DPLL

We now return to DPLL. To facilitate a deeper look at DPLL, we use a high-level framework called *Abstract DPLL*.

- Abstract DPLL uses *states* and *transitions* to model the progress of the algorithm.
- Most states are of the form $M \parallel F$, where
  - $M$ is a *sequence of* annotated *literals* denoting a partial truth assignment, and
  - $F$ is the CNF formula being checked, represented as a *set of clauses*.
- The *initial state* is $\emptyset \parallel F$, where $F$ is to be checked for satisfiability.
Abstract DPLL

We now return to DPLL. To facilitate a deeper look at DPLL, we use a high-level framework called Abstract DPLL.

- Abstract DPLL uses states and transitions to model the progress of the algorithm.
- Most states are of the form $M \parallel F$, where
  - $M$ is a sequence of annotated literals denoting a partial truth assignment, and
  - $F$ is the CNF formula being checked, represented as a set of clauses.
- The initial state is $\emptyset \parallel F$, where $F$ is to be checked for satisfiability.
- Transitions between states are defined by a set of conditional transition rules.
Abstract DPLL

The *final state* is either:

- a special fail state: *fail*, if $F$ is unsatisfiable, or
- $M \models G$, where $G$ is a CNF formula equisatisfiable with the original formula $F$, and $M$ satisfies $G$

We write $M \models C$ to mean that for every truth assignment $v$, $v(M) = True$ implies $v(C) = True$. 
Abstract DPLL Rules

UnitProp:

\[ M \parallel F, C \lor l \implies M \parallel F, C \lor l \text{ if } \begin{cases} 
M \models \neg C \\
\text{l is undefined in } M 
\end{cases} \]
Abstract DPLL Rules

UnitProp:

\[ M \parallel F, C \lor l \implies M \downarrow l \parallel F, C \lor l \]

if \[ \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases} \]

PureLiteral:

\[ M \parallel F \implies M \downarrow l \parallel F \]

if \[ \begin{cases} l \text{ occurs in some clause of } F \\ \neg l \text{ occurs in no clause of } F \\ l \text{ is undefined in } M \end{cases} \]

Decide:

\[ M \parallel F \implies M \downarrow l \parallel F \]

if \[ \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases} \]

Backtrack:

\[ M \downarrow l \parallel N \parallel F, C \implies M \neg l \parallel F, C \]

if \[ \begin{cases} M \downarrow l \parallel N \downarrow C \\ N \text{ contains no decision literals} \end{cases} \]

Fail:

\[ M \parallel F, C \implies \text{fail} \]

if \[ \begin{cases} M \downarrow C \downarrow \text{ contains no decision literals} \end{cases} \]
Abstract DPLL Rules

UnitProp:

\[ M \parallel F, C \lor l \implies M \parallel F, C \lor I \]

if \( M \models \neg C \) \land \( l \) is undefined in \( M \)

PureLiteral:

\[ M \parallel F \implies M \parallel F \]

if \( l \) occurs in some clause of \( F \)

\( \neg l \) occurs in no clause of \( F \)

\( l \) is undefined in \( M \)

Decide:

\[ M \parallel F \implies M \parallel F \]

if \( l \) or \( \neg l \) occurs in a clause of \( F \)

\( l \) is undefined in \( M \)
Abstract DPLL Rules

UnitProp:

\[ M \parallel F, C \lor l \implies M l \parallel F, C \lor l \]

if \( M \models \neg C \)

\( l \) is undefined in \( M \)

PureLiteral:

\[ M \parallel F \implies M l \parallel F \]

if \( l \) occurs in some clause of \( F \)

\( \neg l \) occurs in no clause of \( F \)

\( l \) is undefined in \( M \)

Decide:

\[ M \parallel F \implies M l^d \parallel F \]

if \( l \) or \( \neg l \) occurs in a clause of \( F \)

\( l \) is undefined in \( M \)

Backtrack:

\[ M l^d N \parallel F, C \implies M \neg l \parallel F, C \]

if \( M l^d N \models \neg C \)

\( N \) contains no decision literals
Abstract DPLL Rules

UnitProp:

\[ M \parallel F, C \lor l \implies M \parallel F, C \lor l \quad \text{if} \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases} \]

PureLiteral:

\[ M \parallel F \implies M \parallel F \quad \text{if} \begin{cases} l \text{ occurs in some clause of } F \\ \neg l \text{ occurs in no clause of } F \\ l \text{ is undefined in } M \end{cases} \]

Decide:

\[ M \parallel F \implies M \parallel F \quad \text{if} \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases} \]

Backtrack:

\[ M \parallel N \parallel F, C \implies M \parallel \neg l \parallel F, C \quad \text{if} \begin{cases} M \parallel N \models \neg C \\ N \text{ contains no decision literals} \end{cases} \]

Fail:

\[ M \parallel F, C \implies \text{fail} \quad \text{if} \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases} \]
Example

\[ \emptyset \parallel 1 \lor \overline{2}, \quad \overline{1} \lor \overline{2}, \quad 2 \lor 3, \quad \overline{3} \lor 2, \quad 1 \lor 4 \]

Result: Unsatisfiable
Example

\[
\emptyset \ || \ 1 \lor \bar{2}, \ 1 \lor \bar{2}, \ 2 \lor 3, \ 3 \lor 2, \ 1 \lor 4 \quad \implies \quad \text{(PureLiteral)}
\]

\[
4 \ || \ 1 \lor \bar{2}, \ 1 \lor \bar{2}, \ 2 \lor 3, \ 3 \lor 2, \ 1 \lor 4 \quad \implies \quad \text{Backtrack}
\]

Result: Unsatisfiable
Example

∅ || 1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4  \Rightarrow (PureLiteral)

4 || 1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4  \Rightarrow (Decide)

4 1d || 1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4

Result: Unsatisfiable
Example

<table>
<thead>
<tr>
<th></th>
<th>1∨2, 1∨2, 2∨3, 3∨2, 1∨4</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>(PureLiteral)</td>
</tr>
<tr>
<td>4</td>
<td>1∨2, 1∨2, 2∨3, 3∨2, 1∨4</td>
</tr>
<tr>
<td>4 1^d</td>
<td>(Decide)</td>
</tr>
<tr>
<td>4 1^d 2</td>
<td>1∨2, 1∨2, 2∨3, 3∨2, 1∨4</td>
</tr>
<tr>
<td>4 1^d 2</td>
<td>(UnitProp)</td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th></th>
<th>1 ∨ 2, 1 2, 2 ∨ 3, 2 3, 1 ∨ 4</th>
<th>⇒  (PureLiteral)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1 ∨ 2, 1 2, 2 ∨ 3, 2 3, 1 ∨ 4</td>
<td>⇒  (Decide)</td>
</tr>
<tr>
<td>4 1^d</td>
<td>1 ∨ 2, 1 2, 2 ∨ 3, 2 3, 1 ∨ 4</td>
<td>⇒  (UnitProp)</td>
</tr>
<tr>
<td>4 1^d 2</td>
<td>1 ∨ 2, 1 2, 2 ∨ 3, 2 3, 1 ∨ 4</td>
<td>⇒  (UnitProp)</td>
</tr>
<tr>
<td>4 1^d 2 3</td>
<td>1 ∨ 2, 1 2, 2 ∨ 3, 2 3, 1 ∨ 4</td>
<td></td>
</tr>
</tbody>
</table>

Result: Unsatisfiable
Example

\[
\begin{array}{c|ccccc}
\emptyset & 1 \lor \bar{2}, & \bar{1} \lor \bar{2}, & 2 \lor 3, & \bar{3} \lor 2, & 1 \lor 4 \\
4 & 1 \lor \bar{2}, & \bar{1} \lor \bar{2}, & 2 \lor 3, & \bar{3} \lor 2, & 1 \lor 4 \\
4 1^d & 1 \lor \bar{2}, & \bar{1} \lor \bar{2}, & 2 \lor 3, & \bar{3} \lor 2, & 1 \lor 4 \\
4 1^d \bar{2} & 1 \lor \bar{2}, & \bar{1} \lor \bar{2}, & 2 \lor 3, & \bar{3} \lor 2, & 1 \lor 4 \\
4 1^d \bar{2} 3 & 1 \lor \bar{2}, & \bar{1} \lor \bar{2}, & 2 \lor 3, & \bar{3} \lor 2, & 1 \lor 4 \\
4 \bar{1} & 1 \lor \bar{2}, & \bar{1} \lor \bar{2}, & 2 \lor 3, & \bar{3} \lor 2, & 1 \lor 4 \\
\end{array}
\]

\[
= \Rightarrow (PureLiteral) \\
= \Rightarrow (Decide) \\
= \Rightarrow (UnitProp) \\
= \Rightarrow (UnitProp) \\
= \Rightarrow (Backtrack) 
\]
Example

<table>
<thead>
<tr>
<th></th>
<th>1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4</th>
<th>(\Rightarrow) (PureLiteral)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4</td>
<td>(\Rightarrow) (Decide)</td>
</tr>
<tr>
<td>4 1(^d)</td>
<td>1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4</td>
<td>(\Rightarrow) (UnitProp)</td>
</tr>
<tr>
<td>4 1(^d) 2</td>
<td>1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4</td>
<td>(\Rightarrow) (UnitProp)</td>
</tr>
<tr>
<td>4 1(^d) 2 3</td>
<td>1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4</td>
<td>(\Rightarrow) (Backtrack)</td>
</tr>
<tr>
<td>4 (\overline{1})</td>
<td>1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4</td>
<td>(\Rightarrow) (UnitProp)</td>
</tr>
<tr>
<td>4 (\overline{1}) 2 3</td>
<td>1 ∨ 2, 1 ∨ 2, 2 ∨ 3, 3 ∨ 2, 1 ∨ 4</td>
<td></td>
</tr>
</tbody>
</table>
Example

\[
\begin{array}{c|cccc}\emptyset & 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 & \Rightarrow & (\text{PureLiteral}) \\
4 & 1 \lor 2, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 & \Rightarrow & (\text{Decide}) \\
\underline{4} \quad \underline{1} \quad 2 & 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 & \Rightarrow & (\text{UnitProp}) \\
\underline{4} \quad \underline{1} \quad \underline{2} & 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 & \Rightarrow & (\text{UnitProp}) \\
\underline{4} \quad \underline{1} \quad \underline{2} \quad 3 & 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 & \Rightarrow & (\text{Backtrack}) \\
\underline{4} \quad \underline{1} & 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 & \Rightarrow & (\text{UnitProp}) \\
\underline{4} \quad \underline{1} \quad \underline{2} \quad \underline{3} & 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 & \Rightarrow & (\text{Fail}) \\
\end{array}
\]

fail

Result: Unsatisfiable
Example

\[
\begin{align*}
\emptyset & || 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 \quad \Rightarrow \quad (\text{PureLiteral}) \\
4 & || 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 \quad \Rightarrow \quad (\text{Decide}) \\
4 \, 1^d & || 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 \quad \Rightarrow \quad (\text{UnitProp}) \\
4 \, 1^d \, \overline{2} & || 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 \quad \Rightarrow \quad (\text{UnitProp}) \\
4 \, 1^d \, \overline{2} \, 3 & || 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 \quad \Rightarrow \quad (\text{Backtrack}) \\
4 \, \overline{1} & || 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 \quad \Rightarrow \quad (\text{UnitProp}) \\
4 \, \overline{1} \, \overline{2} \, 3 & || 1 \lor \overline{2}, \overline{1} \lor \overline{2}, 2 \lor 3, \overline{3} \lor 2, 1 \lor 4 \quad \Rightarrow \quad (\text{Fail}) \\
\end{align*}
\]

\textit{fail}

Result: \textit{Unsatisfiable}
Abstract DPLL: Backjumping and Learning

The basic rules can be improved by replacing the Backtrack rule with the more powerful Backjump rule and adding a Learn rule:

Backjump:

\[ M \vdash^d N \parallel F, C \implies M \vdash^d N \parallel F, C \]

if

\[ \begin{cases} M \vdash^d N \models \neg C, \text{ and there is some clause } C' \vee l' \text{ such that:} \\ F, C \models C' \vee l' \text{ and } M \models \neg C', \\ l' \text{ is undefined in } M, \text{ and} \\ l' \text{ or } \neg l' \text{ occurs in } F \text{ or in } M \vdash^d N \end{cases} \]

Learn:

\[ M \parallel F \implies M \parallel F, C \]

if

\[ \begin{cases} \text{all atoms of } C \text{ occur in } F \\ F \models C \end{cases} \]
Abstract DPLL: Backjumping and Learning

The Backjump rule is best understood by introducing the notion of *implication graph*, a directed graph associated with a state $M \parallel F$ of Abstract DPLL:

- The vertices are the *variables* in $M$
- There is an edge from $v_1$ to $v_2$ if $v_2$ was assigned a value as the result of an application of UnitProp using a clause containing $v_2$.

When we reach a state in which $M \models \neg C$ for some $C \in F$, we add an extra *conflict* vertex and edges from each of the variables in $C$ to the conflict vertex.
Abstract DPLL: Backjumping and Learning

The clause to use for backjumping (called the *conflict clause*) is obtained from the resulting graph:

- We first cut the graph along edges in such a way that it separates the conflict vertex from all of the decision vertices.
- Then, every vertex with an outgoing edge that was cut is marked.
- For each literal $l$ in $M$ whose variable is marked, $\neg l$ is added to the conflict clause.

To avoid ever having the same conflict again, we can learn the conflict clause using the *learn* rule.