Given a logic network, is there a sequence of inputs such that the network returns true as output?

Let the logical network be represented by the equation:

\[ F = (A + B)(B + \overline{C})(A + \overline{B}) \]

Davis-Putnam: Recursive Algorithm that creates a binary search tree by making assignments to the remaining variables at each stage of the algorithm

- Performs better on average than an exhaustive search
- Effectively prunes failed branches from the search

Let’s apply the Davis-Putnam Algorithm to:

\[ F = (A + B)(B + \overline{C})(A + \overline{B}) \]

Davis-Putnam Algorithm:

```plaintext
procedure split(F) {
    if F has an empty clause, then return
    if F has no clauses, then exit with current partial assignment
    select next unassigned variable, \( x \), in F
    split(F(x=\overline{0}))
    split(F(x=0))
}
```

DP Algorithm Continued

\[ F = (A + B)(B + \overline{C})(A + \overline{B}) \]

\[ (A + B)(B + \overline{C})(A + \overline{B}) \]

0 0 0

0 0

0 0

0 (SAT) 0

1 (SAT)
DP Algorithm complexity

- Unfortunately in the worst case the Davis-Putnam Algorithm is still non-polynomial wrt number of variables (exponential, i.e. NP-complete)
- Currently there is no known solution that provides a worst case performance better than exponential

DLL Algorithm – an improvement over DP

- Davis, Logemann and Loveland
- Also known as DPLL (Davis-Putnam-Logemann-Loveland) for historical reasons
- Basic framework for many modern SAT solvers

Basic DPLL Procedure – Depth First Search

Example: set of clauses in a CNF logic formula:

\[(a + b + c)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]

Basic DLL Procedure - DFS

Example: set of clauses in a CNF logic formula:

\[(a + b + c)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + b + c)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(b + c + d)\]
\[(a + b + c)\]
\[(a + b + c)\]

Decision

\[(a + b + c)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(b + c + d)\]
\[(a + b + c)\]
\[(a + b + c)\]

Decision

\[(a + b + c)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(b + c + d)\]
\[(a + b + c)\]
\[(a + b + c)\]

Decision

\[(a + b + c)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(b + c + d)\]
\[(a + b + c)\]
\[(a + b + c)\]

Decision

\[(a + b + c)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(a + c + d)\]
\[(b + c + d)\]
\[(a + b + c)\]
\[(a + b + c)\]

Decision

\[c=0\]
\[d=0\]
\[a=0\]
\[d=1\]
\[c=0\]
\[a=0\]

Conflict!
Backtrack

Forced Decision

Conflict!
Basic DLL Procedure - DFS

\[ (a + b + c) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (b + c + d) \]
\[ (b + c + d) \]
\[ (a + b + c) \]
\[ (a + b + c) \]

\( \iff \) Backtrack

\[ \begin{align*}
&\begin{array}{c}
(a + b + c) \\
(a + c + d) \\
(b + c + d) \\
(a + b + c)
\end{array}
\end{align*} \]

Basic DLL Procedure - DFS

\[ (a + b + c) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (a + b + c) \]
\[ (b + c + d) \]
\[ (a + b + c) \]
\[ (a + b + c) \]

\( \iff \) Forced Decision

\[ \begin{align*}
&\begin{array}{c}
(a + b + c) \\
(a + c + d) \\
(a + c + d) \\
(a + b + c)
\end{array}
\end{align*} \]

Basic DLL Procedure - DFS

\[ (a + b + c) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (b + c + d) \]
\[ (a + b + c) \]
\[ (a + b + c) \]

\( \iff \) Forced Decision

\[ \begin{align*}
&\begin{array}{c}
(a + b + c) \\
(a + c + d) \\
(a + c + d) \\
(a + b + c)
\end{array}
\end{align*} \]

Basic DLL Procedure - DFS

\[ (a + b + c) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (a + c + d) \]
\[ (b + c + d) \]
\[ (a + b + c) \]
\[ (a + b + c) \]

\( \iff \) Forced Decision

\[ \begin{align*}
&\begin{array}{c}
(a + b + c) \\
(a + c + d) \\
(a + c + d) \\
(a + b + c)
\end{array}
\end{align*} \]
Advantages of DLL over DP

- Eliminates the exponential memory requirements
- Exponential time is still a problem
- Limited practical applicability – largest use seen in automatic theorem proving
- Limited size of problems are allowed
  - Problem size limited by total size of clauses (1300 clauses reported)