1.5 The DPLL Procedure

Goal:
Given a propositional formula in CNF (or alternatively, a finite set N of clauses), check whether it is satisfiable (and optionally: output *one* solution, if it is satisfiable).

Assumption:
Clauses contain neither duplicated literals nor complementary literals.

Notation:
$\overline{L}$ is the complementary literal of L, i. e., $\overline{P} = \neg P$ and $\overline{\neg P} = P$.

Satisfiability of Clause Sets

$\mathcal{A} \models N$ if and only if $\mathcal{A} \models C$ for all clauses C in N.

$\mathcal{A} \models C$ if and only if $\mathcal{A} \models L$ for some literal $L \in C$.

Partial Valuations

Since we will construct satisfying valuations incrementally, we consider *partial valuations* (that is, partial mappings $\mathcal{A} : \Pi \to \{0, 1\}$).

Every partial valuation $\mathcal{A}$ corresponds to a set M of literals that does not contain complementary literals, and vice versa:

$\mathcal{A}(L)$ is true, if $L \in M$.

$\mathcal{A}(L)$ is false, if $\overline{L} \in M$.

$\mathcal{A}(L)$ is undefined, if neither $L \in M$ nor $\overline{L} \in M$.

We will use $\mathcal{A}$ and M interchangeably.

A clause is true under a partial valuation $\mathcal{A}$ (or under a set M of literals) if one of its literals is true; it is false (or *"conflicting"*) if all its literals are false; otherwise it is undefined (or *"unresolved"*).

Unit Clauses

Observation:
Let $\mathcal{A}$ be a partial valuation. If the set N contains a clause C, such that all literals but one in C are false under $\mathcal{A}$, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and makes the remaining literal L of C true.

C is called a *unit clause;* L is called a *unit literal.*

Pure Literals

One more observation:
Let $\mathcal{A}$ be a partial valuation and P a variable that is undefined under $\mathcal{A}$. If P occurs only positively (or only negatively) in the unresolved clauses in N, then the following properties are equivalent:

- there is a valuation that is a model of N and extends $\mathcal{A}$.
- there is a valuation that is a model of N and extends $\mathcal{A}$ and assigns true (false) to P.

P is called a *pure literal.*

The Davis-Putnam-Logemann-Loveland Proc.

```
boolean DPLL(literal set M, clause set N) {
    if (all clauses in N are true under M) return true;
    elsif (some clause in N is false under M) return false;
    elsif (N contains unit clause P) return DPLL(M ∪ {P}, N);
    elsif (N contains unit clause ¬P) return DPLL(M ∪ {¬P}, N);
    elsif (N contains pure literal P) return DPLL(M ∪ {P}, N);
    elsif (N contains pure literal ¬P) return DPLL(M ∪ {¬P}, N);
    else {
        let P be some undefined variable in N;
        if (DPLL(M ∪ {¬P}, N)) return true;
        else return DPLL(M ∪ {P}, N);
    }
}
```

Initially, DPLL is called with an empty literal set and the clause set N.

DPLL Iteratively

In practice, there are several changes to the procedure:

The pure literal check is often omitted (it is too expensive).

The branching variable is not chosen randomly.

The algorithm is implemented iteratively;
the backtrack stack is managed explicitly
(it may be possible and useful to backtrack more than one level).

Information is reused by learning.

Branching Heuristics

Choosing the right undefined variable to branch is important for efficiency, but the branching heuristics may be expensive itself.

State of the art: use branching heuristics that need not be recomputed too frequently.

In general: choose variables that occur frequently.

The Deduction Algorithm

For applying the unit rule, we need to know the number of literals in a clause that are not false.

Maintaining this number is expensive, however.

Better approach: *"Two watched literals"*:

In each clause, select two (currently undefined) "watched" literals.

For each variable P, keep a list of all clauses in which P is watched and a list of all clauses in which $\neg P$ is watched.

If an undefined variable is set to 0 (or to 1), check all clauses in which P (or $\neg P$) is watched and watch another literal (that is true or undefined) in this clause if possible.

Watched literal information need not be restored upon backtracking.

Conflict Analysis and Learning

Goal: Reuse information that is obtained in one branch in further branches.

Method: *Learning:*

> If a conflicting clause is found, derive a new clause from the conflict and add it to the current set of clauses.

Problem: This may produce a large number of new clauses; therefore it may become necessary to delete some of them afterwards to save space.

Backjumping

Related technique:
non-chronological backtracking ("backjumping"):

> If a conflict is independent of some earlier branch, try to skip over that backtrack level.

Restart

Runtimes of DPLL-style procedures depend extremely on the choice of branching variables.

If no solution is found within a certain time limit, it can be useful to *restart* from scratch with another choice of branchings (but learned clauses may be kept).

Formalizing DPLL with Refinements

The DPLL procedure is modelled by a transition relation $\Rightarrow_{\mathrm{DPLL}}$ on a set of states.

States:

- *fail*
- $M \parallel N$,

where M is a *list of annotated literals* and N is a set of clauses.

Annotated literal:

- L: deduced literal, due to unit propagation.
- L^{d}: decision literal (guessed literal).

Unit Propagate:

$M \parallel N \cup \{C \vee L\} \ \Rightarrow_{\text{DPLL}} \ M\ L \parallel N \cup \{C \vee L\}$

if C is false under M and L is undefined under M.

Decide:

$M \parallel N \ \Rightarrow_{\text{DPLL}} \ M\ L^{\text{d}} \parallel N$

if L is undefined under M.

Fail:

$M \parallel N \cup \{C\} \ \Rightarrow_{\text{DPLL}} \ \mathit{fail}$

if C is false under M and M contains no decision literals.

Backjump:

$M'\ L^{\text{d}}\ M'' \parallel N \ \Rightarrow_{\text{DPLL}} \ M'\ L' \parallel N$

if there is some "backjump clause" $C \vee L'$ such that
$N \models C \vee L'$,
C is false under M', and
L' is undefined under M'.

We will see later that the Backjump rule is always applicable, if the list of literals M contains at least one decision literal and some clause in N is false under M.

There are many possible backjump clauses. One candidate: $\overline{L_1} \vee \ldots \vee \overline{L_n}$, where the L_i are all the decision literals in $M\ L^{\text{d}}\ M'$. (But usually there are better choices.)

Lemma 1.9 *If we reach a state $M \parallel N$ starting from $\emptyset \parallel N$, then:*

(1) M does not contain complementary literals.

(2) Every deduced literal L in M follows from N and decision literals occurring before L in M.

Proof. By induction on the length of the derivation. □

Lemma 1.10 *Every derivation starting from $\emptyset \parallel N$ terminates.*

[The proof is relatively easy but requires techniques that will be introduced later in the lecture.]

Lemma 1.11 *Suppose that we reach a state $M \parallel N$ starting from $\emptyset \parallel N$ such that some clause $D \in N$ is false under M. Then:*

(1) If M does not contain any decision literal, then "Fail" is applicable.

(2) Otherwise, "Backjump" is applicable.

(Proof follows)

Proof. (1) Obvious.

(2) Let $L_1, \dots, L_n$ be the decision literals occurring in M (in this order). Since $M \models \neg D$, we obtain, by Lemma 1.9, $N \cup \{L_1, \dots, L_n\} \models \neg D$. Since $D \in N$, $N \models \overline{L_1} \vee \cdots \vee \overline{L_n}$. Now let $C = \overline{L_1} \vee \cdots \vee \overline{L_{n-1}}$, $L' = \overline{L_n}$, $L = L_n$, and let M' be the list of all literals of M occurring before L_n, then the condition of "Backjump" is satisfied. $\square$

Theorem 1.12 *(1) If we reach a final state $M \parallel N$ starting from $\emptyset \parallel N$, then N is satisfiable and M is a model of N.*

(2) If we reach a final state fail starting from $\emptyset \parallel N$, then N is unsatisfiable.

(Proof follows)

Proof. (1) Observe that the "Decide" rule is applicable as long as literals are undefined under M. Hence, in a final state, all literals must be defined. Furthermore, in a final state, no clause in N can be false under M, otherwise "Fail" or "Backjump" would be applicable. Hence M is a model of every clause in N.

(2) If we reach *fail*, then in the previous step we must have reached a state $M \parallel N$ such that some $C \in N$ is false under M and M contains no decision literals. By part (2) of Lemma 1.9, every literal in M follows from N. On the other hand, $C \in N$, so N must be unsatisfiable. $\square$

Getting Better Backjump Clauses

Suppose that we have reached a state $M \parallel N$ such that some clause $C \in N$ (or following from N) is false under M.

Consequently, every literal of C is the complement of some literal in M.

(1) If every literal in C is the complement of a decision literal of M. Then C is a backjump clause.

(2) Otherwise, $C = C' \vee \overline{L}$, such that L is a deduced literal.

    For every deduced literal L, there is a clause $D \vee L$, such that $N \models D \vee L$ and D is false under M.

    Consequently, $N \models D \vee C'$ and $D \vee C'$ is also false under M.