Lecture 6
The DPLL Algorithm

Print version of the lecture in *Introduction to Logic for Computer Science*

presented by Prof Hongseok Yang

These lecture notes are very minor variants of the ones made by Prof James Worrell and Dr Christoph Haase for their 'Logic and Proof' course at Oxford.

6.1

1 The DPLL Algorithm

The *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm is a procedure that combines search and deduction to decide the satisfiability of CNF formulas. This algorithm underlies most modern SAT solvers. While the basic procedure itself is 50 years old, practical DPLL-based SAT solvers only started to appear from the mid 1990s as a result of enhancements such as clause learning, non-chronological backtracking, branching heuristics, restart strategies, and lazy data structures.

The DPLL algorithm is based around backtrack search for an ordered variant of a satisfying partial assignment, called **valuation**. Here we describe a version of the algorithm with *clause learning* and *non-chronological backtracking*. At every unsuccessful leaf of the search tree (called a *conflict*) the algorithm uses resolution to compute a *conflict clause*. This clause is added to the formula whose satisfiability is being determined. One can think of conflict clauses as "caching" previous search results. Conflict clauses also determine backtracking, as will be explained below.

2 The Main Procedure

The DPLL algorithm is shown in Figure 1. Essentially the algorithm looks for a satisfying valuation of a given CNF-formula by depth-first search. At any time the *state* of the algorithm is a pair $(F, \mathcal{A})$, where F is a CNF-formula and $\mathcal{A}$ is a valuation. We say that such a state is *successful* if $\mathcal{A}$ sets some literal in each clause of F to true, that is, $\mathcal{A} \models F$.[1] *conflict state* is one in which $\mathcal{A}$ sets all literals in some clause of F to false, that is, $\mathcal{A} \models \neg F$.

We represent a CNF-formula F as a set of clauses, with each clause being a set of literals. A valuation is represented as a sequence of assignments $\langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$, where $p_1, \ldots, p_k$ are distinct propositional variables and $b_1, \ldots, b_k \in \{0, 1\}$.

We classify each assignment $p_i \mapsto b_i$ in a valuation as either a *decision assignment* or an *implied assignment*. We call the variable p_i in a decision assignment $p_i \mapsto b_i$ a *decision variable*. We sometimes use the notation $p_i \overset{C}{\mapsto} b_i$ to denote an

[1]Here $\mathcal{A}$ is an ordered variant of a partial map from propositional variables to $\{0, 1\}$. Formally, $\mathcal{A} \models F$ means that for all assignments $\mathcal{B}$ that are total maps from propositional variables, if $\mathcal{B}(p) = \mathcal{A}(p)$ for each p in the domain of $\mathcal{A}$, we have $\mathcal{B} \models F$.

```
Input: CNF formula $F$.
1. Initialise $\mathcal{A}$ to be the empty list of assignments.
2. While there is a unit clause $\{L\}$ in $F|_{\mathcal{A}}$, add assignment $L \mapsto 1$ to $\mathcal{A}$.
3. If $F|_{\mathcal{A}}$ contains no clauses then stop and output $\mathcal{A}$.
4. If $F|_{\mathcal{A}}$ contains the empty clause then apply the learning procedure to add a
   new clause $C$ to $F$. If $C$ is the empty clause then stop and output "UNSAT".
   Otherwise backtrack to the highest level at which $C$ is a unit clause. Go to
   Line 2.
5. Apply the decision strategy to determine a new decision assignment $P \mapsto b$ to
   be added to $\mathcal{A}$. Go to Line 2.
```

Figure 1: DPLL Algorithm

implied assignment arising through unit propagation (see below) on clause C. The *decision level* of an assignment $p_i \mapsto b_i$ in a given state valuation $\mathcal{A}$ is the number of decision assignments in $\mathcal{A}$ that precede $p_i \mapsto b_i$.

By $F|_{\mathcal{A}}$ we denote the set of clauses obtained by deleting from F any clause that contains a true literal under $\mathcal{A}$ and deleting from each remaining clause all literals that are false under $\mathcal{A}$. Note that $(F, \mathcal{A})$ is a conflict state if $F|_{\mathcal{A}}$ contains the empty clause $\Box$ and $(F, \mathcal{A})$ is a successful state if $F|_{\mathcal{A}}$ is the empty set of clauses.

2.1 Unit Propagation

A *unit clause* is a clause with a single literal. The while loop in Line 2 of the DPLL algorithm adds the assignment $L \mapsto 1$ to the state whenever there is a unit clause $\{L\}$ in $F|_{\mathcal{A}}$. This subroutine is called *unit propagation*.

Example 1. Consider an execution of the DPLL algorithm starting with the set of clauses $F = \{C_1, \ldots, C_5\}$, where

$$\begin{aligned}
C_1 &: \{\neg p_1, \neg p_4, p_5\} \\
C_2 &: \{\neg p_1, p_6, \neg p_5\} \\
C_3 &: \{\neg p_1, \neg p_6, p_7\} \\
C_4 &: \{\neg p_1, \neg p_7, \neg p_5\} \\
C_5 &: \{p_1, p_4, p_6\} .
\end{aligned}$$

Suppose that the current valuation is given by the following sequence of decision assignments $\mathcal{A} = \langle p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0, p_4 \mapsto 1 \rangle$. Notice that $F|_{\mathcal{A}}$ contains the unit clause $\{p_5\}$. From this, unit propagation generates the further sequence of implied assignments $\langle p_5 \overset{C_1}{\mapsto} 1, p_6 \overset{C_2}{\mapsto} 1, p_7 \overset{C_3}{\mapsto} 1 \rangle$. This leads to a conflict, with clause C_4 being made false.

2.2 Conflict Analysis

On termination of unit propagation, if the procedure is neither in a conflict state nor a successful state, then another decision assignment is made (Line 5). If, on the other hand, a conflict has been reached then a *learned clause* is added to the current state (Line 4). Intuitively the learned clause summarises the reason for the conflict.

We will explain in detail one method of clause learning in the next section. For now we just note the properties that a learned clause is required to have under our learning scheme.

If the state of the algorithm is $(F, \mathcal{A})$ then we say that a clause C is a *conflict clause* if all literals in C are made false by $\mathcal{A}$. If $(F, \mathcal{A})$ is a conflict state and clause C is learned then it is required that:

1. $F \equiv F \cup \{C\}$;
2. C be a conflict clause;
3. all variables in C be decision variables.

2.3 Correctness

We first argue termination of the algorithm. To this end, notice that a sequence of decisions that leads to a conflict cannot be repeated. This is because all the variables in the learned clause C are decision variables. Thus if all but one of the literals in C are made false in some future assignment then the remaining variable cannot be a decision variable since its value is determined by the unit propagation rule.

Given termination, correctness is straightforward. By Condition 1 above we have $F \equiv F \cup \{C\}$ for any learned clause C. Thus if the empty clause is learned then the original formula was unsatisfiable. On the other hand, if the algorithm terminates with a satisfying valuation $\mathcal{A}$ then the input formula is also satisfied by $\mathcal{A}$.

3 Clause Learning

Let $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$ be a sequence of assignments leading to a conflict. We compute an associated sequence of clauses $A_1, \ldots, A_{k+1}$ by backward induction as follows:

1. Define A_{k+1} to be any conflict clause under the assignment $\mathcal{A}$.
2. If $p_i \mapsto b_i$ is a decision assignment or if p_i is not mentioned in A_{i+1} then define $A_i = A_{i+1}$.
3. If $p_i \overset{C_i}{\mapsto} b_i$ is an implied assignment and p_i is mentioned in A_{i+1} then define A_i to be the resolvent of A_{i+1} and C_i with respect p_i.

The final clause A_1 is the learned clause.[2]

Example 2. Consider the conflict described in Example 1. From this situation the learning procedure generates clauses $A_8, A_7, \ldots, A_1$ as shown below.

$$\begin{array}{lr}
A_8 := \{\neg p_1, \neg p_7, \neg p_5\} & \text{(clause } C_4\text{)} \\
A_7 := \{\neg p_1, \neg p_5, \neg p_6\} & \text{(resolve } A_8, C_3\text{)} \\
A_6 := \{\neg p_1, \neg p_5\} & \text{(resolve } A_7, C_2\text{)} \\
A_5 := \{\neg p_1, \neg p_4\} & \text{(resolve } A_6, C_1\text{)} \\
\quad\vdots & \\
A_1 := \{\neg p_1, \neg p_4\} &
\end{array}$$

The learned clause A_1 is a conflict clause that contains only decision variables, including an occurrence of the top-level decision variable p_4. This clause captures the intuition that the conflict arose from the decision to make both p_1 and p_4 true (and is nothing to do with p_2 and p_3, which are not even mentioned in the formula). The addition of clause A_1 to F ensures that valuations in which p_1 and p_4 are both true are no longer reachable. Indeed, the algorithm backtracks to the highest level in which A_1 is a unit clause (namely the valuation $p_1 \mapsto 1$) and then unit propagation immediately leads to the assignment $p_4 \mapsto 0$.

The following proposition shows that the above learning procedure fulfils the desiderata listed in Section 2.2.

[2]Note that we are describing one policy for learning clauses and that alternative policies exist.

Proposition 3. Suppose that state $(F, \mathcal{A})$ is a conflict and let C be the learned clause. Then C is a conflict clause, all variables occurring in C are decision variables in $\mathcal{A}$, and $F \equiv F \cup \{C\}$.

Proof. Suppose that $\mathcal{A} = \langle p_1 \mapsto b_1, \ldots, p_k \mapsto b_k \rangle$. Then clause learning produces a sequence of clauses $A_{k+1}, A_k, \ldots, A_1$, with A_{k+1} the conflict clause and $A_1 = C$ the learned clause. Since C is obtained by a resolution proof from F, we have $F \equiv F \cup \{C\}$ by the Resolution Lemma.

Now we claim that the following hold for each clause A_i with $1 \leq i \leq k+1$,

1. A_i is a conflict clause (i.e., is made false by $\mathcal{A}$); and
2. the non-decision variables appearing in A_i lie all in $\{p_1, \ldots, p_{i-1}\}$;

The proof of claim is by "backward" induction on i, from $k+1$ down to 1.

The base case is $i = k+1$. By definition A_{k+1} is a conflict clause. It follows that A_{k+1} can only mention variables in the domain of $\mathcal{A}$, and so Condition 2 is satisfied.

The induction step divides into two cases. Suppose that A_{i+1} satisfies Conditions 1 and 2 above. The first case is that p_i is either a decision variable or does not occur in A_{i+1} then $A_i = A_{i+1}$, and clearly A_i satisfies Conditions 1 and 2.

The second case is that $p_i \overset{C_i}{\mapsto} b_i$ is an implied assignment and p_i occurs in A_{i+1}. Then A_i is a resolvent of A_{i+1} and C_i (with respect to p_i). In particular A_i does not mention the variable p_i. It remains to check that Conditions 1 and 2 hold for A_i.

Now C_i is a unit clause under the assignment $\langle p_1 \mapsto b_1, \ldots, p_{i-1} \mapsto b_{i-1} \rangle$. Thus, other than p_i, C_i can only mention variables in the set $\{p_1, \ldots, p_{i-1}\}$ and all literals in C_i pertaining to these variables are made false by $\mathcal{A}$. This shows that Conditions 1 and 2 hold for the resolvent A_i. □

Note that we have proved that all variables that appear in a learned clause C are decision variables. By a similar inductive argument it can be shown that C must mention the top-level decision variable. It follows that if a top-level decision assignment $p \mapsto b$ leads to conflict after unit propagation, then after backtracking the assignment $p \mapsto 1 - b$ is performed by the unit propagation rule before any further decisions.

4 Other Enhancements

Modern SAT solvers include numerous enhancements beyond what is discussed above. These include techniques, such as *watched literals*, to efficiently discover which clauses become unsat, decision heuristics for choosing decision variables, clause removal (removing learned clauses), and random restarts.