CO519 - Theory of Computing - Logic

Part C : Satisfiability for propositional logic

Dominic Orchard
School of Computing, University of Kent

Last updated on October 9, 2017

If you spot any errors or have suggested edits, the notes are written in LaTeX and are available on GitHub at `http://github.com/dorchard/co519-logic`. Please fork and submit a pull request with any suggested changes.

1 Satisfiable formula

This section covers *satisfiability* of propositional formula. Recall from Part A that a formula is *satisfiable* if it is true for a particular "assignment" of either true or false to variables in the formula. For example, $a \wedge b$ is satisfiable since we can make it true by setting a to be true and b to be true. We say this is the *satisfying assignment*, and will write this as:

$$\{a = \top,\ b = \top\}$$

A valid formula is trivially satisfiable since it is true no matter how we assign truth or falsehood to the variables.

The general problem of finding a satisfying assignment for a formula is known in computer science as the *Boolean satisfiability problem* or simply *SAT* for short. There are various algorithmic approaches to solving the *SAT* problem, *i.e.*, for calculating a satisfying assignment for a propositional formula. SAT is used in many areas of formal verification, such as AI, planning, circuit design, and theorem provers, and is applied to problems with thousands of variables.

One approach is to exhaustively list all possible assignments of true/false to variables in a formula by constructing its truth tables. Thus for a formula with n variables we need to calculate 2^n rows. This is completely infeasible for problems with hundreds or even thousands of variables. Instead, we'll look at the *DPLL algorithm* which utilises properties of propositional logic to be more efficient for many problems.

We can use SAT solving algorithms to prove validity of a formula P by applying SAT to $\neg P$. If the algorithm shows us that $\neg P$ is unsatisfiable, that implies $\neg P$ is false for any assignment of its variables and thus P is true for any assignment of its variables, *i.e.*, *valid*. This is very useful for modelling problems where we have a complex model of some system in logic, and a complex specification, and we want to prove the validity of:

$$\mathit{model} \rightarrow \mathit{specification} \tag{1}$$

(see Part B of the course on modelling in logic). Instead of proving this valid using natural deduction, we can instead use an algorithm to show unsatisfiability of $\neg(\mathit{model} \rightarrow \mathit{specification})$. If however we find a satisfying assignment, then we have a counterexample to the original property, that is, a set of variable assignments which makes equation (1) false.

2 The DPLL algorithm

DPLL stands for *Davis-Putnam-Logemann-Loveland* (who proposed this algorithm)[1]. We'll go over the technique since its a nice algorithm and makes some clever use of the properties of propositional logic to simplify the exploration of the state space. Despite the fact it is more than 50 years old, DPLL still forms the basis of many SAT solvers, though there has been some progress since.

2.1 CNF

The input to the DPLL algorithm is a formula in *conjunctive normal form* (or CNF for short). A formula in Conjunction Normal Form (CNF) is a conjunction of disjunctions of literals. A literal is either a variable or the negation of a variable. Thus, CNF formula are of the form:

$$(x \vee \neg y \vee ...) \wedge (z \vee \ldots) \wedge \ldots \wedge (\neg w \vee x \vee y)$$

We refer to a disjunction of literals as a *clause*. For example, the following highlights the middle clause:

$$(x \vee \neg w) \wedge (y \vee z) \wedge (\neg z \vee \neg x) \tag{2}$$

A CNF formula consists of a set of clauses all of which have to be true, since they are combined using conjunction. Within each clause, just one of literals has to be true since they are combined via disjunction.

Any formula can be converted into CNF by applying algebraic properties of logic to rewrite a formula. Informally, this can be done by doing the following:

- Replace implication $\rightarrow$ with disjunction $\vee$ and negation $\neg$, via:

$$P \rightarrow Q = \neg P \vee Q$$

- Use De Morgan's law to push negation inside of disjunction and conjunction:

$$\neg(P \vee Q) = \neg P \wedge \neg Q$$
$$\neg(P \wedge Q) = \neg P \vee \neg Q$$

[1] Actually the history is more complicated as in the history of most science: in 1960 Davis and Putnam did some work on automatically showing validity of formulae following a different technique, and David, Logemann, and Loveland built on some of the ideas to create their SAT in algorithm 1962.

- Push disjunction inside and pull conjunction out (using distributive laws):

$$(P \vee Q) \vee R = (P \vee R) \wedge (Q \vee R)$$
$$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$$

- Eliminate double negation $\neg\neg P = P$

By repeatedly applying these equations as rewrites (orienting the equalities from left to the right), we end up with a formula in CNF which is ready for DPLL.

2.2 DPLL, step-by-step

DPLL can be summarised in pseudo code as follows, which has four main steps which I remember using the acronym: **TUPS**:

$\mathsf{DPLL}(\top) =$*satisfiable*
$\mathsf{DPLL}(\bot) =$*unsatisfiable*
$\mathsf{DPLL}(P) =$

1. ***T**autology elimination*
2. ***U**nit propagation*
3. ***P**ure literal elimination*
4. ***S**plit on a variable: choose a variable v*
   $\mathsf{DPLL}(P')\{v = \top\}$
   $\mathsf{DPLL}(P')\{v = \bot\}$

Note that this is a recursive algorithm which branches into two executions at the last step. The four steps gradually reduce the size of the input CNF formula and create an assignment for its variables (setting variables to be true or false).

The first step is a simplification step using properties of propositional logic and the structure of CNF formula. The next two steps provide simplification and can give us some satisfying assignments. The last step splits the problem into two by picking a variable and repeating the DPLL procedure with that variable assigned to be true in one branch (written above as $\{v = \top\}$) and repeating DPLL with that variable assigned to false ($\{v = \bot\}$).

Step 1. Tautology elimination Consider a formula in CNF where one clause (highlighted in yellow bellow) contains both x and $\neg x$, e.g., a formula of the form:

$$P \wedge (\ \ldots x \vee \ldots \vee \neg x\ ) \wedge Q$$

Such a formula can be simplified by completely removing the highlighted clause. This is because a disjunction of a formula with its negation is a tautology (always true): $P \vee \neg P = \top$. Furthermore, since disjunction of anything with truth is equivalent to truth: $P \vee \top = \top$ and conjunction $P \wedge \top = \top$, we can completely filter out clauses that have a tautology, *i.e.*

$$P \wedge (\ \ldots x \vee \ldots \vee \neg x\ ) \wedge Q \quad \longrightarrow \quad P \wedge (\top) \wedge Q \quad \longrightarrow \quad P \wedge Q$$

Note that this doesn't tell us whether to assign true or false to x yet. So far we just know that this clause didn't depend on the truth or falsehood of x because it contained this tautology.

Step 2: Unit propagation In CNF terminology, a "unit" is a clause which contains just one literal (a variable or negation of a variable), *e.g.*,

$$P \wedge (\,x\,) \wedge Q \quad \textit{or} \quad P \wedge (\,\neg x\,) \wedge Q$$

where the "unit" clauses are highlighted in yellow.

If we see a formula in the left form, then we know that x must be true if we want the overall formula to be true, since we are taking the conjunction of x with all the other formula. If we have a formula with the right form, then we know that x must be false so that $\neg x$ is true to make the overall formula true. Thus in the left case we get the assignment $x = \top$ and in the right case we get $x = \bot$. We then *propagate* this assignment to the rest of the clauses P and Q, replacing any occurrences of x with its assignment:

$$\begin{array}{rrcll} & P \wedge (\,x\,) \wedge Q & \longrightarrow & \textit{replace}(x, \top, P) \wedge \textit{replace}(x, \top, Q) & \{x = \top\} \\ \textit{or} & P \wedge (\,\neg x\,) \wedge Q & \longrightarrow & \textit{replace}(x, \bot, P) \wedge \textit{replace}(x, \bot, Q) & \{x = \bot\} \end{array}$$

The assignment that is output by this step is written on the right-hand side. Note that we have also removed the unit clause since it has been made true and is therefore redundant now.

Here we are using a function on the syntax of formulas called *replace* where $\textit{replace}(x, \top, P)$ means replace/substitute any occurrence of x in P with $\top$, and similarly $\textit{replace}(x, \bot, P)$ replaces any occurrence of x in P with $\bot$. (This idea of substituting one proposition for another will crop up again in part D when we look at first-order logic proofs).

Thus, unit propagation gives us both a simplification and an assignment when we have "unit" clauses in our formula.

Example 1. The following formula has a unit clause highlighted in yellow:

$$(\neg x \vee \neg y) \wedge \,x\, \wedge (y \vee x)$$

Since it is "positive" (*i.e.*, not negated) then unit propagation emits the assignment $\{x = \top\}$ and then propagates this assignment by replacement:

$$\textit{replace}(x, \top, (\neg x \vee \neg y)) \wedge \textit{replace}(x, \top, (y \vee x))$$

Applying the replacement function then gives us:

$$(\neg \top \vee \neg y) \wedge (y \vee \top)$$

which simplifies via the following steps

$$\begin{aligned}
&(\neg\top \vee \neg y) \wedge (y \vee \top) \\
=\ &(\neg\top \vee \neg y) \wedge T \\
=\ &(\neg\top \vee \neg y) \\
=\ &(\bot \vee \neg y) \\
=\ &\neg y
\end{aligned}$$

In DPLL, we need only apply simplifications that involve disjunction or conjunction with $\top$ or $\vee$. It is straightforward to build this into an implementation so that these steps are implicit and automatic: essentially the "replacement" can remove entire clauses when we know we are making a literal true, or remove literals when they are false.

Performing just one step of unit propagation has greatly simplified our original formula from $(\neg x \vee \neg y) \wedge (x) \wedge (y \vee x)$ to $\neg y$, along with giving the assignment $\{x = \top\}$. Since $\neg y$ is itself a unit, we apply unit propagation again, yielding the assignment $\{y = \bot\}$ and the formula $\top$. Thus, we have reached a satisfying assignment $\{x = \top, y = \bot\}$ just by applying unit propagation twice.

Exercise 2.1. Convince yourself that $\{x = \top, y = \bot\}$ is a satisfying assignment for $(\neg x \vee \neg y) \wedge (x) \wedge (y \vee x)$ by substituting the variables for their assignment and simplifying.

Exercise 2.2. Perform unit propagation on the formula $(y \vee \neg x) \wedge (\neg y)$.

Step 3: Pure literal elimination In DPLL terminology, a pure literal is a literal whose negation does not appear anywhere else in the entire formula, *e.g.*

$$P \wedge (\ldots \vee x \vee \ldots) \wedge Q$$

where its dual $\neg x$ does not appear in P nor is it in Q. For example, x is a pure literal in this formula:

$$(x \vee y \vee \neg z) \wedge (\neg y \vee z \vee x)$$

The literal x can appear multiple times, what makes it "pure" is that its negation never appears anywhere. This means we can assign x to be true. We could assign x to be false, but it might be the wrong decision later, for example if the other literals in the clause turn out to false as well. It turns out that the most useful approach is to assign x to true, by following the principle of progressing towards a true formula as quickly as possible.

A pure literal might also be negative, for example:

$$P \wedge (\ldots \vee \neg x \vee \ldots) \wedge Q$$

where x does not appear in P and Q. In this case, we can assign x to be false, and propagate this assignment into P and Q.

Pure literal elimination therefore has the two dual rules:

$$\begin{array}{lllll} & P \wedge (\ldots \vee \neg x \vee \ldots) \wedge Q & \longrightarrow & \mathit{replace}(x, \top, P) \wedge \mathit{replace}(x, \top, Q) & \{x = \top\} \\ \mathit{or} & P \wedge (\ldots \vee x \vee \ldots) \wedge Q & \longrightarrow & \mathit{replace}(x, \bot, P) \wedge \mathit{replace}(x, \bot, Q) & \{x = \bot\} \end{array}$$

Example 2. The following formula has pure literal x:

$$(x \vee y \vee \neg z) \wedge (\neg y \vee x) \wedge (y \vee z)$$

Pure literal elimination then produces the assignment $\{x = \top\}$, eliminates the first clause (since it is now true), and rewrites the rest of the formula as follows:

$$\begin{aligned} & \mathit{replace}(x, \top, (\neg y \vee x) \wedge (y \vee z)) \\ = {} & (\neg y \vee \top) \wedge (y \vee z) \\ = {} & (y \vee z) \end{aligned}$$

We are left with a clause where both y and z are now pure literals, so pure literal elimination can be applied to either.

Let's pick y and assign it to $\{y = \top\}$. This gives us $\top$ as the resulting formula. Thus, we have found that $(x \vee y \vee \neg z) \wedge (\neg y \vee x) \wedge (y \vee z)$ is satisfiable with $\{x = \top, y = \top\}$, and it doesn't matter whether z is true or false.

Exercise 2.3. Apply pure literal elimination to the formula:

$$(\neg x \vee y \vee \neg z) \wedge (\neg y \vee z) \wedge (\neg x \vee z)$$

Step 4: Split a variable The previous steps have applied the rules of logic, and the shape of CNF formula, to make simplifications and assignments. Once we've done all that we can with those steps, the last step falls back to a "brute force" approach. We pick a variable, and "split it", that is, we assign it to be true and apply DPLL on the result and separately assign it to false and apply DPLL on the result. This results in us running two DPLL separate procedures separately from this point forwards.

That is, given a formula P, splitting recursively calls the DPLL algorithm under two new assignments:

$$\begin{array}{lll} & \mathsf{DPLL}(\mathit{replace}(x, \top, P) & \{x = \top\} \\ \mathit{and} & \mathsf{DPLL}(\mathit{replace}(x, \bot, P) & \{x = \bot\} \end{array}$$

These recursive calls will be separate, producing possibly different assignments from this point on.

Example 3. The following formula has no tautologies, units, or pure literals (*i.e.*, none of the first three steps of DPLL apply):

$$(x \vee \neg y) \wedge (y \vee z) \wedge (\neg z \vee \neg x)$$

The splitting step chooses any variable in the formula, let's say z, and splits the DPLL process. Let's follow the branch with the assignment $\{z = \top\}$, which is then propagated to the rest of the formula by replacement:

$$\begin{aligned} &= \quad \mathit{replace}(z, \top, (x \vee \neg y) \wedge (y \vee z) \wedge (\neg z \vee \neg x)) \qquad \{z = \top\} \\ &= \quad (x \vee \neg y) \wedge (y \vee \top) \wedge (\bot \vee \neg x) \\ &= \quad (x \vee \neg y) \wedge (\neg x) \end{aligned}$$

The other branched DPLL process has assignment $\{z = \bot\}$ which produces:

$$\begin{aligned} &= \quad \mathit{replace}(z, \bot, (x \vee \neg y) \wedge (y \vee z) \wedge (\neg z \vee \neg x)) \qquad \{z = \bot\} \\ &= \quad (x \vee \neg y) \wedge (y \vee \bot) \wedge (\top \vee \neg x) \\ &= \quad (x \vee \neg y) \wedge (y) \end{aligned}$$

We then return to step 1 for both branches, and continue with two separate instance of the DPLL procedure on the above two formula.

Exercise 2.4. Apply the splitting step to the following formula on variable y:

$$(x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (y \vee z)$$

Write down the two resulting formula after replacement and trivial simplifications have been performed.

The following gives an example putting the steps together.

Example 4. Consider the following formula:

$$(\neg x \to y) \wedge x$$

Is it satisfiable? Before we apply DPLL we first have to convert it to CNF:

$$\begin{aligned} & \quad (\neg x \to y) \wedge x \\ = & \quad (\neg\neg x \vee y) \wedge x \quad \{\to \mathit{as} \vee\} \\ = & \quad (x \vee y) \wedge x \quad \{\mathit{Double\ negation\ elimination}\} \end{aligned}$$

Now we have the formula in CNF, we can apply DPLL. I'll write out the steps in tabular form:

Step	Note	Resulting CNF formula	Satisfying assignments
	start	$(x \vee y) \wedge x$	
1	Tautology elim	$(x \vee y) \wedge x$	
2	Unit propagation x	$\top$	$\{x = \top\}$

Thus the original equation is satisfiable with assignment $x = \top$ (and y can be anything). Note that we reached this conclusion very quickly, just by applying tautology elimination and unit propagation.

This tabulated form is handy for small examples; you might like to use it for the class exercises.

Exercise 2.5. Apply the full DPLL procedure to the following formula (already in CNF):

$$(x \vee y \vee z) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y \vee \neg z)$$

Example 5. The following is the proposition $\neg(\textit{model} \rightarrow \textit{specification})$ for the traffic light example shown in Part B (see lectures), in CNF:

$$(\neg r \vee g') \wedge (\neg r \vee \neg r') \wedge (\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$$
$$(r \vee g) \wedge (\neg r \vee \neg g) \wedge (\neg r' \vee r')$$
$$(\neg r' \vee g') \wedge (\neg g' \vee r') \wedge (\neg g' \vee g)$$

We'll apply DPLL to it here. Due to the large size of the formula, I'll use a more free-form style rather than the tabulated form used above.

Step 1: Tautology elimination - There are two immediate tautologies in the above formula which are highlighted in yellow. These are eliminated to give:

$$(\neg r \vee g') \wedge (\neg r \vee \neg r') \wedge (\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$$
$$(r \vee g) \wedge (\neg r \vee \neg g) \wedge$$
$$(\neg r' \vee g') \wedge (\neg g' \vee r')$$

There are no units or pure literals so we move to step 4.

Step 4: Splitting a variable - Choose r:

$\{r = \top\}$	$\{r = \bot\}$
yielding:	yielding:
$(\neg\top \vee g') \wedge (\neg\top \vee \neg r') \wedge$ $(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(\top \vee g) \wedge (\neg\top \vee \neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$	$(\neg\bot \vee g') \wedge (\neg\bot \vee \neg r') \wedge$ $(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(\bot \vee g) \wedge (\neg\bot \vee \neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$
which simplifies to	which simplifies to
$g' \wedge (\neg r') \wedge (\neg g \vee \neg g') \wedge$ $(\neg g \vee r') \wedge$ $(\neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$	$(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$
1. *Unit propagation* on the unit g' $g' \wedge (\neg r') \wedge (\neg g \vee \neg g') \wedge$ $(\neg g \vee r') \wedge$ $(\neg g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$ yields assignment $\{g' = \top\}$ giving	1. *Unit propagation* on the unit g $(\neg g \vee \neg g') \wedge (\neg g \vee r') \wedge$ $(g) \wedge$ $(\neg r' \vee g') \wedge (\neg g' \vee r')$ yields assignment $\{g = \top\}$ giving

$$(\neg r') \land (\neg g) \land (\neg g \lor r') \land (\neg g) \land (r')$$

1. Unit propagation: on the unit $\neg r'$

$$(\neg r') \land (\neg g) \land (\neg g \lor r') \land (\neg g) \land (r')$$

yields assignment $\{r' = \bot\}$ giving

$$(\neg g) \land (\neg g) \land (\neg g) \land \bot$$

simplifies to

$$\bot$$

$$(\neg g') \land (r') \land (\neg r' \lor g') \land (\neg g' \lor r')$$

1. Unit propagation: on the unit $\neg g'$

$$(\neg g') \land (r') \land (\neg r' \lor g') \land (\neg g' \lor r')$$

yields assignment $\{g' = \bot\}$ giving

$$(r') \land (\neg r')$$

1. Unit propagation: on the unit r'
yields assignment $\{r' = \top\}$ giving

$$\bot$$

Both branches have ended with $\bot$ hence the original formula is unsatisfiable.

3 Exercises

This section collects the exercises given in these notes.

Exercise 2.1. Convince yourself that $\{x = \top, y = \bot\}$ is a satisfying assignment for $(\neg x \lor \neg y) \land (x) \land (y \lor x)$ by substituting the variables for their assignment and simplifying.

Exercise 2.2. Perform unit propagation on the formula $(y \lor \neg x) \land (\neg y)$.

Exercise 2.3. Apply pure literal elimination to the formula:

$$(\neg x \lor y \lor \neg z) \land (\neg y \lor z) \land (\neg x \lor z)$$

Exercise 2.4. Apply the splitting step to the following formula on variable y:

$$(x \lor y \lor \neg z) \land (x \lor \neg y \lor z) \land (y \lor z)$$

Write down the two resulting formula after replacement and trivial simplifications have been performed.

Exercise 2.5. Apply the full DPLL procedure to the following formula (already in CNF):

$$(x \lor y \lor z) \land (\neg x \lor y) \land (\neg x \lor \neg y \lor \neg z)$$