Automated Deduction

Laura Kovács
TU Wien

Decision Procedures for Propositional Satisfiability

More sophisticated decision procedures:

- Splitting;
- DPLL;
- (BDDs – binary decision diagrams);
- (resolution).

We next look at the DPLL method, which uses the splitting algorithm.

(DPLL: Davis-Putnam-Logemann-Loveland, 1962)

DPLL works on a restricted set of formulas, we need to first look at the needed normal forms and normal form conversions.

Literal, clause

- Literal: either an atom p (positive literal) or its negation $\neg p$ (negative literal).
- The complementary literal to L:

$$\overline{L} \stackrel{\text{def}}{=} \begin{cases} \neg L, & \text{if } L \text{ is positive;} \\ p, & \text{if } L \text{ has the form } \neg p. \end{cases}$$

  In other words, p and $\neg p$ are complementary.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$, $n \geq 0$ of literals.
  - Empty clause, denoted by $\square$: $n = 0$ (the empty clause is false in every interpretation).
  - Unit clause: $n = 1$.
  - Horn clause: a clause with at most one positive literal.

CNF

- A formula A is in conjunctive normal form, or simply CNF, if it is either $\top$, or $\bot$, or a conjunction of disjunctions of literals:

$$A = \bigwedge_i \bigvee_j L_{i,j}.$$

  (In other words, A is a conjunction of clauses.)
- A formula B is called a conjunctive normal form of a formula A if B is equivalent to A and B is in conjunctive normal form.

Satisfiability on CNF

- An interpretation I satisfies a formula in CNF

$$A = \bigwedge_i \bigvee_j L_{i,j}.$$

  if and only if it satisfies every clause

$$\bigvee_j L_{i,j}.$$

  in it.
- An interpretation I satisfies a clause

$$L_1 \vee \ldots \vee L_k$$

  if and only if it satisfies at least one literal L_m in this clause.

CNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg\neg A & \Rightarrow & A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n & \Rightarrow & (A_1 \vee B_1 \vee \ldots \vee B_n) \quad \wedge \\
 & & \qquad\qquad \ldots \qquad\qquad \wedge \\
 & & (A_m \vee B_1 \vee \ldots \vee B_n).
\end{array}
$$

A formula to which no rewrite rule is applicable

- contains no $\leftrightarrow$;
- contains no $\rightarrow$;
- may only contain $\neg$ applied to atoms;
- cannot contain $\wedge$ in the scope of $\vee$;
- (hence) is in CNF.

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A,
\end{aligned}$$

$$(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n \Rightarrow \begin{array}{l} (A_1 \vee B_1 \vee \ldots \vee B_n) \quad \wedge \\ \qquad\qquad \ldots \qquad\qquad\quad \wedge \\ (A_m \vee B_1 \vee \ldots \vee B_n). \end{array}$$

CNF and satisfiability

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\ldots$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

Therefore, the formula

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$$

has the same models as the set consisting of four clauses

$$\neg p \vee q$$
$$\neg p \vee \neg q \vee r$$
$$p$$
$$\neg r$$

The CNF transformation reduces the satisfiability problem for formulas to the satisfiability problem for sets of clauses.

Problem with CNF

Compute the CNF of

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))).$$

$$\begin{aligned}
&p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))) \Rightarrow \\
&(\neg p_1 \vee (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \Rightarrow \\
&(\neg p_1 \vee \quad ((\neg p_2 \vee (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))\wedge \\
&\qquad\qquad (p_2 \vee \neg(p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))) \wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))
\end{aligned}$$

If we continue, the formula will grow exponentially.

CNF is exponential

There are formulas for which the shortest CNF has an exponential size.

Is there any way to avoid exponential blowup?

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

After several steps

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$

$$\begin{aligned} &p_1 \leftrightarrow (p_2 \leftrightarrow n_3);\\ &n_3 \leftrightarrow (p_3 \leftrightarrow n_4);\\ &n_4 \leftrightarrow (p_4 \leftrightarrow n_5);\\ &n_5 \leftrightarrow (p_5 \leftrightarrow p_6). \end{aligned}$$

The conversion of the original formula to CNF introduces 32 copies of p_6.

The conversion of the new set of formulas to CNF introduces 4 copies of p_6.

Definitional Translations

Lemma [Naming]

Let S be a set of formulas and A a formula. Let n be a boolean variable not occuring in S, nor in A.

Then S is satisfiable if and only if so is the set of formulas $S \cup \{n \leftrightarrow A\}$.

Clausal Form

- Clausal form of a formula A: a set of clauses which is satisfiable if and only if A is satisfiable.
- Clausal form of a set S of formulas: a set of clauses which is satisfiable if and only if so is S.

We can require even more: that A and its clausal form have the same models in the language of A.

Using clausal normal forms instead of conjunctive normal forms we can convert any formula to a set of clauses in almost linear time.

Definitional Clause Form Transformation

This algorithm converts a formula A into a set of clauses S such that S is a clausal normal form of A.
If A has the form $C_1 \wedge \ldots \wedge C_n$, where $n \geq 1$ and each C_i is a clause, then $S \stackrel{\text{def}}{=} \{C_1, \ldots, C_n\}$.
Otherwise, introduce a name for each subformula B of A such that B is not a literal and use this name instead of the formula.

Example

	subformula	definition	clauses
			n_1
n_1	$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$	$n_1 \leftrightarrow \neg n_2$	$\neg n_1 \vee \neg n_2$ $n_1 \vee n_2$
n_2	$(p \to q) \wedge (p \wedge q \to r) \to (p \to r)$	$n_2 \leftrightarrow (n_3 \to n_7)$	$\neg n_2 \vee \neg n_3 \vee n_7$ $n_3 \vee n_2$ $\neg n_7 \vee n_2$
n_3	$(p \to q) \wedge (p \wedge q \to r)$	$n_3 \leftrightarrow (n_4 \wedge n_5)$	$\neg n_3 \vee n_4$ $\neg n_3 \vee n_5$ $\neg n_4 \vee \neg n_5 \vee n_3$
n_4	$p \to q$	$n_4 \leftrightarrow (p \to q)$	$\neg n_4 \vee \neg p \vee q$ $p \vee n_4$ $\neg q \vee n_4$
n_5	$p \wedge q \to r$	$n_5 \leftrightarrow (n_6 \to r)$	$\neg n_5 \vee \neg n_6 \vee r$ $n_6 \vee n_5$ $\neg r \vee n_5$
n_6	$p \wedge q$	$n_6 \leftrightarrow (p \wedge q)$	$\neg n_6 \vee p$ $\neg n_6 \vee q$ $\neg p \vee \neg q \vee n_6$
n_7	$p \to r$	$n_7 \leftrightarrow (p \to r)$	$\neg n_7 \vee \neg p \vee r$ $p \vee n_7$ $\neg r \vee n_7$

Converting a formula to clausal form.Take all subformulas that are not literals.Introduce names for these formulas.Introduce definitions.Convert the resulting formula to CNF using the standard algorithm.

Optimised Definitional Clause Form Transformation

If we introduce a name n for a subformula B and the occurence of the subformula B in formula A is either only positive or negative, then an implication is used instead of equivalence.

- if B has only positive occurences in A, then $n \rightarrow B$ is used instead of $n \leftrightarrow B$;
- if B has only negative occurences in A, then $B \rightarrow n$ is used instead of $n \leftrightarrow B$;

Example

	subformula	definition	clauses
			n_1
n_1	$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$	$n_1 \to \neg n_2$	$\neg n_1 \vee \neg n_2$ $n_1 \vee n_2$
n_2	$(p \to q) \wedge (p \wedge q \to r) \to (p \to r)$	$(n_3 \to n_7) \to n_2$	$\neg n_2 \vee \neg n_3 \vee n_7$ $n_3 \vee n_2$ $\neg n_7 \vee n_2$
n_3	$(p \to q) \wedge (p \wedge q \to r)$	$n_3 \to (n_4 \wedge n_5)$	$\neg n_3 \vee n_4$ $\neg n_3 \vee n_5$ $\neg n_4 \vee \neg n_5 \vee n_3$
n_4	$p \to q$	$n_4 \to (p \to q)$	$\neg n_4 \vee \neg p \vee q$ $p \vee n_4$ $\neg q \vee n_4$
n_5	$p \wedge q \to r$	$n_5 \to (n_6 \to r)$	$\neg n_5 \vee \neg n_6 \vee r$ $n_6 \vee n_5$ $\neg r \vee n_5$
n_6	$p \wedge q$	$(p \wedge q) \to n_6$	$\neg n_6 \vee p$ $\neg n_6 \vee q$ $\neg p \vee \neg q \vee n_6$
n_7	$p \to r$	$(p \to r) \to n_7$	$\neg n_7 \vee \neg p \vee r$ $p \vee n_7$ $\neg r \vee n_7$

All clauses shown in the red color are not generated by the optimised transformation.
The optimised transformation gives fewer clauses.

Satisfiability-checking for sets of clauses

The CNF transformation of

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$$

gives the set of four clauses:

$$\neg p \vee q$$
$$\neg p \vee \neg q \vee r$$
$$p$$
$$\neg r$$

Every interpretation that satisfies this set of clauses must assign 1 to p and 0 to r, so we do not have to guess values of these variables.

In fact, we can do even better and establish unsatisfiability without any guessing.

Searching for satisfiability

$$\{p \mapsto 1, r \mapsto 0, q \mapsto 1\}$$

$$\neg p \vee q$$
$$\neg p \vee \neg q \vee r \square$$
$$p$$
$$\neg r$$

This set of clauses is unsatisfiable.

Unit propagation

Let S be a set of clauses. Unit propagation: repeatedly performing the following transformation: if S contains a unit clause, i.e. a clause consisting of one literal L, then

1. remove from S every clause of the form $L \vee C'$;
2. replace in S every clause of the form $\overline{L} \vee C'$ by the clause C'.

Unit Propagation, Example

n_1
$\neg n_1 \vee \neg n_2$
$n_1 \vee n_2$
$\neg n_2 \vee \neg n_3 \vee n_7$
$n_3 \vee n_2$
$\neg n_7 \vee n_2$
$\neg n_3 \vee n_4$
$\neg n_3 \vee n_5$
$\neg n_4 \vee \neg n_5 \vee n_3$
$\neg n_4 \vee \neg p \vee q$
$p \vee n_4$
$\neg q \vee n_4$
$\neg n_5 \vee \neg n_6 \vee r$
$n_6 \vee n_5$
$\neg r \vee n_5$
$\neg n_6 \vee p$
$\neg n_6 \vee q$
$\neg p \vee \neg q \vee n_6 \square$
$\neg n_7 \vee \neg p \vee r$
$p \vee n_7$
$\neg r \vee n_7$

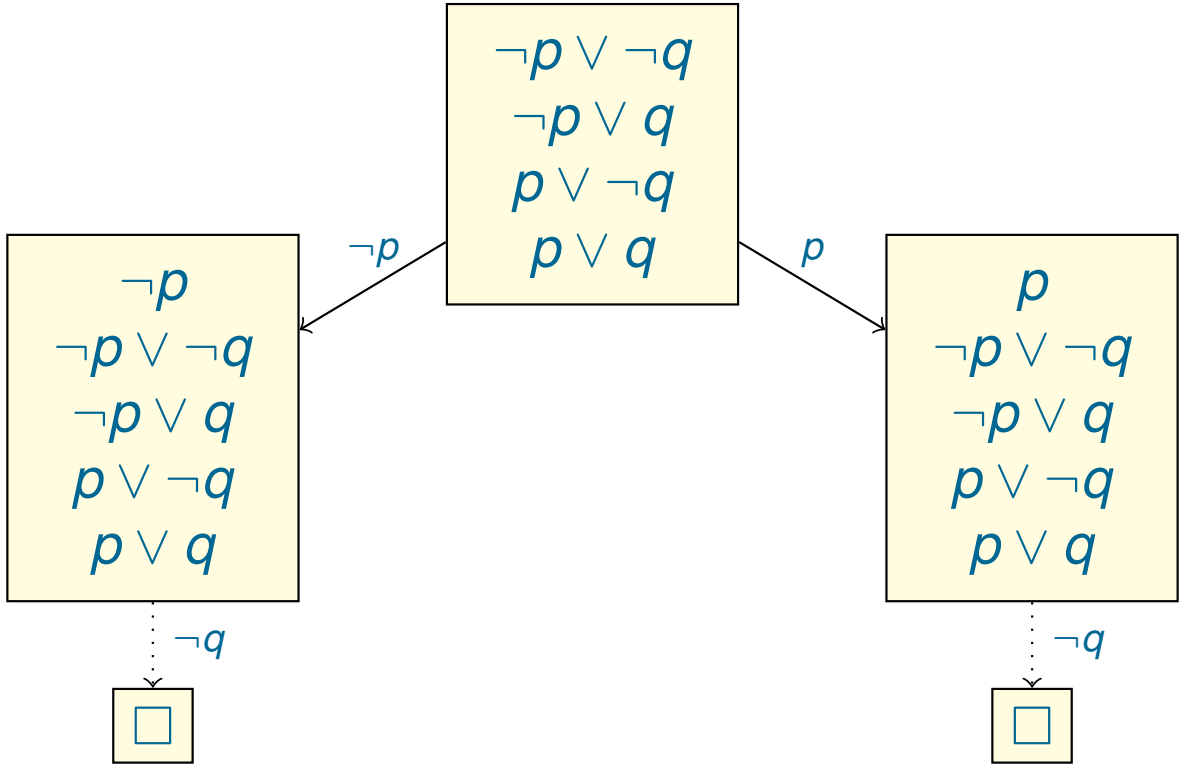
We established unsatisfiability of this set of clauses in a completely deterministic way, by unit propagation.

DPLL = splitting + unit propagation

procedure *DPLL*(S)
input: set of clauses S
output: *satisfiable* or *unsatisfiable*
parameters: function *select_literal*
begin
  S := *propagate*(S)
  if S is empty **then return** *satisfiable*
  if S contains $\square$ **then return** *unsatisfiable*
  L := *select_literal*(S)
  if *DPLL*($S \cup \{L\}$) = *satisfiable*
    then return *satisfiable*
    else return *DPLL*($S \cup \{\overline{L}\}$)
end

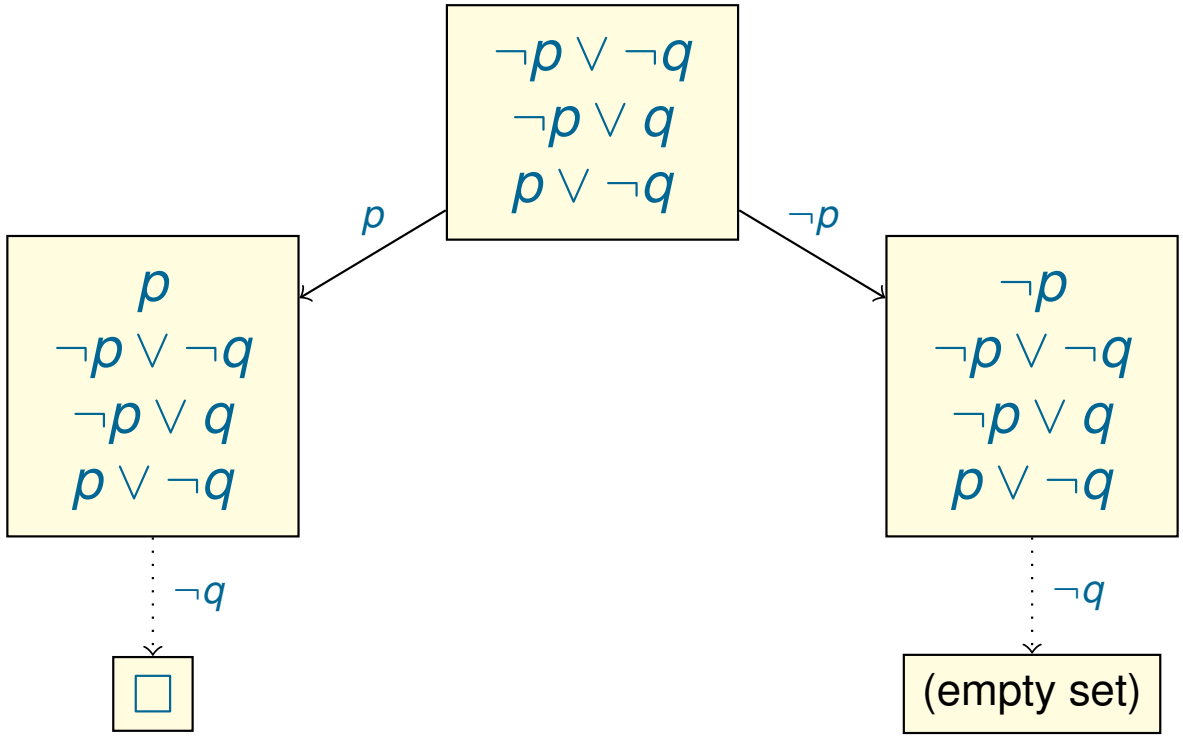
DPLL. Example 1

Can be illustrated using DPLL trees (similar to splitting trees).



Since all branches end up in a set contaning the empty clause, the initial set of clauses is unsatisfiable.

DPLL. Example 2



The set of clauses is satisfiable.
The model can be obtained by collecting all selected literals and literals used in unit propagation on the branch resulting in the empty set.
This DPLL tree gives us the model $\{p \mapsto 0, q \mapsto 0\}$.

Two optimisations

Any clause $p \vee \neg p \vee C$ is a tautology. Tautologies can be removed.
A literal L in S is called pure if S contains no clauses of the form $\overline{L} \vee C$

All clauses containing a pure literal can be satisfied by assigning a suitable truth value to the variable of this literal.

Hence, clauses containing pure literals can be removed, too.

Pure literals: example

$$\neg p_2 \vee \neg p_3$$
$$p_1 \vee \neg p_2$$
$$\neg p_1 \vee p_2 \vee \neg p_3$$
$$\neg p_1 \vee \neg p_3$$
$$p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee \neg p_3$$

The literal $\neg p_3$ is pure in this set. We can remove all clauses containing this literal.

Now the literal p_1 is pure in this set. We can remove all clauses containing this literal.

We obtain the empty set of clauses.

This gives us two models:

$$\{p_1 \mapsto 1, p_2 \mapsto 0, p_3 \mapsto 0\}$$
$$\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 0\}$$

Running a SAT solver

Very simple but efficient SAT solver: MiniSat, `http://minisat.se/`.

$$p_1$$
$$\neg p_1 \vee p_2$$
$$\neg p_1 \vee \neg p_2 \vee p_3$$
$$\neg p_2 \vee \neg p_3$$

DIMACS input format:

```
p cnf 3 4
1 0
-1 2 0
-1 -2 3 0
-2 -3 0
```

3 variables, 4 clauses.

$\neg p_1 \vee \neg p_2 \vee p_3$