SAT and DPLL

Espen H. Lian

Ifi, UiO

May 4, 2010

Introduction

- SAT is the problem of determining if a propositional formula (on *conjunctive normal form*) is satisfiable.
- The DPLL (Davis-Putnam-Logemann-Loveland) procedure from 1962 [2] is an algorithm solving SAT.
- DPLL is a refinement of the DP (Davis-Putnam) procedure from 1960 [3].
- We present (a version of) DPLL as a calculus.
- DPLL is interesting because it works well in practice, ie. the best SAT solvers are based on DPLL.

Preliminaries

A literal is a propositional variable or its negation.

Our connectives are $\neg$, $\wedge$, $\vee$ and $\supset$.

$\top$ and $\bot$ are the truth constants.

We will use the following notation.

- propositional variables: P, Q, R, S (possibly subscripted)
- literals: x, y, z (possibly subscripted)
- general formulae: X, Y, Z

The complement of a literal is defined as follows.

- $\overline{P} = \neg P$, and
- $\overline{\neg P} = P$.

NNF

A formula is on negation normal form (NNF) if negations occur only in front of propositional variables and implications does not occur at all.

Any formula can be put on NNF using the following rewrite rules.

$$\neg\neg X \rightarrow X$$
$$X \supset Y \rightarrow \neg X \lor Y$$
$$\neg(X \land Y) \rightarrow \neg X \lor \neg Y$$
$$\neg(X \lor Y) \rightarrow \neg X \land \neg Y$$

Some additional rewrite rules are needed for formula containing $\top$ and $\bot$.

We will assume that a formula X on NNF does not contain $\top$ or $\bot$ unless $X = \top$ or $X = \bot$.

CNF and DNF

A formula is on conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals.

Example 1

$(\neg P \vee Q) \wedge (P \vee \neg Q \vee R) \wedge (Q \vee S) \wedge (P \vee \neg R)$

A formula on NNF can be put on CNF using the following rewrite rules.

$$(X \wedge Y) \vee Z \to (X \vee Z) \wedge (Y \vee Z)$$
$$Z \vee (X \wedge Y) \to (Z \vee X) \wedge (Z \vee Y)$$

A formula is on disjunctive normal form (DNF) if it is a disjunction of conjunctions of literals.

DNF is like CNF, only with $\wedge$ and $\vee$ exchanged.

Example

The following formula expresses "$P \wedge Q$ or $R \wedge S$ but not both," (ie. *exclusive* or).

$$((P \wedge Q) \vee (R \wedge S)) \wedge (\neg(P \wedge Q) \vee \neg(R \wedge S))$$

NNF: $((P \wedge Q) \vee (R \wedge S)) \wedge ((\neg P \vee \neg Q) \vee (\neg R \vee \neg S))$

CNF: $(P \vee R) \wedge (P \vee S) \wedge (Q \vee R) \wedge (Q \vee S) \wedge (\neg P \vee \neg Q \vee \neg R \vee \neg S)$

The NNF to CNF part is performed as follows.

$$\begin{aligned}
&(P \wedge Q) \vee (R \wedge S) \\
\rightarrow\ &(P \vee (R \wedge S)) \wedge (Q \vee (R \wedge S)) \\
\rightarrow\ &(P \vee R) \wedge (P \vee S) \wedge (Q \vee (R \wedge S)) \\
\rightarrow\ &(P \vee R) \wedge (P \vee S) \wedge (Q \vee R) \wedge (Q \vee S)
\end{aligned}$$

Size increase

Rewriting a formula from DNF to CNF (or vice versa) may cause an exponential increase in size.

$$(P_1 \wedge P_2) \vee (P_3 \wedge P_4) \vee (P_5 \wedge P_6)$$

On CNF:

$$\begin{aligned}
&(P_1 \vee P_3 \vee P_5) \wedge (P_1 \vee P_3 \vee P_6) \wedge \\
&(P_1 \vee P_4 \vee P_5) \wedge (P_1 \vee P_4 \vee P_6) \wedge \\
&(P_2 \vee P_3 \vee P_5) \wedge (P_2 \vee P_3 \vee P_6) \wedge \\
&(P_2 \vee P_4 \vee P_5) \wedge (P_2 \vee P_4 \vee P_6)
\end{aligned}$$

Clauses and clause sets

For the sake of notational simplicity, instead of using formula on CNF, we will use *clause sets*.

A clause is a disjunction of literals.

A unit clause is a singleton clause.

A clause set is a finite set of clauses (interpreted conjunctively).

We will represent non-empty clauses by the set of its literals using a Prolog-like notation.

- An empty clause is the empty disjunction $\perp$.
- $x_1 \vee \cdots \vee x_n$ is represented by the set $[x_1 \ldots x_n]$, for $n > 0$ (n is the length).
- We will sometimes write [] for $\perp$.

Observe that $[] \neq \emptyset$ (see next foil).

Example

Some clauses:

1. $[P \ \neg Q \ R]$
2. $[P \ \neg P]$
3. $[]$, the empty clause

Some clause sets:

1. $\{[P \ \neg Q \ R]\}$
2. $\{[P \ \neg P], [], [P \ \neg Q \ R]\}$
3. $\emptyset$, the empty clause set
4. $\{[]\}$, the clause set containing exactly the empty clause

Valuation

Let Γ be a clause set and C a clause.

As clauses are disjunctions, it follows that they are valuated as follows.

$$v(C) = 1 \text{ iff } v(x) = 1 \text{ for some } x \in C.$$

We will interpret clause sets conjunctively, ie.

$$v(\Gamma) = 1 \text{ iff } v(C) = 1 \text{ for every } C \in \Gamma.$$

Observe that

- if C is empty, then $v(C) = 0$, while
- if Γ is empty, then $v(\Gamma) = 1$.

Thus we may use clause sets to represent formula on CNF.

Example:

$$v(\{[P\ \neg Q\ R], [\neg P\ \neg R]\}) = v((P \vee \neg Q \vee R) \wedge (\neg P \vee \neg R))$$

Subsumption

Let C_1 and C_2 be clauses. If $C_1 \subseteq C_2$, we say that C_1 subsumes C_2.

Lemma 2 (Subsumption)

If C_1 subsumes C_2, and $v(C_1) = 1$, then $v(C_2) = 1$.

Proof.

- If $v(C_1) = 1$, then $v(x) = 1$ for some $x \in C_1$.
- If $C_1 \subseteq C_2$, then $x \in C_2$.
- Thus $v(C_2) = 1$. □

Example: $\models P \supset (P \vee Q)$ as $[P]$ subsumes $[P\ Q]$.

Subsumption

Define $\Gamma_x = \{C \cup [x] \mid C \in \Gamma\}$, ie. x is added to every clause.

Example

1. $\{[P\ Q], [\neg Q], [\neg P\ \neg Q]\}_x = \{[P\ Q\ x], [\neg Q\ x], [\neg P\ \neg Q\ x]\}$.
2. $\{[P\ Q], [\neg Q], [\neg P\ \neg Q]\}_P = \{[P\ Q], [P\ \neg Q], [P\ \neg P\ \neg Q]\}$.
3. $\{\bot\}_x = \{[]\}_x = \{[x]\}$.
4. $\emptyset_x = \emptyset$.

Corollary 3 (of the Subsumption Lemma)

If $v(\Gamma) = 1$, *then* $v(\Gamma_x) = 1$.

Proof.

Every clause in Γ subsumes one in Γ_x. $\square$

Subsumption

A similar lemma for clause sets, only the other way as clause sets are interpreted conjunctively and clauses disjunctively.

Lemma 4

Let Γ and Δ be clause sets. If $\Delta \subseteq \Gamma$ and $v(\Gamma) = 1$, then $v(\Delta) = 1$.

Proof.

- If $v(\Gamma) = 1$, then $v(C) = 1$ for every $C \in \Gamma$.
- If $\Delta \subseteq \Gamma$, then $v(C) = 1$ for every $C \in \Delta$.
- Thus $v(\Delta) = 1$. $\square$

Some lemmata

Let Γ and Δ be clause sets and C a clause.

- Γ, Δ means $\Gamma \cup \Delta$.
- Γ, C means $\Gamma \cup \{C\}$.
- We say that x occurs in Γ if $x \in C$ for some $C \in \Gamma$.

Lemma 5

Let Γ be a non-empty clause set without any occurence of x or $\overline{x}$. If Γ is satisfiable, there is some valuation v such that $v(\Gamma, [x]) = 1$.

Some lemmata

Lemma 6

If $v(x) = 1$, then

1. $v(\Gamma_x) = 1$.
2. $v(\Gamma_{\overline{x}}) = v(\Gamma)$.

Proof.

1. If $v(x) = 1$, then
   - $v(C \cup [x]) = 1$ for any clause C,...
   - ...in particular every one in Γ.
   - Thus $v(\Gamma_x) = 1$.
2. *Left as exercise.* $\square$

The core of DPLL

The preceding lemma comes close to the core of DPLL.

If we make x true, we can

1. remove every clause containing x, and
2. remove $\overline{x}$ from every clause containing it.

Example 7

Let $\Gamma = \{[P\ Q], [\neg P\ \neg Q], [Q\ \neg R]\}$.

If $v(P) = 1$, then we can

1. remove $[P\ Q]$, and
2. remove $\neg P$ from $[\neg P\ \neg Q]$.

In other words, $v(\Gamma) = v(\{[\neg Q], [Q\ \neg R]\})$.

Preliminaries

The DPLL calculus operates not on general formulae but on a clause set Γ.

We start by removing from Γ

- any clause C such that $\{x, \overline{x}\} \subseteq C$ for some x.

This obviously does not affect satisfiability.

- If $\{x, \overline{x}\} \subseteq C$, then $v(C) = 1$, thus $v(\Gamma) = v(\Gamma \setminus \{C\})$.

The rules

Let Γ, Λ and Δ be clause sets without any occurence of x or $\overline{x}$ such that Γ and Λ are non-empty.

Definition 8

An axiom is any clause set where the empty clause occurs, ie. of the form $\perp, \Delta$.

Why are the axioms unsatisfiable?

- In terms of sequent calculus, that Γ is satisfiable may be expressed as $\Gamma \nvdash \perp$ or $\Gamma \nvdash \emptyset$.
- DPLL can be viewed as a left-calculus, ie. the right hand side of the sequent is empty.
- Thus in sequent calculus terms, $\perp, \Delta$ means $\perp, \Delta \vdash \perp$, which is valid.

Monotone literal fixing

If it's the case that some x occurs in some clauses and $\overline{x}$ does not, we say that x is monotone, and we make x true, because this makes the clauses x occurs in true and does not affect the other clauses.

$$\frac{\Delta}{\Gamma_x, \Delta}\ \textbf{Mon}$$

This rule is sometimes called the Affirmative-Negative Rule.

Example: $\neg Q$ is monotone.

$$\frac{[P \neg Q\ R], [\neg P \neg R], [P \neg R]}{[P \neg Q\ R], [\neg P \neg R], [P \neg R]}\ \textbf{Mon}$$

Unit subsumption

If it's the case that

- the unit clause $[x]$ occurs,
- x occur in some other clauses, and
- $\overline{x}$ occurs in yet others,

$[x]$ subsumes the others where x occurs.

$$\frac{[x], \Lambda_{\overline{x}}, \Delta}{[x], \Gamma_x, \Lambda_{\overline{x}}, \Delta}\ \textbf{Sub}$$

Example: $[Q]$ subsumes $[\neg P\ Q]$.

$$\frac{[Q], [\neg P\ Q], [\neg P\ \neg Q], [R]}{[Q], [\neg P\ Q], [\neg P\ \neg Q], [R]}\ \textbf{Sub}$$

Unit resolution

If it's the case that

- the unit clause $[x]$ occurs,
- x does not occur anywhere else but
- $\overline{x}$ does,

make x true.

$$\frac{\Lambda, \Delta}{[x], \Lambda_{\overline{x}}, \Delta}\ \textbf{Res}$$

Example:

$$\frac{[Q], [P\ \neg Q], [\neg P\ \neg Q], [R]}{[Q], [P\ \neg Q], [\neg P\ \neg Q], [R]}\ \textbf{Res}$$

Split

If it's the case that

- some x occurs in some clauses, and
- $\overline{x}$ occurs in others,

we can make two branches: one where x is true and one where x is false.

$$\frac{\Gamma, \Delta \qquad \Lambda, \Delta}{\Gamma_x, \Lambda_{\overline{x}}, \Delta}\ \textbf{Split}$$

Example: Split on P.

$$\frac{[P\ \neg Q], [\neg P\ Q] \qquad [P\ \neg Q], [\neg P\ Q]}{[P\ \neg Q], [\neg P\ Q]}\ \textbf{Split}$$

Example 1

The following formula is valid.

$$(P \supset (Q \supset R)) \supset ((P \supset Q) \supset (P \supset R))$$

Its negation is equivalent to the following clause set.

$$\{[P], [\neg R], [\neg P\ Q], [\neg P\ \neg Q\ R]\}$$

It is unsatisfiable, hence it should be provable.

We show unsatisfiability using only **Res**.

$$\dfrac{\dfrac{\dfrac{\overset{\times}{[P], [\neg R], [\neg P\ Q], [\neg P\ \neg Q\ R]}}{[P], [\neg R], [\neg P\ Q], [\neg P\ \neg Q\ R]}\ \textbf{Res}}{[P], [\neg R], [\neg P\ Q], [\neg P\ \neg Q\ R]}\ \textbf{Res}}{[P], [\neg R], [\neg P\ Q], [\neg P\ \neg Q\ R]}\ \textbf{Res}$$

Example 2

$\{[\neg P\ Q], [P\ \neg Q\ R], [Q\ S], [P\ \neg R]\}$ is satisfiable:

$$
\dfrac{
  \dfrac{
    \dfrac{\emptyset}{[P\ R],[P\ \neg R]}\ \textbf{Mon}
    \qquad
    \dfrac{\dfrac{\emptyset}{[\neg R]}\ \textbf{Mon}}{[\neg P],[P\ \neg R]}\ \textbf{Res}
  }{[\neg P\ Q],[P\ \neg Q\ R],[P\ \neg R]}\ \textbf{Split}
}{[\neg P\ Q],[P\ \neg Q\ R],[Q\ S],[P\ \neg R]}\ \textbf{Mon}
$$

Derivable rules

Res is, in fact superfluous, and can be derived from **Split**:

$$\dfrac{\dfrac{\times}{\bot, \Delta} \qquad \Lambda, \Delta}{[x], \Lambda_{\overline{x}}, \Delta} \; \textbf{Split}$$

If we allow Γ and Λ to be empty, the following rule is called *Unit propagation* (on x).

$$\dfrac{\Lambda, \Delta}{[x], \Gamma_x, \Lambda_{\overline{x}}, \Delta} \; \textbf{Prop}$$

It can be derived from the other rules.

Unit propagation

We can derive **Prop** as follows.

If Γ and Λ are non-empty:

$$\dfrac{\dfrac{\Lambda, \Delta}{[x], \Lambda_{\overline{x}}, \Delta}\ \textbf{Res}}{[x], \Gamma_x, \Lambda_{\overline{x}}, \Delta}\ \textbf{Sub}$$

If $\Lambda = \emptyset$, then $\Lambda_{\overline{x}} = \emptyset$:

$$\dfrac{\Delta}{[x], \Gamma_x, \Delta}\ \textbf{Mon}$$

If $\Gamma = \emptyset$, then $\Gamma_x = \emptyset$:

$$\dfrac{\Lambda, \Delta}{[x], \Lambda_{\overline{x}}, \Delta}\ \textbf{Res}$$

Termination

Lemma 9

A maximal derivation ends in an axiom or $\emptyset$.

Proof.

Assume the opposite: that there is a maximal derivation whose leaf node Γ is neither an axiom nor $\emptyset$.

- Thus there is some x occurring in Γ.
- If $\overline{x}$ does not occur in Γ, **Mon** is applicable.
- If $\overline{x}$ does occur in Γ, **Split** (or in some cases **Sub**) is applicable.

In either case we get a contradiction, as the derivation is not maximal. □

Termination

Theorem 10 (Termination)

Any proof attempt terminates.

Proof.

- **Sub** and **Mon** both reduce the number of clauses.
- **Split** reduces the number of distinct variables.
- Both are finite, thus we have termination. $\square$

Soundness and completeness

Lemma 11 (**Mon**)

Γ_x, Δ *is satisfiable iff* Δ *is satisfiable.*

Proof.

Only if: Follows directly from Lemma 4.

If: Assume that Δ is satisfiable.

- By Lemma 5, there is a v such that $v(\Delta) = v(x) = 1$.
- By Lemma 6(1), $v(\Gamma_x) = 1$.
- Thus $v(\Gamma_x, \Delta) = 1$. □

Soundness and completeness

Lemma 12 (**Sub**)

$[x], \Gamma_x, \Lambda_{\overline{x}}, \Delta$ is satisfiable iff $[x], \Lambda_{\overline{x}}, \Delta$ is satisfiable.

Proof.

Only if: Follows directly from Lemma 4.

If: Follows from Lemma 6(1). □

Lemma 13 (**Split**)

$\Gamma_x, \Lambda_{\overline{x}}, \Delta$ is satisfiable iff Γ, Δ or Λ, Δ are satisfiable.

Proof.

Left as exercise. □

Soundness and completeness

Theorem 14 (Soundness)

If there exists a proof of Γ, then Γ is unsatisfiable.

Proof.

We show this contrapositively: if Γ is satisfiable, then Γ is not provable.

- Assume that Γ is satisfiable.
- Rules preserve satisfiability upwards.
- Thus any derivation π has at least one satisfiable leaf node Λ.
- As axioms are unsatisfiable, Λ is not an axiom, thus π is not a proof. $\square$

Soundness and completeness

Theorem 15 (Completeness)

If Γ is unsatisfiable, there exists a proof of Γ.

Proof.

We show this contrapositively: if there exists no proof of Γ, then Γ is satisfiable.

- Assume that there exists no proof of Γ.
- Then any maximal derivation has at least one open leaf node.
- Termination lets us assume that a derivation is maximal, hence with an open leaf node $\emptyset$, which is satisfiable.
- Rules preserve satisfiability downwards.
- Thus Γ is satisfiable. □

NP-completeness

The first problem to be proven NP-complete was SAT.

Theorem 16 (Cook's Theorem)

SAT is NP-complete.

Proof.

Non-trivial. See [1] or [8]. □

We know from the previous lecture (in 2008 at least) that propositional satisfiablity is NP-complete.

- NP-hardness: follows directly from Cook's Theorem.
- NP-membership: a non-deterministic machine can guess a satisfying valuation and verify it in polynomial time.

Size

A problem (instance) is an instance of SAT, ie. a clause set. If

- the number of clauses is n,
- there occurs m distinct propositional variables, and
- every clause is of length $\leqslant c$,

the problem size is represented by the triple

$$n \times m \times c.$$

Example. The following problem has size $2 \times 4 \times 3$.

$$\{[P \neg Q\ R], [Q\ R \neg S]\}$$

Reduction to CNF

As mentioned, reducing a propositional formula to CNF can cause exponential increase in size.

A formula of the form $(x_1 \land y_1) \lor \cdots \lor (x_n \land y_n)$ reduced to CNF has size

$$2^n \times 2n \times n,$$

that is 2^n clauses of length n.

Example 17

If $n = 3$, we get a $8 \times 6 \times 3$ problem:

$$(x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_2 \lor y_3) \land (x_1 \lor x_3 \lor y_2) \land (x_1 \lor y_2 \lor y_3) \land$$
$$(x_2 \lor x_3 \lor y_1) \land (x_2 \lor y_1 \lor y_3) \land (x_3 \lor y_1 \lor y_2) \land (y_1 \lor y_2 \lor y_3)$$

But the reason for using DPLL in the first place is effectivity!

Equivalence

- Two formulae X and Y are equivalent if

$$v(X) = v(Y) \text{ for every valuation } v.$$

- Equivalence can be expressed in our logical language.
  - Let $(X \equiv Y)$ denote $(X \supset Y) \land (Y \supset X)$.
- So far we have reduced a formula to an equivalent one on CNF:
  - $X \to Y$, where
  - X and Y are equivalent, and
  - Y is on CNF.
- This, in fact, is not strictly necessary.

Equisatisfiability

- For our purposes, it suffices that X and Y are equisatisfiable:

$$X \text{ is satisfiable iff } Y \text{ is satisfiable.}$$

- Until now, the procedure for generating input to DPLL has been
  - $X \xrightarrow{\textbf{NNF}} Y \xrightarrow{\textbf{CNF}} Z \xrightarrow{\textbf{Clause}} \Gamma$, where
  - X, Y, Z and Γ are equivalent, and
  - Z may be exponentially larger than Y.
- Our next approach is as follows.
  - $X \xrightarrow{\textbf{NNF}} Y \xrightarrow{\textbf{Tseitin}} \Gamma$, where
  - Y and Γ are *not* equivalent, and
  - Γ is no more than polynomially larger than X.

Tseitin encoding

Problem: given an arbitrary formula on NNF, find an equisatisfiable formula on CNF (or the corresponding clause set).

Solution: Represent each subformulae (except for literals) with a propositional variable, recursively.

Usually attributed to Tseitin [9].

Example 18

$((P \wedge \neg Q) \vee R)$ has two non-literal subformulae, one of which is itself.

$$\overbrace{(\underbrace{(P \wedge \neg Q)}_{P_2} \vee R)}^{P_1}$$

Tseitin encoding

- For each new variable P_k, we generate a formula expressing that P_k is equivalent to the formula it represents:
  - $(P_1 \equiv (P_2 \vee R))$
  - $(P_2 \equiv (P \wedge \neg Q))$
- In addition we want the variable expressing the entire formula, in our case P_1 to be true.
- Let φ denote the following conjunction.

$$
\begin{aligned}
&P_1 \wedge \\
&(P_1 \equiv (P_2 \vee R)) \wedge \\
&(P_2 \equiv (P \wedge \neg Q))
\end{aligned}
$$

- Then φ is equisatisfiable to $((P \wedge \neg Q) \vee R)$.

Tseitin encoding

- In fact $\models \varphi \supset ((P \land \neg Q) \lor R)$.
- If $v(\varphi) = 1$, then v makes the three conjuncts true:
  1. $v(P_1) = 1$
  2. $v(P_1 \equiv (P_2 \lor R)) = 1$
     - Thus $v(P_1) = v(P_2 \lor R) = 1$.
     - Thus $v(P_2) = 1$ or $v(R) = 1$.
  3. $v(P_2 \equiv (P \land \neg Q)) = 1$
     - Thus $v(P_2) = v(P \land \neg Q)$.
     - If $v(P_2) = 1$, then $v(P \land \neg Q) = 1$.
- Hence $v((P \land \neg Q) \lor R) = 1$.
- Observe that $\not\models ((P \land \neg Q) \lor R) \supset \varphi$.

Tseitin encoding

In order to convert φ to CNF, we use the following functions.

$$\langle x \wedge y \rangle^P = \{[\neg P\ x], [\neg P\ y], [P\ \overline{x}\ \overline{y}]\}$$
$$\langle x \vee y \rangle^P = \{[P\ \overline{x}], [P\ \overline{y}], [\neg P\ x\ y]\}$$
$$\langle x \supset y \rangle^P = \{[P\ x], [P\ \overline{y}], [\neg P\ \overline{x}\ y]\}$$

Lemma 19 (Clause representation)

$\langle X \rangle^P$ is equivalent to $(P \equiv X)$.

E.g., $\{[P\ \overline{x}], [P\ \overline{y}], [\neg P\ x\ y]\}$ is equivalent to $P \equiv (x \vee y)$.

Tseitin encoding

Recall that φ is the formula

$$P_1 \wedge (P_1 \equiv (P_2 \vee R)) \wedge (P_2 \equiv (P \wedge \neg Q)).$$

Using the lemma, φ is *equivalent* to the clause set

$$\{\{[P_1]\} \cup \langle P_2 \vee R\rangle^{P_1} \cup \langle P \wedge \neg Q\rangle^{P_2}\},$$

which again equals

$$\begin{aligned}\{&[P_1],\\ &[P_1\ \neg P_2], [P_1\ \neg R], [\neg P_1\ P_2\ R],\\ &[\neg P_2\ P], [\neg P_2\ \neg Q], [P_2\ \neg P\ Q]\}.\end{aligned}$$

Tseitin encoding

Is this any better than the original CNF translation?

- We will use the number of binary connectives (n) as a measure of the size of our original formula on NNF.
- We let m denote the number of distinct propositional variables.
- Then the size of the equisatisfiable clause set generated is

$$(3n + 1) \times (m + n) \times 3.$$

Hence we obtain an instance of 3SAT with a linear (in n) number of clauses, with n new variables.

Pseudocode algorithm

DPLL can be implemented as follows, where DPLL(Γ) $= \mathrm{true}$ iff Γ is satisfiable.

proc LookAhead(Γ)
  while Γ contains unit clause $[x]$
    perform unit propagation on x

proc DPLL(Γ)
  LookAhead(Γ)
  if $\Gamma = \emptyset$ **return** true
  if $\perp \in \Gamma$ **return** false
  $x :=$ ChooseLiteral(Γ)
  return DPLL($\Gamma, [x]$) **or** DPLL($\Gamma, [\overline{x}]$)

Jeroslow Wang heuristic

- The only non-deterministic part is which literal is chosen.
- Picking the optimal literal is in general NP-hard *and* coNP-hard [7].
- Thus it is *harder* than deciding satisfiability of the formula!
- But there exists heuristics.
- Let $\Gamma(x)$ denote the subset of Γ where x occurs.
- Pick the x that maximizes $w(\Gamma(x))$, where w is the weight function

$$w(\Gamma) = \sum_{k \geqslant 1} \frac{n(\Gamma, k)}{2^k},$$

  and $n(\Gamma, k)$ is the number of clauses in Γ of length k.
- "Pick an x that occurs in many short clauses."

Example 2

Let $\Gamma = \{[\neg P\ Q], [P\ \neg Q\ R], [Q\ S], [P\ \neg R]\}$.

What is DPLL(Γ)?

- Γ contains no unit clause.
- We calculate $w(\Gamma(x))$ for each x occurring in Γ.

x	$\neg P$	P	$\neg Q$	Q	$\neg R$	R	$\neg S$	S
$w(\Gamma(x))$	$\frac{2}{8}$	$\frac{3}{8}$	$\frac{1}{8}$	$\frac{4}{8}$	$\frac{2}{8}$	$\frac{1}{8}$	$\frac{0}{8}$	$\frac{2}{8}$

- Q has the highest weight in Γ.
- DPLL(Γ) is true if DPLL($\Gamma, [Q]$) or DPLL($\Gamma, [\neg Q]$) are.

Example 2

- Unit propagation is performed on $\Gamma, [\neg Q]$:

$$\frac{[P\ R], [P\ \neg R]}{\Gamma, [Q]}\ \textbf{Prop}$$

- Let $\Gamma' = \{[P\ R], [P\ \neg R]\}$.

x	$\neg P$	P	$\neg R$	R
$w(\Gamma'(x))$	$\frac{0}{4}$	$\frac{2}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

- P has the highest weight in Γ'.

Example 2

- DPLL(Γ) is true if
  - DPLL($\Gamma', [P]$) or
  - DPLL($\Gamma', [\neg P]$) or
  - DPLL($\Gamma, [\neg Q]$) are.
- Unit propagation is performed on $\Gamma', [P]$:

$$\cfrac{\cfrac{\emptyset}{[\neg R]}\ \textbf{Prop}}{\Gamma', [P]}\ \textbf{Prop}$$

- DPLL($\Gamma', [P]$) returns true, thus
- DPLL(Γ) returns true, which means
- Γ is satisfiable.

MiniSAT

MiniSAT won the following categories at SAT Competition 2005:

- Industrial SAT+UNSAT
- Industrial UNSAT
- Industrial SAT
- Crafted UNSAT

It didn't do that well at SAT Competition 2007 though.

We can try it on an $3358 \times 1015 \times 3$ problem.

MiniSAT

```
villasayas: MiniSat_v1.14> ./minisat ../DIMACS/par16-5.cnf
==================================[MINISAT]===================================
| Conflicts |     ORIGINAL     |              LEARNT              | Progress |
|           | Clauses Literals |   Limit Clauses Literals  Lit/Cl |          |
==============================================================================
|         0 |    2218     6602 |     739       0        0     nan |  0.000 % |
|       102 |    2218     6602 |     812     102      953     9.3 | 38.227 % |
|       252 |    2218     6602 |     894     252     3313    13.1 | 38.227 % |
|       477 |    2218     6602 |     983     477     5729    12.0 | 38.227 % |
|       814 |    2218     6602 |    1081     814     9112    11.2 | 38.227 % |
|      1321 |    2218     6602 |    1190    1321    13584    10.3 | 38.227 % |
|      2081 |    2218     6602 |    1309    1292    10791     8.4 | 38.227 % |
|      3220 |    2220     6602 |    1440    1576    12234     7.8 | 38.227 % |
==============================================================================
restarts              : 8
conflicts             : 4670           (11121 /sec)
decisions             : 4911           (11695 /sec)
propagations          : 1075868        (2561981 /sec)
conflict literals     : 39668          (36.40 % deleted)
Memory used           : 2.97 MB
CPU time              : 0.419936 s

SATISFIABLE
villasayas: MiniSat_v1.14>
```

Bibliography I

S. A. Cook.
The complexity of theorem-proving procedures.
In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
Commun. ACM, 5(7):394–397, 1962.

Bibliography II

M. Davis and H. Putnam.

A Computing Procedure for Quantification Theory.

J. ACM, 7(3):201–215, 1960.

N. Eén and N. Sörensson.

An Extensible SAT-solver.

In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

M. Fitting.

First-Order Logic and Automated Theorem Proving.

Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2. edition, 1996.

Bibliography III

R. G. Jeroslow and J. Wang.

Solving Propositional Satisfiability Problems.

Annals of Mathematics and Artificial Intelligence, 1(1):167–187, 1990.

P. Liberatore.

On the complexity of choosing the branching literal in DPLL.

Artificial Intelligence, 116(1-2):315–326, 2000.

C. H. Papadimitriou.

Computational Complexity.

Addison Wesley, Reading, Massachusetts, 1994.

Bibliography IV

G. S. Tseitin.

On the Complexity of Derivation in Propositional Calculus.

In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, Berlin, Heidelberg, 1983.