INF3170 – Logikk

Forelesning 3: SAT and DPLL

Espen H. Lian

Institutt for informatikk, Universitetet i Oslo

24. september 2013

Dagens plan

Introduction

- SAT is the problem of determining if a propositional formula is satisfiable.
- SAT can also refer to the problem of determining if a propositional formula on *conjunctive normal form* is satisfiable.
- Both problems are NP-complete.
- 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 [1].
- We present (a version of) DPLL as a calculus.
- DPLL is interesting because it works well in practice, ie. some of the best SAT solvers are based on DPLL.

Preliminaries

A literal is a propositional variable or its negation.

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.

$$\neg\neg X \to X$$

$$X \supset Y \to \neg X \lor Y$$

$$\neg(X \land Y) \to \neg X \lor \neg Y$$

$$\neg(X \lor Y) \to \neg X \land \neg Y$$

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

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

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."

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

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 of the left conjunct can be performed as follows.

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

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:

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

We will deal with the increase in size later.

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 finite set $\{x_1, \ldots, x_n\}$ of literals,
  - written as $[x_1 \ldots x_n]$, and
  - interpreted disjunctively.
- A unit clause is a singleton clause
  - i.e. of the form $[x]$.
- A clause set is a finite set $\{C_1, \ldots, C_n\}$ of clauses,
  - interpreted conjunctively.

We use '[' and ']' for *clauses*, and '{' and '}' for *clause sets* because they are interpreted differently:

$$v([x_1 \ldots x_n]) = v(x_1) \vee \cdots \vee v(x_n)$$
$$v(\{C_1, \ldots, C_n\}) = v(C_1) \wedge \cdots \wedge v(C_n)$$

Example

Some clauses and the formulae they represent:

1. $[P \neg Q \; R]$ — $P \vee \neg Q \vee R$
2. $[P \neg P]$ — $P \vee \neg P$
3. $[\,]$, the empty clause — $\bot$, the empty disjunction

Some clause sets and the formulae they represent:

1. $\{[P \neg Q \; R]\}$ — $P \vee \neg Q \vee R$
2. $\{[P \neg P], [\,], [P \neg Q \; R]\}$ — $(P \vee \neg P) \wedge \bot \wedge (P \vee \neg Q \vee R)$
3. $\{\}$, the empty clause set — $\top$, the empty conjunction

Clauses and clause sets

We will use the following notation.

- clauses: C, D (possibly subscripted)
- clause sets: Γ, Δ, Λ

We will also write $\bot$ for $[\,]$, and $\varnothing$ for $\{\}$.

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. $\varnothing_x = \varnothing$.

Subsumption

If $C \subseteq D$, we say that C subsumes D.

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

Subsumption Lemma

If C subsumes D, then $v(C) = 1$ implies $v(D) = 1$.

Proof.

- If $v(C) = 1$, then $v(x) = 1$ for some $x \in C$.
- If $C \subseteq D$, then $x \in D$, thus $v(x) = 1$ for some $x \in D$.
- Hence $v(D) = 1$. □

Introduction

The DPLL calculus operates not on formulae but on a clause sets.

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 $C \in \Gamma$.

- $\neg Q$ occurs in $\{[P\ \neg Q], [\neg P\ R]\}$, while Q does not.

In derivations we drop '$\{$' and '$\}$' from clause sets.

Introduction

A branch is closed if the empty clause occurs in its leaf node.

An example derivation is:

$$\dfrac{\dfrac{\begin{array}{c}\times\\ [\,],[S]\end{array}}{[Q],[\neg Q],[S]} \qquad \dfrac{[Q],[S]}{[Q],[\neg R],[S]}}{[P\ \ Q],[P\ \neg Q],[\neg P\ \ Q],[\neg P\ \neg R],[S]}$$

The left branch is closed; the right branch is not.

The Idea

The main idea is to try to satisfy the clause set.

If we make a literal x true, we can

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

Example

Let $\Gamma = \{[P\ Q], [\neg P\ \neg Q], [Q\ \neg R]\}$. *If* $v(P) = 1$, *we can*

- *remove* $[P\ Q]$ *from* Γ, *and*
- *remove* $\neg P$ *from* $[\neg P\ \neg Q]$.

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

The Idea

We start by removing

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

This does not affect satisfiability.

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

- Γ and Λ are non-empty.

Then given the clause set $\Gamma_x, \Lambda_{\overline{x}}, \Delta$,

- Γ_x is the subset where x occurs;
- $\Lambda_{\overline{x}}$ is the subset where $\overline{x}$ occurs;
- Δ is the subset where neither occur.

Monotone literal fixing

We say that x is monotone in a clause set if it is the case that

- x occurs in some clauses and
- $\overline{x}$ does not occur in any clause.

If x is monotone in a clause set, we make x true, because this makes the clauses x occurs in true and does not affect the other clauses.

Monotone literal fixing

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

This rule is also called the Affirmative-Negative Rule.

Unit subsumption

Observe: $[x]$ subsumes every clause where x occurs.

If it is the case that

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

we may remove the clauses where x occurs (except $[x]$).

Unit subsumption

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

Examples

Example: $\neg Q$ is monotone in $[P\ \neg Q\ R], [\neg P\ \neg R], [P\ \neg R]$.

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

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 is the case that

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

make x true.

Unit resolution

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

Split

If it is the case that

- 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.

Split

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

Note: x is true in the right branch.

Examples

Example: Q occurs only in $[Q]$, while there are occurrences of $\neg Q$.

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

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.

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

In order to prove this, we negate the formula and rewrite it to CNF:

$$\begin{aligned}
&\neg((P \supset (Q \supset R)) \supset ((P \supset Q) \supset (P \supset R))) \\
\rightarrow\ &\neg(\neg(\neg P \lor (\neg Q \lor R)) \lor (\neg(\neg P \lor Q) \lor (\neg P \lor R))) \\
\rightarrow\ &\neg\neg(\neg P \lor (\neg Q \lor R)) \land (\neg\neg(\neg P \lor Q) \land (\neg\neg P \land \neg R)) \\
\rightarrow\ &(\neg P \lor \neg Q \lor R) \land (\neg P \lor Q) \land P \land \neg R \quad (\textbf{NNF}/\textbf{CNF})
\end{aligned}$$

This is equivalent to the following clause set.

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

Example 1

We prove unsatisfiability using only unit resolution.

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

Every branch is closed, thus we have a proof.

Example 2

$$\dfrac{\dfrac{\dfrac{\varnothing}{[P\ R],[P\ \neg R]}\,\textbf{Mon}^5 \qquad \dfrac{\dfrac{\varnothing}{[\neg R]}\,\textbf{Mon}^4}{[\neg P],[P\ \neg R]}\,\textbf{Res}^3}{[\neg P\ Q],[P\ \neg Q\ R],[P\ \neg R]}\,\textbf{Split}^2}{[\neg P\ Q],[P\ \neg Q\ R],[Q\ S],[P\ \neg R]}\,\textbf{Mon}^1$$

1. S is monotone
2. Split on $\neg Q$
3. Unit resolution on $\neg P$
4. $\neg R$ is monotone
5. P is monotone

The rules

These are all the rules.

Monotone literal fixing

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

Unit subsumption

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

Unit resolution

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

Split

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

Derived rules

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

Unit propagation

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

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

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

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

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

Soundness

Recall that a proof is a closed derivation.

Theorem (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 the empty clause is unsatisfiable, π is not closed,

thus π is not a proof. □

Maximal Derivations

Recall that a maximal derivation is one where no rule is applicable.

Lemma

A leaf node in a maximal derivation is either $\varnothing$ or contains the empty clause.

Proof.

Let Γ be a leaf node in a derivation π. We show the following:

- If Γ is neither $\varnothing$ nor contains the empty clause, then π is not maximal.

Assume that Γ is neither $\varnothing$ nor contains the empty clause.

- Then there is some literal 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, π is not maximal. □

Completeness

Theorem (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.

- Let π be a derivation.
- Termination $(*)$ lets us assume that π is maximal.
- Because π is not a proof, it contains at least one open leaf node Γ.
- By the lemma, $\Gamma = \varnothing$, which is satisfiable.
- Rules preserve satisfiability downwards, $(*)$

thus Γ is satisfiable. □

Size

A problem is an instance of SAT, i.e. a clause set. If

- the number of clauses is n,
- there occurs m distinct propositional variables, and
- every clause is of length k,

the problem size is defined as the triple

$$n \times m \times k.$$

Example

Some problems and their sizes:

- $\{[P\ \neg Q\ R], [Q\ R\ \neg S]\}$ *has size* $2 \times 4 \times 3$.
- $\{[P\ \neg Q], [\neg P\ Q], [P\ Q]\}$ *has size* $3 \times 2 \times 2$.

k-SAT and HORNSAT

Definition (k-SAT)

k-SAT is the subset of SAT with problems of size $n \times m \times k$.

Example: 3-SAT:

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

Definition (HORNSAT)

HORNSAT is the subset of SAT where every clause is a Horn clause, i.e. contains at most one positive literal.

Example: Both HORNSAT and 2-SAT:

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

k-SAT and HORNSAT

The complexity k-SAT and HORNSAT is well-known:

- 3-SAT is **NP**-complete.
- 2-SAT is **NL**-complete.
- HORNSAT is **P**-complete.

The relationship between the classes is as follows.

$$\begin{array}{l} \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \\ \mathbf{NL} \neq \qquad\qquad\quad \mathbf{PSPACE} \end{array}$$

Hence

- 2-SAT is not harder than HORNSAT, and
- HORNSAT is not harder than 3-SAT.

Reduction to CNF

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

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

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

that is 2^n clauses of length n.

Example

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

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee y_3) \wedge (x_1 \vee x_3 \vee y_2) \wedge (x_1 \vee y_2 \vee y_3) \wedge$$
$$(x_2 \vee x_3 \vee y_1) \wedge (x_2 \vee y_1 \vee y_3) \wedge (x_3 \vee y_1 \vee y_2) \wedge (y_1 \vee y_2 \vee y_3)$$

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

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 $(X \supset Y) \wedge (Y \supset X)$. Then

$$v(X \equiv Y) = 1 \text{ iff } X \text{ and } Y \text{ are equivalent.}$$

So far we have reduced a formula to an equivalent one on CNF:

- $X \xrightarrow{\textbf{CNF}} Y$, where
- X and Y are equivalent, and
- Y is on CNF.

This is, in fact, 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

- $X \xrightarrow{\textbf{NNF}} Y \xrightarrow{\textbf{CNF}} Z$, where
- X, Y, and Z are equivalent, and
- Z may be exponentially larger than Y.

Our next approach is as follows.

- $X \xrightarrow{\textbf{NNF}} Y \xrightarrow{\textbf{CNF}} Z$, where
- Y and Z are *not* equivalent, but equisatisfiable, and
- Z is no more than polynomially larger than Y.

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 new propositional variable, recursively.
Usually attributed to Tseitin [3].

Example

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

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

Tseitin encoding

For each subformula X, introduce a new variable P_k and generate a formula expressing that P_k is equivalent to X:

- $(P_1 \equiv (P_2 \vee R))$ [not $(P_1 \equiv ((P \wedge \neg Q) \vee R))$]
- $(P_2 \equiv (P \wedge \neg Q))$

In addition we want the variable representing the entire formula – in our case P_1 – to be true. The result is:

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

The formula above and $((P \wedge \neg Q) \vee R)$ are both satisfiable, but they are not equivalent.

Tseitin encoding

In order to convert $P_1 \wedge (P_1 \equiv (P_2 \vee R)) \wedge (P_2 \equiv (P \wedge \neg Q))$ to CNF, we use the following functions.

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

Lemma (Clause representation)

$[x * y]^P$ *is equivalent to* $P \equiv (x * y)$ *for* $* \in \{\wedge, \vee\}$.

Example: $P_2 \equiv (P \wedge \neg Q)$ is equivalent to

- $[P \wedge \neg Q]^{P_2}$, which equals
- $\{[\neg P_2\ P], [\neg P_2\ \neg Q], [P_2\ \neg P\ Q]\}$.

Tseitin encoding

In conclusion:

$((P \wedge \neg Q) \vee R)$ is equisatisfiable to

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

which is *equivalent* to

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

which *equals* the clause set

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

Tseitin encoding

Tseitin encoding:

$$\dfrac{\dfrac{\dfrac{\dfrac{\varnothing}{[\neg P\ Q]}\ \textbf{Mon} \qquad \dfrac{\dfrac{\varnothing}{[P]}\ \textbf{Mon}}{[P], [\neg Q]}\ \textbf{Mon}}{[\neg P_2\ P], [\neg P_2\ \neg Q], [P_2\ \neg P\ Q]}\ \textbf{Split}}{[P_2\ R], [\neg P_2\ P], [\neg P_2\ \neg Q], [P_2\ \neg P\ Q]}\ \textbf{Mon}}{[P_1], [P_1\ \neg P_2], [P_1\ \neg R], [\neg P_1\ P_2\ R], [\neg P_2\ P], [\neg P_2\ \neg Q], [P_2\ \neg P\ Q]}\ \textbf{Prop}$$

Equivalent CNF encoding:

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

Tseitin encoding

Is this any better (in general) 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

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

- This means that that there are
  - $3n + 1$ clauses,
  - m auxiliary variables, and
  - each clause has at most length 3.

Pseudocode algorithm

A minimal version of DPLL can be implemented as follows.

```
proc LookAhead(Γ)
    while Γ contains some unit clause [x]
        perform unit propagation on x
    return Γ

proc DPLL(Γ)
    Γ := LookAhead(Γ)
    if Γ = ∅ return 1
    if ⊥ ∈ Γ return 0
    x := ChooseLiteral(Γ)
    return DPLL(Γ, [x]) or DPLL(Γ, [x̄])
```

Correctness

Correctness of the algorithm

DPLL(Γ) returns 1 if Γ is satisfiable, and 0 if not.

- The idea is that branching and adding a unit clause $[x]$ to one branch and $[\overline{x}]$ to the other, and then performing unit propagation is basically the same as splitting:

$$\dfrac{\dfrac{\Gamma, \Delta}{[\overline{x}], \Gamma_x, \Lambda_{\overline{x}}, \Delta} \qquad \dfrac{\Lambda, \Delta}{[x], \Gamma_x, \Lambda_{\overline{x}}, \Delta}}{\Gamma_x, \Lambda_{\overline{x}}, \Delta}$$

- (This is not a proof in the calculus.)
- If x is monotone, it gets a little trickier.

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 [4].
- Thus it is *harder* than deciding satisfiability of the formula!
- But there exists heuristics [5].
- Let $\Gamma|_x$ denote the subset of Γ where x occurs: $\{C \in \Gamma \mid x \in C\}$
- 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

Let us apply the algorithm to

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

What is $\texttt{DPLL}(\Gamma)$?

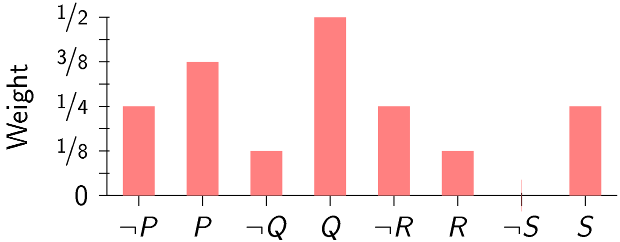
- Γ contains no unit clause, thus $\texttt{LookAhead}(\Gamma)$ returns Γ, and
- Γ is neither empty nor contains the empty clause,
- hence we must choose some literal to split on.
- In order to do this, we apply the heuristic.

Example

- We calculate $w(\Gamma|_x)$ for each x occurring in

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

- E.g., the weight of P in Γ: $w(\Gamma|_P) = {}^0/_{2^1} + {}^1/_{2^2} + {}^1/_{2^3} = {}^3/_8$.



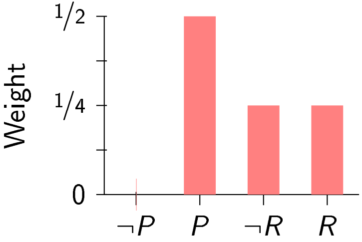
- Q has the highest weight in Γ.

Example

- We add $[Q]$ to Γ and perform unit propagation.

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

- We calculate $w(\Gamma'|_x)$ for each x occurring in $\Gamma' = \{[P\ R], [P\ \neg R]\}$.



- P has the highest weight in Γ'.

Example

- We add $[P]$ to Γ' and perform unit propagation.

$$\dfrac{\dfrac{\varnothing}{[\neg R]}\ \textbf{Prop}}{[P\ R],[P\ \neg R],[P]}\ \textbf{Prop}$$

- $\texttt{DPLL}([P\ R],[P\ \neg R],[P],[P])$ returns 1, thus so does
- $\texttt{DPLL}(\Gamma,[Q])$, thus so does
- $\texttt{DPLL}(\Gamma)$.
- Hence Γ is satisfiable.

SAT Solvers

- A SAT solver is a program that determines whether a propositional formula or clause set is satisfiable.
- Many modern SAT solvers are based on the SAT solver MiniSAT, which again is based on DPLL.
- MiniSAT won all the industrial categories at SAT 2005.

- We can try it on an $3030 \times 1015 \times 3$ problem.

MiniSAT

```
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:          1015                                         |
|  Number of clauses:            3030                                         |
|  Parse time:                   0.00 s                                       |
|                                                                             |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|       100 |     627     1932     5162 |      708      100     11 | 38.228 % |
|       250 |     627     1932     5162 |      779      250     12 | 38.227 % |
|       475 |     627     1932     5162 |      857      475     11 | 38.227 % |
|       812 |     627     1932     5162 |      942      812     10 | 38.227 % |
|      1318 |     627     1932     5162 |     1037     1318     10 | 38.227 % |
|      2077 |     627     1932     5162 |     1140     1359      9 | 38.227 % |
|      3216 |     627     1932     5162 |     1254      966      8 | 38.227 % |
|      4924 |     627     1932     5162 |     1380     1026      8 | 38.227 % |
===============================================================================
restarts              : 27
conflicts             : 4998           (15971 /sec)
decisions             : 5388           (0.00 % random) (17217 /sec)
propagations          : 1131352        (3615098 /sec)
conflict literals     : 44646          (31.69 % deleted)
Memory used           : 6.00 MB
CPU time              : 0.312952 s

SATISFIABLE
```

Bibliography I

[1] Martin Davis and Hilary Putnam, **A Computing Procedure for Quantification Theory**, *J. ACM*, 7(3):201–215, 1960.

[2] Martin Davis, George Logemann and Donald Loveland, **A machine program for theorem-proving**, *Commun. ACM*, 5(7):394–397, 1962.

[3] G. S. Tseitin, **On the Complexity of Derivation in Propositional Calculus**.

[4] Paolo Liberatore, **On the complexity of choosing the branching literal in DPLL**, *Artificial Intelligence*, 116(1–2):315–326, 2000.

[5] Robert G. Jeroslow and Jinchang Wang, **Solving Propositional Satisfiability Problems**, *Annals of Mathematics and Artificial Intelligence*, 1(1):167–187, 1990.