Propositional Resolution

Second Part: Algorithms

Stéphane Devismes Pascal Lafourcade Michel Lévy
Jean-François Monin (jean-francois.monin@imag.fr)

Université Joseph Fourier, Grenoble I

January 30, 2015

Last course

- Resolution

Reminder

(1) $A \vdash B$

B is deduced from A.
There is a proof by resolution of B starting from A

(2) $A \models B$

B is consequence of A.
Every model of A is also a model of B

Correctness

(1) $\Rightarrow$ (2)

Completeness

(2) $\Rightarrow$ (1)

Resolution

Correct and complete

Overview

Homework : solution

- (H1) : $p \Rightarrow \neg j$, rewritten as $\neg p \vee \neg j$
- (H2) : $\neg p \Rightarrow j$, rewritten as $p \vee j$
- (H3) : $j \Rightarrow m$, rewritten as $\neg j \vee m$
- $(\neg$ C) : $\neg m \wedge \neg p$

Clauses : $\{\neg p \vee \neg j, p \vee j, \neg j \vee m, \neg m, \neg p\}$

$$\dfrac{\dfrac{\dfrac{p \vee j \qquad \neg j \vee m}{p \vee m} \qquad \neg m}{p} \qquad \neg p}{\perp}$$

Presentation of the two algorithms

How to « systematically » decide whether Γ is inconsistent or not ?

- Complete strategy
  Construction of ALL the deductible clauses (resolvents) from Γ
- The Davis-Putnam-Logemann-Loveland Algorithm
  « Intelligent » traversal of the possible assignments of Γ

Remark

Exponential solutions in time in the worst case.

Exponential complexity

Consider that two clauses having the same set of literals are equal.

If the length of $s(\Gamma) = n$, then we have at most 2^n clauses deduced from Γ.

Reduction of a set of clauses

In order to accelerate the algorithm, we reduce the set of clauses.

How to proceed with reduction ?

Remove the valid clauses and the clauses containing another clause of the set.

A set of clauses is reduced if it is not reducible anymore.

Reduced set of clauses

Definition 2.1.26

A set of clauses is reduced if it does not contain any valid clause and none of the clauses is included in another clause.

Example 2.1.27

The reduction of the set of clauses
$\{p \vee q \vee \neg p,\ p \vee r,\ p \vee r \vee \neg s,\ r \vee q\}$ gives the reduced set :

Justification

Property 2.1.28

A set of clauses E is equivalent to the reduced set of clauses obtained from E.

Proof.

$\square$

Result of the algorithm : minimum deduction clauses

Definition 2.1.29

Let Γ a set of clauses. A minimum clause for the deduction from Γ is a non-valid clause deduced from Γ and *strictly* not containing any clause deduced from Γ.

Example 2.1.30

Let us consider the set of clauses $\Gamma = \{a \vee \neg b, b \vee c \vee d\}$ the clause $a \vee c \vee d$ is a minimum deduction clause.

However, if we add the clause $\neg a \vee c$ to Γ then $a \vee c \vee d$ is not a minimum clause since we can deduce $c \vee d$ which is included in the clause $a \vee c \vee d$.

Property

Property 2.1.31

Let Θ the set of minimum deduction clauses for the set of clauses Γ.
The set Γ is unsatisfiable if and only if $\perp \in \Theta$.

Proof.

- Suppose $\perp \in \Theta$, then $\Gamma \vdash \perp$, hence by resolution correctness, Γ is unsatisfiable.
- Suppose Γ unsatisfiable, by resolution completeness, $\Gamma \vdash \perp$. Consequently $\perp$ is minimum clause for the deduction of Γ, therefore $\perp \in \Theta$.

$\square$

Interpretation

When the following algorithm terminates :

- $\bot \in \Theta_k$: Γ is unsatisfiable
- $\bot \notin \Theta_k$: Γ is satisfiable, but what does Θ_k represent ?

Θ_k = minimum clauses for the consequence

Definition 2.1.32

Let Γ a set of clauses. A minimum clause for the consequence of Γ is a non valid consequence clause of Γ *strictly* not containing any consequence clause of Γ.

Theorem 2.1.35

Let Γ a set of clauses. A clause is minimum for the deduction of Γ if and only if it is minimum for the consequence of Γ.

Proofs given in the course support.

Example

Example 2.1.33

Consider the set of clauses $\Gamma = \{a \vee d, \neg a \vee b, \neg b \vee c\}$. The clause $d \vee c$ is minimum for the consequence of Γ.

Consequence : $d \vee c$ is a consequence of Γ since in all model of Γ, either d is true or c is true.

Minimality : There exist models of Γ which are not models of d (respectively c) : $a \mapsto 1$, $d \mapsto 0$, $c \mapsto 1$ and $b \mapsto 1$ (respectively $a \mapsto 0$, $d \mapsto 1$, $c \mapsto 0$ and $b \mapsto 0$).

Principle of the algorithm : Construct all the clauses deduced from Γ

Following the height of the proof trees.

Algorithm

For any integer i
While it is possible to construct new clauses
Construct the reduced set of all the clauses having a proof tree of height at most i.

In practice :
Maintain two sequences of the sets of clauses, $\Delta_{i(i \geq 0)}$ and $\Theta_{i(i \geq 0)}$

Two sequences of sets of clauses

$\Delta_{i(i \geq 0)}$

Clauses deduced from Γ by a proof of height i, after clauses removal :

- valid clauses
- clauses including another clause of the proof of height at most i.

Δ_0 is obtained by reducing Γ

Two sequences of sets of clauses

$\Theta_{i(i \geq 0)}$

Clauses deduced from Γ by a proof of height less than i after clauses removal :

- valid clauses
- clauses including another clause of the proof of height at most i.

Θ_0 is the empty set.

Details of the method

If $\Delta_k = \emptyset$, stop the construction :

- $k - 1$ is then the maximum height of a proof
- Θ_k is the reduced set of the clauses deduced from Γ

Construction of the sequences $\Delta_{i(i\geq 0)}$ and $\Theta_{i(i\geq 0)}$

Δ_{i+1}

- Construct all the resolvents of Δ_i and $\Delta_i \cup \Theta_i$
- Reduce this set
- Remove the new resolvents including a clause of $\Delta_i \cup \Theta_i$

Θ_{i+1}

Remove from $\Delta_i \cup \Theta_i$ the clauses which include one of the clauses of Δ_{i+1}.

Example 2.2.1

Let $\Gamma = \{a \vee b \vee \neg a,\ a \vee b,\ a \vee b \vee c,\ a \vee \neg b,\ \neg a \vee b,\ \neg a \vee \neg b\}$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Resolvents of Δ_i and $\Delta_i \cup \Theta_i$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Resolvents of Δ_i and $\Delta_i \cup \Theta_i$
0	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	$\emptyset$	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	b, $b \vee \neg b$, a, $a \vee \neg a$, $\neg a$, $\neg b$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Resolvents of Δ_i and $\Delta_i \cup \Theta_i$
0	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	$\emptyset$	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	b, $b \vee \neg b$, a, $a \vee \neg a$, $\neg a$, $\neg b$
1	a, b, $\neg b$, $\neg a$	$\emptyset$	a, b, $\neg b$, $\neg a$	$\perp$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Resolvents of Δ_i and $\Delta_i \cup \Theta_i$
0	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	$\emptyset$	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	b, $b \vee \neg b$, a, $a \vee \neg a$, $\neg a$, $\neg b$
1	a, b, $\neg b$, $\neg a$	$\emptyset$	a, b, $\neg b$, $\neg a$	$\perp$
2	$\perp$	$\emptyset$	$\perp$	$\emptyset$

Example 2.2.1 (contd.)

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	Resolvents of Δ_i and $\Delta_i \cup \Theta_i$
0	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	$\emptyset$	$a \vee b$, $a \vee \neg b$, $\neg a \vee b$, $\neg a \vee \neg b$	b, $b \vee \neg b$, a, $a \vee \neg a$, $\neg a$, $\neg b$
1	a, b, $\neg b$, $\neg a$	$\emptyset$	a, b, $\neg b$, $\neg a$	$\perp$
2	$\perp$	$\emptyset$	$\perp$	$\emptyset$
3	$\emptyset$	$\perp$		

Example 2.2.2

$$\{a, c, \neg a \vee \neg b, \neg c \vee e\}$$

i	Δ_i	Θ_i	$\Delta_i \cup \Theta_i$	$\Delta_i \sharp (\Delta_i \cup \Theta_i)$
0	$a, c, \neg a \vee \neg b, \neg c \vee e$	$\emptyset$	$a, c, \neg a \vee \neg b, \neg c \vee e$	$e, \neg b$
1	$e, \neg b$	a, c	$a, \neg b, c, e$	$\emptyset$
2	$\emptyset$	$a, \neg b, c, e$		

Termination of the algorithm : idea

There are at most 2^n clauses deduced from Γ.

$\Delta_{i(i \geq 0)}$ contains only clauses deduced from Γ

$\Delta_{i(i \geq 0)}$ are mutually disjoint (To demonstrate)

Hence there are at most $2^n + 1$ sets, therefore $k \leq 2^n + 1$

$\Delta_{i(i \geq 0)}$ are mutually disjoint

Property 2.2.3

Let $i \leq k$. Any clause of $\bigcup_{j \leq i} \Delta_j$ contains a clause of $\Delta_i \cup \Theta_i$.

Proof.

By induction.

- For $i = 0$ the property is trivial since $\Theta_0 = \emptyset$.
- Suppose the property true for i, let us show that it is also true for $i + 1$. Let $C \in \bigcup_{j \leq i+1} \Delta_j$. Let us show that C contains a clause of $\Delta_{i+1} \cup \Theta_{i+1}$. We examine all the possible cases for C.
    1. $C \in \Delta_{i+1}$. Hence C contains a clause of $\Delta_{i+1} \cup \Theta_{i+1}$.
    2. $C \in \bigcup_{j \leq i} \Delta_j$. By induction hypothesis, C contains a clause $D \in \Delta_i \cup \Theta_i$. We distinguish two situations for D.
        - 2.1 $D \in \Theta_{i+1}$. Hence C contains a clause of $\Delta_{i+1} \cup \Theta_{i+1}$.
        - 2.2 $D \notin \Theta_{i+1}$. By construction of Θ_{i+1}, since $D \in \Delta_i \cup \Theta_i$ and $D \notin \Theta_{i+1}$, it means that D contains a clause of Δ_{i+1}. Or C contains D, hence C also contains a clause of $\Delta_{i+1} \cup \Theta_{i+1}$.

$\square$

$\Delta_{i(i \geq 0)}$ are mutually disjoint

Property 2.2.4

For all $i \leq k$, the sets Δ_i are mutually disjoint.

Proof.

We perform an induction on the sets Δ_j with $0 \leq j \leq i$ and $i \leq k$.

The base case (basis) : If $i = 0$, there is only one set, hence the property is verified.

Inductive step : Let $i < k$. Suppose that all the sets Δ_j where $j \leq i$ are mutually disjoint. Let us show that Δ_{i+1} is disjoint with respect to these sets.
Let $C \in \Delta_{i+1}$. Suppose, on the contrary, that $C \in \bigcup_{j \leq i} \Delta_j$.
According to the previous property, C includes a clause of $\Delta_i \cup \Theta_i$.
Hence by construction of Δ_{i+1}, $C \notin \Delta_{i+1}$, contradiction.
Consequently, $C \notin \bigcup_{j \leq i} \Delta_j$.

$\square$

Hence, the algorithm terminates.

Result of the algorithm

- Γ and Θ_k are equivalent
- $\Theta_k =$ set of minimum deduction clauses.

Γ and Θ_k are equivalent

Property 2.2.5

For all $i < k$, the sets $\Delta_i \cup \Theta_i$ and $\Delta_{i+1} \cup \Theta_{i+1}$ are equivalent.

Proof.

1. Any clause of $\Delta_{i+1} \cup \Theta_{i+1}$ is a consequence of $\Delta_i \cup \Theta_i$. Any clause of $\Delta_{i+1} \cup \Theta_{i+1}$ is an element of $\Delta_i \cup \Theta_i$ or a resolvent of two elements of this set, therefore it is a consequence of this set.
2. Any clause of $\Delta_i \cup \Theta_i$ is a consequence of $\Delta_{i+1} \cup \Theta_{i+1}$. Let $C \in \Delta_i \cup \Theta_i$. We distinguish two possible cases :
   - 2.1 $C \in \Theta_{i+1}$, thus C is a consequence of $\Delta_{i+1} \cup \Theta_{i+1}$.
   - 2.2 $C \notin \Theta_{i+1}$, thus C contains a clause of Δ_{i+1} hence is a consequence of $\Delta_{i+1} \cup \Theta_{i+1}$.

□

Γ and Θ_k are equivalent

Property 2.2.6

The sets Γ and Θ_k are equivalent.

Proof.

- Δ_0 is the set obtained by reduction of Γ, according to property 2.1.28, these two sets are equivalent.
- Since Θ_0 is empty, Γ is equivalent to $\Delta_0 \cup \Theta_0$.
- According to property 2.2.5 and by induction, $\Delta_0 \cup \Theta_0$ is equivalent to the set of clauses $\Delta_k \cup \Theta_k$.
- Since the algorithm terminates when Δ_k is the empty set, the sets Γ and Θ_k are equivalent.

$\square$

Θ_k = set of minimum deduction clauses

Property 2.2.13

Θ_k is the set of minimum deduction clauses of Γ.

Proof.

Cf. Course support (Poly) $\square$

Example from 1.6.2 : $maj(x, y, z) = (x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z)$.

History

The Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

- Introduced by Martin Davis and Hilary Putnam in 1960, then refined by Martin Davis, George Logemann and Donald Loveland in 1962
- Indicates if a set of clauses is satisfiable.
- Basis for (most efficient) complete SAT-solvers such as **chaff, zchaff and satz**.

Principle I

Two types of formulae transformation :

1. preserving the truth value : transforming a formula into an equivalent formula
   - reduction
2. preserving the satisfiability only : transforming a satisfiable formula into another satisfiable formula
   - removal of clauses containing isolated literals
   - unit resolution

DPLL is efficient since it uses these two transformations.

Principle II

≪ Branching/Backtracking ≫ (splitting rule)

- **Branching** : After simplification, assign to **true** a heuristically chosen variable (branching literal).
- Continue the algorithm recursively.
- **Backtracking** : If we arrive to a contradiction, we return to the last choice, and we ≪ branch ≫ by assigning **false** to the chosen variable.

Removal of clauses having isolated literals.

Definition 2.3.1 Isolated literal L

If none of the clauses of Γ contains L^c.

Lemme 2.3.2

Removing clauses with an isolated literal preserves the satisfiability.

Proof : see exercise 48.

Example 2.3.3

Let Γ the set of clauses

(1) $p \vee q \vee r$

(2) $\neg q \vee \neg r$

(3) $q \vee s$

(4) $\neg s \vee t$

Simplify Γ by removing clauses having isolated literals.

Unit resolution

Definition 2.3.4

A unit clause is a clause which contains only one literal.

Lemma 2.3.5

Let L the set of literals of the unit clauses of Γ. Let Θ the set of clauses obtained starting from Γ, as follows

- if L contains two complementary literals, then $\Theta = \{\perp\}$.
- else Θ is obtained as follows
  - removing the clauses containing an element of L
  - in the remaining clauses, remove the complementary literals of the elements of L

Γ has a model if and only if Θ has a model.

Proof : The proof is requested in exercise 49.

Example 2.3.6 Unit resolution

Simplify the following sets of clauses by unit resolution :

- **Let Γ the set of clauses :** $p \vee q$, $\neg p$, $\neg q$
- **Let Γ the set of clauses :** $a \vee b \vee \neg d$, $\neg a \vee c \vee \neg d$, $\neg b$, d, $\neg c$**.**
- **Let Γ' the set of clauses :** p, q, $p \vee r$, $\neg p \vee r$, $q \vee \neg r$, $\neg q \vee s$**.**

Removal of valid clauses

Lemma 2.3.7

Let Θ the set of clauses obtained by removing the valid clauses of Γ.

$$\Gamma \text{ has a model iff } \Theta \text{ has a model.}$$

Proof.

- Suppose that Γ has a model v, since Θ is a subset of clauses of Γ, v is also model of Θ. Hence Θ has a model.
- Suppose that Θ has a model v. Let v' a truth assignment of Γ so that $v'(x) = v(x)$ for all variable x belonging to both Γ and Θ. Let C a clause of Γ. If C is also a clause of Θ, then v' is a model of C since v and v' give the same value to C. If C is not a clause of Θ, then C is valid, consequently all truth assignment, v' in particular, is model of C. Hence Γ has a model : v'.

$\square$

The DPLL Algorithm(figure 2.1)

bool function Algo_DPLL(Γ : set of clauses)
0 Remove the valid clauses of Γ.
If $\Gamma = \emptyset$, return (`true`).
Else return (`DPLL`(Γ))

bool function `DPLL`(Γ : non-valid set of clauses)
The function returns true if and only if Γ is satisfiable
1 **If** $\perp \in \Gamma$, return(`false`).
If $\Gamma = \emptyset$, return (`true`).
2 Reduce Γ : simply remove any clause containing *another* clause.
3 Remove from Γ the clauses containing isolated literals (cf. paragraph 2.3.1).
If the set Γ has been modified, goto 1.
4 Apply to Γ the unit resolution (cf paragraph 2.3.2).
If the set Γ has been modified, goto 1.
5 Select x, an arbitrary variable of Γ
return (`DPLL`($\Gamma[x := 0]$) or then `DPLL`($\Gamma[x := 1]$))

Example 2.3.8

Let Γ the set of clauses :
$\neg a \vee \neg b$, $a \vee b$, $\neg a \vee \neg c$, $a \vee c$, $\neg b \vee \neg c$, $b \vee c$.

$$
\begin{tikzcd}
 & \Gamma \arrow[dl, "a \mapsto 0"'] \arrow[dr, "a \mapsto 1"] & \\
b, c, \neg b \vee \neg c, b \vee c \arrow[d, "\mathrm{RE}"'] & & \neg b, \neg c, \neg b \vee \neg c, b \vee c \arrow[d, "\mathrm{RE}"'] \\
b, c, \neg b \vee \neg c \arrow[d, "\mathrm{UR}"'] & & \neg b, \neg c, b \vee c \arrow[d, "\mathrm{UR}"'] \\
\bot & & \bot
\end{tikzcd}
$$

Since all leaves contain the empty clause, the set Γ is unsatisfiable.

Example 2.3.8

Let Γ the set of clauses : $\neg p \lor \neg q$, $\neg p \lor s$, $p \lor q$, $\neg p \lor \neg s$.

$$\begin{tikzcd}
 & \neg p \lor \neg q, \neg p \lor s, p \lor q, \neg p \lor \neg s \arrow[dl, "s \mapsto 0"] \arrow[dr, "s \mapsto 1"] & \\
\neg p \lor \neg q, \neg p, p \lor q \arrow[d, "\mathrm{RE}"'] & & \neg p \lor \neg q, p \lor q, \neg p \\
\neg p, p \lor q \arrow[d, "\mathrm{ELLI} : q=1"'] & & \\
\neg p \arrow[d, "\mathrm{ELLI} : p = 0"'] & & \\
\emptyset & &
\end{tikzcd}$$

Since one leaf contains the empty clause, the set Γ is satisfiable. It is useless to continue the construction of the right branch.

Theorems 2.3.9 et 2.3.10

The algorithm Algo_DPLL is correct and terminates.

Termination proof

Reminder of property 2.1.21 : Γ has a model iff $\Gamma[x := 0]$ is satisfiable or $\Gamma[x := 1]$ is satisfiable.

Correctness proof

Invariant : $\boxed{\text{the current value of } \Gamma \text{ has a model iff } \Gamma \text{ has a model.}}$
Verified at step 0, 1 and 5, hence correct answers. Suppose the recursive calls are correct :

- if DPLL($\Gamma[x := 0]$) is true , then by induction $\Gamma[x := 0]$ is satisfiable, hence Γ is satisfiable, according to property 2.1.21. which corresponds to the true value of DPLL(Γ).
- if DPLL($\Gamma[x := 0]$) is false, then by induction $\Gamma[x := 0]$ is unsatisfiable. In this case, DPLL(Γ) equals DPLL($\Gamma[x := 1]$) :
  - Suppose that DPLL($\Gamma[x := 1]$) is true, then by induction $\Gamma[x := 1]$ is satisfiable, hence Γ is satisfiable, which corresponds to the true value of DPLL(Γ).
  - Suppose that DPLL($\Gamma[x := 1]$) is false, then by induction $\Gamma[x := 1]$ is unsatisfiable. Hence Γ is unsatisfiable, which corresponds to the false value of DPLL(Γ).

Remarks 2.3.11 and 2.3.12

- **Forgetting simplifications :** DPLL stays correct if we forget the reduction (2), the removal of the isolated literals (3) and/or the unit reduction (4).
- **Choice of the variable (branching literal) :**
  - A good choice for the variable x from step (5), is to choose the variable that appears most often.
  - A better choice is to choose the variable which will lead to the most of simplifications

Cf. Sub-section 2.3.5, for the principal branching heuristics

Planning of the Semester

TODAY

- Propositional logic
- Propositional resolution *
- Propositional natural deduction
- First order logic

MIDTERM EXAM

- Basis for the automated proof (≪ first order resolution ≫)
- First order natural deduction

EXAM

Conclusion : Next course

- Natural deduction

Conclusion

Thank you for your attention.

Questions ?