Propositional Resolution

Hans de Nivelle, Marc Bezem

abstract

The goal of this sequence of slides is to quickly introduce all fundamental concepts of saturation-based theorem proving, using propositional resolution as a base.

We first introduce propositional clauses. After that, we introduce propositional resolution and a simple saturation procedure.

Then, we immediately proceed to introduce ordered resolution, redundancy, and simplification. Using those, we give a realistic saturation procedure.

We introduce the notions of fairness, persistent clause, saturated set, and prove the completeness of the saturation procedure.

We end with a discussion of redundancy.

We feel that in most introductions, too much time is spent to introducing resolution as calculus. We want to stress that deletion and simplification are as important as the resolution principle itself, and give them a prominent place from the beginning.

DISCLAIMER: As said, the goal of these slides is to prepare the reader for understanding saturation-based theorem proving for full first-order logic. The methods introduced are not intended for competitive, propositional theorem proving.

Definition: We assume a set of propositional symbols $\mathcal{P}$. We call the elements of $\mathcal{P}$ atoms.

A literal is an atom A or a negated atom $\neg A$. We will assume that $\neg\neg A = A$.

Definition: A clause is a finite multiset of literals

$$[A_1, \ldots, A_p]$$

.

The meaning of a clause $[A_1, \ldots, A_n]$ is the disjunction $A_1 \vee \cdots \vee A_n$.

The meaning of $[\,]$ is $\perp$.

An interpretation I is a set of literals which does not contain a conflicting pair.

An interpretation I is a model of a clause C if it contains a literal from C.

It is a model of set of clauses S if it is a model of every $C \in S$.

Theorem:

Let S be a clause set. S has a model in our sense iff the set of meanings of S has a model in the usual sense.

proof

From right-to-left is trivial. From left-to-right the missing atoms need to be assigned. This can be done by making them all false.

Propositional Resolution

Resolution: Let $[A] \cup R_1$ and $[\neg A] \cup R_2$ be clauses. The clause $R_1 \cup R_2$ is a resolvent of $[A] \cup R_1$ and $[\neg A] \cup R_2$.

Factoring: Let $[A, A] \cup R$ be a clause. The clause $[A] \cup R$ is a factor of $[A, A] \cup R$.

theorem:

Resolution is a sound and complete calculus. Let $C_1, \ldots, C_n$ be a sequence of clauses: $C_1, \ldots, C_n \vdash^*_{\mathrm{RES+FACT}} [\,]$ iff $C_1, \ldots, C_n$ is unsatisfiable.

The implication $\Rightarrow$ is called soundness.

The implication $\Leftarrow$ is called completeness.

A first Algorithm

Resolution can be used as algorithm for propositional theorem proving in the following way.

Start with initial clause set S.

As long as as there exists a resolvent or factor C that is derivable from clauses in S but not in S, add C to S.

If S contains [ ], then return unsatisfiable
else return satisfiable.

Definition: The final set S, to which no more clauses can be added, is called a saturated set.

Examples:

Try $[A, B]$, $[A, \neg B]$, $[\neg A, B]$, $[\neg A, \neg B]$.

Or $[A, B, C]$, $[A, B, \neg C]$, $[A, \neg B, C]$, $[A, \neg B, \neg C]$, $[\neg A, B, C]$, $[\neg A, B, \neg C]$, $[\neg A, \neg B, C]$, $[\neg A, \neg B, \neg C]$.

Improvements

On the implementation level, the resolution algorithm still can be improved:

1. It is not a good idea to wait until the end with checking for presence of [ ].
2. Short clauses should be preferred over long clauses.

On the level of the calculus, there are even bigger improvements possible:

1. Resolution and factoring can be restricted (ordering refinements)
2. Often clauses can be deleted. (subsumption, redundancy).

Ordering Refinements

An order is a binary relation $\succ$ that meets the following requirements:

O1: Never $x \succ x$.

O2: $x \succ y$ and $y \succ z$ imply $x \succ z$.

$\succ$ is called a total order if its also fulfills:

O3: Always $x \succ y$ or $x = y$ or $y \succ x$.

An L-order is a total order on propositional literals.

Ordering Refinements (2)

Definition: Let A be a literal, let R be (multi)set of literals. We write $A \succ L$ if for every $B \in L$, $\;A \succ B$.

We write $A \succeq L$ if for every $B \in L$, $\;A \succ B$ or $A = B$.

L-ordered resolution is defined as follows:

Resolution: Let $[A] \cup R_1$ and $[\neg A] \cup R_2$ be clauses, s.t. $A \succ R_1$ and $\neg A \succ R_2$. The clause $R_1 \cup R_2$ is an ordered resolvent of $[A] \cup R_1$ and $[\neg A] \cup R_2$.

Factoring: Let $[A, A] \cup R$ be a clause, s.t. $A \succeq R$. The clause $[A] \cup R$ is an an ordered factor of $[A, A] \cup R$.

Subsumption

Definition: Let C_1 and C_2 be clauses. C_1 subsumes C_2 if $C_1 \subseteq C_2$.

We say that C_1 strictly subsumes C_2 if $C_1 \subset C_2$.

In case that C_1 subsumes C_2, C_2 can be deleted.

Simplification (1)

Simplification means: By deduction trying to obtain new clauses that subsume existing clauses.

Assume that clauses $C_1, \ldots, C_n, D_1, \ldots, D_m$ logically imply E and that E subsumes all of $D_1, \ldots, D_m$ with $m > 0$.

Then, in case the system contains all the clauses $C_1, \ldots, C_n, D_1, \ldots, D_m$, it can delete all of $D_1, \ldots, D_m$ and replace them by E.

Simplification (2)

It is not possible to generate all consequences of the current set of clauses, and check whether they subsume an existing clause.

In practice, one can use simple deduction rules for rules for trying to find simplifications.

RESSIMP: Let $[A] \cup R_1$ and $[\neg A] \cup R_2$ be clauses s.t. R_1 subsumes R_2. Then

$$[A] \cup R_1,\ [\neg A] \cup R_2 \vdash R_2.$$

FACTSIMP: In each case, where $[A] \cup R$ is a factor of $[A, A] \cup R$, we have

$$[A, A] \cup R \vdash [A] \cup R.$$

Algorithm with Subsumption and Simplification

We write S for the set of clauses derived so far. We use $\vdash$ for the state transitions of the algorithm:

SUBS If $C_1 \neq C_2$, and C_1 subsumes C_2, then $C_1, C_2, S \vdash C_1, S$.

SIMP If $C_1, \ldots, C_n, D_1, \ldots, D_m$ imply E, and E subsumes all of $D_1, \ldots, D_m$, then

$$C_1, \ldots, C_n, D_1, \ldots, D_m, S \vdash C_1, \ldots, C_n, E, S.$$

ORDRES If D is an ordered resolvent of C_1 and C_2, then

$$C_1, C_2, S \vdash C_1, C_2, D, S.$$

ORDFACT If D is an ordered factor of C, then $C, S \vdash C, D, S$.

Completeness

It is not hard to see that the algorithm is sound.

Is it complete?

$\Rightarrow$ Yes and No.

Yes Whenever S is unsatisfiable, there exists a computation $S \vdash^* S'$, s.t. S' contains [ ].

No But often there also exist cycling computations: A clause is derived, then deleted, then rederived and redeleted, etc .

We want strong completeness. Whatever strategy the algorithm chooses, we want to be sure to eventually derive [ ].

Fairness

Let $S_1 \vdash S_2 \vdash \cdots \vdash S_i \vdash \cdots$ be a run of the algorithm. We call the run fair if:

Whenever there is an i, s.t. for all $j \geq i$, S_j contains two clauses C_1, C_2 that have an ordered resolvent D, there exists a $k \geq i$, s.t. S_k either contains D itself, or a clause D' that subsumes D.

Whenever there is an i, s.t. for all $j \geq i$, S_j contains a clause C that has an ordered factor D, there exists a $k \geq i$, s.t. S_k either contains D itself, or a clause D' that subsumes D.

Completeness

Theorem: Let $S_1 \vdash S_2 \vdash S_3 \vdash \cdots$ be a fair run of the algoritm:

If S_1 is unsatisfiable, then there is an S_i that contains $[\ ]$.

Forward Reasoning Rules

Definition: We classify the rules ordered resolution and ordered factoring as forward reasoning rules.

So now we have:

1. Two forward reasoning rules: Ordered resolution and ordered factoring.
2. One deletion rule: Subsumption.
3. Two simplification rules: Factoring and simplifying resolution.

Saturated Sets

Definition: Let S be a set of clauses. We call S a saturated set if

- For every clause D that can be obtained by a forward reasoning rule from clauses $C_1, \ldots, C_n$ with $C_1, \ldots, C_n \in S$,
- there is a clause $D' \in S$, s.t. either $D' = D$ or D' subsumes D.

Let I be some initial set of clauses. We call S a saturation of I if S is saturated, and

- for every clause $C \in I$,
- there is a clause $C' \in S$, s.t. either $C' = C$, or C' subsumes C.

Persistent Clauses

Definition: Let $S_1 \vdash S_2 \vdash S_3 \vdash \cdots$ be a run of the algorithm.

A clause C is persistent if there is an i, s.t. for all $j \geq i$, S_j contains C.

Theorem: Let $S_1 \vdash S_2 \vdash S_3 \vdash \cdots$ be a fair run of the algorithm.

Let S be the set of clauses that are persistent in the run.

Then S is a saturated set of S_1.

proof:

We first prove a lemma:

Lemma: Let C be a clause that occurs in an S_i. If C is not persistent, then there is a persistent clause C', s.t. C' subsumes C.

proof Suppose that C is not persistent. Then it is deleted in an S_j with $j > i$. It must be deleted either by SUBS, or by SIMP. In both cases, there is a clause $C^1 \in S_j$, s.t. C^1 subsumes C.

For this clause, the same argument can be applied. If it is not persistent, then there is an $k > j$, s.t. S_k contains a clause C^2 that subsumes C^1.

Now consider the sequence $C, C^1, C^2, C^3, \ldots$ It cannot be infinite, because each C^{k+1} subsumes C_k, and $C^{k+1} \neq C^k$. (which implies that at least one element is dropped). Therefore, some C^k must be a persistent clause.

By transitivity of subsumption C^k subsumes C.

We now show that S is a saturated set.

Suppose that S contains clauses $C_1, \ldots, C_n$, s.t. there is a clause D that can be obtained from $C_1, \ldots, C_n$ by forward reasoning.

Then the clauses $C_1, \ldots, C_n$ are persistent in the run $S_1 \vdash S_1 \vdash S_2 \vdash \cdots$.

Since $n \leq 2$, it is finite. (We have only resolution and factoring) Therefore, there is an i, s.t. for all $j \geq i$, S_j contains all of $C_1, \ldots, C_n$.

Then, by fairness, some S_k with $k \geq i$ contains either D itself or a clause D' that subsumes D. Using the previous lemma, in both cases there is a persisent clause that subsumes D.

It remains to show that S is a saturation of S_1. For this we also use the lemma. For each $C \in S_1$ there exists a persistent clause C' which subsumes C. This clause is present in S.

It remains to show the following:

theorem: Every saturated set S that does not contain $[\,]$ has a model.

proof:

Ranking the Clauses

Using the order $\succ$, we can sort the clauses. In each clause, we put the maximal elements first, then the second elements, etc.

On these sorted clauses, we can use alphabetic, lexicographic comparison.

As a result, all clauses in S are sorted, and we can assign numbers to them.

Ranking the Clauses

Assume that $\neg A \succ B \succ \neg B \succ A$.

Clause		after sorting		ranking
$[A, B]$	$\Rightarrow$	$[B, A]$	$\Rightarrow$	2,
$[A, \neg B]$	$\Rightarrow$	$[\neg B, A]$	$\Rightarrow$	0,
$[\neg B, \neg B]$	$\Rightarrow$	$[\neg B, \neg B]$	$\Rightarrow$	1,
$[\neg A, B]$	$\Rightarrow$	$[\neg A, B]$	$\Rightarrow$	4,
$[\neg A, \neg B]$	$\Rightarrow$	$[\neg A, \neg B]$	$\Rightarrow$	3,
$[\neg A, \neg A]$	$\Rightarrow$	$[\neg A, \neg A]$	$\Rightarrow$	5.

Model Construction

Using the ranking on clauses in S, we construct the following sequence of interpretations $I_0, I_1, I_2, \ldots, I_n$:

$I_0 = \{\}$.

At each rank i, let $C_i \in S$ be the clause at rank i.

- If C_i has two occurrences of its maximal literal, then $I_{i+1} = I_i$.
- If C_i has a literal that occurs in I_i, then $I_{i+1} = I_i$.
- IF C_i has one occurrence of its maximal literal and there is no literal in C_i that occurs in I_i, then let A be the maximal literal in C_i. Put $I_{i+1} = I_i \cup \{A\}$.

There are two things to show:

1. I_n contains a literal from every clause.
2. I_n does not contain a pair of conflicting literals $A, \neg A$.

We show by induction on i that each I_{i+1} contains a literal from C_i.

If I_i already contains a literal that occurs in C_i, then we are done. Otherwise, there are two possibilities:

1. The maximal element in C_i is not repeated. Then case 3 of the construction applies and I_{i+1} contains the maximal element of C_i.
2. The maximal element of C_i is repeated. Then C_i has an ordered factor D. Because S is a saturated set, either D itself or a clause subsuming D, is present in S.

   In both cases we have a clause $D' \in S$, with the following properties: $D' \subset C$, and D' comes before C in the ranking.

Hence, there exists a $j < i$, s.t. $C_j = D'$. By induction we may assume that I_{j+1} contains a literal from C_j. Because $D' \subset C$, we see that I_{j+1} contains a literal from C. Because $I_{j+1} \subseteq I_i$, this literal also occurs in I_i

Now suppose that I_n contains a complementary pair $A, \neg A$.

There must be a C_i that contains A as non-repeated, maximal element. We can write $C_i = [A] \cup R_1$. There is no literal in R_1 that occurs in I_i.

There must also exist a C_j which contains $\neg A$ as non-repeated, maximal element. Write $C_j = [\neg A] \cup R_2$. There is no literal in R_2 that occurs in I_j.

Because S is saturated, there must be a clause $D \in S$ which subsumes the resolvent $R_1 \cup R_2$.

Since I_n contains a literal from every clause, it must contain a literal from D and hence $R_1 \cup R_2$.

If the literal is from R_1, it must already have been present in I_i. In that case, A would not have been added to I_{i+1}.

Otherwise, it was already present in I_j. In that case, $\neg A$ would not have been inserted in I_{j+1}.

Redundancy

Subsumption is a special instance of a more general notion, which is called redundancy.

Definition: Clauses $C_1, \ldots, C_n$ make clause D redundant if D is a logical consequence of $C_1, \ldots, C_n$ and all of the $C_1, \ldots, C_n$ have a rank lower than D.

Examples:

If $C_1 \subset C_2$ then C_1 makes C_2 redundant.

If C is a tautology (contains a complementary pair $A, \neg A$), then the empty sequence makes C redundant.

If $A \succ B \succ C \succ \neg C$ then $[B, C]$ and $[\neg C]$ make $[A, B]$ redundant.

Using redundancy (instead of subsumption), the notion of saturated set becomes:

- For every clause D that can be obtained by a forward reasoning rule from clauses $C_1, \ldots, C_n$ with $C_1, \ldots, C_n \in S$,
- either $D \in S$ or there are clauses $D_1, \ldots, D_m \in S$ that make D redundant.

S is a saturation of I if S is saturated, and

- for every clause $C \in I$,
- either $C \in S$, or there are clauses $D_1, \ldots, D_m \in S$ that make D redundant.

Summary

We have seen that a saturation calculus consists of :

- Forward reasoning rules (resolution , factoring)
- Simplification rules (simplifying resolution)
- Redundancy (subsumption, tautology elimination)

We have introduced an abstract saturation algorithm, and introduced the notions of fairness, persistent clause, and saturated set.