Propositional Resolution

Hans de Nivelle
Abstract

The goal of this set of slides is to fairly quickly introduce the 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 with selection, a weak form of 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 full redundancy.
In our view, most introductions to automated reasoning spend too much time introducing resolution as a calculus, or as a proof system. We want to stress that deletion and simplification are at least as important as deduction, and give them a prominent place from the beginning.

**DISCLAIMER:** 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 $A_1 \lor \cdots \lor A_n$.

The meaning of $[ ]$ is $\bot$.

An interpretation $I$ is a set of positive atoms.

An interpretation $I$ is a model of a clause $C$ if either

1. $C$ contains a positive literal that occurs in $I$, or
2. $C$ contains a negative literal $\neg A$, for which $A$ does not occur in $I$.

It is a model of set of clauses $S$ if it is a model of every $C \in S$. 
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: $C_1, \ldots, C_n \vdash_{\text{RES+FACT}} \star$ 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 set of clauses $C$. 

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

If $S$ contains $\bot$, 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 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, even bigger improvements are possible:

1. Resolution and factoring can be restricted (ordering refinements, selection)
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 $y \succ x$. 
Ordering Refinements (2)

An $A$-order is a total order on propositional atoms. An $A$-order $\succ$ is extended to literals as follows:

For two atoms $A \neq B$, $\pm A \succ \pm B$ iff $A \succ B$.

For each atom $A$, $\neg A \succ A$. 
Ordering Refinements (3)

**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 $A = B$.

*$A$-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 $[A, A] \cup R$. 

Ordering/Selection Refinements

A selection function $\Sigma$ is a function that assigns to clause $R$ a subset (not multiset!), s.t. $\Sigma(R) \subseteq R$ and $\Sigma(R)$ contains only negative literals.

Let $[L] \cup R$ be a clause. Let $\Sigma$ be a selection function, let $\succ$ be an $A$-order.

$L$ is called *eligible* in $[L] \cup R$ if either

1. $L \in \Sigma([L] \cup R)$, or

2. $\Sigma([L] \cup R)$ is empty, and $L \succeq R$.  


Ordering/Selection Refinements (2)

**Definition:** $A$-ordered resolution with selection is defined as follows:

**Resolution:** Let $[A] \cup R_1$ and $[\neg A] \cup R_2$ be clauses, s.t. $A \notin R_1$, $\neg A \notin R_2$, and both $A$ and $\neg A$ are eligible in their clauses.

Then clause $R_1 \cup R_2$ obtained by ordered resolution with selection from $[A] \cup R_1$ and $[\neg A] \cup R_2$.

**Factoring:** Let $[A, A] \cup R$ be a clause, s.t. $A$ is eligible in $[A, A] \cup R$. Then the clause $[A] \cup R$ is obtained by ordered factoring with selection from $[A, A] \cup R$. 
So, a selection function can be seen as a guard, which either accepts the decision of the $A$-order, (When it returns $\emptyset$), or rejects the decision of the $A$-order, and instead decides that only some negative literals can be used for resolution.

In case there are no negative literals in the clause, the selection function has no choice but to accept the decision of the $A$-order.
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. (Note that $\subset$ on multisets takes cardinality into account)
**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 \([-A] \cup R_2\) be clauses s.t. \(R_1\) subsumes \(R_2\). Then
\[
[A] \cup R_1, \ [-A] \cup R_2 \vdash R_2.
\]

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 OWS-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 OWS-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?

⇒ Yes and No.

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

No But often there also exist cycling computations: A clause is derived, then deleted, then rederived and redleted, 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 OWS-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 OWS-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 OWS-resolution and OWS-factoring as forward reasoning rules.

So now we have:

1. Two forward reasoning rules: OWS-resolution and OWS-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 $j > i$. The deletion must be caused 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 persistent clause that subsumes $D$. 
It remains to show that $S$ is a saturation of $S_1$. For this we also use the previous lemma. For each $C \in S_1$ there exists a persistent clause $C'$ which subsumes $C$. This clause has to be 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

Let $S$ be a set of clauses.

Using the extension of the $A$-order $\triangleright$ to literals, we can sort the clauses. In each clause, put the maximal elements first, then the second elements, etc.

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

As a result, one obtains a complete sorting of the clauses in $S$, and the clauses can be ranked, using natural numbers, assigning higher numbers to bigger clauses.
## Ranking the Clauses (2)

Assume that \( \neg A \succ A \succ \neg B \succ B \).

<table>
<thead>
<tr>
<th>Clause</th>
<th>after sorting</th>
<th>ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>([B, B]) \Rightarrow [B, B] \Rightarrow 0,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>([A, \neg A]) \Rightarrow [\neg A, A] \Rightarrow 4,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>([\neg B, A]) \Rightarrow [A, \neg B] \Rightarrow 2,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>([B, A]) \Rightarrow [A, B] \Rightarrow 1,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>([\neg A, \neg B]) \Rightarrow [\neg A, \neg B] \Rightarrow 3,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>([\neg A, \neg A]) \Rightarrow [\neg A, \neg A] \Rightarrow 5.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Model Construction

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

$I_0 = \{}$.

For each rank $i$ with $i < n$, let $C_i \in S$ be the clause at rank $i$.

- If the maximal literal in $C_i$ is positive, occurs only once, then we can write $C_i = [A] \cup R$ with $A \succ R$. If no negative literal in $C_i$ is selected, and $R$ is false in $I_i$, then put $I_{i+1} = I_i \cup \{A\}$.

- In all other cases, put $I_{i+1} = I_i$. 
Lemma A

For every atom $A \in I_n$, there exists a clause of form $[A] \cup R$ in $S$, s.t. $A \succ R$, $\Sigma([A] \cup R) = \emptyset$ and $R$ is false in $I_n$.

proof

Let $i$ be the smallest number for which $A \in I_{i+1}$. By the last case of the model construction, there is a clause $C_i$ that can be written as $[A] \cup R_1$ with $A \succ R_1$, s.t. $\Sigma(C_i) = \emptyset$, and $R_1$ is false in $I_i$.

We have to show that $R_1$ is also false in $I_n$.

Let $B$ be a positive atom in $R_1$. Suppose that $B$ would occur in $I_n$. Let $j$ be the smallest number for which $B \in I_{j+1}$. Clearly $j > i$.

Then $C_j$ can be written in form $[B] \cup R_2$ and $B$ is maximal in this clause. Since $B$ occurs in $R_1$ and $A \succ R_1$, it would follow that $[B] \cup R_2$ comes before $[A] \cup R_1$ in the ranking. This contradicts the fact that $j > i$. 

34
Let \( \neg B \) be a negative literal in \( R_1 \), for which \( B \in I_i \). Since \( I_i \subseteq I_n \), \( B \) has to be also present in \( I_n \).
We now have to show that: If $S$ is a saturated set which does not contain $[]$, then $I_n$ is a model of $S$.

We show by induction on $i$ that every clause $C_i$ is true in $I_n$.

- Suppose that $C_i$ has form $[-A] \cup R_1$, where $-A$ is either selected or maximal. If $C_i$ were false, then $A \in I_n$ and $R_1$ is false in $I_n$. By the previous lemma, there exists a clause of form $[A] \cup R_2$ in $S$, s.t. $A \succ R_2$, nothing is selected in $A \cup R_2$, and $R_2$ is false in $I_n$.

It follows that the resolvent $R_1 \cup R_2$ is either present or subsumed in $S$. There must exist a clause $C'$ in $S$ that subsumes $R_1 \cup R_2$. This clause $C'$ comes before $C_i$ in the ranking. Therefore, we may assume that $C'$ is true. This contradicts the fact that $R_1$ and $R_2$ are false in $I_n$. 
Suppose that $C_i$ has form $[A, A] \cup R_1$, nothing is selected in $C_i$, and $[A] \geq R_1$.

$C_i$ fulfills the conditions for factoring. Because $S$ is saturated, there exists a clause $C'$ that subsumes $[A] \cup R_1$ in $S$. Since $C'$ comes before $C$ in the ranking, it must be the case that $C'$ is true in $I_n$. But then $C_i$ is also true in $I_n$. 
Suppose that $C_i$ has form $[A] \cup R_1$, that nothing is selected in $C_i$, and $A \succ R_1$.

If $R_1$ is false in $I_i$, then $I_{i+1}$ contains $A$, and therefore $C_i$ is true in $I_n$.

If $R_1$ is true in $I_i$, we have to show that $R_1$ is true in $I_n$ as well.

If there exists a positive literal $B$ in $I_i$, s.t. $B$ is in $R_1$, then clearly $R_1$ is true in $I_n$.

If there exists a negative literal $\neg B$ in $I_i$, s.t. $B$ is not in $I_i$, then we need to show that $B$ is also not present in $I_n$.

If $B$ were present in $I_n$, then there would exist a clause $C_j$ with $j \succ i$, in which $B$ were maximal. But then $C_j$ should have been ranked before $C_i$. 
Full 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 $A \succ B \succ \neg C \succ C$ 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.
Arbitrary Selection (1)

We give an example that shows that all this complicated machinery (A-orderings, selection functions) for controlling which literals can be resolved away, is really necessary. We show that if one selects literals for resolution completely arbitrarily, then completeness is lost. Consider the following set $S$. Selected literals are underlined:

$\begin{align*}
[A, B] & \quad [B, \neg A] \\
[\neg A, \neg B] & \quad [\neg B, A] \\
\end{align*}$

$\begin{align*}
[A, \neg A] & \quad [B, \neg B]
\end{align*}$

You may check that $S$ is unsatisfiable, but that nothing outside of $S$ can be obtained by resolution.
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 fundamental notions of fairness, persistent clause, and saturated set.