A Semi-Decision Procedure for Validity

We now have a semi-decision procedure for validity of a formula:

- Negate the formula
- Transform the result into clausal form
- Generate an appropriate set of ground instances
- Check to see if the set of ground instances is satisfiable (using the obvious variation on propositional resolution).

The problem is, how do we pick the appropriate set of ground instances?
Unification

Suppose we have the clauses:

$$\forall x (\neg p(x)) \quad p(a) \lor \neg q(b) \quad \forall y (q(y))$$

What is the ground-resolution that shows that this set is unsatisfiable?

How about for the set:

$$\forall x (p(x)) \quad \neg p(a) \lor \neg p(b)$$

Or the set:

$$\forall x \forall y p(g(x), x, y) \quad \forall w \neg p(w, a, f(w))$$

Or the set:

$$\forall x \forall y p(g(x), x, y) \quad \forall w \forall z \neg p(w, z, f(w))$$
Unification

Our goal is to determine the ground instantiation lazily, as we are forced into it by resolution.

The solution is Unification, which given two terms (or atoms) determines the assignments to the variables in the terms such that the terms could then be made identical. That is, two terms \( t_1 \) and \( t_2 \) are unifiable iff there is a substitution \( \sigma \) such that \( t_1\sigma = t_2\sigma \).

The process is naturally described in terms of making the trees of the two terms (or atoms) identical. For example:

\[
\begin{array}{c}
p \\
g \quad x \quad y \\
\quad x \\
\end{array}
\quad
\begin{array}{c}
p \\
w \quad a \quad f \\
\quad w \\
\end{array}
\]
Unification

The algorithm can be described roughly as follows:

To unify term $t$ with term $s$, yielding a unifying answer substitution

1. If $s$ is a variable, succeed, with $\sigma = \{s := t\}$.
2. If $t$ is a variable, succeed, with $\sigma = \{t := s\}$.
3. If $s$ is a constant, then:
   (a) If $t$ is the same constant, succeed with $\sigma = \emptyset$.
   (b) Otherwise, fail.
4. If $s$ is a compound term $f(s_1, \ldots, s_n)$, then
   (a) If $t$ is not a compound term $f(t_1, \ldots, t_n)$, fail.
   (b) If $t$ is a compound term $f(t_1, \ldots, t_n)$, then
      i. Set $\sigma_0 = \emptyset$
      ii. For $i \in [1, \ldots, n]$ attempt to unify $s_i\sigma_0 \ldots \sigma_{i-1}$ with $t_i\sigma_0 \ldots \sigma_{i-1}$, yielding $\sigma_i$. If any fails, fail.
      iii. Succeed with $\sigma = \sigma_1 \circ \ldots \circ \sigma_n$
Occur’s Check

This algorithm is not quite right.

In defining the algorithm we must be careful of one thing. Consider an attempt to resolve the set:

\[
\forall x \ p(x, f(x)) \quad \forall y \ \neg p(y, y)
\]

Therefore, the first two steps must be modified to read:

1. If \( s \) is a variable, then:
   (a) if \( s \not\in FV(t) \), succeed, with \( \sigma = \{ s := t \} \),
   (b) otherwise, fail.

2. If \( t \) is a variable, then:
   (a) if \( t \not\in FV(s) \), succeed, with \( \sigma = \{ t := s \} \),
   (b) otherwise, fail.

This extra step is called the \textit{occurs check}.
Mr. MGU

It can be shown that for any two unifiable terms, there exists a substitution that makes the fewest possible substitutions for variables of the two terms. This unifying substitution is called the Most General Unifier, or MGU.

More precisely, if \( \sigma \) is the MGU of terms \( t_1 \) and \( t_2 \), then for any other unifier \( \sigma' \) there is a substitution \( \sigma'' \) such that \( \sigma' = \sigma \circ \sigma'' \).

We can further prove that for any two terms, the given unification algorithm always produces the most general unifier if the terms are unifiable.
Unification, By The Book

**Definition:** (4.3.2) A set of term equations is in *solved form* if:

- All equations are of the form $x = t$
- Each variable $x$ that appears on the left-hand side of an equation does not appear elsewhere in the set.
Unification, By The Book

The unification algorithm can be described in terms of sets of term equations as follows:

To unify the terms \( s \) and \( t \), begin with the set of term equations \( \{ s = t \} \). Perform the following transformations on the set of term equations as long as any one of them is applicable:

1. Replace any equation of the form \( t = x \) where \( t \) is not a variable, with \( x = t \).
2. Erase any equation of the form \( x = x \) or \( c = c \)
3. If there is an equation of the form \( s = t \) where neither \( s \) nor \( t \) is a variable:
   (a) If \( s \) is a constant \( c \), then if \( t \) is not also the constant \( c \), terminate the algorithm, indicating failure.
   (b) If \( s \) is a term of the form \( f(s_1, \ldots, s_n) \) then if \( t \) is not a term of the form \( f(t_1, \ldots, t_n) \), terminate the algorithm, indicating failure. Otherwise, remove the equation \( s = t \) and replace it with the \( n \) equations \( s_1 = t_1, \ldots, s_n = t_n \).
   (c) If there is an equation of the form \( x = t \), then if \( x \) occurs in \( t \), terminate, indicating failure. Otherwise, replace all other occurrences of the variable \( x \) in the set with the term \( t \).
Resolution by the Book

The General Resolution Rule is stated as follows:

Let $C_1$ and $C_2$ be clauses with no variables in common. Let $l_1 \in C_1$ and $l_2 \in C_2$ be literals such that $l_1$ and $l_2^c$ can be unified by an mgu $\sigma$. $C_1$ and $C_2$ are said to be clashing clauses and to clash on the literals $l_1$ and $l_2$. $C$, the resolvent of $C_1$ and $C_2$ is the clause:

$$Res(C_1, C_2) = (C_1\sigma - \{l_1\sigma\}) \cup (C_2\sigma - \{l_2\sigma\})$$

For example:

$$p(f(x), g(y)) \lor q(x, y) \quad \neg p(f(f(a)), g(z)) \lor q(f(a), g(z))$$
**Standardizing Apart**

The restriction that the clauses have disjoint sets of variables is crucial to ensure completeness. If we did not use this restriction, which may require doing an $\alpha$-conversion on the clauses before resolving, we would not be able to resolve the clauses:

$$p(f(x)) \quad \neg p(x)$$

due to the occurs check. Yet these represent independent clauses:

$$\forall x(p(f(x))) \quad \forall x(\neg p(x))$$

which are mutually unsatisfiable.

The process of assigning new variables to the clauses is called *standardizing apart*, or, applying a *separating pair of substitutions*.
The General Resolution Rule is Very General

While it is not explicitly stated in the discussion of the resolution rule in BenAri, it is important to recognize that the clause \((C_i \sigma - \{l_i \sigma\})\) may be more than one literal smaller than the clause \(C_i\). Consider, for example:

\[
\neg p(x, a) \lor \neg p(x, y) \lor \neg p(y, x) \quad p(z, f(z)) \lor p(z, a)
\]
The Lifting Lemma

The completeness of General Resolution is, effectively, a corollary of the following theorem:

**Theorem (4.4.1 – Lifting Lemma):** Let $C'_1$ and $C''_2$ be ground instances of the clauses $C_1$ and $C_2$, respectively. Let $C''$ be a (ground) resolvent of $C'_1$ and $C''_2$, obtained by resolving on the literal $l'$. Then there is a (general) resolvent, $C$, of $C_1$ and $C_2$, such that $C''$ is a ground instance of $C$.

This is captured in the following figure:

```
\[
\begin{array}{c}
\text{Instance} \\
C_1', C_2' \\
\end{array} \quad \begin{array}{c}
\text{General Resolution} \\
\quad \rightarrow \quad C \\
\end{array} \quad \begin{array}{c}
\text{Instance} \\
C'' \\
\end{array}
\]
\[
\begin{array}{c}
\text{Instance} \\
C_1, C_2 \\
\end{array} \quad \begin{array}{c}
\text{Ground Resolution} \\
\quad \rightarrow \quad C'' \\
\end{array}
\]```

General Resolution Procedure

The *General Resolution Procedure* is given as:

Let $S$ be a set of clauses and define $S_0 = S$. Assume that $S_i$ has been constructed. Choose clashing clauses $C_1, C_2 \in S_i$ and let $C = Res(C_1, C_2)$. If $C = \emptyset$ then terminate the procedure – $S$ is unsatisfiable. Otherwise, construct $S_{i+1} = S_i \cup \{C\}$. If $S_{i+1} = S_i$ for all pairs of clashing clauses, terminate the procedure – $S$ is satisfiable.