for each predicate and function:

\[
\begin{align*}
\forall x \quad x &= y \\
\forall x, y \quad x &= y & y &= x \\
\forall x, y, z \quad x &= y, y = z & x &= z \\
\forall x, y \quad x &= y = (P_1(x) \in FM) \\
\forall a, y \quad my \Rightarrow (P_2(x) & P_2(y)) \\
\forall x, y, z \quad w &= y, z = x = z & (F_2(x) = F_2(y, z))
\end{align*}
\]

Given these sentences, a standard inference procedure such as resolution can perform tasks requiring equality reasoning, such as solving mathematical equations. However, these axioms will generate a lot of conclusions, most of them not helpful to a proof. So there has been a search for more efficient ways of handling equality. One alternative is to add inference rules rather than axioms. The simplest rule, demodulation, takes a unit clause \(x = y\) and some clause \(a\) that contains the term \(x\), and yields a new clause formed by substituting \(y\) for \(x\) within \(a\). It works if the term within \(a\) unifies with \(x\); it need not be exactly equal to \(x\). Note that demodulation is directional; given \(x = y\), the \(x\) always gets replaced with \(y\), never vice versa. That means that demodulation can be used for simplifying expressions using demodulators such as \(x + 0 = x\) or \(x \cdot 0 = x\). As another example, given

\[
\text{Father (Father (x))} = \text{PaternalGrandfather (x)} \\
\text{Birthdate (Birthdate (Father (Father (Bella)), 1926)}
\]

we can conclude by demodulation

\[
\text{Birthdate (PaternalGrandfather (Bella), 1926)}
\]

More formally, we have

- **Demodulation**: For any terms \(x, y,\) and \(z,\) where \(2\) appears somewhere in literal \(\ell_n\), and where \(\text{UNIFY}(x, z_n) = \theta\),

\[
x = y, \quad m_1 \lor \cdots \lor m_n,
\]

\[
\text{SUB(SUBST}(\theta, x), \text{SUBST}(\theta, y)) \lor \cdots \lor \text{SUBST}(\theta, z_n)
\]

where \(\text{SUBST}\) is the usual substitution of a binding list, and \(\text{SUB}(x, y)\) means to replace \(x\) with \(y\) everywhere that \(x\) occurs within \(\ell_n\).

The rule can also be extended to handle non-unit clauses in which an equality literal appears:

- **Paramodulation**: For any terms \(x, y,\) and \(z,\) where \(z\) appears somewhere in literal \(\ell_n\), and where \(\text{UNIFY}(x, z) = \theta,\)

\[
\lor \cdots \lor \ell_k, x = y, \quad m_1 \lor \cdots \lor m_n
\]

\[
\text{SUB(SUBST}(\theta, x), \text{SUBST}(\theta, y)), \text{SUBST}(\theta, \ell_1) \lor \cdots \lor \text{SUBST}(\theta, \ell_k) \lor m_1 \lor \cdots \lor m_n
\]

For example, from

\[
P(F[x, y], x) \lor Q(x) \quad \text{and} \quad F(A, y) = y V R(y)
\]
we have $0 = \text{UNIFY}(F(A, p), F(x, B)) = \{x/A, y/B\}$, and we can conclude by \textit{paramodulation} the sentence

$$P(B, A) \lor Q(A) \lor R(B)$$

Paramodulation yields a complete inference procedure for first-order logic with equality.

A third approach handles equality reasoning entirely within an extended unification algorithm. That is, terms are unifiable if they are \textit{provably} equal under some substitution, where "provably" allows for equality reasoning. For example, the terms $1 + 2$ and $2 + 1$ normally are not unifiable, but a unification algorithm that knows that $x = y$ could unify them with the empty substitution. \textit{Equational unification} of this kind can be done with efficient algorithms designed for the particular axioms used (commutativity, associativity, and so on) rather than through explicit inference with those axioms. Theorem provers using this technique are closely related to the CLP systems described in Section 9.4.

9.5.6 Resolution strategies

We know that repeated applications of the resolution inference rule will eventually find a proof if one exists. In this subsection, we examine strategies that help find proofs \textit{efficiently}.

\textbf{Unit preference}: This strategy prefers to do resolutions where one of the sentences is a single literal (also known as a \textit{unit clause}). The idea behind the strategy is that we are trying to produce an empty clause, so it might be a good idea to prefer inferences that produce shorter \textit{clauses}. Resolving a unit sentence (such as $P$) with any other sentence (such as $P \lor Q \lor R$) always yields a clause (in this case, $P \lor R$) that is shorter than the other clause. When the unit preference strategy was first tried for propositional inference in 1964, it led to a dramatic speedup, making it feasible to prove theorems that could not be handled without the preference. Unit \textit{resolution} is a restricted form of resolution in which every resolution step must involve a unit clause. Unit resolution is incomplete in general, but complete for Horn \textit{clauses}. Unit resolution proofs on Horn clauses resemble forward chaining.

The \textsc{OTTER} theorem prover (Organized Techniques for Theorem-proving and Effective Research, McCune, 1992), uses a form of best-first search. Its heuristic function measures the "weight" of each clause, where lighter clauses are preferred. The exact choice of heuristic is up to the user, but generally, the weight of a clause should be correlated with its size or difficulty. Unit clauses are treated as light the search can thus be seen as a generalization of the unit preference strategy.

\textbf{Set of support}: Preferences that try certain resolutions first are helpful, but in general it is more effective to try to eliminate some potential resolutions altogether. For example, we can insist that every resolution step involve at least one element of a special set of clauses—the \textit{set of support}. The resolvent is then added into the set of support. If the \textit{set} of support is small relative to the whole knowledge base, the search space will be reduced dramatically.

We have to be careful with this approach because a bad choice for the set of support will make the algorithm incomplete. However, if we choose the set of support $S$ so that the remainder of the sentences are jointly satisfiable, then a resolution is \textit{complete}. For example, one can use the negated query as the set of support, on the assumption that the