Inference in first-order logic

Chapter 9
Outline

• Reducing first-order inference to propositional inference
• Unification
• Generalized Modus Ponens
• Forward chaining
• Backward chaining
• Resolution
Review

• Inference in First-Order Logic
  – **Could convert first-order logic (FOL) to propositional logic (PL) and use PL inference**
    • Must convert (reduce) Universal and Existential Quantifiers into PL
    • Potential problem with infinite number of substitutions
  – **Could lift Modus Ponens to FOL**
    • Unification required to make expressions look identical
How to perform inference in FOL

• Use a knowledge base
  – Insert data with “STORE”
  – Retrieve “unified” sentences using “FETCH”
    • Search for substitutions that unite query with every sentence in KB

• How can we make FETCH fast?
  – Only attempt unifications if they have a chance of succeeding
Universal instantiation (UI)

• Every instantiation of a universally quantified sentence is entailed by it:
  \[ \forall v \alpha \]
  \[ \text{Subst}\{\{v/g\}, \alpha\} \]
  for any variable \(v\) and ground term \(g\).

• \(\text{Subst}\{\{v/g\}, \alpha\} :\) Substitution

• E.g., \(\forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x)\) yields:
  \[\begin{align*}
  &\text{King(John)} \land Greedy(John) \Rightarrow Evil(John) \{x/John\} \\
  &\text{King(Richard)} \land Greedy(Richard) \Rightarrow Evil(Richard) \{x/Richard\} \\
  &\text{King(Father(John))} \land Greedy(Father(John)) \Rightarrow Evil(Father(John)) \{x/Father(John)\}
  \end{align*}\]
Existential instantiation (EI)

• For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

\[
\exists v \alpha \\
\text{Subst}\{\{v/k\}, \alpha\}
\]

• E.g., $\exists x \; \text{Crown}(x) \land \text{OnHead}(x, John)$ yields:

\[
\text{Crown}(C_1) \land \text{OnHead}(C_1, John)
\]

provided $C_1$ is a new constant symbol, called a Skolem constant.

• Existential Instantiation is a special case of a more general process called skolemization.
• Universal Instantiation can be applied many times to product many different consequence.
• Existential Instantiation can be applied once, and then the Existential quantified sentence can be discarded.
• New knowledge bases is not logically equivalent to the old, but it can be shown to be inferentially equivalent.
• It is satisfiable exactly when the original knowledge base is satisfiable.
Reduction to propositional inference

- Universally quantified sentence can be replaced the set of all possible instantiation.
- Suppose the KB contains just the following:
  \( \forall x \, \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \)
  \( \text{King}(\text{John}) \)
  \( \text{Greedy}(\text{John}) \)
  \( \text{Brother}(\text{Richard}, \text{John}) \)
- Instantiating the universal sentence in all possible ways, we have:
  \( \text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John}) \)
  \( \text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard}) \)
  \( \text{King}(\text{John}) \)
  \( \text{Greedy}(\text{John}) \)
  \( \text{Brother}(\text{Richard}, \text{John}) \)
- The new KB is **propositionalized**: proposition symbols are

  \( \text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard}), \text{etc.} \)
Reduction contd.

• Every FOL KB and query can be \textit{propositionalized} so as to preserve entailment

• (A \textbf{ground sentence} is entailed by new KB iff entailed by original KB)

• Idea: propositionalize KB and query, apply resolution, return result

• Problem: with function symbols, there are \textit{infinitely} many ground terms,
  – e.g., \textit{Father(Father(Father(John)))}
Theorem: Herbrand (1930). If a sentence $\alpha$ is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB.

Idea: For $n = 0$ to $\infty$ do
create a propositional KB by instantiating with depth-$n$ terms
see if $\alpha$ is entailed by this KB

Problem: works if $\alpha$ is entailed, loops if $\alpha$ is not entailed

Theorem: Turing (1936), Church (1936) Entailment for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)
9.2 Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.

- E.g., from:
  \[
  \forall x \ King(x) \land \ Greedy(x) \Rightarrow \ Evil(x)
  \]
  \[
  \begin{align*}
  &\ King(John) \\
  &\ \forall y \ Greedy(y) \\
  &\ Brother(Richard, John)
  \end{align*}
  \]

- it seems obvious that \textit{Evil(John)}, but propositionalization produces lots of facts such as \textit{Greedy(Richard)} that are irrelevant

- With \( p \) \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations.
Generalized Modus Ponens

This inference process can be captured as a single inference rule that we call Generalized Modus Ponens: For atomic sentences $p_i, p_i',$ and $q,$ where there is a substitution $\theta$ such that $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i),$ for all $i,$

$$p_1', \ p_2', \ \ldots, \ p_n', \ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)$$

$\quad$\text{SUBST}(\theta, q).

There are $n + 1$ premises to this rule: the $n$ atomic sentences $p_i'$ and the one implication. The conclusion is the result of applying the substitution $\theta$ to the consequent $q.$ For our example:

$p_1'$ is $\text{King}(\text{John})$  \hspace{1cm}  p_1$ is $\text{King}(x)$
$p_2'$ is $\text{Greedy}(\text{y})$  \hspace{1cm}  p_2$ is $\text{Greedy}(x)$
$\theta$ is $\{x/\text{John}, y/\text{John}\}$  \hspace{1cm}  q$ is $\text{Evil}(x)$
$\text{SUBST}(\theta, q)$ is $\text{Evil}(\text{John}).$

It is easy to show that Generalized Modus Ponens is a sound inference rule. First, we observe that, for any sentence $p$ (whose variables are assumed to be universally quantified) and for any substitution $\theta$,

$$p \models \text{SUBST}(\theta, p).$$
Unification

• Unify should return a substitution that makes the two arguments look the same.

• Unify algorithm takes two sentences and returns a unifer for them if one exist:

• \text{UNIFY}(p, q) = \theta

where \text{SUBSET}(\theta, p) = \text{SUBST}(\theta, q)
Unification

- We can get the inference immediately if we can find a substitution $\theta$ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

$\theta = \{x/John, y/John\}$ works

- $\text{Unify}(p, q) = \theta$ if $\text{SUBST}(p, \theta) = \text{SUBST}(q, \theta)$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John,x)</td>
<td>Knows(John,Jane)</td>
<td></td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td></td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td></td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x,OJ)</td>
<td></td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, OJ)$

Query: $\text{Knows}(John, x)$
Unification

- We can get the inference immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- $\text{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{OJ})$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(x, \text{OJ})$</td>
<td></td>
</tr>
</tbody>
</table>

- Standardizing apart eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$
Unification

- We can get the inference immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- Unify($\alpha, \beta$) = $\theta$ if $\alpha \theta = \beta \theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John,x)</td>
<td>Knows(John,Jane)</td>
<td>${x/\text{Jane}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td>${x/OJ,y/\text{John}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td></td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x,OJ)</td>
<td></td>
</tr>
</tbody>
</table>

- Standardizing apart eliminates overlap of variables, e.g., 
  Knows($z_{17},OJ$)
Unification

- We can get the inference immediately if we can find a substitution \( \theta \) such that \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(y) \)

\[ \theta = \{x/\text{John}, y/\text{John}\} \] works

- \( \text{Unify}(\alpha, \beta) = \theta \) if \( \alpha \theta = \beta \theta \)

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Knows}(\text{John},x) )</td>
<td>( \text{Knows}(\text{John},\text{Jane}) )</td>
<td>( {x/\text{Jane}} )</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John},x) )</td>
<td>( \text{Knows}(y,\text{OJ}) )</td>
<td>( {x/\text{OJ}, y/\text{John}} )</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John},x) )</td>
<td>( \text{Knows}(y, \text{Mother}(y)) )</td>
<td>( {y/\text{John}, x/\text{Mother}(\text{John})} )</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John},x) )</td>
<td>( \text{Knows}(x,\text{OJ}) )</td>
<td></td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g., \( \text{Knows}(z_{17}, \text{OJ}) \)
Unification

• We can get the inference immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$

$\theta = \{x/\text{John}, y/\text{John}\}$ works

• $\text{Unify}(\alpha, \beta) = \theta$ if $\alpha \theta = \beta \theta$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John,x)</td>
<td>Knows(John,Jane)</td>
<td>${x/Jane}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td>${x/OJ,y/\text{John}}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td>${y/\text{John},x/Mother(\text{John})}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x,OJ)</td>
<td>${\text{fail}}$</td>
</tr>
</tbody>
</table>

• **Standardizing apart** eliminates overlap of variables, e.g., Knows($z_{17}, \text{OJ}$)
Unification

• To unify $\text{Knows}(\text{John}, x)$ and $\text{Knows}(y, z)$,
  $\theta = \{y/\text{John}, x/z \}$ or $\theta = \{y/\text{John}, x/\text{John}, z/\text{John}\}$

• The first unifier is more general than the second.

• There is a single most general unifier (MGU) that is unique up to renaming of variables.
  $\text{MGU} = \{ y/\text{John}, x/z \}$
The unification algorithm

function \textsc{Unify}(x, y, \theta) \textbf{returns} a substitution to make \textit{x} and \textit{y} identical
inputs: \textit{x}, a variable, constant, list, or compound
\hspace{1cm} \textit{y}, a variable, constant, list, or compound
\hspace{1cm} \theta, the substitution built up so far

if \theta = \text{failure} then return failure
else if \text{\textit{x} = \textit{y}} then return \theta
else if \text{\textsc{Variable}?(\textit{x})} then return \textsc{Unify-Var}(\textit{x}, \textit{y}, \theta)
else if \text{\textsc{Variable}?(\textit{y})} then return \textsc{Unify-Var}(\textit{y}, \textit{x}, \theta)
else if \text{\textsc{Compound}?(\textit{x}) and \textsc{Compound}?(\textit{y})} then
   return \textsc{Unify} (\textit{\textsc{Args}[x]}, \textit{\textsc{Args}[y]}, \textsc{Unify} (\textit{\textsc{Op}[x]}, \textit{\textsc{Op}[y]}, \theta))
else if \text{\textsc{List}?(\textit{x}) and \textsc{List}?(\textit{y})} then
   return \textsc{Unify} (\textit{\textsc{Rest}[x]}, \textit{\textsc{Rest}[y]}, \textsc{Unify} (\textit{\textsc{First}[x]}, \textit{\textsc{First}[y]}, \theta))
else return failure
The unification algorithm

function \textsc{Unify-Var}(\textit{var}, \textit{x}, \theta) \textbf{returns} a substitution
\hspace{1em} \textbf{inputs:} \textit{var}, a variable
\hspace{2em} \textit{x}, any expression
\hspace{3em} \theta, the substitution built up so far
\hspace{1em} \textbf{if} \{\textit{var}/\textit{val}\} \in \theta \textbf{ then return} \textsc{Unify}(\textit{val}, \textit{x}, \theta)
\hspace{1em} \textbf{else if} \{\textit{x}/\textit{val}\} \in \theta \textbf{ then return} \textsc{Unify}(\textit{var}, \textit{val}, \theta)
\hspace{1em} \textbf{else if} \textsc{Occur-Check?}(\textit{var}, \textit{x}) \textbf{ then return} \textbf{failure}
\hspace{1em} \textbf{else return} add \{\textit{var}/\textit{x}\} to \theta

Occur-check: make the complexity of the algorithm quadratic.
Generalized Modus Ponens (GMP)

\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]

\[ \therefore q_{\theta} \]

where \( p_i'_{\theta} = p_i \theta \) for all \( i \)

\( p_1' \) is \( King(John) \)  
\( p_1 \) is \( King(x) \)

\( p_2' \) is \( Greedy(y) \)  
\( p_2 \) is \( Greedy(x) \)

\( \theta \) is \( \{x/John,y/John\} \)  
\( q \) is \( Evil(x) \)

\( q_{\theta} \) is \( Evil(John) \)

- GMP used with KB of definite clauses (exactly one positive literal)

- All variables assumed universally quantified
Soundness of GMP

• Need to show that
  \[ p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \models q\theta \]
  provided that \( p_i'\theta = p_i\theta \) for all \( i \)

• Lemma: For any sentence \( p \), we have \( p \models p\theta \) by UI

  1. \( (p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)\theta = (p_1\theta \land \ldots \land p_n\theta \Rightarrow q\theta) \)
  2. \( p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models p_1'\theta \land \ldots \land p_n'\theta \)
  3. From 1 and 2, \( q\theta \) follows by ordinary Modus Ponens
How to perform inference in FOL

• More than “ASK” and “Tell”
• Use a knowledge base
  – Insert data with “STORE”
  – Retrieve “unified” sentences using “FETCH”
    • Search for substitutions that unite query with every sentence in KB: \textbf{Knows(John, x)}
• How can we make FETCH fast?
  – Only attempt unifications if they have a chance of succeeding
  – \textbf{Knows(John, x)} & \textbf{Brother(Richard, John)} no change
Predicate Indexing

• We can avoid such unifications by indexing the facts in the KB. That’s puts all the **Knows** facts in one bucket and all the **Brother** facts in another. *(hash table)*
  – Example: unify **Knows (John, x)** with KB:
    • **Knows (John, Jane)**
    • **Knows (y, Bill)**
    • **Brother (Richard, John)**
    • **Knows (y, Mother(y))**
    • **Knows (x, Elizabeth)**

**No need to check**
Predicate Indexing

• Predicate indexing puts all the *Knows* facts in one bucket and all the *Brother* facts in another
  – Might not be a win if there are lots of clauses for a particular predicate symbol
    • Consider *how many* people *Know* one another
  – Instead index by predicate and first argument
    • Clauses may be stored in multiple buckets
Subsumption lattice

- How to construct indices for all possible queries that unify with it
  - Example: Employs (AIMA.org, Richard)

\[
\begin{align*}
\text{Employs}(\text{AIMA.org}, \text{Richard}) & \quad \text{Does AIMA.org employ Richard?} \\
\text{Employs}(x, \text{Richard}) & \quad \text{Who employs Richard?} \\
\text{Employs}(\text{AIMA.org}, y) & \quad \text{Whom does AIMA.org employ?} \\
\text{Employs}(x, y) & \quad \text{Who employs whom?}
\end{align*}
\]
Subsumption Lattice

\begin{align*}
\text{Employs}(\text{AIMA.org, Richard}) & \quad \text{Does AIMA.org employ Richard?} \\
\text{Employs}(x, \text{Richard}) & \quad \text{Who employs Richard?} \\
\text{Employs}(\text{AIMA.org, } y) & \quad \text{Whom does AIMA.org employ?} \\
\text{Employs}(x, y) & \quad \text{Who employs whom?}
\end{align*}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{subsumption_lattice.png}
\caption{(a) The subsumption lattice whose lowest node is the sentence \text{Employs}(\text{AIMA.org, Richard}). (b) The subsumption lattice for the sentence \text{Employs}(\text{John, John}).}
\end{figure}
Subsumption lattice

• Each node reflects making one substitution
• The “highest” common descendent of any two nodes is the result of applying the most general unifier
• Predicate with $n$ arguments will create a lattice with $O(2^n)$ nodes
• Benefits of indexing may be outweighed by cost of storing and maintaining indices
9.3 Forward Chaining

• Start with the atomic sentences in KB and apply Modus Ponens in the forward direction.
• Situation → Response are useful for systems that make inference in response to newly arrived information.
• FOL definite clauses: they are disjunctions of literals of which exactly one is positive.
• A definite clause either is
  – atomic or
  – is an implication
    • whose antecedent is a conjunction of positive literals and whose consequence is a single positive literal.
Forward –chaining

- King(x) \land \text{Greedy}(x) \rightarrow \text{Evil}(x)
- King(John)
- Greedy(John).
- FOL include variable, and assumed to be universally quantified.
- Definite clauses are a suitable normal form for use with Generalized Modus Ponens.
- Not every KB can be converted into a set of definite clauses.
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]

Nono … has some missiles, i.e., \( \exists x \text{ Owns}(\text{Nono},x) \land \text{Missile}(x) \):
\[ \text{Owns}(\text{Nono},M_1) \text{ and } \text{Missile}(M_1) \]

... all of its missiles were sold to it by Colonel West
\[ \text{Missile}(x) \land \text{Owns}(\text{Nono},x) \Rightarrow \text{Sells}(\text{West},x,\text{Nono}) \]

Missiles are weapons:
\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

An enemy of America counts as "hostile“:
\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]

West, who is American …
\[ \text{American}(\text{West}) \]

The country Nono, an enemy of America …
\[ \text{Enemy}(\text{Nono}, \text{America}) \]
Forward chaining algorithm

function FOL-FC-Ask(KB, α) returns a substitution or false

repeat until new is empty
  new ← { }
  for each sentence r in KB do
    (p_1 ∧ ... ∧ p_n ⇒ q) ← STANDARDIZE-APART(r)
    for each θ such that (p_1 ∧ ... ∧ p_n)θ = (p'_1 ∧ ... ∧ p'_n)θ
      for some p'_1, ..., p'_n in KB
        q' ← SUBST(θ, q)
        if q' is not a renaming of a sentence already in KB or new then do
          add q' to new
          φ ← UNIFY(q', α)
          if φ is not fail then return φ
    add new to KB
  return false

A fact is not “new” if it is just a renaming of a known fact.
Likes(x, IceCream), Likes(y, IceCream),
Forward chaining proof
Forward chaining proof

\[ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

\[ \text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x) \]
Forward chaining proof

\[\text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)\]
Properties of forward chaining

- **Sound**: every inference is just an application of GMP
- **Complete** for first-order definite clauses KB
- **Datalog** = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if $\alpha$ is not entailed
- This is unavoidable: entailment with definite clauses is *semidecidable*

NatNum(0)
\forall n \text{ NatNum}(n) \Rightarrow \text{NatNum}(S(n))
Facts: NatNum(S(0)) , NatNum(S(S(0))), NatNum(S(S(S(0)))), …
Problem of FC

• Inner loop of FC involves finding all possible unifiers such that the premise of a rule unifies with a suitable set of facts in KB. -> pattern matching problem

• Algorithm rechecks every rule on every iteration even if very few additions are made to KB on each iteration.

• Might generate many facts that are irrelevant to the goal.
Improvement

- Pattern Matching problem
- Missile(x) ⇒ Weapon(x)
- Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono)
  - Match Owans(Nono, x) first
  - Match Missile(x)
  - Conject ordering problem: find an ordering to solve the conjuncts of the rule premises so that the total cost is minimized. NP-hard problem.
  - Heuristic: most constrained variable.
CSP vs. KB inference

Hard matching example

\[ \text{Diff}(wa, nt) \land \text{Diff}(wa, sa) \land \text{Diff}(nt, q) \land \]
\[ \text{Diff}(nt, sa) \land \text{Diff}(q, nsw) \land \text{Diff}(q, sa) \land \]
\[ \text{Diff}(nsw, v) \land \text{Diff}(nsw, sa) \land \text{Diff}(v, sa) \Rightarrow \]
\[ \text{Colorable}() \]

\[ \text{Diff}(\text{Red}, \text{Blue}) \quad \text{Diff}(\text{Red}, \text{Green}) \]
\[ \text{Diff}(\text{Green}, \text{Red}) \quad \text{Diff}(\text{Green}, \text{Blue}) \]
\[ \text{Diff}(\text{Blue}, \text{Red}) \quad \text{Diff}(\text{Blue}, \text{Green}) \]

- \text{Colorable}() is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence
  matching is NP-hard
Cheer ourselves Up

• Most rules in real world KB are small and simple. (polynomial)

• We can consider subclasses of rules for which matching is efficient.
  – Constraint graph -> linear graph or tree: easy to solve.

• Eliminate redundant rule matching
Efficiency of forward chaining

- **Incremental forward chaining**: no need to match a rule on iteration $k$ if a premise wasn't added on iteration $k-1$
  - $\implies$ match each rule whose premise contains a newly added positive literal

- **Matching itself can be expensive**:
- **Database indexing** allows $O(1)$ retrieval of known facts
  - e.g., query $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

- Forward chaining is widely used in **deductive databases**
  - E.g. Production system: XCON system
  - ACT, SOAR
Irrelevant facts

• Forward chaining makes all allowable inferences based on the known facts, even they are irrelevant to the goal at hand.
• Backward chaining
• Restrict forward chaining to a selected subset of rules
• For deductive DB, rewrite the rule set, using information from the goal, so that only relevant variables binding (magic set) are considered during FC inference.
  – Goal: Criminal(WEST)
  – Rewrite Magic(WEST) ∧ American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
• Perform a sort of “generic” backward inference.
Backward Chaining

• Called with a list of goals
• Take the first goal and finds every clause in KB whose positive literal or head unifies with the goal.
• Each clause creates a new recursive call in which premise(or body) is added to the list(stack).
9.4 Backward chaining algorithm

```plaintext
function FOL-BC-Ask(KB, goals, θ) returns a set of substitutions
    inputs: KB, a knowledge base
        goals, a list of conjuncts forming a query
        θ, the current substitution, initially the empty substitution {}
    local variables: ans, a set of substitutions, initially empty
    if goals is empty then return {θ}
    q' ← SUBST(θ, FIRST(goals))
    for each r in KB where STANDARDIZE-Apart(r) = (p₁ ∧ ... ∧ pₙ ⇒ q)
        and θ' ← UNIFY(q, q') succeeds
        ans ← FOL-BC-Ask(KB, [p₁, ..., pₙ|REST(goals)], COMPOSE(θ, θ')) ∪ ans
    return ans
```

\[\text{SUBST(COMPOSE}(\theta₁, \theta₂), p) = \text{SUBST}(\theta₂, \text{SUBST}(\theta₁, p))\]
Backward chaining example

\textit{Criminal(West)}
Backward chaining example

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]
Backward chaining example
Backward chaining example

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]
Backward chaining example
Backward chaining example

\[ \text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}) \]
Backward chaining example

Enemy(x, America) $\Rightarrow$ Hostile(x)
Backward chaining example
Properties of backward chaining

- **Depth-first recursive** proof search: space is linear in size of proof
- **Incomplete** due to infinite loops (repeated state)
  - ⇒ fix by checking current goal against every goal on stack
- **Inefficient** due to repeated subgoals (both success and failure)
  - ⇒ fix using caching of previous results (extra space)
- Widely used for logic programming
Logic programming: Prolog

• **Algorithm = Logic + Control**

• Basis: backward chaining with Horn clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
  Compilation techniques ⇒ 60 million LIPS

• Program = set of clauses = \texttt{head :- literal}_1, \ldots \texttt{ literal}_n.
  \texttt{criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).}

• Depth-first, left-to-right backward chaining
• Built-in predicates for arithmetic etc., e.g., \texttt{X is Y*Z+3}
• Built-in predicates that have side effects (e.g., input and output
  predicates, assert/retract predicates)
• **Closed-world assumption** ("negation as failure")
  – e.g., given \texttt{alive(X) :- not dead(X).}
  – \texttt{alive(joe)} succeeds if \texttt{dead(joe)} fails
Prolog

• Appending two lists to produce a third:
  
  ```prolog
  append([], Y, Y).
  append([X|L], Y, [X|Z]) :-
      append(L, Y, Z).
  ```

• query:    ```prolog
  append(A, B, [1, 2]) ?
  ```

• answers:  
  
  ```prolog
  A=[]   B=[1, 2]
  A=[1, 2] B=[]
  ```
Speedup by parallelization

• Compiler or Interpreter
  – Open code(WAM)
• OR-parallelism
• AND-parallelism
• Prolog is IMCOMPLETE.
Redundant inference and infinite loops

\[
\begin{align*}
\text{path}(X,Z) & : \text{- } \text{link}(X,Z). \\
\text{path}(X,Z) & : \text{- } \text{path}(X,Y), \text{ link}(Y,Z).
\end{align*}
\]

A simple three-node graph, described by the facts \text{link}(a,b) and \text{link}(b,\text{ in Figure 9.9(a). With this program, the query path}(a,c) generates the pre} in Figure 9.10(a). On the other hand, if we put the two clauses in the order

\[
\begin{align*}
\text{path}(X,Z) & : \text{- } \text{path}(X,Y), \text{ link}(Y,Z). \\
\text{path}(X,Z) & : \text{- } \text{link}(X,Z).
\end{align*}
\]
Figure 9.9  (a) Finding a path from A to C can lead Prolog into an infinite loop. (b) A graph in which each node is connected to two random successors in the next layer. Finding a path from $A_1$ to $J_4$ requires 877 inferences.
9.5 Resolution

• Is existence of complete proof procedure?
• If TRUE
  – All conjectures can be established mechanically.
  – All of mathematics can be established as the logic consequence of a set of fundamental axioms.
• 1930, Kurt proved that completeness theorem for FOL, show that any entailed sentence has a finite proof.
• 1931, incompleteness theorem. For a logical system that includes the principle of induction is necessarily incomplete.
• 1965 Resolution-based prover is OK.
First-order CNF

– For all x, American(x) ^ Weapon(y) ^ Sells(x, y, z) ^ Hostile (z) => Criminal(x)

– ~American(x) V ~Weapon(y) V ~Sells(x, y, z) V ~Hostile(z) V Criminal(x)

• Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence (they are both unsatisfiable in same conditions)
Example

• Everyone who loves all animals is loved by

\[ \forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

◊ Eliminate implications:

\[ \forall x \ [\forall y \ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y \ Loves(y, x)] . \]

◊ Move \( \neg \) inwards: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have

\[ \neg \forall x \ p \quad \text{becomes} \quad \exists x \ \neg p \]

\[ \neg \exists x \ p \quad \text{becomes} \quad \forall x \ \neg p . \]

Our sentence goes through the following transformations:

\[ \forall x \ [\exists y \ \neg (\neg Animal(y) \lor Loves(x, y))] \lor [\exists y \ Loves(y, x)] . \]

\[ \forall x \ [\exists y \ \neg \neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] . \]

\[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] . \]

Notice how a universal quantifier \((\forall y)\) in the premise of the implication has become an existential quantifier. The sentence now reads “Either there is some animal that \(x\) doesn’t love, or (if this is not the case) someone loves \(x\).” Clearly, the meaning of the original sentence has been preserved.
Example

- **Standardize variables**: For sentences like \((\forall x\ P(x)) \lor (\exists x\ Q(x))\) which use the same variable name twice, change the name of one of the variables. This avoids confusion later when we drop the quantifiers. Thus, we have

\[
\forall x\ [\exists y\ Animal(y) \land \neg Loves(x, y)] \lor [\exists z\ Loves(z, x)].
\]

**Skolemize**: **Skolemization** is the process of removing existential quantifiers by elimination. In the simple case, it is just like the Existential Instantiation rule of Section 9.1: translate \(\exists x\ P(x)\) into \(P(A)\), where \(A\) is a new constant. If we apply this rule to our sample sentence, however, we obtain

\[
\forall x\ [Animal(A) \land \neg Loves(x, A)] \lor Loves(B, x)
\]

which has the wrong meaning entirely: it says that everyone either fails to love a particular animal \(A\) or is loved by some particular entity \(B\). In fact, our original sentence allows each person to fail to love a different animal or to be loved by a different person. Thus, we want the Skolem entities to depend on \(x\):

\[
\forall x\ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x).
\]
Example

\[ \forall x \ [\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x))] \lor \text{Loves}(G(x), x) \]

- F and G are Skolem Functions
  - arguments of function are universally quantified variables in whose scope the existential quantifier appears
Example

Drop universal quantifiers: At this point, all remaining variables must be universally quantified. Moreover, the sentence is equivalent to one in which all the universal quantifiers have been moved to the left. We can therefore drop the universal quantifiers:

\[ \text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x)) \lor \text{Loves}(G(x), x) . \]

Distribute \( \land \) over \( \lor \):

\[ \text{Animal}(F(x)) \lor \text{Loves}(G(x), x) \land \neg \text{Loves}(x, F(x)) \lor \text{Loves}(G(x), x) . \]

This step may also require flattening out nested conjunctions and disjunctions.

- Two clauses
- \( F(x) \) refers to the animal potentially unloved by \( x \)
- \( G(x) \) refers to someone who might love \( x \)
Resolution inference rule

\[ \ell_1 \lor \cdots \lor \ell_k, \quad m \]

\[ \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \]

\( l_i \) and \( m \) are complementary literals

- A lifted version of propositional resolution rule
  - two clauses must be standardized apart
    - no variables are shared
  - can be resolved if their literals are complementary
    - one is the negation of the other
    - if one unifies with the negation of the other
Resolution

\[ \ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n \]
\[ \text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \]

where \( \text{UNIFY}(\ell_i, \neg m_j) = \theta \). For example, we can resolve the two clauses

\[ [\text{Animal}(F(x)) \lor \text{Loves}(G(x), x)] \quad \text{and} \quad [\neg \text{Loves}(u, v) \lor \neg \text{Kills}(u, v)] \]

by eliminating the complementary literals \( \text{Loves}(G(x), x) \) and \( \neg \text{Loves}(u, v) \), with unifier \( \theta = \{u/G(x), v/x\} \), to produce the \textbf{resolvent} clause

\[ [\text{Animal}(F(x)) \lor \neg \text{Kills}(G(x), x)]. \]
Conversion to CNF

• Every sentence of FOL can be converted into an inferentially equivalent CNF sentence.
• Need to eliminate existential quantifiers.
• Everyone who loves all animals is loved by someone:
  \[ \forall x [\forall y \text{Animal}(y) \Rightarrow \text{Loves}(x,y)] \Rightarrow [\exists y \text{Loves}(y,x)] \]

• 1. Eliminate biconditionals and implications
  \[ \forall x [\neg\forall y \neg\text{Animal}(y) \lor \text{Loves}(x,y)] \lor [\exists y \text{Loves}(y,x)] \]

• 2. Move \(\neg\) inwards:
  \[ \neg\forall x p \equiv \exists x \neg p, \quad \neg\exists x p \equiv \forall x \neg p \]
  \[ \forall x [\exists y \neg(\neg\text{Animal}(y) \lor \text{Loves}(x,y))] \lor [\exists y \text{Loves}(y,x)] \]
  \[ \forall x [\exists y \neg\neg\text{Animal}(y) \land \neg\text{Loves}(x,y)] \lor [\exists y \text{Loves}(y,x)] \]
  \[ \forall x [\exists y \text{Animal}(y) \land \neg\text{Loves}(x,y)] \lor [\exists y \text{Loves}(y,x)] \]
Conversion to CNF contd.

3. **Standardize variables**: each quantifier should use a different one
\[ \forall x \left[ \exists y \ Animal(y) \land \neg Loves(x,y) \right] \lor \left[ \exists z \ Loves(z,x) \right] \]

4. **Skolemize**: a more general form of existential instantiation.
Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
\[ \forall x \ [Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x) \]

5. **Drop universal quantifiers**:
\[ [Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x) \]

6. **Distribute** \( \lor \) over \( \land \):
\[ [Animal(F(x)) \lor Loves(G(x),x)] \land [\neg Loves(x,F(x)) \lor Loves(G(x),x)] \]
Resolution: brief summary

• Full first-order version:

\[
\frac{l_1 \lor \cdots \lor l_k, \quad m_1 \lor \cdots \lor m_n}{(l_1 \lor \cdots \lor l_{i-1} \lor l_{i+1} \lor \cdots \lor l_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta}
\]

where \( \text{Unify}(l_i, \neg m_j) = \theta \).

• The two clauses are assumed to be standardized apart so that they share no variables.
• For example,

\[
\neg \text{Rich}(x) \lor \text{Unhappy}(x)
\]

\[
\frac{}{\text{Rich(Ken)}}
\]

\[
\frac{}{\text{Unhappy(Ken)}}
\]

with \( \theta = \{x/\text{Ken}\} \)

• Apply resolution steps to \( \text{CNF}(\text{KB} \land \neg \alpha) \); complete for FOL
Binary resolution & factoring

- Binary resolution is not complete
- Factoring: the removal of redundant literals.
- Binary resolution & factoring is complete.
Example

\[\neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x, y, z) \lor \neg \text{Hostile}(z) \lor \text{Criminal}(x) .\]
\[\neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono}, x) \lor \text{Sells}(\text{West}, x, \text{Nono}) .\]
\[\neg \text{Enemy}(x, \text{America}) \lor \text{Hostile}(x) .\]
\[\neg \text{Missile}(x) \lor \text{Weapon}(x) .\]
\[\text{Owns}(\text{Nono}, M_1) .\]
\[\text{Missile}(M_1) .\]
\[\text{American}(\text{West}) .\]
\[\text{Enemy}(\text{Nono}, \text{America}) .\]

We also include the negated goal \(\neg \text{Criminal}(\text{West})\). The resolution proof is shown in Fig-
Resolution proof: definite clauses

\[ \neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x, y, z) \lor \neg \text{Hostile}(z) \lor \text{Criminal}(x) \]

\[ \neg \text{Criminal}(\text{West}) \]

\[ \text{American}(\text{West}) \]

\[ \neg \text{American}(\text{West}) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(\text{West}, y, z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Weapon}(y) \lor \neg \text{Sells}(\text{West}, y, z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Missile}(x) \lor \text{Weapon}(x) \]

\[ \text{Missile}(M1) \]

\[ \neg \text{Missile}(y) \lor \neg \text{Sells}(\text{West}, y, z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Sells}(\text{West}, M1, z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Missile}(M1) \lor \neg \text{Owns}(\text{Nono}, x) \lor \text{Sells}(\text{West}, x, \text{Nono}) \]

\[ \text{Missile}(M1) \]

\[ \neg \text{Missile}(M1) \lor \neg \text{Owns}(\text{Nono}, M1) \lor \neg \text{Hostile}(\text{Nono}) \]

\[ \neg \text{Sells}(\text{West}, M1, z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Owns}(\text{Nono}, M1) \lor \neg \text{Hostile}(\text{Nono}) \]

\[ \neg \text{Owns}(\text{Nono}, M1) \lor \neg \text{Hostile}(\text{Nono}) \]

\[ \neg \text{Enemy}(x, \text{America}) \lor \text{Hostile}(x) \]

\[ \neg \text{Hostile}(\text{Nono}) \]

\[ \neg \text{Enemy}(\text{Nono}, \text{America}) \]

\[ \text{Enemy}(\text{Nono}, \text{America}) \]
Example 2

Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by no one.
Jack loves all animals.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

First, we express the original sentences, some background knowledge, and the negated goal $G$ in first-order logic:

A. $\forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)]$
B. $\forall x \ [\exists y \ Animal(y) \wedge Kills(x, y)] \Rightarrow [\forall z \ \neg Loves(z, x)]$
C. $\forall x \ Animal(x) \Rightarrow Loves(Jack, x)$
D. $Kills(Jack, Tuna) \lor Kills(Curiosity, Tuna)$
E. $Cat(Tuna)$
F. $\forall x \ Cat(x) \Rightarrow Animal(x)$
G. $\neg Kills(Curiosity, Tuna)$
First, we express the original sentences, some background knowledge, and the negated goal $G$ in first-order logic:

A. $\forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)]$

B. $\forall x \ [\exists y \ Animal(y) \land Kills(x, y)] \Rightarrow [\forall z \neg Loves(z, x)]$

C. $\forall x \ Animal(x) \Rightarrow Loves(Jack, x)$

D. $Kills(Jack, Tuna) \lor Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

F. $\forall x \ Cat(x) \Rightarrow Animal(x)$

$\neg G. \neg Kills(Curiosity, Tuna)$

Now we apply the conversion procedure to convert each sentence to CNF:

A1. $Animal(F(x)) \lor Loves(G(x), x)$

A2. $\neg Loves(x, F(x)) \lor Loves(G(x), x)$

B. $\neg Animal(y) \lor \neg Kills(x, y) \lor \neg Loves(z, x)$

C. $\neg Animal(x) \lor Loves(Jack, x)$

D. $Kills(Jack, Tuna) \lor Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

F. $\neg Cat(x) \lor Animal(x)$

$\neg G. \neg Kills(Curiosity, Tuna)$

The resolution proof that Curiosity killed the cat is given in Figure 9.12. In English, the proof
Figure 9.12 A resolution proof that Curiosity killed the cat. Notice the use of factoring in the derivation of the clause $Loves(G(\text{Jack}), \text{Jack})$. 
Nonconstructive proofs

Unfortunately, resolution can produce nonconstructive proofs for existential goals. For example, $\neg\text{Kills}(w, \text{Tuna})$ resolves with \text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna})$ to give \text{Kills}(\text{Jack}, \text{Tuna}), which resolves again with $\neg\text{Kills}(w, \text{Tuna})$ to yield the empty clause. Notice that $w$ has two different bindings in this proof; resolution is telling us that, yes, someone killed Tuna—either Jack or Curiosity. This is no great surprise! One solution is to restrict the allowed resolution steps so that the query variables can be bound only once in a given proof; then we need to be able to backtrack over the possible bind-

Answer literal
1. First, we observe that if \( S \) is unsatisfiable, then there exists a particular set of *ground instances* of the clauses of \( S \) such that this set is also unsatisfiable (Herbrand’s theorem).

2. We then appeal to the **ground resolution theorem** given in Chapter 7, which states that propositional resolution is complete for *ground* sentences.

3. We then use a **lifting lemma** to show that, for any propositional resolution proof using the set of ground sentences, there is a *corresponding* first-order resolution proof using the first-order sentences from which the *ground* sentences were obtained.
Any set of sentences $S$ is representable in clausal form

Assume $S$ is unsatisfiable, and in clausal form

Some set $S'$ of ground instances is unsatisfiable

Resolution can find a contradiction in $S'$

There is a resolution proof for the contradiction in $S'$

**Figure 9.13** Structure of a completeness proof for resolution.
◊ **Herbrand universe:** If $S$ is a set of clauses, then $H_S$, the Herbrand universe of $S$, is the set of all ground terms constructible from the following:

   a. The function symbols in $S$, if any.
   b. The constant symbols in $S$, if any; if none, then the constant symbol $A$.

For example, if $S$ contains just the clause $\neg P(x, F(x, A)) \lor \neg Q(x, A) \lor R(x, B)$, then $H_S$ is the following infinite set of ground terms:

$$\{ A, B, F(A, A), F(A, B), F(B, A), F(B, B), F(A, F(A, A)), \ldots \}.$$  

◊ **Saturation:** If $S$ is a set of clauses and $P$ is a set of ground terms, then $P(S)$, the saturation of $S$ with respect to $P$, is the set of all ground clauses obtained by applying all possible consistent substitutions of ground terms in $P$ with variables in $S$.

◊ **Herbrand base:** The saturation of a set $S$ of clauses with respect to its Herbrand universe is called the Herbrand base of $S$, written as $H_S(S)$. For example, if $S$ contains solely the clause just given, then $H_S(S)$ is the infinite set of clauses

$$\{ \neg P(A, F(A, A)) \lor \neg Q(A, A) \lor R(A, B),$$

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

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

$$\neg P(F(A, B), F(F(A, B), A)) \lor \neg Q(F(A, B), A) \lor R(F(A, B), B), \ldots \}$$

80
Let $C_1$ and $C_2$ be two clauses with no shared variables, and let $C'_1$ and $C'_2$ be ground instances of $C_1$ and $C_2$. If $C'$ is a resolvent of $C'_1$ and $C'_2$, then there exists a clause $C$ such that (1) $C$ is a resolvent of $C_1$ and $C_2$ and (2) $C'$ is a ground instance of $C$.

This is called a lifting lemma, because it lifts a proof step from ground clauses up to general first-order clauses. In order to prove his basic lifting lemma, Robinson had to invent unification and derive all of the properties of most general unifiers. Rather than repeat the proof here, we simply illustrate the lemma:

\[
\begin{align*}
C_1 &= \neg P(x, F(x, A)) \lor \neg Q(x, A) \lor R(x, B) \\
C_2 &= \neg N(G(y), z) \lor P(H(y), z) \\
C'_1 &= \neg P(H(B), F(H(B), A)) \lor \neg Q(H(B), A) \lor R(H(B), B) \\
C'_2 &= \neg N(G(B), F(H(B), A)) \lor P(H(B), F(H(B), A)) \\
C' &= \neg N(G(B), F(H(B), A)) \lor \neg Q(H(B), A) \lor R(H(B), B) \\
C &= \neg N(G(y), F(H(y), A)) \lor \neg Q(H(y), A) \lor R(H(y), B). 
\end{align*}
\]
equals for equals in any predicate or function. So we need three basic axioms, and then one for each predicate and function:

\[ \forall x \ x = x \]
\[ \forall x, y \ x = y \Rightarrow y = x \]
\[ \forall x, y, z \ x = y \land y = z \Rightarrow x = z \]
\[ \forall x, y \ x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)) \]
\[ \forall x, y \ x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)) \]
\[
\vdots
\]
\[ \forall w, x, y, z \ w = y \land x = z \Rightarrow (F_1(w, x) = F_1(y, z)) \]
\[ \forall w, x, y, z \ w = y \land x = z \Rightarrow (F_2(w, x) = F_2(y, z)) \]
\[
\vdots
\]

Given these sentences, a standard inference procedure such as resolution can perform tasks requiring equality reasoning, such as solving mathematical equations.

Another way to deal with equality is with an additional inference rule. The simplest rule, demodulation, takes a unit clause \( x = y \) and substitutes \( y \) for any term that unifies with \( x \) in some other clause. More formally, we have
\*\*Demodulation\*\*: For any terms \(x, y,\) and \(z,\) where \(\text{UNIFY}(x, z) = \theta\) and \(m_n[z]\) is a literal containing \(z:\)

\[
\frac{x = y, \quad m_1 \lor \cdots \lor m_n[z]}{m_1 \lor \cdots \lor m_n[\text{SUBST}(\theta, y)]}. \]

Demodulation is typically used for simplifying expressions using collections of assertions such as \(x + 0 = x,\ x^1 = x,\) and so on. 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 \(\text{UNIFY}(x, z) = \theta,\)

\[
\frac{\ell_1 \lor \cdots \lor \ell_k \lor x = y, \quad m_1 \lor \cdots \lor m_n[z]}{\text{SUBST}(\theta, \ell_1 \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_n[y])}. \]

Unlike demodulation, 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 provably equal under some substitution, where “provably” allows for some amount of equality reasoning. For example, the terms \(1 + 2\) and \(2 + 1\) normally are not unifiable, but a unification algorithm that knows that \(x + y = y + x\) could unify them with the empty substitution. \textbf{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 constraint logic programming systems described in Section 9.4.
Practical theorem provers

- Verification and synthesis of hard/soft ware
  - Software (axiomize all syntactic elements of programming language)
    - Verify a program’s output is correct for all inputs
    - There exists a program, P, that satisfies a specification
      - Synthesize P during search
  - Hardware (axiomize all interactions between signal and circuit elements)
    - Verify that interactions between signals and circuits is robust
      - Will CPU work in all conditions?
    - There exists a circuit, C, that satisfies a specification
      - Synthesize C during search
Theorem provers

• Logical inference is a powerful way to “reason” automatically
  – Prover should be independent of KB syntax
  – Prover should use control strategy that is fast
  – Prover can support a human by
    • Checking a proof by filling in voids
    • Person can kill off search even if semi-decidable
Practical theorem provers

– Boyer-Moore (1979)
  • First rigorous proof of Godel Incompleteness Theorem

– OTTER (1997)
  • Solved several open questions in combinatorial logic

– EQP
  • Solved Robbins algebra, a proof of axioms required for Boolean algebra
  – Problem posed in 1933 and solved in 1997 after eight days of computation
**procedure** OTTER\( (sos, usable) \)

**inputs**: \( sos \), a set of support—clauses defining the problem (a global variable) 
\( usable \), background knowledge potentially relevant to the problem

**repeat**

\[ \text{clause} \leftarrow \text{the lightest member of } sos \]

\[ \text{move clause from } sos \text{ to } usable \]

**PROCESS**\( (\text{INFER(clause, usable), sos}) \)

**until** \( sos = [] \) or a refutation has been found

**function** INFER\( (\text{clause, usable}) \) **returns** clauses

**resolve** \( \text{clause} \) with each member of \( usable \)

**return** the resulting clauses after applying \( \text{FILTER} \)

**procedure** PROCESS\( (\text{clauses, sos}) \)

**for each** \( \text{clause} \) in \( \text{clauses} \) **do**

\[ \text{clause} \leftarrow \text{SIMPLIFY(clause)} \]

merge identical literals

discard \( \text{clause} \) if it is a tautology

\[ sos \leftarrow [\text{clause} \mid sos] \]

**if** \( \text{clause} \) has no literals **then** a refutation has been found

**if** \( \text{clause} \) has one literal **then** look for \( \text{unit refutation} \)

---

**Figure 9.14** Sketch of the OTTER theorem prover. Heuristic control is applied in the