Inference in first-order logic

Chapter 9
(& finish chapter 8)
Outline

• Knowledge engineering example (circuits)
• Reducing first-order inference to propositional inference
• Unification
• Generalized Modus Ponens
• Forward chaining
• Backward chaining
• Resolution
Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants (ontology)
4. Encode general knowledge about the domain (epistemology)
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base
The electronic circuits domain

One-bit full adder
The electronic circuits domain

1. Identify the task
   - Does the circuit actually add properly? (circuit verification)

2. Assemble the relevant knowledge
   - Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   - Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary (ontology, what exists)
   - Alternatives:
     Type($X_1$) = XOR
     Type($X_1$, XOR)
     XOR($X_1$)
The electronic circuits domain

4. Encode general knowledge of the domain (epistemology)
   - \( \forall t_1, t_2 \) Connected\( (t_1, t_2) \) \( \Rightarrow \) Signal\( (t_1) = \) Signal\( (t_2) \)
   - \( \forall t \) Signal\( (t) = 1 \lor \) Signal\( (t) = 0 \)
   - \( 1 \neq 0 \)
   - \( \forall t_1, t_2 \) Connected\( (t_1, t_2) \) \( \Rightarrow \) Connected\( (t_2, t_1) \)
   - \( \forall g \) Type\( (g) = \text{OR} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) = 1 \iff \exists n \) Signal\( (\text{In}(n,g)) = 1 \)
   - \( \forall g \) Type\( (g) = \text{AND} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) = 0 \iff \exists n \) Signal\( (\text{In}(n,g)) = 0 \)
   - \( \forall g \) Type\( (g) = \text{XOR} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) = 1 \iff \) Signal\( (\text{In}(1,g)) \neq \) Signal\( (\text{In}(2,g)) \)
   - \( \forall g \) Type\( (g) = \text{NOT} \) \( \Rightarrow \) Signal\( (\text{Out}(1,g)) \neq \) Signal\( (\text{In}(1,g)) \)
The electronic circuits domain

5. Encode the specific problem instance

Type($X_1$) = XOR    Type($X_2$) = XOR
Type($A_1$) = AND    Type($A_2$) = AND
Type($O_1$) = OR

Connected(Out(1,$X_1$),In(1,$X_2$))    Connected(In(1,$C_1$),In(1,$X_1$))
Connected(Out(1,$X_1$),In(2,$A_2$))    Connected(In(1,$C_1$),In(1,$A_1$))
Connected(Out(1,$A_2$),In(1,$O_1$))    Connected(In(2,$C_1$),In(2,$X_1$))
Connected(Out(1,$A_1$),In(2,$O_1$))    Connected(In(2,$C_1$),In(2,$A_1$))
Connected(Out(1,$X_2$),Out(1,$C_1$))    Connected(In(3,$C_1$),In(2,$X_2$))
Connected(Out(1,$O_1$),Out(2,$C_1$))    Connected(In(3,$C_1$),In(1,$A_2$))
The electronic circuits domain

6. Pose queries to the inference procedure

What are the possible sets of values of all the terminals for the adder circuit?

\[ \exists i_1, i_2, i_3, o_1, o_2 \quad \text{Signal}(\text{In}(1, C_1)) = i_1 \land \text{Signal}(\text{In}(2, C_1)) = i_2 \land \text{Signal}(\text{In}(3, C_1)) = i_3 \land \text{Signal}(\text{Out}(1, C_1)) = o_1 \land \text{Signal}(\text{Out}(2, C_1)) = o_2 \]

7. Debug the knowledge base

May have omitted assertions like \( 1 \neq 0 \)
Universal instantiation (UI)

• Every instantiation of a universally quantified sentence is entailed by it:

\[
\forall v \alpha \\
\text{Subst(\{v/g\}, \alpha)}
\]

for any (every) variable \( v \) and ground term \( g \). \( \{v/g\} \) is a binding of variable \( v \) to expression, and \( \text{Subst(\{v/g\}, \alpha)} \) does this binding within sentence \( \alpha \).

• E.g., \( \forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \) yields:

\[\begin{align*}
\text{King}(John) \land \text{Greedy}(John) & \Rightarrow \text{Evil}(John) \\
\text{King}(Richard) \land \text{Greedy}(Richard) & \Rightarrow \text{Evil}(Richard) \\
\text{King}(&\text{Father}(John)) \land \text{Greedy}(\text{Father}(John)) & \Rightarrow \text{Evil}(\text{Father}(John))
\end{align*}\]
Existential instantiation (EI)

- For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:
  
  $$\exists v \alpha \quad \text{Subst}\{{v/k}, \alpha\}$$

- E.g., $\exists x \ Crown(x) \land OnHead(x,John)$ yields:
  
  $$\text{Crown}(C_1) \land OnHead(C_1,John)$$

  provided $C_1$ is a new constant symbol, called a Skolem constant
Existential instantiation (EI)

• UI can be applied several times to add new sentences; the new KB is logically equivalent to the old one.

• EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old, but it is satisfiable iff the old KB was satisfiable.
Reduction to propositional inference

Suppose the KB contains just the following:
\[
\forall x \text{ King}(x) \land \text{ Greedy}(x) \Rightarrow \text{ Evil}(x)
\]
King(John)
Greedy(John)
Brother(Richard,John)

• Instantiating the universal sentence in all possible ways, we have:
  King(John) \land \text{ Greedy}(John) \Rightarrow \text{ Evil}(John)
  King(Richard) \land \text{ Greedy}(Richard) \Rightarrow \text{ Evil}(Richard)
  King(John)
  Greedy(John)
  Brother(Richard,John)

• The new KB is \textbf{propositionalized}: proposition symbols are
  King(John), \text{ Greedy}(John), \text{ Evil}(John), \text{ King}(Richard), \text{ etc.}
Reduction contd.

- Claim: a ground sentence is entailed by new KB iff entailed by original KB.
- Claim: Every FOL KB can be propositionalized so as to preserve entailment.
- Idea: propositionalize KB and query, apply resolution, return result.
- Problem: with function symbols, there are infinitely many ground terms,
  – e.g., $\text{Father}(\text{Father}(\text{Father}(\text{John})))$
Reduction contd.

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 forever 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.)
Problems with propositionalization

• Propositionalization seems to generate lots of irrelevant sentences.

• E.g., from:
  \[\forall x \text{ King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)\]
  \[\text{King(John)}\]
  \[\forall y \text{Greedy}(y)\]
  \[\text{Brother(Richard,John)}\]

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

• With \(p\) \(k\)-ary predicates and \(n\) constants, there are \(p \cdot n^k\) instantiations. With function symbols, it is even more than this.
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 $\alpha\theta = \beta\theta$, (meaning $\text{subst}(\theta, \alpha) = \text{subst}(\theta, \beta)$)

<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., $\text{Knows}(z_{17}, \text{OJ})$
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 $\alpha\theta = \beta\theta$, (meaning $\text{subst}(\theta, \alpha) = \text{subst}(\theta, \beta)$)

<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

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

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

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

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

<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/John}</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td>{y/John,x/Mother(John)}</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., \( \text{Knows}(z_{17},OJ) \)
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 $\alpha\theta = \beta\theta$, (meaning $\text{subst}(\theta, \alpha) = \text{subst}(\theta, \beta)$)

<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>${\text{fail}}$</td>
</tr>
</tbody>
</table>

• The idea of “Standardizing apart” eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$ so that the last case won’t fail.
Unification

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

- 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.
  
  $\text{MGU} = \{y/John, x/z\}$
  
  – it is the MGU that we desire.
The unification algorithm

function \text{UNIFY}(x, y, \theta) \text{ returns} \text{ a substitution to make } x \text{ and } y \text{ identical}

\text{inputs: } x, \text{ a variable, constant, list, or compound}
\hspace{1cm} y, \text{ a variable, constant, list, or compound}
\hspace{1cm} \theta, \text{ the substitution built up so far}

\text{if } \theta = \text{failure} \text{ then return failure}
\text{else if } x = y \text{ then return } \theta
\text{else if } \text{VARIABLE?}(x) \text{ then return } \text{UNIFY-VAR}(x, y, \theta)
\text{else if } \text{VARIABLE?}(y) \text{ then return } \text{UNIFY-VAR}(y, x, \theta)
\text{else if } \text{COMPOUND?}(x) \text{ and } \text{COMPOUND?}(y) \text{ then}
\hspace{1cm} \text{return } \text{UNIFY}([\text{ARGS}[x], \text{ARGS}[y], \text{UNIFY}([\text{OP}[x], \text{OP}[y], \theta)])
\text{else if } \text{LIST?}(x) \text{ and } \text{LIST?}(y) \text{ then}
\hspace{1cm} \text{return } \text{UNIFY}([\text{REST}[x], \text{REST}[y], \text{UNIFY}([\text{FIRST}[x], \text{FIRST}[y], \theta)])
\text{else return failure}
The unification algorithm

function UNIFY-VAR(var, x, θ) returns a substitution
inputs: var, a variable
        x, any expression
        θ, the substitution built up so far

if \{var/val\} ∈ θ then return UNIFY(val, x, θ)
else if \{x/val\} ∈ θ then return UNIFY(var, val, θ)
else if OCCUR-CHECK?(var, x) then return failure
else return add \{var/x\} to θ

• the occur-check function makes sure that var does not occur in x expression (if it does, it would lead to circularity)
Generalized Modus Ponens (GMP)

\[
\frac{p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{q_\theta}
\]

where \( p_i'\theta = p_i \theta \) for all \( i \)

- \( p_1' \) is \text{King}(\text{John})
- \( p_2' \) is \text{Greedy}(\text{y})
- \( \theta \) is \{x/\text{John}, y/\text{John}\}
- \( q \) is \text{Evil}(x)
- \( q_\theta \) is \text{Evil}(\text{John})

- GMP used with KB of definite clauses (exactly one positive literal, recall Horn clauses). This is same as an implication: e.g.,

\[
p_1 \land p_2 \land \ldots \land p_n \Rightarrow q
\]

- 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
Example knowledge base

• The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

• Prove that Col. West is a criminal
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

\[ American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x) \]

Nono … has some missiles, i.e., \( \exists x \) Owns(Nono,x) \land Missile(x):

\[ Owns(Nono,M_1) \land Missile(M_1) \]

... all of its missiles were sold to it by Colonel West

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

Missiles are weapons:

\[ Missile(x) \Rightarrow Weapon(x) \]

An enemy of America counts as "hostile":

\[ Enemy(x,America) \Rightarrow Hostile(x) \]

West, who is American …

\[ American(West) \]

The country Nono, an enemy of America …

\[ Enemy(Nono,America) \]
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
    \((p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)\)
    for each \(\theta\) such that \((p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta\)
      for some \(p'_1, \ldots, p'_n\) in KB
        \(q' \leftarrow \text{SUBST}(\theta, q)\)
        if \(q'\) is not a renaming of a sentence already in KB or new then do
          add \(q'\) to new
          \(\phi \leftarrow \text{UNIFY}(q', \alpha)\)
          if \(\phi\) is not fail then return \(\phi\)
        add new to KB
  return false
Forward chaining proof
Forward chaining proof
Forward chaining proof
Properties of forward chaining

- Sound and complete for first-order definite clauses
- **Datalog** = first-order definite clauses + no functions (e.g., the crime KB)
- 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
Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration \( k \) if a premise wasn't added on iteration \( k-1 \) 
\[ \Rightarrow \text{match each rule whose premise contains a newly added positive literal} \]

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**
Hard matching example

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

\begin{align*}
\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})
\end{align*}

- \text{Colorable()}\text{ is inferred iff the CSP has a solution}

- CSPs include 3SAT as a special case, hence matching is NP-hard
Backward chaining algorithm

function FOL-BC-Ask(KB, goals, \( \theta \)) returns a set of substitutions
text
inputs: KB, a knowledge base
goals, a list of conjuncts forming a query
\( \theta \), the current substitution, initially the empty substitution \{ \}
local variables: ans, a set of substitutions, initially empty
text
if goals is empty then return \{\( \theta \)\}
\( q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals)) \)
for each \( r \) in KB where STANDARDIZE-APART(r) = (\( p_1 \land \ldots \land p_n \Rightarrow q \))
and \( \theta' \leftarrow \text{UNIFY}(q, q') \) succeeds
\( \text{ans} \leftarrow \text{FOL-BC-Ask}(KB, [p_1, \ldots, p_n|\text{REST}(goals)], \text{COMPOSE}(\theta, \theta')) \cup \text{ans} \)
return \( \text{ans} \)

\[ \text{SUBST}(	ext{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p)) \]
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Properties of backward chaining

• Depth-first recursive proof search: space is linear in size of proof
• Incomplete due to infinite loops
  – ⇒ 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)
  – memoization
• 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 = head :- literal₁, ... literalₙ.
  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., 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 alive(X) :- not dead(X).
  - alive(joe) succeeds if dead(joe) fails
Prolog

- Appending two lists to produce a third:
  
  \[
  \text{append}([], Y, Y).
  \]
  
  \[
  \text{append}([X|L], Y, [X|Z]) :- \text{append}(L, Y, Z).
  \]

- query: \( \text{append}(A, B, [1, 2]) \) ?

- answers: \[
A=[] \quad B=[1,2]
\]

\[
\]

\[
A=[1,2] \quad B=[]
\]
Resolution: brief summary

- Full first-order version:
  \[
  \begin{array}{c}
  \ell_1 \lor \cdots \lor \ell_k, \\
  m_1 \lor \cdots \lor m_n
  \end{array}
  \]

  \[
  \begin{array}{c}
  (\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta
  \end{array}
  \]

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

- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,
  \[
  \neg \text{Rich}(x) \lor \text{Unhappy}(x)
  \]

  \[
  \begin{array}{c}
  \text{Rich}(\text{Ken})
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{Unhappy}(\text{Ken})
  \end{array}
  \]

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

- Apply resolution steps to \(\text{CNF}(\text{KB} \land \neg \alpha)\); complete for FOL
Conversion to CNF

• Everyone who loves all animals is loved by someone:
  \[ \forall x \left[ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x,y) \right] \Rightarrow [\exists y \text{Loves}(y,x)] \]

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

• 2. Move \( \neg \) inwards:
  \[ \neg \forall x \ p \equiv \exists x \ \neg p, \ \neg \exists x \ p \equiv \forall x \ \neg p \]

  \[ \forall x \left[ \exists y \neg (\neg\text{Animal}(y) \lor \text{Loves}(x,y)) \right] \lor [\exists y \text{Loves}(y,x)] \]
  \[ \forall x \left[ \exists y \neg\text{Animal}(y) \land \neg\text{Loves}(x,y) \right] \lor [\exists y \text{Loves}(y,x)] \]
  \[ \forall x \left[ \exists y \text{Animal}(y) \land \neg\text{Loves}(x,y) \right] \lor [\exists y \text{Loves}(y,x)] \]
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:
\[ \forall x \left[ Animal(F(x)) \land \neg Loves(x,F(x)) \right] \lor Loves(G(x),x) \]

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

6. Distribute \lor over \land:
\[ \left[ Animal(F(x)) \lor Loves(G(x),x) \right] \land \left[ \neg Loves(x,F(x)) \lor Loves(G(x),x) \right] \]
Resolution proof: definite clauses

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

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

\[
\neg Missile(x) \lor Weapon(x)
\]

\[
\neg Missile(x) \lor \neg Owns(Nono,x) \lor Sells(West,x,Nono)
\]

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

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

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

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

\[
\neg Hostile(Nono)
\]

\[
\neg Enemy(Nono,America)
\]

\[
\neg Enemy(Nono,America)
\]
Summary

• Knowledge engineering example (circuits)
• Reducing first-order inference to propositional inference
• Unification
• Generalized Modus Ponens
• Forward chaining
• Backward chaining
• Resolution