Set 8: Inference in First-order logic

ICS 271 Fall 2012
Chapter 9: Russell and Norvig
Outline

◊ Reducing first-order inference to propositional inference
◊ Unification
◊ Generalized Modus Ponens
◊ Forward and backward chaining
◊ Logic programming
◊ Resolution
Universal instantiation (UI)

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

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

  \[
  \text{King}(\text{John}) \land \text{Greedy}(\text{John}) \supset \text{Evil}(\text{John}) \\
  \text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \supset \text{Evil}(\text{Richard}) \\
  \text{King}(\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John})) \supset \text{Evil}(\text{Father}(\text{John}))
  \]

  Obtained by substituting \(\{x/\text{John}\}, \{x/\text{Richard}\}\) and \(\{x/\text{Father}(\text{John})\}\)
Existential instantiation (EI)

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

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

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

$$Crown(C_1) \land OnHead(C_1, John)$$

provided $C_1$ is a new constant symbol, called a Skolem constant
Reduction to propositional inference

Suppose the KB contains just the following:

\[ \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \]

King(John)
Greedy(John)
Brother(Richard,John)

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

- The new KB is propositionalized: proposition symbols are

  King(John), Greedy(John), Evil(John), King(Richard), etc.
Reduction contd.

- Every FOL KB can be propositionalized so as to preserve entailment
  
  (A 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 infinitely many ground terms,
  
    - e.g., $Father(Father(Father(John)))$
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 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:

\[
\forall x \text{ King}(x) \land \text{ Greedy}(x) \Rightarrow \text{ Evil}(x)
\]
\[
\text{King}(\text{John})
\]
\[
\forall y \text{ Greedy}(y)
\]
\[
\text{Brother}(\text{Richard}, \text{John})
\]

- Given query “evil(x) it seems obvious that Evil(John), but propositionalization produces lots of facts such as Greedy(Richard) that are irrelevant
- With \( p \) \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations.
Generalized Modus Ponens (GMP)

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

\[ \frac{q}{q' \theta} \]

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

- \( p_1' \) is \( \text{King}(\text{John}) \)
- \( p_1 \) is \( \text{King}(x) \)
- \( p_2' \) is \( \text{Greedy}(y) \)
- \( p_2 \) is \( \text{Greedy}(x) \)
- \( \theta \) 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)

- All variables assumed universally quantified
Soundness of GMP

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

- Lemma: For any sentence \( p \), we have \( p \vdash p\theta \) by UI

1. \( (p_1 \land \ldots \land p_n \Rightarrow q) \vdash (p_1 \land \ldots \land p_n \Rightarrow q)\theta = (p_1\theta \land \ldots \land p_n\theta \Rightarrow q\theta) \)
2. \( p_1', \ldots, ;p_n' \vdash p_1' \land \ldots \land p_n' \vdash p_1'\theta \land \ldots \land p_n'\theta \)
3. From 1 and 2, \( q\theta \) follows by ordinary Modus Ponens
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></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>
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 $\alpha\theta = \beta\theta$

\[
\begin{array}{ccc}
\text{p} & \text{q} & \theta \\
\text{Knows(John,x)} & \text{Knows(John,Jane)} & \{x/Jane\} \\
\text{Knows(John,x)} & \text{Knows(y,OJ)} & \\
\text{Knows(John,x)} & \text{Knows(y,Mother(y))} & \\
\text{Knows(John,x)} & \text{Knows(x,OJ)} & \\
\end{array}
\]
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

• $Unify(\alpha, \beta) = \theta$ 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>${x/OJ,y/John}$</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 \( King(x) \) and \( Greedy(x) \) match \( King(John) \) and \( Greedy(y) \)

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

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

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

• To unify $Knows(John, x)$ and $Knows(y, z)$, 
  $\theta = \{y/John, x/z\}$ 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.
  MGU = $\{y/John, x/z\}$
The unification algorithm

function \textsc{Unify}(x, y, \theta) \textbf{returns} a substitution to make \textit{x} and \textit{y} identical

\textbf{inputs:} \textit{x}, a variable, constant, list, or compound

\textit{y}, a variable, constant, list, or compound

\textit{\theta}, the substitution built up so far

if \theta = \text{failure} then return failure
else if \textit{x} = \textit{y} then return \theta
else if \text{\textsc{Variable}(x)} then return \textsc{Unify-Var}(x, y, \theta)
else if \text{\textsc{Variable}(y)} then return \textsc{Unify-Var}(y, x, \theta)
else if \text{\textsc{Compound}(x) and \textsc{Compound}(y)} then
    \textbf{return} \textsc{Unify}(\textsc{Args}[x], \textsc{Args}[y], \textsc{Unify}(\textsc{Op}[x], \textsc{Op}[y], \theta))
else if \text{\textsc{List}(x) and \textsc{List}(y)} then
    \textbf{return} \textsc{Unify}(\textsc{Rest}[x], \textsc{Rest}[y], \textsc{Unify}(\textsc{First}[x], \textsc{First}[y], \theta))
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
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
... 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., \( \exists x \text{ Owns}(\text{Nono},x) \land \text{Missile}(x) \):

\[ \text{Owns}(\text{Nono},M_1) \land \text{Missile}(M_1) \]
... 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 ... 

\[ \text{Enemy}(\text{Nono},\text{America}) \]
function FOL-FC-Ask(KB, α) returns a substitution or false
    repeat until new is empty
        new ← {}
        for each sentence r in KB do
            (p₁ ∧ ... ∧ pₙ ⇒ q) ← STANDARDIZE-APART(r)
            for each θ such that (p₁ ∧ ... ∧ pₙ)θ = (p₁' ∧ ... ∧ pₙ')θ
                for some p₁', ..., pₙ' 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
Forward chaining proof

American(West)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)
Forward chaining proof

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

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

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

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

*American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
*Owns(Nono,M1) and Missile(M1)
*Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)
*Missile(x) ⇒ Weapon(x)
*Enemy(x,America) ⇒ Hostile(x)
*American(West)
*Enemy(Nono,America)
Properties of forward chaining

• Sound and complete for first-order definite clauses

• **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
Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration $k$ if a premise wasn't added on iteration $k-1$

$\Rightarrow$ 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 $Missile(x)$ retrieves $Missile(M_1)$

Forward chaining is widely used in deductive databases
Hard matching example

- **Colorable()** is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard

\[
\begin{align*}
\text{Diff}(wa, nt) & \land \text{Diff}(wa, sa) \land \text{Diff}(nt, q) \land \\
\text{Diff}(nt, sa) & \land \text{Diff}(q, nsw) \land \text{Diff}(q, sa) \land \\
\text{Diff}(nsw, v) & \land \text{Diff}(nsw, sa) \land \text{Diff}(v, sa) \Rightarrow \\
\text{Colorable}()
\end{align*}
\]

\[
\begin{align*}
\text{Diff}(\text{Red}, \text{Blue}) & \quad \text{Diff}(\text{Red}, \text{Green}) \\
\text{Diff}(\text{Green}, \text{Red}) & \quad \text{Diff}(\text{Green}, \text{Blue}) \\
\text{Diff}(\text{Blue}, \text{Red}) & \quad \text{Diff}(\text{Blue}, \text{Green})
\end{align*}
\]
Backward chaining example

Criminal(West)
Backward chaining example

```
American(x)  Weapon(y)  Sells(x,y,z)  Hostile(z)
```

```
Criminal(West)
{x/West}
```
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining algorithm

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 {} 
   q' ← Subst(θ, First(goals)) 
   for each r in KB where Standardize-Apart(r) = (p_1 ∧ ... ∧ p_n ⇒ q) and θ' ← Unify(q, q') succeeds 
      ans ← FOL-BC-Ask(KB, [p_1, ..., p_n | Rest(goals)], Compose(θ, θ')) ∪ ans 
   return ans

\[
\text{Subst(Compose(θ_1, θ_2), p)} = \text{Subst(θ_2, Subst(θ_1, p))}
\]
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)
- 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., \( X \text{ is } Y \times Z + 3 \)
- 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 \( \text{alive}(X) :- \text{not } \text{dead}(X) \).
  - \( \text{alive}(joe) \) succeeds if \( \text{dead}(joe) \) 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=\[] B=[1, 2]


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

- Full first-order version:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
\ell_1 \lor \cdots \lor \ell_i-1 \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n)\theta
\]

where \text{Unify}(\ell, \neg m) = \emptyset.

- The two clauses are assumed to be standardized apart so that they share no variables.

- For example,

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

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

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

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:
  \( \forall x [\forall y \text{Animal}(y) \Rightarrow \text{Loves}(x,y)] \Rightarrow [\exists y \text{Loves}(y,x)] \)

• 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: \( \neg \forall x \ p \equiv \exists x \ \neg p, \ \neg \exists x \ p \equiv \forall x \ \neg p \)
  
  \begin{align*}
  \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 \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)]
  \end{align*}
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 $\land$:

$$\left[ Animal(F(x)) \lor Loves(G(x),x) \right] \land \left[ \neg Loves(x,F(x)) \lor Loves(G(x),x) \right]$$
... 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 …

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

Missile(M1)

\neg Missile(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

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

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

Missile(M1)

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

Owns(Nono,M1)

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

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

\neg Hostile(Nono)

Enemy(Nono, America)

Enemy(Nono, America)
Converting to clause form

\( \forall x, y \ P(x) \land P(y) \land I(x, 27) \land I(y, 28) \rightarrow S(x, y) \)

\( P(A), P(B) \)

\( I(A, 27) \lor I(A, 28) \)

\( I(B, 27) \)

\( \neg S(B, A) \)

Prove \( I(A, 27) \)
Example: Resolution
Refutation Prove \( I(A,27) \)
Example: Answer Extraction

\[\neg I(A, u) \vee \text{Ans}(u) \]
(negation of wff to be proved with answer literal)

\[I(A, 28) \vee \text{Ans}(27) \]

\[\neg P(x) \vee \neg P(y) \vee \neg I(x, 27) \vee \neg I(y, 28) \vee S(x, y)\]