Set 8: Inference in First-order logic

ICS 271 Fall 2015
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:

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

for any variable \( v \) and ground term \( g \)

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

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

*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:

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

- 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 (not used so far) constant term, called a Skolem constant.

- Skolemization: $\exists$ elimination
  - $\forall x \ \exists y \ Loves(y,x)$
  - Incorrect inference: $\forall x \ Loves(A,x)$ – $y$ may be different for each $x$
  - Correct inference: $\forall x \ Loves(f(x),x)$
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 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., $\text{Father(Father(Father(John)))}$
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 (will terminate) 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 non-entailed 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)

\[
p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \quad \frac{\text{where } p_i'\theta = p_i \theta \text{ for all } i}{q\theta}
\]

King(John), Greedy(y), (King(x) \land Greedy(x) \Rightarrow Evil(x))

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

- 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
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 $\alpha \theta = \beta \theta$
  - note: replace variables with terms!

<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., 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 $Greedy(y)$

$\theta = \{x/John, y/John\}$ 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></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

• 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>${x/\text{OJ}, y/\text{John}}$</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

• 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>${y/\text{John}, x/\text{Mother(John)}}$</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., Knows($z_{17}$,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 $Greedy(y)$

$\theta = \{x/John,y/John\}$ 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>$\emptyset$</td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g., $\text{Knows}(z_{17},\text{OJ})$
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.
  $\text{MGU} = \{y/John, x/z\}$
The unification algorithm

function UNIFY(x, y, θ) returns a substitution to make x and y identical
inputs: x, a variable, constant, list, or compound
        y, a variable, constant, list, or compound
        θ, the substitution built up so far

if θ = failure then return failure
else if x = y then return θ
else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
else if LIST?(x) and LIST?(y) then
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
else return failure
The unification algorithm

function $\text{UNIFY-VAR}(\text{var}, x, \theta)$ returns a substitution

inputs: $\text{var}$, a variable
$x$, any expression
$\theta$, the substitution built up so far

if $\{\text{var}/\text{val}\} \in \theta$ then return $\text{UNIFY}(\text{val}, x, \theta)$
else if $\{x/\text{val}\} \in \theta$ then return $\text{UNIFY}(\text{var}, \text{val}, \theta)$
else if $\text{OCCUR-CHECK?}(\text{var}, x)$ then return failure
else return add $\{\text{var}/x\}$ to $\theta$
Unification

- Basic task: unify
  - \( p_1, p_2, \ldots, p_n \)
  - \( q_1, q_2, \ldots, q_n \)
- Proceed left to right, carry along current substitution \( \theta \)
- Compare \( p_i \) with \( q_i \),
  - predicates must match
  - apply existing substitution
  - unify instantiated pair, producing \( \theta_i \)
  - add new substitution to existing \( \theta = \theta \cup \theta_i \)
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, cont.

... 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 …

\[ \text{Enemy}(\text{Nono},\text{America}) \]
function FOL-FC-Ask(\(KB, \alpha\)) returns a substitution or false

repeat until new is empty

\(new \leftarrow \{\}\)

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

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

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

Missile(x) \Rightarrow Weapon(x)

Enemy(x,America) \Rightarrow Hostile(x)
Forward chaining proof

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

*American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)
*Owns(Nono,M1) \land Missile(M1)
*Missile(x) \land Owns(Nono,x) \Rightarrow Sells(West,x,Nono)
*Missile(x) \Rightarrow Weapon(x)
*Enemy(x,America) \Rightarrow Hostile(x)
*American(West)
*Enemy(Nono,America)
Properties of forward chaining

• Forward chaining is widely used in deductive databases

• Sound and complete for first-order definite clauses
• May not terminate in general if $\alpha$ is not entailed
• This is unavoidable: entailment with definite clauses is semidecidable

• Datalog = first-order definite clauses + no functions
  – FC terminates for Datalog in finite number of iterations ($p \cdot n^k$ ground terms)
Matching facts against rules:

Hard matching example

- $\text{Colorable()}$ is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard
- Query complexity vs. data complexity

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

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

\[ p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)_{q\theta} \]

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

- Pattern matching itself can be expensive:
  - Use indexing to unify sentences that have a chance of unifying
    - Knows(x,y) vs Brother(u,v)
  - Database indexing allows O(1) retrieval of known facts
  - e.g., query \( \text{Missile}(x) \) retrieves \( \text{Missile}(M_1) \)
Efficiency of forward chaining

\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \quad \text{where } p_i' \theta = p_i \theta \text{ for all } i \]

- Matching rules against known facts

Conjunct ordering problem
Missile(x) \land Owns(Nono,x) \Rightarrow Sells(West, x, Nono)

NP-hard in general, but can use heuristics used for CSPs

Rule-matching tractable when CSP is tractable
1. Incremental forward chaining: no need to match a rule on iteration $k$ if a premise wasn't added on iteration $k-1$

   ⇒ match each rule whose premise contains a newly added positive literal

2. Retain partial matches and complete them incrementally as new facts arrive
Efficiency of forward chaining

Forward chaining infers everything, most of which can be irrelevant to the goal

- Solution: allow only those bindings that are relevant to the goal
  - Use generic backward chaining
- Add Magic(x) extra conjunct to rules and Magic(c) to the KB
  - E.g. Magic(West)
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example

```
Criminal(West)  \{x/West, y/M1, z/Nono\}

American(West) \{}\n
Weapon(y) \n
Sells(West, M1, z)  \{z/Nono\}

Hostile(z) \n
Missile(y)  \{y/M1\}

Missile(M1) \n
Owns(Nono, M1) \n```
Backward chaining example
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
```

**SUBST(COMPOSE(θ₁, θ₂), p) = SUBST(θ₂, SUBST(θ₁, p))**
Properties of backward chaining

• Depth-first recursive proof search: space is linear in size of proof
  – But not in size of data (bindings)
• 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 (Prolog)
Prolog

• Appending two lists to produce a third:

append([], Y, Y).
append([X|L], Y, [X|Z]) :- append(L, Y, Z).

• query: 
append(A, B, [1, 2]) ?

• answers: 
A= []   B= [1, 2]  
A= [1, 2] B= []
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 (within rule), top-down (within rule-set) 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)
- No occurs-check in unification – may produce results not entailed
- No checks for infinite loops – incomplete even for definite clauses
- Prolog : no caching; Tabled Logic Programming : memoization
- Database semantics :
  – Unique names assumption
  – Closed-world assumption ("negation as failure")
    • e.g., given alive(X) :- not dead(X).
    • alive(joe) succeeds if dead(joe) fails
  – Closed domain assumption
Resolution: brief summary

- Full first-order version:

\[
\begin{array}{c}
\ell_1 \lor \cdots \lor \ell_k \\
\lor \cdots \lor m_1 \lor \cdots \lor m_n \\
\end{array}
\]

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

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

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

- For example,

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

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

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

- Apply resolution steps to CNF(KB \land \neg \alpha); complete (with factoring) for FOL
Conversion to CNF

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

• 1. Eliminate biconditionals and implications
  - \( A \iff B \) becomes \((A \Rightarrow B) \land (B \Rightarrow A)\)
  - \( A \Rightarrow B \) becomes \( \neg A \lor B \)

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

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

  \( \forall x \ [\exists y \ \neg(\neg Animal(y) \lor Loves(x,y))] \lor [\exists y \ Loves(y,x)] \)
  \( \forall x \ [\exists y \ \neg Animal(y) \land \neg Loves(x,y)] \lor [\exists y \ Loves(y,x)] \)
  \( \forall x \ [\exists y \ Animal(y) \land \neg Loves(x,y)] \lor [\exists y \ Loves(y,x)] \)
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., \(\exists x \text{ Owns}(\text{Nono},x) \land \text{Missile}(x)\):
\[\text{Owns}(\text{Nono},M_1) \land \text{Missile}(M_1)\]

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

American(West)

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

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

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

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)

Missile(M1)

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

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

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

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

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

\neg Enemy(Nono,America)

\neg Enemy(Nono,America)
Efficient Resolution

• Resolution proofs can be long

• Strategies:
  – Unit Preference
  – Set of support
  – Input resolution
    • Complete for Horn clauses
  – Linear Resolution
    • Complete in general
Converting to clause form (Try this example)

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

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

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

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

\[-P(x) \lor -P(y) \lor -I(x,27) \lor -I(y,28) \lor S(x,y)\]

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

\[-P(x) \lor -P(A) \lor -I(x,27) \lor S(x,A) \lor \text{Ans}(27)\]

\[P(A)\]

\[P(B)\]

\[I(B,27)\]

\[-I(B,27) \lor S(B,A) \lor \text{Ans}(27)\]

\[S(B,A) \lor \text{Ans}(27)\]

\[\text{Ans}(27)\]
Unit resolution is (refutation) complete for Horn clauses: example

(Non-unit) resolution is complete!

Can transform any non-unit resolution proof to unit resolution proof
Unit resolution is (refutation) complete for Horn clauses: general case

Only unit resolution here. $S_2$ eliminated one at a time.
Non-Horn clauses?