First Order Predicate Logic

- Propositional logic
  - assumes the world contains propositions
  - has limited expressive power
- First-order predicate logic (like natural language)
  - assumes the world contains
    - Objects:
      - people, flowers, houses, numbers, students,
    - Relations:
      - red, round, prime, brother of, bigger than, part of
    - Functions:
      - father of, best friend, plus, …
  - Allows one to talk about some or all of the objects
Ontological and Epistemological Commitments

<table>
<thead>
<tr>
<th>Ontological Commitments</th>
<th>Epistemological Commitments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional Logic</td>
<td>Facts</td>
</tr>
<tr>
<td>First Order Predicate Logic</td>
<td>Facts, Objects, Relations</td>
</tr>
<tr>
<td>Probability Theory</td>
<td>Facts</td>
</tr>
</tbody>
</table>

Syntax of FOL: Basic elements

- Constants
  - Oksana, 2, Iowa-State-University
- Predicates
  - Brother, Father, Teacher, Red
- Functions
  - Successor ( ), Plus
- Variables x, y, a, b,...
- Connectives ¬, ⇒, ∧, ∨, ⇔
- Equality =
- Quantifiers ∀, ∃
Atomic sentences

• Term
  – function \((\text{term}_1, \ldots, \text{term}_n)\),
    - e.g. house_of (John)
  – constant
    - e.g., John, 5
  – or variable
    - e.g., x, y, z
• Predicates
  – E.g., Brother(George, Jeb)

Compound sentences

• Compound sentences are made from atomic sentences using connectives
  - \(\neg S\),
  - \(S_1 \land S_2\),
  - \(S_1 \lor S_2\),
  - \(S_1 \Rightarrow S_2\),
  - \(S_1 \Leftrightarrow S_2\),

E.g. Brother(George, Jeb) \(\Rightarrow\) Sibling(George, Jeb)
Truth in first-order logic

- Sentences are true with respect to a model and an interpretation
- Model contains relations among objects
- Interpretation specifies referents for
  - constant symbols $\rightarrow$ objects
  - predicate symbols $\rightarrow$ relations
  - function symbols $\rightarrow$ functions

- An atomic sentence $\text{predicate}(\text{term}_1, \ldots, \text{term}_n)$ is true iff the objects referred to by $\text{term}_1, \ldots, \text{term}_n$ are in the relation referred to by $\text{predicate}$

FOL Models - Example

- Object Constants: $A$, $B$, $\text{Table}$
- Relation Constant: $\text{On}$

Model

$\text{On} (A, B)$
$\text{On} (B, \text{Table})$
Models for FOL

- In principle, we can enumerate the models for a given KB vocabulary
- Computing entailment by enumerating the models will not be easy!!

For each number of domain elements $n$ from 1 to $\infty$
For each $k$-ary predicate $P_k$ in the vocabulary
For each possible $k$-ary relation on $n$ objects
For each constant symbol $C$ in the vocabulary
For each choice of referent for $C$ from $n$ objects...

Quantifiers

- Allow us to express properties of collections of objects instead of enumerating objects by name
  - Universal: “for all” $\forall$
  - Existential: “there exists” $\exists$

$\forall x \ Human(x) \Rightarrow Mortal(x)$
$\forall z \ Petdog(z) \Rightarrow \exists y \ Human(y) \land Caresfor(y,z)$
Universal quantification

- $\forall<\text{variables}> <\text{sentence}>$
- Everyone at ISU is smart: $\forall x \ At(x, ISU) \Rightarrow Smart(x)$
- $\forall x \ P(x)$ is true in a model $m$ iff $P$ is true with $x$ instantiated to each possible object in the world
- Roughly speaking, $\forall x \ P(x)$ is equivalent to the conjunction of instantiations of $P$

\[
\begin{align*}
(At(Matt, ISU) \Rightarrow Smart(Matt)) \land \\
(At(Oksana, ISU) \Rightarrow Smart(Oksana)) \land \\
(At(Fido, ISU) \Rightarrow Smart(Fido)) \land \\
\ldots.
\end{align*}
\]

A common mistake to avoid

- A universally quantifier is also equivalent to a set of implications over all objects
- Common mistake: using $\land$ as the main connective with $\forall$:

\[
\forall x \ At(x, ISU) \land Smart(x)
\]

Means
- “Everyone is at ISU and everyone is smart” as opposed to
- “Everyone at ISU is smart”
Existential quantification

\[ \exists \langle \text{variables} \rangle \langle \text{sentence} \rangle \]

Someone at ISU is smart:
\[ \exists x \, \text{At}(x, ISU) \land \text{Smart}(x) \]

\[ x P \text{ is true in a model } m \text{ iff } P \text{ is true with } x \text{ being some possible object in the model} \]

• Roughly speaking, equivalent to the disjunction of instantiations of \( P(x) \)

\[ \text{At}(Matt, ISU) \land \text{Smart}(Matt) \lor \text{At}(Oksana, ISU) \land \text{Smart}(Oksana) \lor \text{At}(Fido, ISU) \land \text{Smart}(Fido) \]

Another common mistake to avoid

• Common mistake: using \( \Rightarrow \) as the main connective with \( \exists \):

\[ \exists x \, \text{At}(x, ISU) \Rightarrow \text{Smart}(x) \]

is true even if there is someone that is smart who is not at ISU!
Properties of quantifiers

∀x ∃y is the same as ∃y ∀x
∃x ∀y is the same as ∀y ∃x

∃x ∃y is not the same as ∃y ∃x
∃x ∃y Loves(x,y)
  – “There is a person who loves everyone in the world”
∀y ∃x Loves(x,y)
  – “Everyone in the world is loved by someone”

Quantifier Duality

Duality: “Everyone dislikes Parsnips” ≡
“there is no one who likes Parsnips”
∀x ¬Likes(x, Parsnips) ≡ ¬∃x Likes(x, Parsnips)

De Morgan Rules:

∀x ¬P ≡ ¬∃x P  ¬P ∧ ¬Q ≡ ¬(P ∨ Q)
¬∀x P ≡ ∃x ¬P ¬(P ∧ Q) ≡ ¬P ∨ ¬Q
∀x P ≡ ¬∃x ¬P  P ∨ Q ≡ ¬(¬P ∨ ¬Q)
¬∀x ¬P ≡ ∃x P  (¬P ∧ ¬Q) ≡ P ∨ Q

Equality

- $\text{term}_1 = \text{term}_2$ is true under a given interpretation if and only if $\text{term}_1$ and $\text{term}_2$ refer to the same object

- E.g., definition of $\text{Sibling}$ in terms of $\text{Parent}$:
  \[
  \forall x, y \; \text{Sibling}(x,y) \iff [\neg (x = y) \land \exists m, f \; (m = f) \land \text{Parent}(m,x) \land \text{Parent}(f,x) \land \text{Parent}(m,y) \land \text{Parent}(f,y)]
  \]

Interacting with FOL KBs

- Given a sentence $S$ and a substitution $\alpha$,
  - $S\alpha$ denotes the result of plugging $\alpha$ into $S$; e.g.,
    \[
    S = \text{Smarter}(x,y)
    \]
    \[
    \alpha = \{x/\text{Hillary}, y/\text{Bill}\}
    \]
    \[
    S\alpha = \text{Smarter}(\text{Hillary}, \text{Bill})
    \]

- $\text{Ask}(\text{KB}, S)$ returns some/all $\alpha$ such that $\text{KB} \models S\alpha$. 
Using FOL

The kinship domain:
• Brothers are siblings
  \( \forall x, y \; \text{Brother}(x, y) \iff \text{Sibling}(x, y) \)
• One’s mother is one’s female parent
  \( \forall m, c \; \text{Mother}(c) = m \iff (\text{Female}(m) \land \text{Parent}(m, c)) \)
• “Sibling” is symmetric
  \( \forall x, y \; \text{Sibling}(x, y) \iff \text{Sibling}(y, x) \)
• A first cousin is a child of a parent’s sibling
  \( \forall x, y \; \text{FirstCousin}(x, y) \iff \exists u, v \; \text{Parent}(u, x) \land \text{Sibling}(v, u) \land \text{Parent}(v, y) \)

FOL Examples

• Predicates:
  \( \text{Purple}(x), \text{Mushroom}(x), \text{Poisonous}(x), \text{Equal}(x, y) \)
• All purple mushrooms are poisonous
  \( \forall x \; \text{Purple}(x) \land \text{Mushroom}(x) \Rightarrow \text{Poisonous}(x) \)
• Some purple mushrooms are poisonous
  \( \exists x \; \text{Purple}(x) \land \text{Mushroom}(x) \land \text{Poisonous}(x) \)
• No purple mushrooms are poisonous
  \( \forall x \; \text{Purple}(x) \land \text{Mushroom}(x) \Rightarrow \neg \text{Poisonous}(x) \)
• There is exactly one mushroom
  \( \exists x \; \text{Mushroom}(x) \land \forall y \; \text{Mushroom}(y) \Rightarrow \text{Equal}(x, y) \)
Knowledge engineering in FOL

1. Identify the task (what will the KB be used for)
2. Assemble the relevant knowledge (knowledge acquisition)
3. Decide on a vocabulary of predicates, functions, and constants
   - Translate domain-level knowledge into logic-level names
4. Encode general knowledge about the domain
   - define axioms
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
Example – Electronic circuit domain

- Identify the task
  - Does the circuit actually add properly? (circuit verification)
- Assemble the relevant knowledge
  - Composed of wires and gates
  - Types of gates (AND, OR, XOR, NOT)
  - Connections between terminals
  - Irrelevant: size, shape, color, cost of gates
- Decide on a vocabulary
  - Alternatives:
    - Type($X_1$) = XOR
    - Type($X_1$, XOR)
    - XOR($X_1$)

Encode knowledge

Encode general knowledge of the domain

\[
\forall t_1, t_2 \, \text{Connected}(t_1, t_2) \Rightarrow \text{Signal}(t_1) = \text{Signal}(t_2) \\
\forall t \, \text{Signal}(t) = 1 \lor \text{Signal}(t) = 0 \\
1 \neq 0 \\
\forall t_1, t_2 \, \text{Connected}(t_1, t_2) \Rightarrow \text{Connected}(t_2, t_1) \\
\forall g \, \text{Type}(g) = \text{OR} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 1 \iff \exists n \, \text{Signal}(\text{In}(n,g)) = 1 \\
\forall g \, \text{Type}(g) = \text{AND} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 0 \iff \exists n \, \text{Signal}(\text{In}(n,g)) = 0 \\
\forall g \, \text{Type}(g) = \text{XOR} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 1 \iff \text{Signal}(\text{In}(1,g)) \neq \text{Signal}(\text{In}(2,g)) \\
\forall g \, \text{Type}(g) = \text{NOT} \Rightarrow \text{Signal}(\text{Out}(1,g)) \neq \text{Signal}(\text{In}(1,g))
\]
The electronic circuits domain

Encode the specific problem instance

Type(X₁) = XOR  Type(X₂) = XOR
Type(A₁) = AND  Type(A₂) = AND
Type(O₁) = OR

Connected(Out(1,X₁),In(1,X₂))  Connected(In(1,C₁),In(1,X₁))
Connected(Out(1,X₁),In(2,A₂))  Connected(In(1,C₁),In(1,A₁))
Connected(Out(1,A₂),In(1,O₁))  Connected(In(2,C₁),In(2,X₁))
Connected(Out(1,A₁),In(2,O₁))  Connected(In(2,C₁),In(2,A₁))
Connected(Out(1,X₂),Out(1,C₁))  Connected(In(3,C₁),In(2,X₂))
Connected(Out(1,O₁),Out(2,C₁))  Connected(In(3,C₁),In(1,A₂))

Pose queries to the inference procedure

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

∃i₁,i₂,i₃,o₁,o₂  Signal(In(1,C₁_1)) = i₁ ∧ Signal(In(2,C₁)) = i₂ ∧
   Signal(In(3,C₁)) = i₃ ∧ Signal(Out(1,C₁)) = o₁ ∧
   Signal(Out(2,C₁)) = o₂

Debug the knowledge base

May have omitted assertions like 1 ≠ 0
Summary

• First-order logic:
  – objects and relations are semantic primitives
  – syntax: logical symbols, constants, functions, predicates, equality, quantifiers

• Increased expressive power

Inference in FOL

• Convert a FOL KB into a corresponding propositional KB
• Adapt techniques developed for propositional inference
  – How to eliminate universal quantifiers?
    • Instantiate variables
  – How to convert existential quantifiers?
    • Skolemization
Universal instantiation (UI)

Example

\( \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) : \)

- \( King(John) \land Greedy(John) \Rightarrow Evil(John) \)
- \( King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard) \)
- \( King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John)) \)

- Every instantiation of a universally quantified sentence is entailed by it
- \( \forall v \alpha, \) entails instantiations obtained by substituting \( v \) with ground terms:
- \( \text{Subst}(\{v/g\}, \alpha) \) denotes instantiation of \( \alpha \) by substituting variable \( v \) with term \( g \)

(Subst \((x|y)\) = substitution of \(y\) by \(x\))

Existential instantiation (EI)

- E.g., \( \exists x \ House(x) \land Ownedby(x,John) \)
- There exists a house owned by John
- Let us name the house whose existence is asserted by the above, John-Villa
- Now, John-Villa is a house, and it is owned by John

\( \text{House}(John-Villa) \land Ownedby(John-Villa,John) \)

John-Villa, a unique name that refers to the house obtained by eliminating the existential quantifier above is called a Skolem constant
Skolemization Examples

Eg: “Everyone has a heart.”

$$\forall X \text{person}(X) \Rightarrow \exists Y \text{heart}(Y) \land \text{has}(X,Y)$$

Incorrect: $$\forall X \text{Person}(X) \Rightarrow \text{heart}(H_1) \land \text{has}(x,H_1)$$

?everyone has the SAME heart H_1?

Correct: $$\forall X \text{person}(X) \Rightarrow \text{heart}(h(X)) \land \text{has}(X,h(X))$$

where h is a new symbol (“Skolem function”)

• Skolem function arguments:
  all enclosing universally quantified variables

Skolemization

• Skolemizing procedure (to remove existentials)

For each existential X, let Y_1, ..., Y_m be the universally quantified variables that are quantified to the LEFT of X’s “∃X”.
Generate new function symbol, g_X, of m variables. Replace each X with g_X(Y_1, ..., Y_m).
(Write g_X() as g_X.)

$$\forall Y \exists X \phi(X) \land \rho(Y) \Rightarrow \forall Y \phi(\frac{g_X(Y)}{X}) \land \rho(Y)$$

$$\exists X \forall Y \phi(X) \land \rho(Y) \Rightarrow \forall Y \phi(\frac{g_X}{X}) \land \rho(Y)$$
Skolemization Theorem

If \[ T_1 = \{ \alpha_1, \ldots, \alpha_2, \exists X \forall Y \varphi(X, Y), \ldots \} \]

is consistent

then \[ s(T_1) = \{ \alpha_1, \alpha_2, \forall Y \varphi(c_1, Y), \ldots \} \]

is consistent.

\[ \ldots \text{if } s(T) \text{ is inconsistent, then } T \text{ is inconsistent} \ldots \]

Universal versus Existential Instantiation

- Universal Instantiation
  - can be applied many times to add new sentences;
  - the new KB is logically equivalent to the old
- Existential Instantiation (Skolemization)
  - can be applied once to eliminate each existential quantifier;
  - the resulting existential quantifier free KB is not equivalent to the old
  - The new KB is satisfiable if the old KB was satisfiable
Reduction to propositional inference

• Suppose the KB contains just the following:

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

• After universal instantiation we get a variable-free, quantifier-
  free KB – a propositionalized KB

  \[ King(John) \land Greedy(John) \Rightarrow Evil(John) \]
  \[ King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard) \]
  \[ King(John) \]
  \[ Greedy(John) \]
  \[ Brother(Richard,John) \]

Reduction of FOL inference to PL inference

• CLAIM: A ground sentence is entailed by a new KB iff
  entailed by the original KB.

• CLAIM: Every FOL KB can be propositionalized so as to
  preserve entailment

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

• PROBLEM: when function symbols are present, it is
  possible to generate infinitely many ground terms:

  e.g., \[ Father(Father(Father(John))) \]
Reduction of FOL inference to PL inference

• **THEOREM**: Herbrand (1930).
  If a sentence \( \alpha \) is entailed by a 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 semi decidable
  – algorithms exist that say yes to every sentence that is entailed by the KB
    • Prove a theorem that in fact follows from the axioms
  – No algorithm exists that also says no to sentence that is not entailed by the KB
    • Algorithm may not terminate
Problems with propositionalization

Given:

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

King(John)

[1] Greedy(y)

Brother(Richard, John)

- 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!
  - Can we avoid unnecessary instantiation of unneeded facts?

Lifting and Unification

- Instead of translating the knowledge base to PL, we can redefine the inference rules into FOL.
  - Lifting: only make those substitutions that are needed to allow particular inferences to proceed
  - Unification: identify the relevant substitutions
Unification

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

Substituting $x$ by John and $y$ by John works
$\alpha = \{x/John, y/John\}$

Unification

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

- 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.
  $MGU = \{ y/John, x/z\}$
Unification Examples

\[ p = P(x, f(y), B) \]
\[ q = P(z, f(w), B) \]
\[ \alpha = \{z|x, w|y\} \]

\[ p = P(x, f(y), B) \]
\[ q = Q(z, f(w), B) \]
\[ p \text{ and } q \text{ not unifiable (why?)} \]

\[ p = P(x, B) \]
\[ q = P(f(x), B) \]
\[ p \text{ and } q \text{ not unifiable (why?)} \]

\[ p = P(y, B) \]
\[ q = P(f(x), B) \]
\[ \alpha = \{y|f(x)\} \]

Unification examples

\[ p = P(g(x), B) \]
\[ q = P(f(x), B) \]
\[ p \text{ and } q \text{ are not unifiable (Why?)} \]

\[ p = P(x, A) \]
\[ q = P(y, B) \]
\[ p \text{ and } q \text{ are not unifiable (Why?)} \]

\[ p = P(x, y, z, f(w)) \]
\[ q = P(A, y, z, f(u)) \]
\[ \alpha = \{x|A, w|u\} \]
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(REST[x], REST[y], θ))
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

Applying Substitution

- Given \( t \) - a term
  \( \sigma \) - a substitution

  "\( t\sigma \)" is the term resulting from applying substitution \( \sigma \) to term \( t \).

- Small Examples:
  \[ X{X/a} = a \]
  \[ f(X){X/a} = f(a) \]

Examples

- Example: Using \( t = f( a, h(Y, b), X ) \)
  \[ f( a, h(Y, b), X ){X/b} = f( a, h(Y, b), b ) \]
  \[ f( a, h(Y, b), X ){X/b Y/f(Z)} = f( a, h(f(Z), b), b ) \]
  \[ f( a, h(Y, b), X ){X/Z Y/f(Z,a)} = f( a, h(f(Z,a),b), Z ) \]
  \[ f( a, h(Y, b), X ){W/Z} = f( a, h(Y, b), X ) \]

- \( \sigma \) need not include all variables in \( t \);
  \( \sigma \) can include variables not in \( t \).
Most General Unifier

- $\sigma$ is a mgu for $t_1$ and $t_2$ \iff
  - $\sigma$ unifies $t_1$ and $t_2$, \quad and
  - $\forall \mu$: unifier of $t_1$ and $t_2$,
    $\exists \ $ substitution, $\theta$, s.t. $\sigma \circ \theta = \mu$.
  $\text{(Ie, for all terms } t, \ t\mu = (t\sigma)\theta.)$

MGU example

- Example: $\sigma = \{x/y\}$ is mgu for $f(x)$ and $f(y)$.
  Consider unifier $\mu = \{x/a, y/a\}$.
  Use substitution $\theta = \{y/a\}$:

  $f(x)\mu = f(x)\{x/a, y/a\} = f(a)$

  $f(x) [\sigma \circ \theta] = (f(x) \sigma) \theta

  = (f(x) \{x/y\}) \theta

  = f(y) \{y/a\}

  = f(a)$

  Similarly, $f(y)\mu = f(a) = f(y) [\sigma \circ \theta]$

  ($\mu$ is NOT a mgu, as $\nexists \theta'$ s.t. $\mu \circ \theta' = \sigma \|$)
Notes on MGU

- If two terms are unifiable, then there exists a MGU
- There can be more than one MGU, but they differ only in variable names
- Not every unifier is a MGU
- A MGU uses constants only as necessary

FOL Modus Ponens Example

All Men are Mortal
Socrates is a Man
Socrates is mortal

\[ \forall x \text{Man}(x) \Rightarrow \text{Mortal}(x) \]
\[ \text{Man}(\text{Socrates}) \]
\[ \frac{}{\text{Mortal}(\text{Socrates})} \]

\[ \text{MGU} = \{ \text{Socrates} \mid x \} \]
Generalized Modus Ponens (GMP)

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

\[ \frac{p_1' \land p_2' \land \ldots \land p_n'}{q'_{\theta}} \]

where \((p_1 \land p_2 \land \ldots \land p_n)_{\theta} = p_1' \land p_2' \land \ldots \land p_n'\)

- \(p_1'\) 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.
Resolution Rule in FOL

- Example:
  - father(John, Kim),
  - $\forall x \forall y \neg\text{father}(x,y) \lor \text{parent}(x,y)$
  - parent(John, Kim)?

- Resolution with propositional logic:
  - Find complementary literals

- Resolution with FOL
  - Create complementary literals with substitution

Conversion to CNF

0: $\forall x \ [ (\forall y \ P(x,y)) \Rightarrow \neg\forall y \ Q(x,y) \Rightarrow R(x,y) ]$

1: **Eliminate implication, iff, ...**
$\forall x \ [(\neg (\forall y \ P(x,y)) \lor [\neg\forall y \neg Q(x,y) \lor R(x,y) ])]$

2: **Move $\neg$ inwards**
$\forall x \ [(\exists y \neg P(x,y)) \lor [\exists y Q(x,y) \land \neg R(x,y) ] ]$

3: **Standardize variables**
$\forall x \ [(\exists y \neg P(x,y)) \lor [\exists z Q(x,z) \land \neg R(x,z) ] ]$

4: **Move quantifiers left**
$\forall x \exists y \exists z \ [\neg P(x,y) \lor [Q(x,z) \land \neg R(x,z) ] ]$
Conversion to CNF

5: Skolemize (remove existentials); Drop $\forall$s
$\neg P(x, F_1(x)) \lor [Q(x, F_2(x)) \land \neg R(x, F_2(x))]$

6: Distribute $\land$ over $\lor$
$[\neg P(x, F_1(x)) \lor Q(x, F_2(x))]$
$\land [\neg P(x, F_1(x)) \lor \neg R(x, F_2(x))]$

7: Change to SET notation
$\begin{cases}
\neg P(x, F_1(x)) \lor Q(x, F_2(x)), \\
\neg P(x, F_1(x)) \lor \neg R(x, F_2(x))
\end{cases}$

8: Make variables unique
$\begin{cases}
\neg P(x_1, F_1(x_1)) \lor Q(x_1, F_2(x_1)), \\
\neg P(x_2, F_1(x_2)) \lor \neg R(x_2, F_2(x_2))
\end{cases}$

Theorem Proving in FOL

Example:
If a course is interesting, some students are happy.
if a course has a final, no student is happy.
Prove: If a course has a final, then it is not interesting.
Putting this in FOPL we get:
1. $\forall c \text{ Interesting}(c) \Rightarrow \exists s [\text{Student}(s, c) \land \text{Happy}(s)]$
2. $\forall s \forall c \text{ Final}(c) \land \text{Student}(s, c) \Rightarrow \neg \text{Happy}(s)$
Theorem to prove: $\forall c \text{ Final}(c) \Rightarrow \neg \text{Interesting}(c)$

Negation of theorem:
3. $\neg [\forall c \text{ Final}(c) \Rightarrow \neg \text{Interesting}(c)]$
Theorem Proving in FOL

a. \( \neg \text{Interesting}(c) \lor \text{Student}(skf(c), c) \)
b. \( \neg \text{Interesting}(x) \lor \text{happy}(skf(x)) \)
c. \( \neg \text{Final}(z) \lor \neg \text{Student}(s, z) \lor \neg \text{Happy}(s) \)
d. \( \text{Final}(skf) \)
e. \( \text{Interesting}(skf) \)

f. \( \text{Student}(skf(skf), skf) \)
   \[ \sigma = \{ skf | skf \} \]
c. \( \neg \text{Final}(z) \lor \neg \text{Student}(s, z) \lor \neg \text{Happy}(s) \)
   \[ \sigma = \{ skf(skf) | s, skf | z \} \]

i. \( \neg \text{Interesting}(skf) \)
e. \( \text{Interesting}(skf) \)

[Null clause]
Example #2

- Jack owns a dog
- Every dog owner is an animal lover
- No animal lover kills an animal
- Either Jack or Curiosity killed the cat (named Tuna)
- Did Curiosity kill the cat?

Properties of Resolution

Resolution by refutation is
- Sound
- Refutation Complete
  - If $\text{KB} \models \alpha$, refutation will prove it
  - Otherwise, in the general setting (infinite number of models) refutation procedure may not terminate
- Complexity
  - Exponential in the size of $\text{KB}$ for Propositional Logic (worst case)
Example #3

- The law says that it is a crime for an American to sell weapons to hostile nations
- Missiles are weapons
- The country Nono, an enemy of America, has some missiles
- All of Nono's its missiles were sold to it by Colonel West
- Colonel West is an American

- Prove that Col. West is a criminal

Example knowledge base

... 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
    ( 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 example

[Input example]

{ American(Own) Missle(1) Own(1) Own(Enemy, Own) }
Forward chaining example

Properties of forward chaining

• Sound and complete for first-order definite clauses
• $\textbf{Datalog} =$ first-order definite clauses + no functions (e.g. crime KB)
  – FC terminates for Datalog in finite number of iterations

• May not terminate in general DF clauses with functions 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 $k-1$
  – 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 $\text{Missile}(M_1)$

• Matching conjunctive premises against known facts is NP-hard. (Pattern matching)

• Forward chaining is widely used in deductive databases
Hard matching example

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

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

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

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 {θ}
qu′ ← 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(α₁, α₂), p) = SUBST(α₂, SUBST(α₁, p))
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!!)
- Widely used for **logic programming**

Logic programming

- Logic programming
  - Identify problem
  - Assemble information
  - Encode info in KB
  - Encode problem instances as facts
  - Ask queries
  - Find false facts.
- Procedural programming
  - Identify problem
  - Assemble information
  - Figure out solution
  - Program solution
  - Encode problem instance as data
  - Apply program to data
  - Debug procedural errors
Logic programming: Prolog

- **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).

  - Efficient unification and retrieval of matching clauses.
  - 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

Theorem Proving in Predicate Logic
Resolution by Refutation

If $KB \models \sigma$
then $\exists$ resolution proof of $\{\}$
from $KB \cup \{\neg \sigma\}$

- Add $\neg \sigma$ to KB
- Convert KB to CNF
- Apply Resolution Procedure
  - Derive $\{\}$: $\sigma$ is proved
  - Deadend:
    $\sigma$ is not a consequence of KB
Search Control in Theorem Proving

- Unit preference strategy
  
  \[ P(x) \]
  
  \[ \neg P(y) \lor R(y) \lor Q(y) \]
  
  \[ P(z) \lor \neg S(z) \]

- Which pair of clauses to choose?
  
  \[ P(x) \]
  
  \[ \neg P(y) \lor R(y) \lor Q(y) \]

- Why?
Search Control in Theorem Proving

- **Set of support** (SOS)
  - All clauses in negated theorem belong to SOS
  - Any clause derived from resolving a member of SOS with another clause belongs to SOS

- **Set of support strategy**
  - Each resolution step must choose a member of SOS as one of the two clauses to be resolved

- **Theorem**: SOS is refutation complete for FOL. That is, if there is a proof for a theorem, it can be found using SOS strategy.

**Set of Support Strategy: Example**

**Axioms**: \{I(A), D(A), \neg R(x) \lor L(x), \neg D(y) \lor \neg L(y)\}

**Negated Theorem**: \{\neg I(z) \lor R(z)\}

```

¬I(z) ∨ R(z)    I(A)
                 /               /
               /                 /
             R(A)             ¬R(x) ∨ L(x)
                      /               /
                    /                 /
                  L(A)             ¬D(y) ∨ ¬L(y)
                  /               /
                /                 /
              ¬D(A)             D(A)
```

Search Control for Theorem Proving

• Eliminate
  – clauses containing pure literals (literals whose complements do not appear in any other clause in the KB)
  – tautologies e.g., \( R(x) \lor \neg R(x) \)
  – any clause that is subsumed by another clause

A clause \( \phi \) subsumes a clause \( \psi \) iff

\[ \exists \text{ a substitution } \sigma \text{ such that } \phi \sigma \subseteq \psi \]

\[ P(x) \text{ subsumes } P(x) \lor R(y) \]
\[ P(x) \lor Q(y) \text{ subsumes } P(f(A)) \lor R(z) \lor Q(B) \]

Elimination of subsumed clauses

• **Theorem**: Unsatisfiability of a set \( S \) of clauses is unaffected by elimination of clauses in \( S \) that are subsumed by other clauses in \( S \)

• **Proof**: WLOG consider propositional KB

Let \( S = \{ c_1, \ldots, c_n, c, c' \} = q \)
\[ S' = \{ c_1, \ldots, c_n, c \} = S - \{ c' \} \]
\[ S'' = \{ c_1, \ldots, c_n \} = S - \{ c, c' \} = S' - \{ c' \} \]

Let \( c = P; \ c' = P \lor Q; \) So \( c \) subsumes \( c' \)
\[ M_S = M_{S'} \cap M_{c \lor c'} = M_{S'} \cap M_c \cap M_{c'} \]
\[ = M_{S''} \cap M_c = M_S \]
Green’s Trick for Answer Extraction

• We are often interested in instantiation that makes a theorem true (e.g., queries in deductive databases)

  KB:
  \[ \forall x \, \text{At}(\text{Bumstead}, x) \Rightarrow \text{At}(\text{Daisy}, x) \]
  \[ \text{At}(\text{Bumstead}, \text{Couch}) \]

  Query:
  \[ \exists x \, \text{At}(\text{Daisy}, x) \]

  Substitute in \[ \text{At}(\text{Daisy}, x) \]

  the same substitutions used to prove the query to answer the question Where is Daisy?