Deliberative Agents
Knowledge Representation II

Vasant Honavar
Artificial Intelligence Research Laboratory
Department of Computer Science
Bioinformatics and Computational Biology Program
Center for Computational Intelligence, Learning, & Discovery
Iowa State University
honavar@cs.iastate.edu
www.cs.iastate.edu/~honavar/
www.cild.iastate.edu/
www.bcb.iastate.edu/
www.igert.iastate.edu

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></th>
<th>Ontological Commitments</th>
<th>Epistemological Commitments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional Logic</td>
<td>Facts</td>
<td>True, False, Unknown</td>
</tr>
<tr>
<td>First Order Predicate Logic</td>
<td>Facts, Objects, Relations</td>
<td>True, False, Unknown</td>
</tr>
<tr>
<td>Probability Theory</td>
<td>Facts</td>
<td>Degree of belief $\in {0,1}$</td>
</tr>
</tbody>
</table>

Syntax of FOL: Basic elements

- Constants
  - Oksana, 2, Iowa-State-University
- Predicates
  - Brother, Father, Teacher, Red
- Functions
  - Successor (), Plus
- Variables $x, y, a, b,\ldots$
- Connectives $\neg, \Rightarrow, \land, \lor, \leftrightarrow$
- Equality $=$
- Quantifiers $\forall, \exists$
Atomic sentences

- Term
  - function \((\text{term}_1, \ldots, \text{term}_n)\),
  - e.g., house_of \((\text{John})\)
  - constant
    - e.g., John, 5
- or variable
  - e.g., \(x, y, z\)

- Predicates
  - e.g., \(\text{Brother}(\text{George}, \text{Jeb})\)

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. \(\text{Brother}(\text{George}, \text{Jeb}) \Rightarrow \text{Sibling}(\text{George}, \text{Jeb})\)
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 → objects
  - predicate symbols → relations
  - function symbols → 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: $A$, $B$, $Table$
- Relation Constant: $On$

Model

$On (A, B)$
$On (B, Table)$
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

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

\[
(At(Matt, ISU) ⇒ Smart(Matt)) \land \\
(At(Oksana, ISU) ⇒ Smart(Oksana)) \land \\
(At(Fido, ISU) ⇒ Smart(Fido)) \land \\
.....
\]

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 \(∀\):

\[
∀x \ At(x, ISU) \land Smart(x)
\]

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

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

Someone at ISU is smart:

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

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

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

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

Another common mistake to avoid

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

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

- The above assertion is true even if there is someone that is smart who is not at ISU!
- What we wanted to assert was instead that there is someone at ISU who is smart!
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:**

<table>
<thead>
<tr>
<th></th>
<th>∀x P</th>
<th>¬∀x P</th>
<th>¬P ∧ ¬Q</th>
<th>¬(P ∨ Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀x ¬P</td>
<td>¬∃x P</td>
<td>∃x ¬P</td>
<td>¬Q</td>
<td>¬P ∨ ¬Q</td>
</tr>
<tr>
<td>¬∀x P</td>
<td>∃x ¬P</td>
<td>¬(P ∧ Q)</td>
<td>¬(P ∨ ¬Q)</td>
<td></td>
</tr>
<tr>
<td>∀x P</td>
<td>∃x ¬P</td>
<td>P ∧ Q</td>
<td>¬(¬P ∨ ¬Q)</td>
<td></td>
</tr>
<tr>
<td>¬∀x ¬P</td>
<td>∃x P</td>
<td>¬(¬P ∧ ¬Q)</td>
<td>P ∨ Q</td>
<td></td>
</tr>
<tr>
<td>∀x ¬¬P</td>
<td>∃x P</td>
<td>¬¬P</td>
<td>P ∨ Q</td>
<td></td>
</tr>
<tr>
<td>¬∀x ¬¬P</td>
<td>∃x ¬¬P</td>
<td>¬¬¬P</td>
<td>P ∨ Q</td>
<td></td>
</tr>
</tbody>
</table>
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 \textit{Sibling} in terms of \textit{Parent}:
  \[
  \forall x, y \, \text{Sibling}(x, y) \Leftrightarrow \neg(x = y) \land \exists m, f \, \neg(m = f) \land \text{Parent}(m, x) \land \text{Parent}(f, x) \land \text{Parent}(m, y) \land \text{Parent}(f, y)
  \]

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}(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

- Identify the task (what will the KB be used for)
- Assemble the relevant knowledge (knowledge acquisition)
- Decide on a vocabulary of predicates, functions, and constants
  - Translate domain-level knowledge into logic-level names
- Encode general knowledge about the domain
  - define axioms
- Encode a description of the specific problem instance
- Pose queries to the inference procedure and get answers
- 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,C_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

- Adapt techniques from propositional logic
- Adapt techniques developed for propositional inference
  - How to eliminate universal quantifiers?
    - Instantiate variables
  - How to convert existential quantifiers?
    - Skolemization
Universal instantiation (UI)

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

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

- 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 \( g \)

\( \text{(Subst} (x/y) = \text{substitution of } y \text{ by } x) \)

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

\( House(John-Villa) \land Ownedby(John-Villa,John) \)

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 \( Y_1, \ldots, Y_m \) be the universally quantified variables that are quantified to the left of \( X \)'s “\( \exists X \)”.
Generate new function symbol, \( g_X \), of \( m \) variables. Replace each \( X \) with \( g_X(Y_1, \ldots, Y_m) \).
(Write \( g_X() \) as \( g_X \).)

\[
\begin{align*}
\forall Y \exists X \phi(X) \land \rho(Y) & \quad \Rightarrow \quad \forall Y \phi(\frac{g_X(Y)}{g_X}) \land \rho(Y) \\
\exists X \forall Y \phi(X) \land \rho(Y) & \quad \Rightarrow \quad \forall Y \phi(\frac{g_X}{g_X}) \land \rho(Y)
\end{align*}
\]
Skolemization Theorem

If $$T_1 = \left\{ \begin{array}{c} \alpha_1 \\ \alpha_2 \\ \vdots \\ \exists X \forall Y \varphi(X, Y) \\ \vdots \end{array} \right\}$$ is consistent

then $$s(T_1) = \left\{ \begin{array}{c} \alpha_1 \\ \alpha_2 \\ \vdots \\ \forall Y \varphi(c_1, Y) \\ \vdots \end{array} \right\}$$ is consistent.

... if $$s(T)$$ is inconsistent, then $$T$$ is inconsistent ...

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

• After universal instantiation we get a variable-free, quantifier-free KB — a propositionalized KB
  \[ \text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John}) \]
  \[ \text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard}) \]
  \[ \text{King}(\text{John}) \]
  \[ \text{Greedy}(\text{John}) \]
  \[ \text{Brother}(\text{Richard, John}) \]

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., \[ \text{Father}(\text{Father}(\text{Father}(\text{John}))) \]
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

Reduction of FOL inference to PL inference

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

• 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 \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(\text{y}) \).

Substituting \( x \) by \( \text{John} \) and \( y \) by \( \text{John} \) works
\[ \alpha = \{ x/\text{John}, y/\text{John} \} \]

Unification

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

- The first unifier is more general than the second.

- There is a single most general unifier (MGU) that is unique up to renaming of variables.
\[ \text{MGU} = \{ y/\text{John}, x/z \} \]
Unification Examples

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

\[ p = P(x, f(y), B) \]
\[ q = Q(z, f(w), B) \]

\[ p = P(x, B) \]
\[ q = P(f(x), B) \]

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

Unification examples

\[ p = P(g(x), B) \]
\[ q = P(f(x), B) \]

\[ p = P(x, A) \]
\[ q = P(y, B) \]

\[ p = P(x, y, z, f(w)) \]
\[ q = P(A, y, z, f(u)) \]
\[ \alpha = \{x \mapsto A, w \mapsto u\} \]
The unification algorithm

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

if $\theta = \text{failure}$ then return failure
else if $x = y$ then return $\theta$
else if VARIABLE?($x$) then return UNIFY-VAR($x$, $y$, $\theta$)
else if VARIABLE?($y$) then return UNIFY-VAR($y$, $x$, $\theta$)
else if COMPOUND?($x$) and COMPOUND?($y$) then
  return UNIFY(ARGs[$x$], ARGs[$y$], UNIFY(Op[$x$], Op[$y$], $\theta$))
else if LIST?($x$) and LIST?($y$) then
  return UNIFY(REST[$x$], REST[$y$], UNIFY(First[$x$], First[$y$], $\theta$))
else return failure

The unification algorithm

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

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

- Given \( t - a \text{ term} \)
  \( \sigma - a \text{ substitution} \)
  
  "to\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 \( t=f( a, h(Y,b) , X ) \)
  
  \[ f( a, h(Y,b) , X )\{X/b\} = f( a, h(Y,b) , b ) \]
  \[ f( a, h(Y,b) , X )\{X/Y/f(Z)\} = f( a, h(f(Z),b) , b ) \]
  \[ f( a, h(Y,b) , X )\{X/Z Y/f(Z,a)\} = f( a, h(f(Z,a),b) , Z ) \]
  \[ f( a, h(Y,b) , X )\{W/Z\} = f( a, h(Y,b) , X ) \]

- \( \sigma \) 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 $t_2$, and
  - $\forall \mu$: unifier of $t_1$ and $t_2$,
    $\exists$ substitution, $\theta$, s.t. $\sigma \circ \theta = \mu$.
  (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 $\exists \theta'$ 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

\[
\begin{align*}
\text{All Men are Mortal} \\
\text{Socrates is a Man} \\
\text{Socrates is mortal}
\end{align*}
\]

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

\[
\begin{align*}
\text{Man}(\text{Socrates}) \\
\text{Mortal}(\text{Socrates})
\end{align*}
\]

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

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

\[ p_1' \land p_2' \land ... \land p_n' \]

\[ q^\theta \]

where \( (p_1 \land p_2 \land ... \land p_n)^\theta = p_1' \land p_2' \land ... \land p_n' \)

\( p_1' \) 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 \( p_1', ..., p_n', (p_1 \land ... \land p_n \Rightarrow q) \vdash q^\theta \)

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. \( (p_1 \land ... \land p_n \Rightarrow q) \vdash (p_1 \land ... \land p_n \Rightarrow q)^\theta \)

   \[ = (p_1^\theta \land ... \land p_n^\theta \Rightarrow q^\theta) \]

2. From 1 and 2, \( q^\theta \) follows by ordinary Modus Ponens.
Generalized resolution principle

\[
p_1 \lor p_2 \lor \cdots \lor p_n
\]

\[
p_1' \lor p_2' \lor p_m'
\]

\[
\left( p_1 \lor \cdots \lor p_{i-1} \lor p_{i+1} \lor \cdots \lor p_n \lor p_1' \lor \cdots \lor p_{j-1} \lor p_{j+1} \lor \cdots \lor p_m' \right) \theta
\]

where \( (p_i)\theta = \neg p_j' \)

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: \( \forall x \left[ (\forall y \ P(x, y)) \Rightarrow \neg \forall y \ Q(x, y) \Rightarrow R(x, y) \right] \)

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

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

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

4: Move quantifiers left
   \( \forall x \exists y \exists z \left[ \neg P(x, y) \lor [Q(x, z) \land \neg R(x, z)] \right] \)

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
   \[ \{ \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)) \} \}

8: Make variables unique
   \[ \{ \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)) \} \}

Vasant Honavar, 2009.
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: \( \forall c \text{ Final}(c) \Rightarrow \neg \text{Interesting}(c) \)

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. \( \text{Final}(sk\phi) \)
e. \( \text{Interesting}(sk\phi) \)
Theorem Proving in FOL

Clause normal form

- \( \neg \text{Interesting}(c) \lor \text{Student}(skf(c), c) \)
- \( \neg \text{Interesting}(x) \lor \text{happy}(skf(x)) \)
- \( \neg \text{Final}(z) \lor \neg \text{Student}(s, z) \lor \neg \text{Happy}(s) \)
- \text{Final}(skf)
- \text{Interesting}(skf)
- \( \neg \text{Interesting}(c) \lor \text{Student}(skf(c), c) \)
- \( \text{Interesting}(skf) \)

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 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., \( \exists x \text{Owns}(\text{Nono},x) \land \text{Missile}(x) \):
\[ \text{Owns}(\text{Nono},M,1) \text{ and Missile}(M,1) \]

... 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

```plaintext
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 \(θ\) such that \((p_1 \land \ldots \land p_n)θ = (p'_1 \land \ldots \land p'_n)θ\) for some \(p'_1,\ldots,p'_n\) in KB
            \(q' \leftarrow \text{SUBST}(θ, q)\)
            if \(q'\) is not a renaming of a sentence already in KB or new then do
                add \(q'\) to new
                \(φ \leftarrow \text{UNIFY}(q', α)\)
                if \(φ\) is not fail then return \(φ\)
        add new to KB
    return false
```

Vasant Honavar, 2009.
Forward chaining example

American(West) Missile(M1) Own(M1, None) Enemy(No, America)

Weapon(M1) SetIn(No, West, M1, None) Hostile(None)

American(No) Missile(M1) Own(No, M1) Enemy(No, America)

Vasant Honavar, 2009.
Forward chaining example

Properties of forward chaining

• Sound and complete for first-order definite clauses
• *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 $\text{Missile}(M_i)$

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

- Forward chaining is widely used in deductive databases

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
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}
  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, |goals| REST(goals), COMPOSE(θ, θ’)) ∪ ans
  return ans

SUBST(COMPOSE(α₁, α₂), p) = SUBST(α₂, SUBST(α₁, p))

Backward chaining example
Backward chaining example

![Diagram of backward chaining example]

Vasant Honavar, 2009.
Backward chaining example

Vasant Honavar, 2009.
Backward chaining example
Backward chaining example

\begin{center}
\begin{tikzpicture}
  \node {\texttt{CrimsonWest}}
    child {\node {\texttt{AmericanWest}}
      child {\node {\texttt{Bosnia}}
        child {\node {\texttt{Mazowiezy}}
          child {\node {\texttt{ZyMi}}
            child {\node {\texttt{Emma}(Nona, America)}}
          }
        }
        child {\node {\texttt{Mazowiezy}}
          child {\node {\texttt{ZyMi}}
            child {\node {\texttt{Emma}(Nona, America)}}
          }
        }
      }
      child {\node {\texttt{SellWestM1, z/None}}
        child {\node {\texttt{ZyMi}}
          child {\node {\texttt{Emma}(Nona, America)}}
        }
      }
    }
    child {\node {\texttt{MissileNone}}}
  child {\node {\texttt{x/West, y/RU, z/None}}}
\end{tikzpicture}
\end{center}

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)\}

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
    \[
    \text{A clause } \phi \text{ subsumes a clause } \psi \iff \exists \text{ a substitution } \sigma \text{ such that } \phi\sigma \subseteq \psi
    \]
    \[
    P(x) \text{subsumes } P(x) \lor R(y) \\
    P(x) \lor Q(y) \text{subsumes } P(f(A)) \lor R(z) \lor Q(B)
    \]
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 $c'$
  
  $M_S = M_{S'} \cap M_{c \land c'}$
  
  $= M_{S'} \cap M_c \land M_{c'}$
  
  $= M_{S'} \cap M_c = M_S$

---

Green’s Trick for Answer Extraction

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

  **KB**:

  $\forall x \, At(\text{Bumstead}, x) \Rightarrow At(\text{Daisy}, x)$

  $At(\text{Bumstead}, \text{Couch})$

  **Query**:

  $\exists x \, At(\text{Daisy}, x)$

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

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