COGS 515—Artificial Intelligence for CogSci
First Order Logic

Umut Özge

Department of Cognitive Science
Informatics Institute
Middle East Technical University
Motivation for First Order Logic (FOL)

- The primary motivation for moving from PL to FOL is *expressive power*.
- In FOL you can write more general statements than you can in PL.
- We had to write statements like the following for every cell in the Wumpus World:
  \[ B_{1.1} \leftrightarrow (P_{1,2} \lor P_{2.1}) \]
- In FOL you can write general statements like:
  
  Squares adjacent to pits are breezy.
First Order Logic

Syntax:

- Symbols (or vocabulary) are divided into *constants* and *variables*.
- *Individual constants* stand for the names of objects; i.e. *john, alice, a, b* . . .
- *Predicate constants* stand for the names of relations; i.e. *red, on, between* . . .
- *Functional constants* stand for the names of functions; i.e. *left_leg, president, age* . . .
- We also have *individual variables* like *x, y, z* . . .
First Order Logic (cont.)

Semantics:

- Models of FOL are no longer restricted to truth values but inhabit objects, relations and functions.
- Every model is based on a (possibly infinite) non-empty set of objects, usually called the *domain* of the model.
- A model for FOL basically does two things (at the same time),
  1. It fixes the meaning of the constants;
  2. It fixes the state of affairs (i.e. what the case is).
The ontological commitment of a logical system concerns its commitment to what there is; in other words what sorts of objects are there in its models.

FOL is richer in its ontological commitment than PL.

The epistemological commitment of a logical system concerns the possible epistemic states an agent can have with respect to a given sentence.

<table>
<thead>
<tr>
<th>Language</th>
<th>Ontological Commitment</th>
<th>Epistemological Commitment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional logic</td>
<td>facts</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>First-order logic</td>
<td>facts, objects, relations</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Temporal logic</td>
<td>facts, objects, relations, times</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Probability theory</td>
<td>facts</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Fuzzy logic</td>
<td>facts + degree of truth</td>
<td>known interval value</td>
</tr>
</tbody>
</table>
Examples

Example 1
Consider a KB comprising of $P(a)$ and $P(b)$. Does this KB entail $\forall x. P(x)$?

Example 2
How do you say in FOL *John has at least two brothers*?

Example 3
How do you say in FOL *John has two brothers, Jill and Jack*?

Example 4
Is the sentence $\exists x \exists y. x = y$ valid $\Sigma$?

Example 5
Write down a logical sentence such that every world in which it is true contains exactly one object.
As we saw in Example 3 above, adding certain facts to knowledge bases can be cumbersome in standard semantics of FOL.

Database semantics makes the following assumptions:

- unique-names assumption;
- closed-world assumption;
- domain closure.

These assumptions has the nice consequence of providing finite number of models, but they have a drawback as well . . . .
Wumpus World—Again

- Percepts:
  \[ \text{Percept}([\text{Stench, Breeze, Glitter, None, None}], 5) \]

- Actions:
  \[ \text{Turn(Right), Turn(Left), Forward, Shoot, Grab, Release, Climb} \]

- Query:
  \[ \exists a. \text{BestAction}(a, 5) \]

- Perception Axioms:
  \[ \forall t, s, g, m, c. \text{Percept}([s, Breeze, g, m, c], t) \rightarrow \text{Breeze}(t) \]
  \[ \forall t, s, g, m, c. \text{Percept}([s, b, Glitter, m, c], t) \rightarrow \text{Glitter}(t) \]

- Reflex Behavior:
  \[ \forall t. \text{Glitter}(t) \rightarrow \text{BestAction}(\text{Grab}, t) \]
Wumpus World—Environment

- Squares: If we have the squares as atomic terms like $S_{1.2}$, then we need to state adjacency information for each square separately. How can we obtain a more concise encoding?
Wumpus World—Environment

- Squares: If we have the squares as atomic terms like $S_{1,2}$, then we need to state adjacency information for each square separately. How can we obtain a more concise encoding?

  $$\forall w, x, y, z.\text{Adjacent}([x, y], [w, z]) \leftrightarrow (x = w \land (y = z - 1 \lor y = z + 1)) \lor (y = z \land (x = w - 1 \lor x = w + 1))$$

- What about pits and the Wumpus?
Wumpus World—Environment

- Squares: If we have the squares as atomic terms like $S_{1,2}$, then we need to state adjacency information for each square separately. How can we obtain a more concise encoding?

$$\forall w, x, y, z.\text{Adjacent}([x, y], [w, z]) \leftrightarrow (x = w \land (y = z - 1 \lor y = z + 1)) \lor (y = z \land (x = w - 1 \lor x = w + 1))$$

- What about pits and the Wumpus?

- To capture the dynamics of agent’s location we use $At(Agent, s, t)$. How can we express that an agent cannot be at two distinct locations at a time?
Wumpus World—Environment

- **Squares:** If we have the squares as atomic terms like $S_{1.2}$, then we need to state adjacency information for each square separately. How can we obtain a more concise encoding?

  \[
  \forall w, x, y, z. \text{Adjacent}([x, y], [w, z]) \leftrightarrow (x = w \land (y = z - 1 \lor y = z + 1)) \lor \\
  (y = z \land (x = w - 1 \lor x = w + 1))
  \]

- **What about pits and the Wumpus?**
- **To capture the dynamics of agent’s location we use $At(Agent, s, t)$:** How can we express that an agent cannot be at two distinct locations at a time?

  \[
  \forall x, s_1, s_2, t. At(x, s_1, t) \land At(x, s_2, t) \rightarrow s_1 = s_2
  \]

- **Linking percepts and locations:**

  \[
  \forall s, t. At(Agent, s, t) \land Breeze(t) \rightarrow Breezy(t)
  \]

- **Linking breeziness and pits:**

  \[
  \forall s. Breezy(s) \leftrightarrow \exists r. \text{Adjacent}(r, s) \land Pit(r)
  \]
Inference in FOL

- Inference in FOL is like inference in PL except we need to take care of variables and quantification in the former.
- One way to deal with variables is to **propositionalize** the knowledge base.
- **Universal Instantiation (UI):**
  \[
  \forall v. \alpha \\
  \text{SUBST}([v/g], \alpha)
  \]  
  (1)
- **Existential Instantiation (EI):**
  \[
  \exists v. \alpha \\
  \text{SUBST}([v/k], \alpha)
  \]  
  (2)
  where \(k\) is a **Skolem constant**.
- The KB that is formed after Skolemization is inferentially equivalent to the original; that is, it is satisfiable only when the original KB is satisfiable.
Propositionalization

- Consider the FOL knowledge base:

\[ \forall x. \text{Rich}(x) \land \text{Healthy}(x) \rightarrow \text{Happy}(x) \]

\text{Rich}(John) \\
\text{Healthy}(John) \\
\text{Loves}(John, Mary)

- The universal formula is grounded to:

\[ \text{Rich}(John) \land \text{Healthy}(John) \rightarrow \text{Happy}(John) \]

\[ \text{Rich}(Mary) \land \text{Healthy}(Mary) \rightarrow \text{Happy}(Mary) \]

- This approach runs into trouble with functional terms, because they render infinitely many substitutions.

- Herbrand’s theorem tells that if a FOL knowledge base entails a formula, there exists a finite subset of the converted FOL that proves the entailment.

- Yet, the question of entailment for FOL is \textit{semidecidable}. 
Propositionalization is inefficient due to involvement of irrelevant formulas. E.g. it is pointless to generate sentences about Mary to prove that John is happy in the previous example.

Generalized Modus Ponens: For atomic sentences $p_i, p'_i$ and $q$, when there is a substitution $\theta$ s.t. $\text{SUBST}(\theta, p_i) = \text{SUBST}(\theta, p'_i,)$ for all $i$:

$$p'_1, p'_2, \ldots, p'_n, \ (p_1 \land p_2 \land \ldots \land p_n \rightarrow q) \quad \text{SUBST}(\theta, q)$$

(3)
Unification

- Unification is a mechanism that gives the substitution that makes two expressions equivalent.

\[ \text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q) \]  \hspace{1cm} (4)

- Suppose we are given the query \( \text{Knows}(John, x) \), whom does John know?

\[
\begin{align*}
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(John, Jane)) &= \\
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, Bill)) &= \\
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, \text{Mother}(y))) &= \\
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(x, Elizabeth)) &= 
\end{align*}
\]
Unification

- Unification is a mechanism that gives the substitution that makes two expressions equivalent.

\[
\text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)
\]  

(4)

- Suppose we are given the query \text{Knows}(John, x), whom does John know?

\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(John, Jane)) = [x/\text{Jane}]
\]
\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, \text{Bill})) =
\]
\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, \text{Mother}(y))) =
\]
\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(x, \text{Elizabeth})) =
\]
Unification

Unification is a mechanism that gives the substitution that makes two expressions equivalent.

\[ \text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q) \] (4)

Suppose we are given the query \textit{Knows}(John, x), whom does John know?

\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(John, Jane)) = [x/\text{Jane}] \]
\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, Bill)) = [x/\text{Bill}, y/\text{John}] \]
\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, \text{Mother}(y))) = \]
\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(x, \text{Elizabeth})) = \]
Unification

- Unification is a mechanism that gives the substitution that makes two expressions equivalent.

\[
\text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q) \tag{4}
\]

- Suppose we are given the query \(\text{Knows}(John, x)\), whom does John know?

\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(John, Jane)) = [x/\text{Jane}]
\]
\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, \text{Bill})) = [x/\text{Bill}, y/\text{John}]
\]
\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, \text{Mother}(y))) = [y/\text{John}, x/\text{Mother}(\text{John})]
\]
\[
\text{UNIFY}(\text{Knows}(John, x), \text{Knows}(x, \text{Elizabeth})) =
\]

Unification

Unification is a mechanism that gives the substitution that makes two expressions equivalent.

\[ \text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q) \] (4)

Suppose we are given the query \( \text{Knows}(John, x) \), whom does John know?

\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(John, Jane)) = [x/\text{Jane}] \]
\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, Bill)) = [x/\text{Bill}, y/\text{John}] \]
\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(y, \text{Mother}(y))) = [y/\text{John}, x/\text{Mother}(y)] \]
\[ \text{UNIFY}(\text{Knows}(John, x), \text{Knows}(x, Elizabeth)) = [x/\text{Jane}] \]
A clause is a disjunction of positive and negative literals. E.g.:

\[ P \lor Q \lor \neg R \lor \neg S \]  

A definite clause is a clause that has exactly one positive literal.

\[ P \lor Q \lor R \lor \neg S \]  

The interest in definite clauses is due to the form they take when turned into implications.
Clausal Form and Definite Clauses

- A **clause** is a disjunction of positive and negative literals. E.g.:
  \[ P \lor Q \lor \neg R \lor \neg S \]  
  (5)

- A **definite** clause is a clause that has exactly one positive literal.
  \[ P \lor Q \lor R \lor \neg S \]  
  (6)

- The interest in definite clauses is due to the form they take when turned into implications.
  \[ P \lor Q \lor R \lor \neg S \equiv P \land Q \land R \rightarrow S \]  
  (7)
Databases comprised entirely of definite clauses afford two useful inference strategies.

- Forward-chaining: Data-driven inference, production systems like ACT-R.
- Backward-chaining: Goal-driven inference.
Definite Clause in FOL—An Example

- By convention universal quantification is left implicit in FOL definite clauses.
- “The law says that it is a crime for a doctor to prescribe drugs to healthy people. Jane is a healthy person who has some marijuana, and all that she has is prescribed to her by John, who is a doctor.”
- “it is a crime for a doctor to prescribe drugs to healthy people”
Definite Clause in FOL—An Example

By convention universal quantification is left implicit in FOL definite clauses.

“The law says that it is a crime for a doctor to prescribe drugs to healthy people. Jane is a healthy person who has some marijuana, and all that she has is prescribed to her by John, who is a doctor.”

“it is a crime for a doctor to prescribe drugs to healthy people”

\[ \text{Doctor}(x) \land \text{Drug}(y) \land \text{Presc}(x,y,z) \land \text{Healthy}(z) \rightarrow \text{Criminal}(x) \]

“Jane. . . has some marijuana”
Definite Clause in FOL—An Example

- By convention universal quantification is left implicit in FOL definite clauses.
- “The law says that it is a crime for a doctor to prescribe drugs to healthy people. Jane is a healthy person who has some marijuana, and all that she has is prescribed to her by John, who is a doctor.”
- “it is a crime for a doctor to prescribe drugs to healthy people”

\[ \text{Doctor}(x) \land \text{Drug}(y) \land \text{Presc}(x,y,z) \land \text{Healthy}(z) \rightarrow \text{Criminal}(x) \]

- “Jane... has some marijuana”

\[ \text{Owns}(Jane, M_{47}) \]
\[ \text{Marijuana}(M_{47}) \]

- “all that she has is prescribed to her by John”
Definite Clause in FOL—An Example

- By convention universal quantification is left implicit in FOL definite clauses.
- “The law says that it is a crime for a doctor to prescribe drugs to healthy people. Jane is a healthy person who has some marijuana, and all that she has is prescribed to her by John, who is a doctor.”
- “it is a crime for a doctor to prescribe drugs to healthy people”

\[
\text{Doctor}(x) \land \text{Drug}(y) \land \text{Presc}(x,y,z) \land \text{Healthy}(z) \rightarrow \text{Criminal}(x)
\]

- “Jane... has some marijuana”

\[
\text{Owns}(\text{Jane}, M_{47})
\]
\[
\text{Marijuana}(M_{47})
\]

- “all that she has is prescribed to her by John”

\[
\text{Marijuana}(x) \land \text{Owns}(\text{Jane},x) \rightarrow \text{Presc}(\text{John},x,\text{Jane})
\]
The rest:

\[ \text{Marijuana}(x) \rightarrow \text{Drug}(x) \]
\[ \text{Healthy}(\text{Jane}) \]
\[ \text{Doctor}(\text{John}) \]

In a forward-chaining regime, we can solve a query like \( \text{Criminal}(\text{John})? \) by starting from ground facts and applying modus ponens in each iteration until we arrive at the goal or there are no iterations left (fixed point).

What is the maximum number of iterations for a KB with \( p \) predicates with the average arity of \( k \), and \( n \) constant symbols?
Backward Chaining and Prolog

criminal(X) :- doctor(X), drug(Y), presc(X,Y,Z), healthy(Z).

owns(jane, m47).

marijuana(m47).

presc(john, X, jane) :- marijuana(X), owns(jane, X).

drug(X) :- marijuana(X).

healthy(jane).

doctor(john).
Backward Chaining and Prolog

criminal(X) :- doctor(X), drug(Y), presc(X,Y,Z), healthy(Z).

owns(jane,m47).

marijuana(m47).

presc(john,X,jane) :- marijuana(X), owns(jane,X).

drug(X) :- cocaine(X).

drug(X) :- marijuana(X).

healthy(jane).

doctor(john).
Downsides of Prolog

- Prolog is incomplete.
- Consider the following program/KB:

\[
\begin{align*}
\text{path}(X,Z) & :\text{-- } \text{link}(X,Z). \\
\text{path}(X,Z) & :\text{-- } \text{path}(X,Y), \text{link}(Y,Z).
\end{align*}
\]
Now consider the same KB with the reverse clause order:

```prolog
path(X,Z) :- path(X,Y), link(Y,Z).
path(X,Z) :- link(X,Z).
```

This program runs into an infinite loop.
Another drawback of Prolog is the issue of redundant paths. Redundancy can be avoided by memoization.
A formula in **conjunctive normal form** (CNF) is a conjunction of clauses. For instance the formula,

\[ P \land Q \rightarrow R \land S \]  

(8)

becomes

\[ (\neg P \lor \neg Q \lor R) \land (\neg P \lor \neg Q \lor S) \]  

(9)

in CNF.