First-Order Predicate Logic

Unit-IV

Introduction to First-Order Logic (FOL)

- The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.
    - "Some humans are bad", or
    - "Sachin likes cricket.“
    - “All the humans are intelligent”
- To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic(FOL)/ first-order predicate logic(FOPL).
- First-order Predicate logic (FOPL) models the world in terms of
    - Objects, which are things with individual identities
    - Properties of objects that distinguish them from other objects
    - Relations that hold among sets of objects
    - Functions, which are a subset of relations where there is only one “value” for any given “input”

Introduction to First-Order Logic (FOL)

- The key components of First-order Predicate logic (FOPL) are
  1) ***Constants***: Alice, NewYork, 2, dog33. Name a specific object.
  2) ***Variables:*** Variables stand for unspecified objects in the domain. e.g. X, Y, x, y.
  3) ***Predicates:*** Predicates are functions that return true or false, representing properties of objects or relationships between them. For example, Likes(Alice, Bob) indicates that Alice likes Bob, and GreaterThan(x, 2) means that x is greater than 2.
  4) ***Functions:*** Mapping from objects to objects.
  5) ***Terms:*** Refer to objects
  6) ***Atomic Sentences:*** in(dad-of(X), food6) Can be true or false, Correspond to propositional symbols P, Q.
  7) ***Logical Connectives:*** Logical connectives include conjunction (∧), disjunction (∨), implication (→), biconditional (↔), and negation (¬). These connectives are used to form complex logical statements.

Introduction to First-Order Logic (FOL)

Limitations of Propositional Logic -- Necessity of First-Order Logic (FOL):

(FOL is also known as First-Order Predicate Calculus or Predicate Logic)

Consider the sentences:

	In Propositional Logic	
(i) Socrates is a man.	SMAN	-- We can't draw any similarities here.
Plato is a man.	PMAN	
(ii) All men are mortal.	MORTALMAN	

Fails to capture the relationship between any individual being a man and that individual being a mortal.

Better way: for (i), MAN(SOCRATES)

MAN(PLATO)

for(ii), $\forall x$: MAN(x) $\rightarrow$ MORTAL(x)

- The above notations are very similar to First-Order Logic representation.
- We can draw similarities and we can express the universal scope.

Syntax and Semantics of First-Order Logic

(1) Models for First-Order Logic:

- Models for first-order logic have objects.
- Domain of a model -- is the set of objects it contains (i.e., also known as domain elements)
- The domain is required to be nonempty—every possible world must contain at least one object.
- Mathematically speaking, it doesn’t matter what these objects are—all that matters is how many there are in each model.
- Figure 1. Shows a model with five objects: Richard the Lionheart, King of England from 1189 to 1199; his younger brother, the evil King John, who ruled from 1199 to 1215; the left legs of Richard and John; and a crown.

Syntax and Semantics of First-Order Logic

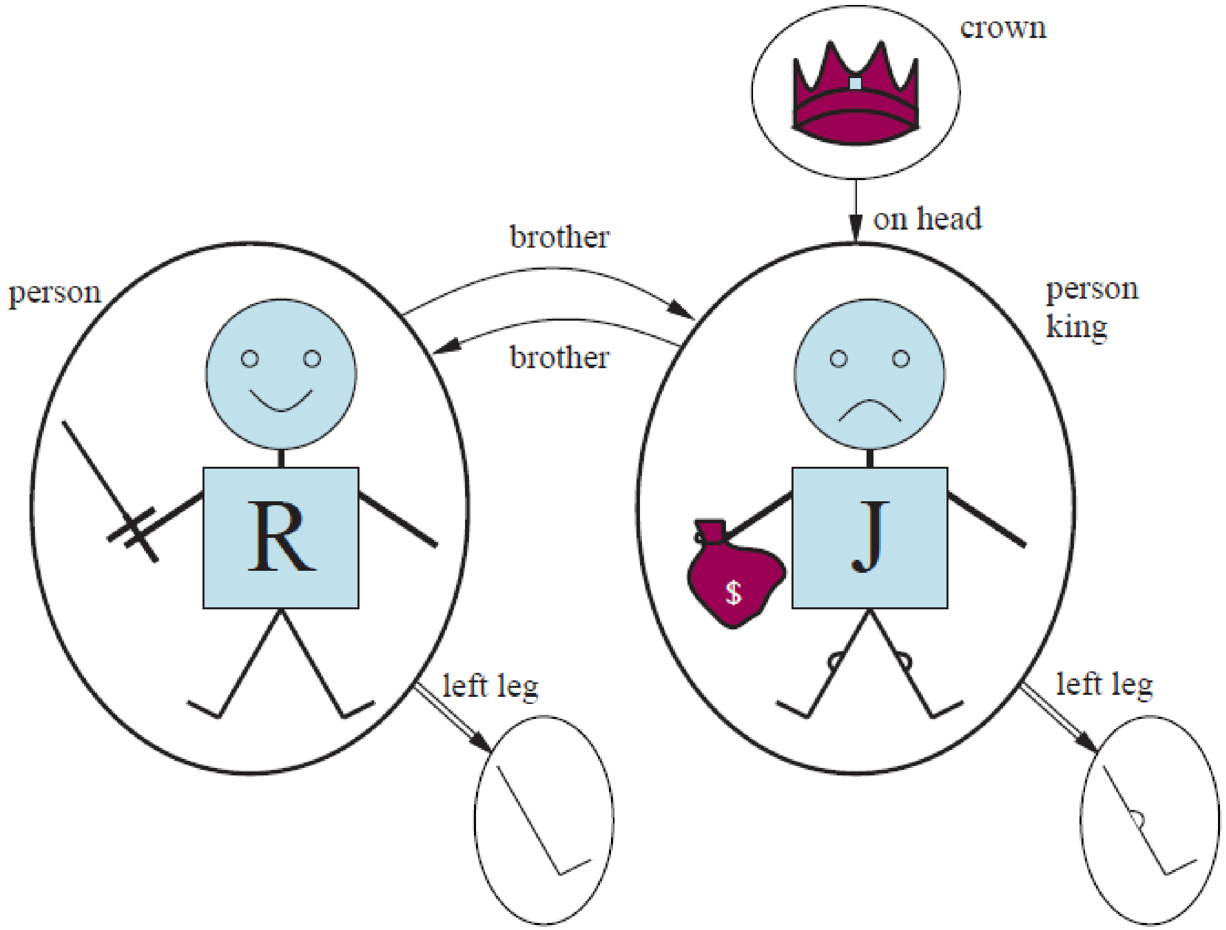

Figure 1. A model containing five objects, two binary relations (brother and on-head), three unary relations (person, king, and crown), and one unary function (left-leg).

Continued...

- Figure shows a model with 5 objects:

  - Richard (King)
  - John (brother of Richard, King)
  - Left leg of Richard
  - Left leg of John
  - Crown

Relation: The set of tuples of objects that are related.

Richard and John are brothers. The brotherhood relation as a tuple is:

{ <Richard, John>, <John, Richard> }

'on head' relation: <crown, John>

Binary relations: 'brother' and 'on head'

Unary relations (or Properties): 'person' (true for Richard and John)

'King' (true for John only)

Continued...

Other relationships (functions):

A given object must be related to exactly one object.

Ex: Each person has one left leg.

<Richard> -- Richard's left leg

<John> -- John's left leg

(2) Symbols and Interpretations:

Symbols:	constant symbols -- stand for objects	(Richard, John)
	predicate symbols -- stand for relations	(Brother, OnHead, Person, King, Crown)
	function symbols -- stand for functions	(LeftLeg)

Semantics: It must relate sentences to models in order to determine truth.

Interpretation:

It specifies exactly which objects, relations and functions are referred to by the constant, predicate and function symbols.

Continued...

<u>Ex:</u> One possible interpretation

Richard -- refers to Richard (the King who is not alive now)

John -- refers to John (the evil King)

Brother -- refers to brotherhood relation

LeftLeg -- leftleg function

<u>The Syntax of First-Order Logic: (the Backus-Naur Form)</u>

Figure 2 The syntax of first-order logic with equality, specified in Backus–Naur form. Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & Predicate \mid Predicate(Term, \ldots) \mid Term = Term \\
ComplexSentence & \rightarrow & (\ Sentence\ ) \\
 & \mid & \neg\ Sentence \\
 & \mid & Sentence \land Sentence \\
 & \mid & Sentence \lor Sentence \\
 & \mid & Sentence \;\Rightarrow\; Sentence \\
 & \mid & Sentence \;\Leftrightarrow\; Sentence \\
 & \mid & Quantifier\ Variable, \ldots\ Sentence \\
 & & \\
Term & \rightarrow & Function(Term, \ldots) \\
 & \mid & Constant \\
 & \mid & Variable \\
 & & \\
Quantifier & \rightarrow & \forall \mid \exists \\
Constant & \rightarrow & A \mid X_1 \mid John \mid \cdots \\
Variable & \rightarrow & a \mid x \mid s \mid \cdots \\
Predicate & \rightarrow & True \mid False \mid After \mid Loves \mid Raining \mid \cdots \\
Function & \rightarrow & Mother \mid LeftLeg \mid \cdots \\
\textsc{Operator Precedence} & : & \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Continued...

(3) Terms:

- A term is a logical expression that refers to an object.
- Constant symbols are terms, but every object cannot have a distinct symbol.

  Ex: King John's left leg.

  Instead of giving a name to 'left leg', we write this as LeftLeg(John) (as a complex term)

  (LeftLeg is not like a function (subroutine) in programming languages)
- $f(t_1,t_2,......,t_n)$ is a term

  (f is a function; $t_1,t_2,......,t_n$ refer to objects in a domain)

(4) Atomic Sentences:

- Formed using terms (objects) and predicate symbols.

  Ex: Richard is the brother of John. Brother(Richard, John)

  Richard's father is married to John's mother. Married(Father(Richard), Mother(John))

Continued...

(5) Complex Sentences:

Atomic sentences are combined using logical connectives to form complex sentences.

Ex: Brother(Richard, John) ∧ Brother(John, Richard)

King(Richard) ∨ King(John)

¬King(Richard) → King(John)

(6) Quantifiers:

(i) Universal Quantifier (∀)

Ex: All kings are persons.

∀x King(x) → Person(x) (x is a variable or term)

- A variable serves as an argument to a function.

  Ex: LeftLeg(x)

- Ground term: A term with no variables

  Ex: LeftLeg()

Continued...

(ii) Existential Quantifier (∃)

Ex: King John has a crown on his head. $\exists x \;\; Crown(x) \wedge OnHead(x, John)$

(iii) Nested Quantifiers

Multiple quantifiers are used to express more complex sentences.

Ex: Brothers are siblings.

$\forall x \, \forall y \;\; Brother(x, y) \rightarrow Sibling(x, y)$

Siblinghood is a symmetric relationship.

$\forall x, y \;\; Sibling(x, y) \Leftrightarrow Sibling(y, x)$

Everybody loves somebody.

$\forall x \, \exists y \;\; Loves(x, y)$

There is someone who is loved by everyone.

$\exists y \, \forall x \;\; Loves(x, y)$ (* Order of quantification is important)

Continued...

(iv) Connections between ∀ and ∃

Ex: Everyone likes ice cream.

$\forall x$ Likes(x, IceCream) is equivalent to $\neg\exists x \neg$Likes(x, IceCream).

- The de Morgan rules for quantified and unquantified sentences are as follows:

$$
\begin{array}{llll}
\neg\exists x\ P \equiv \forall x\ \neg P & \qquad & \neg(P \lor Q) \equiv \neg P \land \neg Q \\
\neg\forall x\ P \equiv \exists x\ \neg P & \qquad & \neg(P \land Q) \equiv \neg P \lor \neg Q \\
\forall x\ P \equiv \neg\exists x\ \neg P & \qquad & P \land Q \equiv \neg(\neg P \lor \neg Q) \\
\exists x\ P \equiv \neg\forall x\ \neg P & \qquad & P \lor Q \equiv \neg(\neg P \land \neg Q)\,.
\end{array}
$$

- Thus, we do not really need both ∀ and ∃, just as we do not really need both ∧ and ∨.
- For readability, we will keep both of the quantifiers.

Continued...

(v) Equality

- When two terms refer to the same object, equality symbol is used.

Exs: Father(John) = Henry

(Object referred to by Father(John) and the object referred to by Henry are the same)

Richard has at least two brothers.

$\exists\, x, y\ \ Brother(x, Richard) \land Brother(y, Richard) \land \neg(x = y)$

(equality with negation)

Using First-Order Logic

(i) Assertions and Queries in FOL:

TELL -- adds sentences (assertions) to a KB

Ex: TELL(KB, King(John))

TELL(KB, ∀x King(x) → Person(x))

ASK -- asks questions of the KB

Ex: ASK(KB, King(John)) -- returns true

ASK(KB, Person(John)) -- returns true

ASK(KB, ∃x Person(x)) -- returns a *substitution* or *binding list* ({ x | John})

(ii) The Kinship Domain:

(the domain of family relationships)

Kinship relations -- parenthood, brotherhood, marriage,…..

Unary predicates -- Male, Female

Binary predicates -- Parent, Sibling, Brother, Sister, Child, Daughter, Son, Spouse, Wife, Husband, Grandparent, Grandchild, Cousin, Aunty, Uncle

Functions -- Mother, Father

Continued...

Examples:

1. One’s mother is one’s female parent.

   $\forall$ m, c Mother(c) = m $\Leftrightarrow$ Female(m) $\wedge$ Parent(m, c)

2. One’s husband is one’s male spouse.

   $\forall$ w, h Husband(h, w) $\Leftrightarrow$ Male(h) $\wedge$ Spouse(h, w)

3. Male and Female are disjoint categories.

   $\forall$x Male(x) $\Leftrightarrow$ $\neg$Female(x)

4. Parent and child are inverse relations.

   $\forall$ p, c Parent(p, c) $\Leftrightarrow$ Child(c, p)

5. A grandparent is a parent of one’s parent.

   $\forall$ g, c Grandparent(g, c) $\Leftrightarrow$ $\exists$ p Parent(g, p) $\wedge$ Parent(p, c)

6. A sibling is another child of one’s parents.

   $\forall$ x, y Sibling(x, y) $\Leftrightarrow$ x $\neq$ y $\wedge$ $\exists$ p Parent(p, x) $\wedge$ Parent(p, y)

Continued...

- Each of these sentences can be viewed as an axiom of the domain (axioms associate with purely mathematical domains).
- Some sentences are <u>theorems</u> -- i.e., they are entailed by axioms.

  $\forall x, y \;\; Sibling(x, y) \Leftrightarrow Sibling(y, x)$ (follows from siblinghood axiom)
- Axioms can also be just <u>plain facts</u>.

  Male(Jim). Spouse(Jim, Laura).

<u>(iii) Numbers, Sets and Lists:</u>

<u>Describing Natural Numbers (or Non-negative Integers):</u>

NatNum -- a predicate

0 -- a constant symbol

S -- a function symbol (for successor)

<u>Exs:</u> 1. Defining natural numbers recursively.

$$NatNum(0).$$

$$\forall n \;\; NatNum(n) \rightarrow NatNum(S(n))$$

Continued...

2. To constrain the successor function.

$\forall n \quad 0 \neq S(n)$

$\forall m,n \quad m \neq n \rightarrow S(m) \neq S(n)$

3. Addition in terms of the successor function.

$\forall m \quad NatNum(m) \rightarrow +(m, 0) = m$

$\forall m, n \quad NatNum(m) \wedge NatNum(n) \rightarrow +(S(m), n) = S(+(m, n))$ Prefix notation

Infix notation: $\forall m, n \quad NatNum(m) \wedge NatNum(n) \rightarrow (m+1)+n = (m+n)+1$

Continued...

Sets:

Empty Set -- { }

Unary predicate -- Set (it is true of sets)

Binary predicates -- $x \varepsilon s$ (x is a member of set s)

$s_1 \subseteq s_2$ (s_1 is a subset of s_2; not necessarily a proper subset)

Binary functions -- $s_1 \cap s_2$ (intersection)

$s_1 \cup s_2$ (union)

{ x | s } (set resulting from adjoining element x to set s)

One possible set of axioms is as follows:

1. The only sets are the empty set and those made by adding something to a set:

$$\forall s\ Set(s) \Leftrightarrow (s=\{\}) \lor (\exists x, s_2\ Set(s_2) \land s=Add(x, s_2)) .$$

2. The empty set has no elements added into it. In other words, there is no way to decompose $\{\}$ into a smaller set and an element:

$$\neg \exists x, s\ Add(x, s)=\{\} .$$

3. Adding an element already in the set has no effect:

$$\forall x, s\ x \in s \Leftrightarrow s=Add(x, s) .$$

4. The only members of a set are the elements that were added into it. We express this recursively, saying that x is a member of s if and only if s is equal to some element y added to some set s_2, where either y is the same as x or x is a member of s_2:

$$\forall x, s\ x \in s \Leftrightarrow \exists y, s_2\ (s=Add(y, s_2) \land (x=y \lor x \in s_2)) .$$

5. A set is a subset of another set if and only if all of the first set's members are members of the second set:

$$\forall s_1, s_2 \;\; s_1 \subseteq s_2 \;\Leftrightarrow\; (\forall x \;\; x \in s_1 \;\Rightarrow\; x \in s_2)\,.$$

6. Two sets are equal if and only if each is a subset of the other:

$$\forall s_1, s_2 \;\; (s_1 = s_2) \;\Leftrightarrow\; (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)\,.$$

7. An object is in the intersection of two sets if and only if it is a member of both sets:

$$\forall x, s_1, s_2 \;\; x \in (s_1 \cap s_2) \;\Leftrightarrow\; (x \in s_1 \wedge x \in s_2)\,.$$

8. An object is in the union of two sets if and only if it is a member of either set:

$$\forall x, s_1, s_2 \;\; x \in (s_1 \cup s_2) \;\Leftrightarrow\; (x \in s_1 \vee x \in s_2)\,.$$

Continued...

Lists:

Similar to sets, but lists are ordered and the same element can appear more than once in a list.

Nil -- constant list with no elements

Functions -- Cons, Append, First, Rest

Predicate -- Find

List? -- a predicate that is true only of lists

Empty list -- [ ]

Cons(x, y) -- written as [x | y] (y is a non-empty list)

Cons(x, Nil) -- written as [x]

list [A, B, C] -- corresponds to Cons(A, Cons(B, Cons(C, Nil)))

Continued...

(iv) The Wumpus World:

FOL Axioms:

Percept([Stench, Breeze, Glitter, None, None], 5)

Percept -- binary predicate

Stench,… -- constants placed in a list

5 --- time at which the percept occurred (when the agent saw what)

Actions

Turn(Right), Turn(left), Forward, Shoot, Grab, Release, Climb

To determine best action, the agent program constructs a query such as,

$$\textsc{AskVars}(KB, BestAction(a, 5)),$$

ASK solves this query and returns a binding list such as { a | Grab }

(Grab is the action to take)

Continued...

The raw percept data implies certain facts about the current state.

$$\forall t, s, g, w, c \;\; Percept([s, Breeze, g, w, c], t) \Rightarrow Breeze(t)$$
$$\forall t, s, g, w, c \;\; Percept([s, None, g, w, c], t) \Rightarrow \neg Breeze(t)$$
$$\forall t, s, b, w, c \;\; Percept([s, b, Glitter, w, c], t) \Rightarrow Glitter(t)$$
$$\forall t, s, b, w, c \;\; Percept([s, b, None, w, c], t) \Rightarrow \neg Glitter(t)$$

Simple ‘reflex’ behavior can be shown by quantification.

$$\forall t \;\; Glitter(t) \Rightarrow BestAction(Grab, t).$$

Representing environment:

Objects: squares, pits, wumpus

Adjacency of any two squares:

$$\forall x, y, a, b \;\; Adjacent([x, y], [a, b]) \Leftrightarrow$$
$$(x = a \land (y = b - 1 \lor y = b + 1)) \lor (y = b \land (x = a - 1 \lor x = a + 1)).$$

Ex: Squares adjacent to [3, 2]: [4, 2], [2, 2], [3, 3], [3, 1]

Continued...

Home(Wumpus)

A function that specifies the square in which wumpus lives

Wumpus

A constant

Agent's location changes over time

At(Agent, s, t) (agent is at square s at time t)

Ex: Agent is at a square and perceives a breeze, then that square is breezy.

$$\forall s,t \ At(Agent,s,t) \wedge Breeze(t) \Rightarrow Breezy(s).$$

****