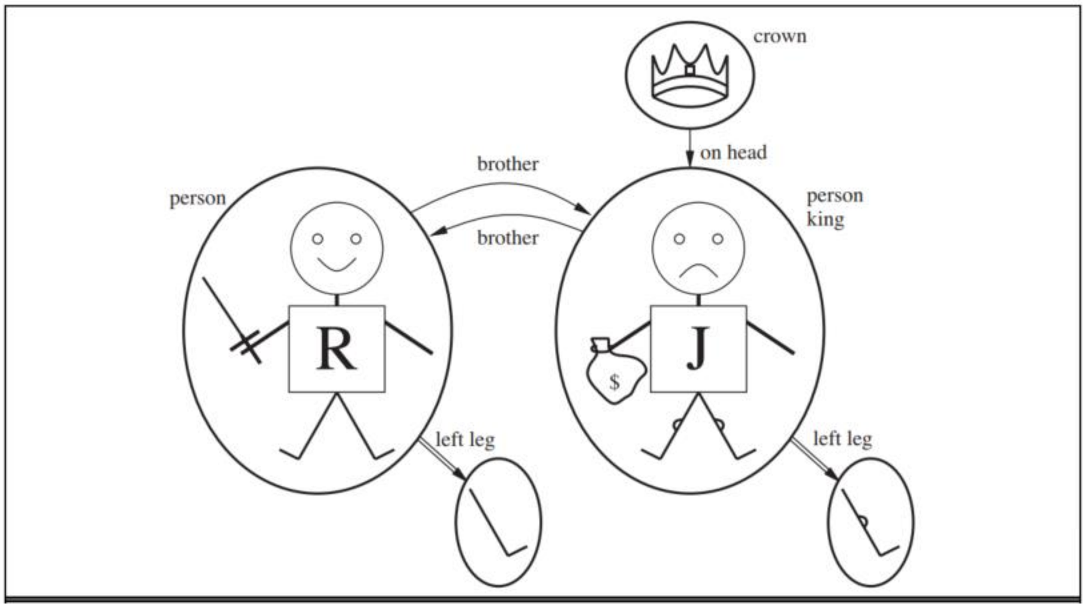
First-Order Logic

In previous lectures, we showed how a knowledge-based agent could represent the world in which it operates and deduce what action to take. We used propositional logic as our representation language because it sufficient to illustrate the basic concepts of logic and knowledge-based agents. Unfortunately, propositional logic is too puny a language to represent knowledge of complex environments in a concise way. Propositional logic lacks the expressive power to concisely describe an environment with many objects. In this lecture, we examine first-order logic which is sufficiently expressive to represent a good deal of our commonsense knowledge.

Syntax and Semantics of First-Order Logic

The model of a logical language are the formal structures that constitute the possible worlds under consideration. Each model links the vocabulary of the logical sentences to elements of the possible world. So that the truth of any sentence can be determined. Thus, models for propositional logic link proposition symbols to predefined truth values. Models for first-order logic are much more interesting. First, they have objects in them, The domain of a model is the set of objects or domain elements it contains. The domain is required to be non-empty, every possible world must contain at least one object. Figure below shows a model with five objects:

1. Richard the Lionheart
2. King John
3. The left leg of Richard
4. The left leg of John
5. Crown.



The objects in the model may be related in various ways, in the figure, Richard and John are brothers. Formally speaking, a relation is just the set of tuples of objects that are related. Thus, the brotherhood relation in this model is the set.

$$\{ \langle \text{Richard the Lionheart}, \text{ King John} \rangle, \langle \text{King John}, \text{ Richard the Lionheart} \rangle \}$$

The crown is on King Johns head, so “on head” relation contain just one tuple.

$$\langle \text{the crown}, \text{ King John} \rangle$$

The “brother” and “on head” relations are ***binary relations***, that is they relates ***pairs of objects***.

The model also contains ***unary relations***, or properties: the “person” property is true of both Richard and John. The king property is true on of John. And the crown property is true only of the crown.

Certain kinds of relationships are best considered as ***functions***, in that a given object must be related to exactly one object in this way for example, each person has one left leg, so the model has a unary “left leg” function that includes the following mappings.

$$\langle \text{Richard the Lionheart} \rangle \rightarrow \text{Richard's left leg}$$
$$\langle \text{King John} \rangle \rightarrow \text{John's left leg} .$$

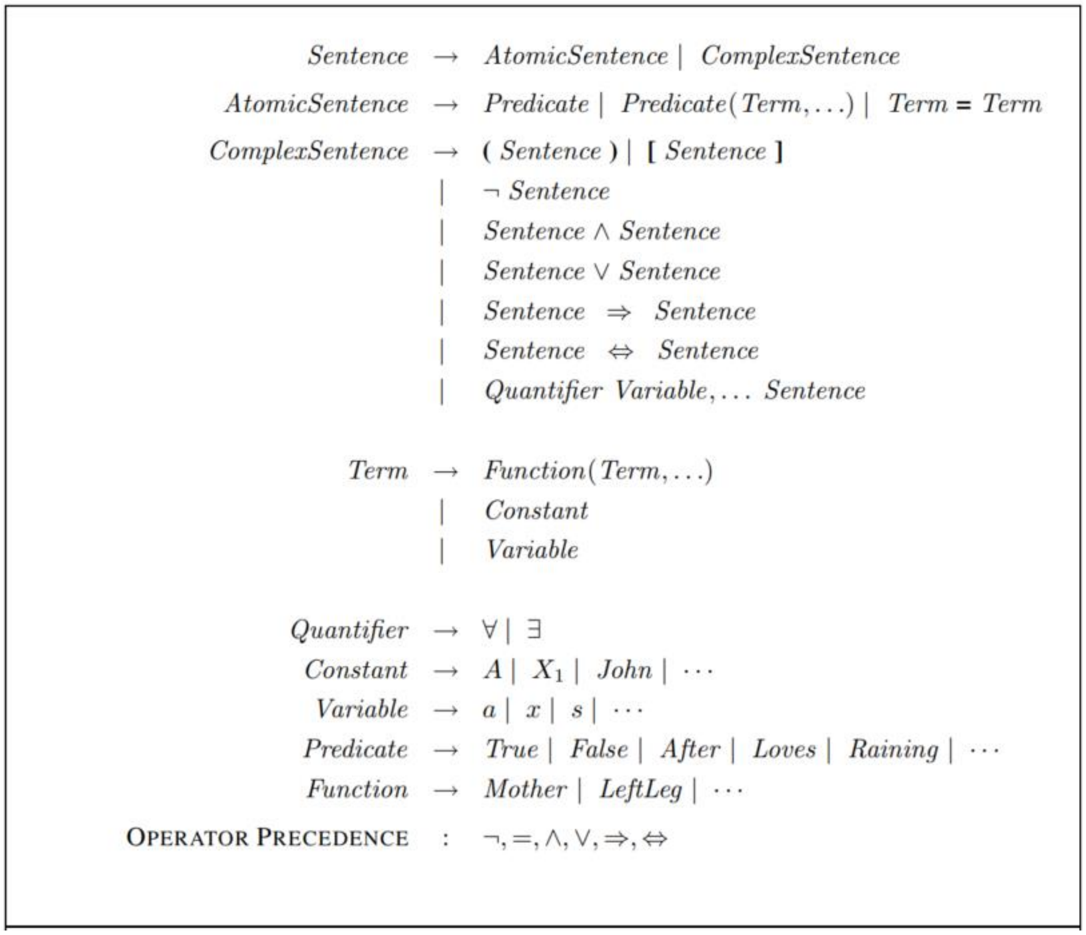
Symbols and Interpretations

We turn now to the syntax of first-order logic. The basic syntactic elements of first-order logic are the symbols that stand for **objects, relations, and functions**. The symbols, therefore, come in three kinds:

- **Constant symbols which stand for object**.
- **Predicate symbols which stand for relations**
- **Function symbols which stand for function.**

We adopt the convention that these symbols will begin with uppercase letters. For example, we might use the constant symbols ***Richard*** and ***John***. The predicate symbols ***Brother***, ***OnHead***, ***Person***, ***King***, and ***Crown***. And the function symbol ***LeftLeg***. As with proposition symbols, the choice of names is entirely up to the user. Each ***predicate*** and ***function*** symbol comes with ***arity*** that fixes the number of arguments.

As in propositional logic, every model must provide the information required to determine if any given sentence is true or false. Thus, in addition to its objects, relations and functions, each model includes an ***interpretation*** that specifies exactly which objects, relations and functions are referred to by the constant, predicate and functions symbols. The complete description of the formal grammar for the first order logic are explained in figure below.



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

In summary, a model in first-order logic consists of a set of objects and interpretation that maps constant symbols to objects, predicate symbols to relations on those objects, and function symbols to functions on those objects. Just as with propositional logic, entailment, validity, and so on are defined in terms of all possible models.

Terms

A ***term*** is a logical expression that refers to an object. Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object. For example, in English we might use the expression “King john’s left leg” rather than giving a name to his leg. This is what function symbols are for: instead of using a constant symbol, we use LeftLeg(John). In the general case, a term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol. It is important to remember that a complex term is just a complicated kind of name.

The formal semantic of terms is straightforward. Consider a term ***f(t*₁*,……, t*ₙ*)***. The function symbol *f* refers to some function in the model (call it F); the argument terms refer to objects in the domain (call them $d_1, \dots, d_n$), and the term as a whole refers to the object that is the value of the function F applied to $d_1, \dots, d_n$. For example, suppose the LeftLeg function symbol refers to the function:

$$\langle \text{King John} \rangle \rightarrow \text{John's left leg} .$$

And John refers to King John, then ***LeftLeg(John)*** refers to King John's left leg. In this way, the interpretation fixes the reference of every term.

Atomic Sentences

Now that we have both terms for referring to objects and predicate symbols for referring to relations, we can put them together to make ***atomic sentences*** that state facts. An ***atomic sentence*** is formed from a ***predicate*** symbol optionally followed by a parenthesized list of terms, such as.

$$Brother(Richard, John)$$

This states, under the intended interpretation given earlier, that Richard the Lionheart is the brother of King John. Atomic sentence can have complex terms as arguments. For example:

$$Married(Father(Richard), Mother(John))$$

States that Richard the Lionheart's father is married of King John's mother.

An atomic sentence is ***true*** in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the argument.

Complex Sentences

We can use **logical connectives** to construct more complex sentences, with the same syntax and semantics as in propositional calculus. Here are four sentences that are true in the previous model.

$$\neg Brother(LeftLeg(Richard), John)$$
$$Brother(Richard, John) \land Brother(John, Richard)$$
$$King(Richard) \lor King(John)$$
$$\neg King(Richard) \Rightarrow King(John) .$$

Quantifiers

Once we have a logic that allows objects, it is only natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. Quantifiers let us do this. First-order logic contains two standard quantifiers, called ***universal*** and ***existential***.

Universal quantification (∀)

Rules such as "All kings are persons" are the bread and butter of first-order logic. The above rule is written in first-order logic as:

$$\forall x \;\; King(x) \Rightarrow Person(x) .$$

∀ is usually pronounced "For all …". Thus, the sentence says, "For all x, if x is a king, then x is a person". The symbol x is called a **variable**. By conventions, variables are lowercase letters. A variable is a term all by itself, and as such can also serve as the argument of a function. For example, LeftLeg(x), A term with no variables is called a **ground term**.

Intuitively, the sentence ∀*x P*, where P is any logical expression, says that P is true for every object x. More precisely, ∀ x P is true in a give model if P is true in all possible **extended interpretations** constructed from the interpretation given in the model, where each extended interpretation specifies a domain element to which x refers.

Existential quantification (∃)

Universal quantification makes statements about every object. Similarly, we can make a statement about some object in the universe without naming it, by using an existential quantifier. To say, for example, that King John has a crown on his head, we write.

$\exists x$ is pronounced "There exists an x such that …..” or "For some x ….”. intuitively, the sentence $\exists x\ P$ is true for at least one object x. More precisely, $\exists x\ P$ is true in a given model if P is true in at least one extended interpretation that assigns x to a domain element.

Nested quantifiers

We will often want to express more complex sentences using multiple quantifiers. The simplest case is where the quantifiers are of the same type. Fore example, “Brother are siblings” can be written as:

$$\forall x \ \forall y \ \ Brother(x, y) \Rightarrow Sibling(x, y)$$

Consecutive quantifiers of the same type can be written as one quantifier with several variables. For example, to say that siblinghood is a symmetric relationship, we can write:

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

In other cases, we will have mixtures. “Everybody loves somebody” means that for every person, there is someone that person loves.

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

On the other hand, to say “There is someone who is loved by everyone. We write:

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

The order of quantification is therefore very important. It becomes clearer if we insert parentheses. $\forall\, x\ (\exists\, y\ Loves(x, y))$ says that everyone has a particular property, namely, the property that hey love someone. On the other hand, $\exists\, y\ (\forall\, x\ Loves(x, y))$ says that someone in the world has a particular property, namely the property of being loved by everybody.

Connections between ∀ and ∃

The two quantifiers are actually intimately connected with each other, through negations. Asserting that **everyone dislikes parsnips** is the same asserting there **does not exist someone who likes them**, and vice versa:

$$\forall x \;\; \neg Likes(x, Parsnips) \quad \text{is equivalent to} \quad \neg\exists x \;\; Likes(x, Parsnips)$$

We can go one step further: “Everyone likes ice cream” means that there is no one who does not like ice cream:

$$\forall x \;\; Likes(x, IceCream) \quad \text{is equivalent to} \quad \neg\exists x \;\; \neg Likes(x, IceCream)$$

Because ∀ is really a conjunction over the universe of objects and ∃ is a disjunction, it should not be surprising that they obey De Morgan’s rules. The De Morgan rules for quantified and unquantified sentences are as follows.

$$\begin{array}{llllll}
\forall x \;\; \neg P & \equiv & \neg\exists x \;\; P & \qquad \neg(P \vee Q) & \equiv & \neg P \wedge \neg Q \\
\neg\forall x \;\; P & \equiv & \exists x \;\; \neg P & \qquad \neg(P \wedge Q) & \equiv & \neg P \vee \neg Q \\
\forall x \;\; P & \equiv & \neg\exists x \;\; \neg P & \qquad P \wedge Q & \equiv & \neg(\neg P \vee \neg Q) \\
\exists x \;\; P & \equiv & \neg\forall x \;\; \neg P & \qquad P \vee Q & \equiv & \neg(\neg P \wedge \neg Q)
\end{array}$$

Equality

First-order logic includes one more way to make atomic sentences, other than using a predicate and terms as described earlier. We can use the **equality symbol** to signify those two terms refer to the same object. For example.

$$Father(John) = Henry$$

Says that the object referred to by ***Father (John)*** and the object referred to by Henry are the same. Because an interpretation fixes the referent of any term, determining the truth of an equality sentence is simply a matter of seeing that the referents of the two terms are the same object.

The equality symbol can be used to state facts about a given function, as we just did for the ***father*** symbol. It can also be used with negation to insist that two terms are not the same object. To say that Richard has at least two brothers, we would write.

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