First-Order Logic

First-order logic is a generalisation of the language of logic from propositions to statements about individuals (or things). For example, instead of a particular sentence about a particular person such as "Amitabh is Rich" being treated as an atomic proposition (and which we will abstract into a propositional variable or letter) that is either given the value *true* or that of *false* (according to the state of the world), we consider the property "x is Rich" for any individual x, and assign truth values to this statement depending on who x represents. Likewise we can have statements such as *"x is married to y"*. In other words, we are interested in relationships between individuals (which may be people or things, depending on our domain of discourse).

Syntax

Let us remember Wittgenstein's claim that the world is made up of facts, not things.

We present the abstract syntax of a first-order logic, denoted $\mathscr{L}(\Sigma, \Pi, \mathscr{X})$, which is parameterised by three mutually disjoint sets of symbols, $\Sigma, \Pi, \mathscr{X}$.

> $\sum$ is, as before, a signature, which specifies the constant and function symbols in the language, and how they are to be used, i.e, by giving their arities.
>
> $\mathscr{X}$ is, as before, a set of variables, which we will assume to be denumerable. Let $x, x_1, \ldots, y, y_1, \ldots \in \mathscr{X}$ be typical "individual variables", or simply variables.
>
> $\prod$ is also similar to a signature, in that it specifies symbols and their arities, but $\prod$ specifies predicate symbols, which will be used to denote relations. Let $P, P', P_1, \ldots, Q, Q', Q_1, \ldots \in \Pi$ be typical "predicate symbols". Note that 0-ary predicate symbols serve the role of our propositional variables.

A language of first-order logic $\mathscr{L}(\Sigma, \Pi, \mathscr{X})$ is a two-level language.

At the lower level is a "language of things", which is our familiar tree algebra $\mathbf{Tree}_\Sigma(\mathscr{X})$, with the carrier set being the set of abstract syntax trees $Tree_\Sigma(\mathscr{X})$. (Note: The effort we put in earlier to define these sets and algebras, and study properties and theorems about this algebra will pay off in our understanding of first-order languages.)

At the upper level, is a "language of facts about things". This language is the language of formulas, which we specify inductively using the (now familiar) presentation of an abstract grammar.

Definition: The set of formulas *Form* is inductively defined, using the following abstract grammar, with typical meta-variables $p, p_1, p_2 \ldots \in Form$, as:

$$
\begin{aligned}
p, p_1, p_2 \ldots \in Form ::= \\
&\mathrm{T} \mid \mathrm{F} && \text{(boolean constants)} \\
&\mid P(t_1, \ldots, t_k) && \text{(atomic formulas)} \\
&\mid \neg p_1 \mid p_1 \wedge p_2 \mid p_1 \vee p_2 \mid p_1 \rightarrow p_2 \mid p_1 \leftrightarrow p_2 && \text{(usual connectives)}
\end{aligned}
$$

$| (\forall x)\ p_1 | (\exists x)\ p_1$ (quantified formulas)

We have highlighted the significant differences between propositions and formulas in a maroon colour. There are several things to note in how we have moved from a 0^{th}-order language (propositions) to a first-order language:
First, we retain many of the basic features about the structure of propositions. In particular, all the constants and the propositional connectives are retained, both in the syntax and in their interpretation as boolean constants and boolean functions.

The base case, apart from the propositional constants, no longer comprises propositional variables, but instead has "atomic formulas". An atomic formula $P(t_1, \ldots, t_k)$ is built using a k-ary predicate symbol P taken from $\prod$, which is provided a k-tuple of arguments which are abstract trees (terms) from $Tree_{\Sigma}(\mathcal{X})$. Note that to be well-formed, the number of arguments given to a predicate symbol in an atomic formula must exactly equal the arity of the symbol as specified in $\prod$.

Two other new constructions involve the quantifiers $\forall$ (called "universal") and $\exists$ (called "existential"). When we want to write either quantifier without specifying which, we will use the meta-symbol $\mathcal{Q}$. Both quantifiers are "binding" operations, and take an (individual) variable e.g., x, which is treated as "bound" in the "scope" of the formula p_1 that is part of the quantified formula.

The language is first-order in the following sense: variables are used to denote "things"; they do not denote facts about things (which are represented by formulas). Furthermore, the quantification is only over individual variables — which, as mentioned above, range over individual "things", and not over facts or sets of things or properties (which can be seen as defined by facts about things).

Definition: An occurrence of an (individual) variable x is called a binding occurrence if it appears immediately after a quantifier $\mathcal{Q}$ in a quantified formula $(\mathcal{Q}x)\ p_1$.
An occurrence of a variable x is called a bound occurrence if it appears in the scope p_1 of a quantification $(\mathcal{Q}x)$ in a quantified formula $(\mathcal{Q}x)\ p_1$. Note that a bound variable x may occur below (in the syntax tree) many quantification instances $(\mathcal{Q}x)$ — this occurrence of x is bound by the nearest binding occurrence of x in the path from this node to the root of the abstract syntax tree.
An occurrence of a variable x is called a free occurrence if it is neither a binding occurrence nor a bound occurrence. That is, in the path from this occurrence of x to the root of the abstract syntax tree, no quantification $(\mathcal{Q}x)$ (with a binding occurrence of the same variable x) appears.

Example: Consider the formula $Q(x, a) \wedge (\forall x)\ P(x)$. The three occurrences of x are (in the order in which they appear) free, binding and bound.

Definition: The set of free variables $fv(p)$ in a formula p can be inductively defined, using case analysis as:

$fv(\mathbb{T}) = \{\}$
$fv(\mathbb{F}) = \{\}$

$$fv(P(t_1, \ldots, t_k)) = \bigcup_{i=1}^{k} vars(t_i)$$

$$fv(\neg p_1) = fv(p_1)$$
$$fv(p_1 \wedge p_2) = fv(p_1) \cup fv(p_2)$$
$$fv(p_1 \vee p_2) = fv(p_1) \cup fv(p_2)$$
$$fv(p_1 \rightarrow p_2) = fv(p_1) \cup fv(p_2)$$
$$fv(p_1 \leftrightarrow p_2) = fv(p_1) \cup fv(p_2)$$

$$fv((\mathcal{Q}x)\ p_1) = fv(p_1) - \{x\}$$

A formula p is called a sentence if $fv(p) = \{\}$, i.e., it has no free variables. Sentences are particularly interesting in the semantics of formulas, and in the development of the semantic results.

Coding formulas in OCaml. Assume we have already defined a data type `tree` for abstract syntax trees based on a signature $\sum$ and a set of variables $\mathcal{X}$. We now define a data type for formulas (superseding our earlier definition of propositions).

```
type formula = T | F  (* propositional constants *)
    | Atom of pred * tree list(* atomic formula *)
    | Not of formula   (* negation *)
    | And of formula * formula  (* conjunction *)
    | Or of formula * formula    (* disjunction *)
    | Imply of formula * formula   (* Implication *)
    | Iff of formula * formula (* Bi-implication or Iff or
equivalence *)
    | Forall of string * formula   (* Universal
quantification *)
    | Exists of string * formula   (* Existential
quantification *)
;;
```

Exercise: Write a recursive function `form_ht: formula -> int`, which returns the height of a formula (not considering the height of any `tree` that appears in the formula).

Exercise: Write a recursive function `ast_ht: formula -> int`, which returns the height of an abstract syntax tree (considering the heights of every `tree` that appears in the formula).

Exercise: Write a recursive function `form_size: formula -> int`, which returns the number of formula constructions in a formula (not considering the size of of any `tree` that appears in the formula).

Exercise: Write a recursive function `ast_size: formula -> int`, which returns the number of nodes in the complete abstract syntax tree (considering the nodes of everyy `tree` that appears in the formula).

Exercise: Write a recursive function `fv: formula -> string set`, which returns the *set* of free variables that appear in a proposition.

Definition: A formula p is in prenex normal form (PNF) if in its syntax tree, no quantifiers appear below any propositional connective ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$) .

Definition: A proposition p is in negation normal form (NNF) if in its syntax tree, no quantifiers or connectives appear below a negation symbol ($\neg$) , i.e., only atomic formulas can appear below a negation.

Semantics

The semantics of formulas in a first-order language $\mathscr{L}(\Sigma, \Pi, \mathscr{X})$ is also presented in two stages.

First, we need to decide on the domain of discourse, and the meaning of all the constant and function symbols. This is where we recourse to the concepts already discussed, namely that of $\sum$-algebras, for the given signature $\sum$ of the language $\mathscr{L}(\Sigma, \Pi, \mathscr{X})$. Let $\mathscr{A} = \langle A, \ldots \rangle$ be a $\sum$-algebra, with carrier set A, and interpretations of 0-ary symbols as elements of A, and k-ary symbols of $\sum$ as k-ary (total) functions in $A^k \rightarrow A$.

Once the domain of discourse is fixed as A, namely the carrier set of the $\sum$-algebra $\mathscr{A} = \langle A, \ldots \rangle$, we can provide meaning to the predicate symbols in $\prod$. Every k-ary predicate symbol P in $\prod$ is interpreted as a k-ary relation over A, i.e., some subset of A^k. Let $\mathscr{I}$ denote the interpretation of predicate symbols that maps each k-ary symbol P in $\prod$ to some k-ary relation over A — we write $\mathscr{I}(P)$ to denote the relation corresponding to P.

Definition: A first-order structure for the language $\mathscr{L}(\Sigma, \Pi, \mathscr{X})$ is defined as $\mathscr{M} = \langle \mathscr{A}, \mathscr{I} \rangle$, where (we first) choose a $\sum$-algebra $\mathscr{A} = \langle A, \ldots \rangle$, and then choose an interpretation $\mathscr{I}$ of the predicate symbols in $\prod$ as k-ary relations over A. This determines the meanings of all the fixed symbols in the language.

The meanings of the variable symbols are given as an A-assignment or A-valuation, i.e., a function $\rho \in [\mathscr{X} \rightarrow A]$, where A is the carrier set of the $\sum$-algebra $\mathscr{A} = \langle A, \ldots \rangle$ in the structure $\mathscr{M} = \langle \mathscr{A}, \mathscr{I} \rangle$.

Definition: The truth value of a formula in a first-order language is defined inductively on the structure of the formula, with respect to a given first-order structure $\mathscr{M} = \langle \mathscr{A}, \mathscr{I} \rangle$ and A-valuation $\rho \in [\mathscr{X} \rightarrow A]$ as the following semantic function $[\![p]\!]^{\mathscr{M}} \rho$ as follows (we will leave the function name "*truth*" unwritten):

$$[\![\mathrm{T}]\!]^{\mathscr{M}} \rho = true$$
$$[\![\mathrm{F}]\!]^{\mathscr{M}} \rho = false$$

$$[\![P(t_1, \ldots t_k)]\!]^{\mathscr{M}}\rho = true \text{ if and only if } (\widehat{\rho_{\mathscr{A}}}\,(t_1), \ldots, \widehat{\rho_{\mathscr{A}}}\,(t_k)) \in \mathscr{I}(P)$$
$$[\![\neg p_1]\!]^{\mathscr{M}}\rho = not\ ([\![p_1]\!]^{\mathscr{M}}\rho)$$
$$[\![p_1 \wedge p_2]\!]^{\mathscr{M}}\rho = ([\![p_1]\!]^{\mathscr{M}}\rho)\ and\ ([\![p_2]\!]^{\mathscr{M}}\rho)$$
$$[\![p_1 \vee p_2]\!]^{\mathscr{M}}\rho = ([\![p_1]\!]^{\mathscr{M}}\rho)\ or\ ([\![p_2]\!]^{\mathscr{M}}\rho)$$
$$[\![p_1 \rightarrow p_2]\!]^{\mathscr{M}}\rho = (not\ ([\![p_1]\!]^{\mathscr{M}}\rho))\ or\ ([\![p_2]\!]^{\mathscr{M}}\rho)$$
$$[\![p_1 \leftrightarrow p_2]\!]^{\mathscr{M}}\rho = ([\![p_1]\!]^{\mathscr{M}}\rho)\ iff\ ([\![p_2]\!]^{\mathscr{M}}\rho)$$
$$[\![(\forall x)\ p_1]\!]^{\mathscr{M}}\rho = true \text{ if (and only if) for all } a \in A, [\![p_1]\!]^{\mathscr{M}}(\rho[x \mapsto a]) = true$$
$$[\![(\exists x)\ p_1]\!]^{\mathscr{M}}\rho = true \text{ if (and only if) for some } a \in A, [\![p_1]\!]^{\mathscr{M}}(\rho[x \mapsto a]) = true$$

We concentrate on the important new cases, highlighted in maroon. The truth value given to an atomic formula $P(t_1, \ldots, t_k)$ depends on both the structure $\mathscr{M} = \langle \mathscr{A}, \mathscr{I} \rangle$ as well as on the A-valuation $\rho \in [\mathscr{X} \rightarrow A]$. First we evaluate the k terms (trees) $t_1, \ldots, t_k$ using $\widehat{\rho_{\mathscr{A}}}$ the (unique homomorphic extension of ρ) to the set $Tree_{\Sigma}(\mathscr{X})$, with the interpretations of symbols in $\sum$ given by the algebra $\mathscr{A}$. This will yield values $a_1, \ldots, a_k$, each of which is in A. We then check if the tuple $(a_1, \ldots, a_k)$ belongs in the interpretation $\mathscr{I}(P)$ of the symbol P, returning *true* as the meaning of the atomic formula if it is, and *false* otherwise.

In the case of quantifiers, the meaning $[\![(\forall x)\ p_1]\!]^{\mathscr{M}}\rho$ of formula $(\forall x)\ p_1$ is *true* if for each a in A, when we consider the variation of A-valuation $\rho \in [\mathscr{X} \rightarrow A]$ that maps x to a, we have $[\![p_1]\!]^{\mathscr{M}}(\rho[x \mapsto a]) = true$.

The meaning $[\![(\forall x)\ p_1]\!]^{\mathscr{M}}\rho$ of formula $(\exists x)\ p_1$ is *true* if there is some a in A, such that when we consider the variation of A-valuation $\rho \in [\mathscr{X} \rightarrow A]$ that maps x to a, we have $[\![p_1]\!]^{\mathscr{M}}(\rho[x \mapsto a]) = true$.

Exercise*: Suppose p is a *sentence* in $\mathscr{L}(\Sigma, \Pi, \mathscr{X})$, and $\mathscr{M} = \langle \mathscr{A}, \mathscr{I} \rangle$ a first-order structure with respect to this language. Show (by induction on the structure of p) that $[\![p]\!]^{\mathscr{M}}\rho$ is independent of $\rho \in [\mathscr{X} \rightarrow A]$, i.e., the value of $[\![p]\!]^{\mathscr{M}}\rho_1 = [\![p]\!]^{\mathscr{M}}\rho_2$ for any $\rho_1, \rho_2 \in [\mathscr{X} \rightarrow A]$.

Exercise:** Suppose we are given a representation of a first order structure, write a module with recursive definitions for some type `fostructure`, with its carrier set being the type `carrier`. Define the function `truth: formula -> fostructure -> (string -> carrier) -> bool`, which evaluates a formula with respect to a given A-valuation for the (individual) variables.

In the following, we will assume a fixed first-order language $\mathscr{L}(\Sigma, \Pi, \mathscr{X})$ and only consider structures $\mathscr{M} = \langle \mathscr{A}, \mathscr{I} \rangle$ with respect to the signatures $\sum$ and $\prod$, and then A-valuations pertaining to the $\sum$-algebra $\mathscr{A} = \langle A, \ldots \rangle$.