**Formulas**

The choice of sets P and F for predicate and function symbols, respectively,is driven by what we intend to describe. For example, if we work on abdatabase representing relations between our kin we might want to considerbP = {M, F, S, D}, referring to being male, being female, being a son of ...band being a daughter of .... Naturally, F and M are unary predicates (they take one argument) whereas D and S are binary (taking two). Similarly, wemay define F = {mother-of, f ather-of}. We already know what the terms over F are. Given that knowledge, we can now proceed to define the formulas of predicate logic.

We define the set of formulas over (F,P) inductively, using the already defined set of terms over F:

- If $P \in P$ is a predicate symbol of arity $n \geq 1$, and if t1, t2,...,tn are terms over F, then P(t1, t2,...,tn) is a formula. If $\varphi$ is a formula, then so is $(\neg\varphi)$.
- If $\varphi$ and $\psi$ are formulas, then so are $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\varphi \rightarrow \psi)$.
- If $\varphi$ is a formula and x is a variable, then $(\forall x\, \varphi)$ and $(\exists x\, \varphi)$ are formulas.
- Nothing else is a formula.

Note how the arguments given to predicates are always terms. This can also be seen in the Backus Naur form (BNF) for predicate logic:

$\varphi ::= P(t1, t2,...,tn) \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (\forall x\, \varphi) \mid (\exists x\, \varphi)$

where $P \in P$ is a predicate symbol of arity $n \geq 1$, ti are terms over F and x is a variable. Recall that each occurrence of $\varphi$ on the right-hand side of the ::= stands for any formula already constructed by these rules.

**Free and bound variables**

The introduction of variables and quantifiers allows us to express the notions of all ... and some ... Intuitively, to verify that $\forall x\, Q(x)$ is true amounts to replacing x by any of its possible values and checking that Q holds for each one of them. There are two important and different senses in which such formulas can be 'true.' First, if we give concrete meanings to all predicate and function symbols involved we have a model and can check whether a formula is true for this particular model. For example, if a formula encodes a required behaviour of a hardware circuit, then we would want to know whether it is true for the model of the circuit. Second, one sometimes would like to ensure that certain formulas are true for all models.

Unfortunately, things are more complicated if we want to define formally what it means for a formula to be true in a given model. Ideally, we seek a definition that we could use to write a computer program verifying that a formula holds in a given model. To begin with, we need to understand that variables occur in different ways. Consider the formula

$\forall x\, ((P(x) \rightarrow Q(x)) \wedge S(x, y))$.

We draw its parse tree in the same way as for propositional formulas, but with two additional sorts of nodes:

- The quantifiers ∀x and ∃y form nodes and have, like negation, just one subtree.
- Predicate expressions, which are generally of the form P(t1, t2,...,tn), have the symbol P as a node, but now P has n many subtrees, namely the parse trees of the terms t1, t2,...,tn.

You can see that variables occur at two different sorts of places. First, they appear next to quantifiers ∀ and ∃ in nodes like ∀x and ∃z; such nodes always have one subtree, subsuming their scope to which the respective quantifier applies.

The other sort of occurrence of variables is leaf nodes containing variables. If variables are leaf nodes, then they stand for values that still have to be made concrete. There are two principal such occurrences:

1. we have three leaf nodes x. If we walk up the tree beginning at any one of these x leaves, we run into the quantifier ∀x. This means that those occurrences of x are actually bound to ∀x so they represent, or stand for, any possible value of x.

2. In walking upwards, the only quantifier that the leaf node y runs into is ∀x but that x has nothing to do with y; x and y are different place holders. So y is free in this formula. This means that its value has to be specified by some additional information, for example, the contents of a location in memory.