Data objects/Atoms

Some objects are only identified by their name, called an **atom**. Thus, it is correct to say that, in such a case, an object *is* an atom. For example, we saw the atom `bob`.

An atom starts with a lower-case letter which can be followed by a string of characters out of lower-case letters, upper-case letters, digits and the underscore character ('_').

For example, the following are valid atoms:

```
anna          alpha_beta_proc
x25           call_Java
x_25          x_
x_25AB        x____y
```

Data objects/Numbers

Numbers in Prolog include integer numbers and floating-point numbers. The syntax of integer is as expected, for example

```
1     1234     0     -97
```

The lower and larger integers are limited by the actual Prolog system in use.

Floating-point numbers follow the usual syntax too, like

```
3.14     -0.06     100.5
```

The general syntax is not given here because Prolog is primarily intended for symbolic computations, not numerical computations.

Atoms and numbers define the group of **constants**.

Data objects/Variables

A **variable** is a name for an object, but, contrary to atoms, variables do not define any object. So a variable can denote several objects. They must start with an upper-case letter or an underscore and may be followed by any number of letters, digit and underscores, in any order. For example, the following are valid variables:

```
X    Obj_List    _23    Object2    Result    ObjList    _x23
```

If a variable appears only once in the body of a clause and not in the head, it is an **unknown variable** and can be replaced by an underscore. For example

```
has_a_child(X) :- parent(X,Y).
```

can be replaced by the equivalent

```
has_a_child(X) :- parent(X,_).
```

Data objects/Variables (cont)

Several underscores can occur in a body. In this case, each of them should be considered an absolutely unique, despite unknown, variable.

For example, consider

```
someone_has_a_child :- parent(_,_).
```

Each underscore could be replaced by a unique variable, like

```
someone_has_a_child :- parent(X,Y).
```

But certainly **not**

```
someone_has_a_child :- parent(X,X).
```

Data objects/Variables and lexical scopes

If an unknown variable appears in a query, then its value, found by the interpreter, is not displayed in the result.

For example, if one wants to know who are the people who have children, the Prolog query is

```
?- parent(X,_).
```

and only all the possible values of `X` that satisfy the query will be displayed.

Given an occurrence of a variable, the part of the program where this variable is usable is called the **scope**.

Data objects/Variables and lexical scopes (cont)

Prolog uses **lexical scoping**, which means

- the same variable always represents the same object inside a clause;
- the same variable in two different clauses represent different objects, in general.

For example, in clause

```
ancestor(X,Y) :- parent(X,Y)
```

the tow occurrences of variable `X` represent the same object. But, if we also have

```
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

the occurrences of `X` denote, in general, objects different from the `X` in the previous clause.

Data objects/Functors and structures

Structured objects, or **structures**, are objects that are composed of several objects. These sub-objects are boxed together by means of a **functor**. For example

```
date(9,july,2006)
```

is a structure composed, in order, of the integer constant `1`, the atom `july`, the number 2006 and tied together by the functor `date`. The sub-objects are called **arguments** and the number of arguments is the **arity**.

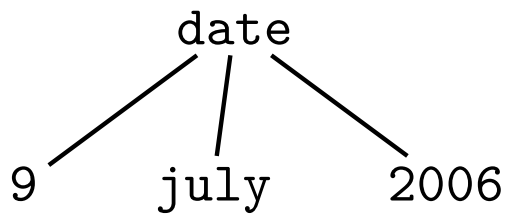
The syntax of functor is the same as the atoms. Functors are used to define facts (i.e. instances of a relation). For example,

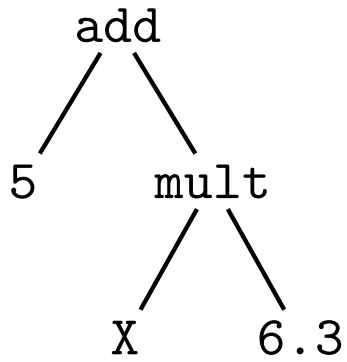
```
female(pam).
```

Data objects/Functors and structures (cont)

Just as it is possible to make a graphical representation of a successful run of the Prolog interpreter (see page 48), it is possible to draw a tree which represents a Prolog structure.

The idea here is to map a functor to a node whose label is the functor, and its arguments to sub-trees, recursively in the same order. The consequence is that the component objects that are not structures are mapped to the leaves. For example





Data objects/Summary

It is also handy to represent all the kinds of Prolog data objects we reviewed until now in a tree. The idea consists in mapping a set of lexical elements to a node whose label is this set name, and the subsets are mapped to subtrees.