Many Sorted Logic

As in typed programming languages, it sometimes helps to have structures with objects of different types.
In logic, types are called *sorts*.

E.g., some objects in a structure M may be lecturers, others may be PCs, numbers, etc.

We can handle this with unary relation symbols, or with *'many-sorted first-order logic'*. We'll use many-sorted logic mainly to specify programs.

Fix a collection $s, s', s'', \ldots$ of sorts. How many, and what they're called, are determined by the application.

These sorts do *not* generate extra sorts, like $s \to s'$ or (s, s').

If you want extra sorts like these, add them explicitly to the original list of sorts. (Their meaning would not be automatic, unlike in Haskell.)

Many Sorted Terms

We adjust the definition of 'term' (Def. 4.2), to give each term a sort:

- each variable and constant comes with a sort s. To indicate which sort it is, we write $x : s$ and $c : s$. There are infinitely many variables of each sort.
- each n-ary function symbol f comes with a template

  $$f : (s_1, \ldots, s_n) \to s$$

  where $s_1, \ldots, s_n$, and s are sorts.
  Note: $(s_1, \ldots, s_n) \to s$ is not itself a sort.
- For such an f and terms $t_1, \ldots, t_n$, if t_i has sort s_i (for each i) then $f(t_1, \ldots, t_n)$ is a term of sort s.
  Otherwise (if the t_i don't all have the right sorts), $f(t_1, \ldots, t_n)$ is not a term — it's just rubbish, like $)\forall)\to$.

Formulas

- Each n-ary relation symbol R comes with a template $R(s_1, \ldots, s_n)$, where $s_1, \ldots, s_n$ are sorts.
  For terms $t_1, \ldots, t_n$, if t_i has sort s_i (for each i) then $R(t_1, \ldots, t_n)$ is a formula. Otherwise, it's rubbish.
- $t = t'$ is a formula if the terms t, t' have the same sort. Otherwise, it's rubbish.
- Other operations ($\wedge, \neg, \forall, \exists$, etc) are unchanged. But it's polite to indicate the sort of a variable in $\forall, \exists$ by writing

  $$\forall x : s\ \phi \quad \text{instead of just} \quad \forall x \phi$$

  $$\exists x : s\ \phi \quad \text{instead of just} \quad \exists x \phi$$

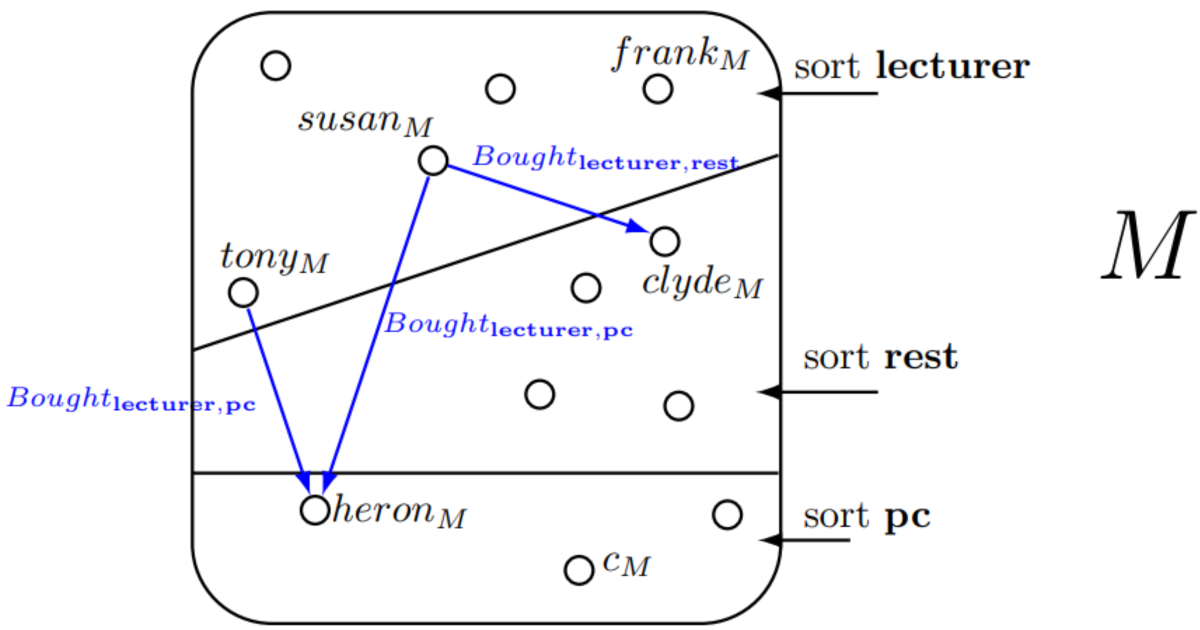
  if x has sort s. Alternatively, declare the variables of each sort.

E.g., roughly, you can write $\forall x : \mathbf{lecturer}\ \exists y : \mathbf{pc}\ (\texttt{Bought}(x, y))$ instead of $\forall x(\texttt{Lecturer}(x) \to \exists y(\texttt{PC}(y) \wedge \texttt{Bought}(x, y)))$.

Example L-structure

Let L be a many-sorted signature. An L-structure is defined as before (Definition 4.4 + slide 26), but additionally it allocates each object in its domain to a single sort. No sort should be empty.

E.g., if L has sorts **lecturer**, **pc**, **rest**, an L -structure looks like:



Interpretation of L-structures

Let M be a many-sorted L-structure.

- For each constant $c : s$ in L, M must say which object of sort s in $dom(M)$ is 'named' by c.
- For each function symbol $f : (s_1, \ldots, s_n) \to s$ in L and all objects $a_1, \ldots, a_n$ in $dom(M)$ of sorts $s_1, \ldots, s_n$, respectively, M must say which object $f_M(a_1, \ldots, a_n)$ of sort s is associated with $(a_1, \ldots, a_n)$ by f.
  M doesn't say anything about $f(b_1, \ldots, b_n)$ if $b_1, \ldots, b_n$ don't all have the right sorts.
- For each relation symbol $R(s_1, \ldots, s_n)$ in L, and all objects $a_1, \ldots, a_n$ in $dom(M)$ of sorts $s_1, \ldots, s_n$, respectively, M must say whether $R(a_1, \ldots, a_n)$ is true or not.
  M doesn't say anything about $R(b_1, \ldots, b_n)$ if $b_1, \ldots, b_n$ don't all have the right sorts.

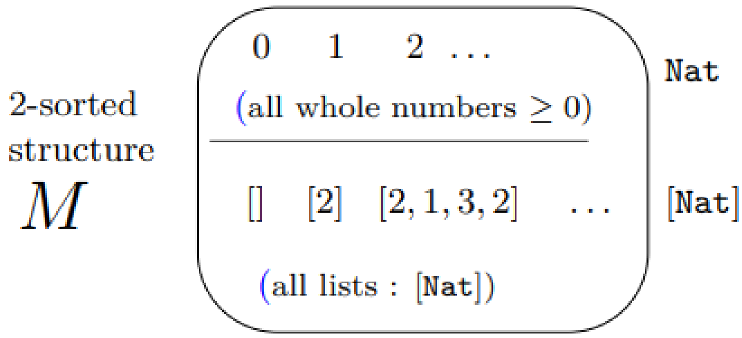
Further Notes

1. Sorts can replace some or all unary relation symbols.
2. As in Haskell, each object has only 1 sort, not 2.
   So for M above, `Human` would have to be implemented as three unary relation symbols: $\mathtt{human}_{\mathbf{lecturer}}$, $\mathtt{Human}_{\mathbf{pc}}$, $\mathtt{Human}_{\mathbf{rest}}$.
   But if (e.g.) you don't want to talk about human objects of sort **pc**, you can omit $\mathtt{human}_{\mathbf{PC}}$.
3. We need a binary relation symbol $\mathtt{Bought}_{s,s'}$ for each pair (s, s') of sorts (unless s-objects are not expected to buy s'-objects).
4. Messy alternative: use sorts for human lecturer, PC lecturer, etc — all possible types of object.

Example 1

Let's have a sort **Nat**, for $0, 1, 2, \ldots$, and a sort [**Nat**] for lists of natural numbers.

The idea is that the structure's domain should look like:



The signature should be chosen to provide access to the objects in such a structure. We may want:

[], : (cons), ++, head, tail, length (we can write as $\sharp$), !!, and $+, -$, etc., for arithmetic.

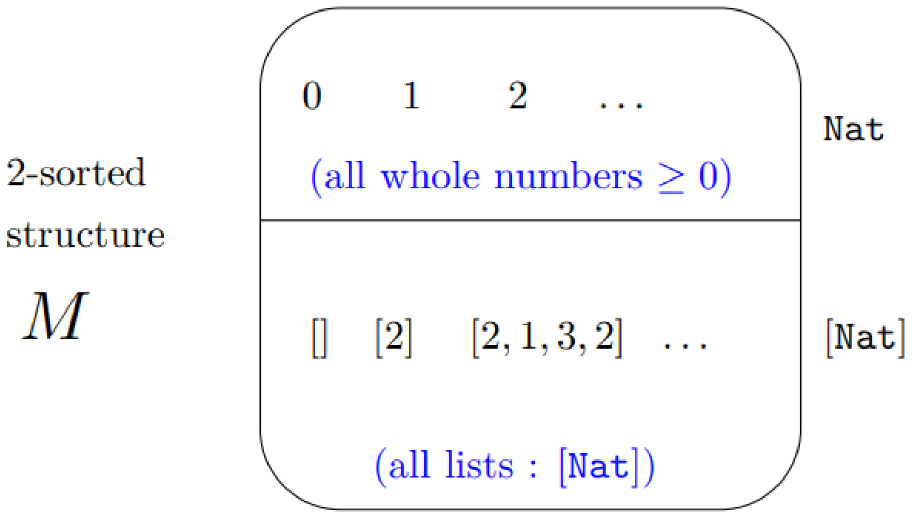
We can represent these using constants, function symbols, or relation symbols

Now we can define a signature L suitable for lists of type [$\texttt{Nat}$].

- L has constants $\underline{0}, \underline{1}, \ldots : \texttt{Nat}$, relation symbols $<, \leq, >, \geq$ of sort ($\texttt{Nat}$,$\texttt{Nat}$), and function symbols
    - $+, -, \times : (\texttt{Nat}, \texttt{Nat}) \to \texttt{Nat}$
    - $[] : [\texttt{Nat}]$ (a constant to name the empty list)
    - $\texttt{cons}(:) \ : (\texttt{Nat}, [\texttt{Nat}]) \to [\texttt{Nat}]$
    - $++ : ([\texttt{Nat}], [\texttt{Nat}]) \to [\texttt{Nat}]$
    - $\texttt{head} : [\texttt{Nat}] \to \texttt{Nat}$
    - $\texttt{tail} : [\texttt{Nat}] \to [\texttt{Nat}]$
    - $\sharp : [\texttt{Nat}] \to \texttt{Nat}$
    - $!! : ([\texttt{Nat}], \texttt{Nat}) \to \texttt{Nat}$

  We write the constants as $\underline{0}, \underline{1}, \ldots$ to avoid confusion with actual numbers $0, 1, \ldots$
- Let $x, y, z, k, n, m \ldots$ be variables of sort $\texttt{Nat}$.
  Let $xs, ys, zs, \ldots$ be variables of sort [$\texttt{Nat}$].

Let M be the L-structure



The L-symbols are interpreted in the natural way: $++$ as concatenation of lists, etc.

We define $34 - 61$, $\texttt{tail}([])$, etc. arbitrarily. So don't assume they have the values you might expect.

Describing Lists

Now we can say a lot about lists.

E.g., the following L-sentences, expressing the definitions of the function symbols, are true in M, because (as we said) the L-symbols are interpreted in M in the natural way:

- $\sharp([\,]) = \underline{0}$
- $\forall x \forall xs((\sharp(x : xs) = \sharp(xs) + \underline{1})$
- $\forall xs(xs \neq [\,] \rightarrow \mathtt{head}(xs) = xs!!\underline{0})$
- $\forall x \forall xs(\mathtt{head}(x : xs) = x)$
- $\forall x \forall xs(\mathtt{tail}(x : xs) = xs)$
- $\forall n \forall m(n < m \wedge m < \sharp(xs) \rightarrow xs!!n \leq xs!!m)$