2.9 Dependent types and modules

We will be able to define modules and abstract data types by extending the existing types in a simple but very expressive way — using so-called *dependent types*.

dependent product

Suppose you are writing a business application and you wish to construct a type representing the date:

$$\begin{aligned} Month &= \{1, \ldots, 12\} \\ Day &= \{1, \ldots, 31\} \\ \\ Date &= Month \times Day \end{aligned}$$

We would need a way to check for valid dates. Currently, $\langle 2, 31\rangle$ is a perfectly legal member of $Date$, although it is not a valid date. One thing we can do is to define

$$\begin{aligned} Day(1) &= \{1, \ldots, 31\} \\ Day(2) &= \{1, \ldots, 29\} \\ &\vdots \\ Day(12) &= \{1, \ldots, 31\} \end{aligned}$$

and we will now write our data type as

$$Date = m : Month \times Day(m).$$

We mean by this that the second element of the pair belongs to the type indexed by the first element. Now, $\langle 2, 20\rangle$ is a legal date since $20 \in Day(2)$, and $\langle 2, 31\rangle$ is illegal because $31 \notin Day(2)$.

Many programming languages implement this or a similar concept in a limited way. An example is Pascal's *variant records*. While Pascal requires the indexing element to be of scalar type, we will allow it to be of any type.

We can see that what we are doing is making a more general product type. It is very similar to $A \times B$. Let us call this type $prod(A, x.B)$. We can display this as $x : A \times B$. The typing rules are:

$$\frac{E \vdash a : A \;\; E \vdash b \in B[a/x]}{E \vdash pair(a, b) : prod(A, x.B)}$$

$$\frac{E \vdash p : prod(A, x.B) \;\; E, u : A, v : B[u/x] \vdash t \in T}{E \vdash spread(p; u, v.t) \in T}$$

Note that we haven't added any elements. We've just added some new typing rules.

dependent functions

If we allow B to be a family in the type $A \to B$, we get a new type, denoted by $fun(A; x.B)$, or $x : A \to B$, which generalizes the type $A \to B$. The rules are:

$$\frac{E, y : A \vdash b[y/x] \in B[y/x]}{E \vdash \lambda(x.b) \in fun(A; x.B)} \; new\ y$$

$$\frac{E \vdash f \in fun(A; x.B) \quad E \vdash a \in A}{E \vdash ap(f; a) \in B[a/x]}$$

Example 2 : Back to our example Dates. We see that $m : Month \to Day[m]$ is just $fun(Month; m.Day)$, where Day is a family of twelve types. And $\lambda(x.maxday[x])$ is a term in it.