`:`	`a -> [a] -> [a]`	Add a single element to the front of a list. `3:[2,3]` ⇝ `[3,2,3]`
`++`	`[a] -> [a] -> [a]`	Join two lists together. `"Ron"++"aldo"` ⇝ `"Ronaldo"`
`!!`	`[a] -> Int -> a`	`xs!!n`returns the `n`th element of `xs`, starting at the beginning and counting from 0. `[14,7,3]!!1` ⇝ `7`
`concat`	`[[a]] -> [a]`	Concatenate a list of lists into a single list. `concat [[2,3],[],[4]]` ⇝ `[2,3,4]`
`length`	`[a] -> Int`	The length of the list. `length "word"` ⇝ `4`

`head,last`	`[a] -> a`	The first/last element of the list.
		`head "word"` ⇝ `'w'`
		`last "word"` ⇝ `'d'`

`tail,init`	`[a] -> [a]`	All but the first/last element of the list.
		`tail "word"` ⇝ `"ord"`
		`init "word"` ⇝ `"wor"`

`replicate`	`Int -> a -> [a]`	Make a list of `n` copies of the item. `replicate 3 'c'` ⇝ `"ccc"`
`take`	`Int -> [a] -> [a]`	Take `n` elements from the front of a list. `take 3 "Peccary"` ⇝ `"Pec"`
`drop`	`Int -> [a] -> [a]`	Drop `n` elements from the front of a list. `drop 3 "Peccary"` ⇝ `"cary"`
`splitAt`	`Int->[a]->([a],[a])`	Split a list at a given position. `splitAt 3 "Peccary"` ⇝ `("Pec","cary")`
`reverse`	`[a] -> [a]`	Reverse the order of the elements. `reverse [2,1,3]` ⇝ `[3,1,2]`
`zip`	`[a]->[b]->[(a,b)]`	Take a pair of lists into a list of pairs. `zip [1,2] [3,4,5]` ⇝ `[(1,3),(2,4)]`
`unzip`	`[(a,b)] -> ([a],[b])`	Take a list of pairs into a pair of lists. `unzip [(1,5),(3,6)]` ⇝ `([1,3],[5,6])`

Figure 6.1 Some polymorphic list operations from `Prelude.hs`

Figure 6.1. Suppose we are looking for a function to make a list from a number of copies of a single element. It must take the item and a count and give a list, so its type will be one of

```
Int -> a -> [a]          a -> Int -> [a]
```

`and`	`[Bool] -> Bool`	The conjunction of a list of Booleans. `and [True,False]` ⇝ `False`
`or`	`[Bool] -> Bool`	The disjunction of a list of Booleans. `or [True,False]` ⇝ `True`
`sum`	`[Integer] -> Integer` `[Float] -> Float`	The sum of a numeric list. `sum [2,3,4]` ⇝ `9`
`product`	`[Integer] -> Integer` `[Float] -> Float`	The product of a numeric list. `product [0.1,0.4 .. 1]` ⇝ `0.028`

Figure 6.2 Some monomorphic list operations from the `Prelude`

Looking at Figure 6.1 we can quickly locate one function, `replicate`, which does have one of these types and is indeed the function which we seek. If we want a function to reverse a list it will have type `[a] -> [a]` and although there is more than one function with this type, the search is very much narrowed by looking at types. We'll see a little later on (page 133) that there's a web service called *hoogle* to look up functions by type.

This insight is not confined to functional languages, but is of particular use when a language supports polymorphic or **generic** functions and operators as we have seen here.

Further functions

We have not described all the functions in the prelude for two different reasons. First, some of the general functions are **higher-order** and we postpone discussion of these until Chapter 10; secondly, some of the functions, such as `zip3`, are obvious variants of things we have discussed here. Similarly, we have not chosen to enumerate the functions in the library `Data.List`; readers should consult the library file itself, which contains type information and comments about the effects of the functions, as well as the Haddock documentation for the library: we talk in detail about documentation in the next section.

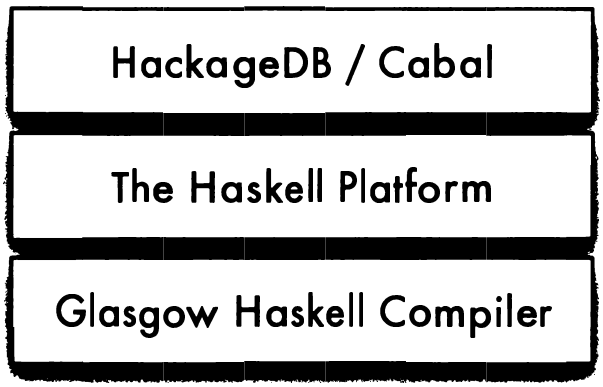

Figure 6.3 The Haskell infrastructure