CURRY IS ON THE MENU

WIM VANDERBAUWHEDE

Partial Application and Currying.

Currying. Consider a type signature of a function with three arguments:

```
f :: X -> Y -> Z -> A
```

The arrow "->" is right-associative, so this is the same as:

```
f :: X -> (Y -> (Z -> A))
```

What this means is that we can consider `f` as a function with a single argument of type `X` which returns a function of type `Y->Z->A`.

The technique of rewriting a function of multiple arguments into a sequence of functions with a single argument is called *currying*. We can illustrate this best using a lambda function:

```
\x y z -> ...
\x -> (\y z -> ...)
\x -> (\y -> (\z -> ... ))
```

The name "currying", is a reference to logician Haskell Curry. The concept was actually proposed originally by another logician, Moses Schnfinkel, but his name was not so catchy.

Partial application. Partial application means that we don't need to provide all arguments to a function. For example, given

```
sq x y = x*x+y*y
```

We note that function application associates to the left, so the following equivalence holds

```
sq x y = (sq x) y
```

We can therefore create a specialised function by partial application of x:

```
sq4 = sq 4 -- = \y -> 16+y*y
sq4 3 -- = (sq 4) 3 = sq 4 3 = 25
```

This is why you can write things like:

```
doubles = map (*2) [1 .. ]
```