TYPE INFERENCE BY EXAMPLE

WIM VANDERBAUWHEDE

Type Inference: An intuitive view. Given a function without a type signature, how can we infer the type?

For example, given `f`:

```
f x xs = x + length xs
```

The most general type for the function `f` is:

```
f ::a -> b -> c
```

Now we can add constraints based on the types of the functions in the expression on the right-hand side. Given that in the *Prelude*:

```
(+) :: Num a => a -> a -> a -- (1)
length :: [a] -> Int -- (2)
```

From (2) it follows that:

```
length xs :: Int
```

From (1) we know that the first and second argument of `(+)` must have the same type. Thus:

```
x :: Int
```

However, for `xs` all we can say is that, from (2), it must be a list. So the final type is:

```
f :: Int -> [a] -> Int
```

Now, if we modify the definition of `f` as follows:

```
f' x xs = x + length xs + head xs
```

From the *Prelude* we know that:

```
head :: [a] -> a -- (3)
```

It follows via (3) and (1) that:

```
head xs :: Int
```

And by applying (3) again it follows that

```
xs :: [Int]
```

So the final type of `f'` is:

```
f :: Int -> [Int] -> Int
```