Language Engineering | Lecture 10

A simple parser gives back the next item in the input stream.

REMEMBER
Parser x ≅ String → [(String, x)]

```
item :: Parser Char
~~item = Parser~~
item = Parser (λts → case ts of
        []     →  []
        (x:xs) →  [(xs, x)]  )
```

To capture the notion of sequencing ie. running one parser and then another, we will use monads.

```
class Monad m where
    return  ::  a → m a
    (>>=)   ::  m a → (a → m b) → m b
```

↖ pronounced "bind". Think of this as being a powerful ";".

To understand return for parsers, we are looking for something of type

a → Parser a

So we will use "produce", since it has the right type.

For bind, we specialise to:

$$(\gg\!=) :: \text{Parser } a \to (a \to \text{Parser } b) \to \text{Parser } b$$

$$\text{Parser } \underbrace{px}_{\text{String} \to [(\text{String},a)]} \gg\!= f = \text{Parser}$$

$$\text{Parser } (\lambda \overset{\text{String}}{ts} \to \text{concat } [\text{parse } \overset{\text{Parser b}}{(f\ x)}\ ts' \mid (\underbrace{ts'}_{\text{String}}, \underbrace{x}_{a}) \leftarrow \underbrace{px\ ts}_{[(\text{String},a)]}\ ])$$

So, to sum up:

```
(>>=) :: Parser a → (a → Parser b) → Parser b
Parser px >>= f = Parser (λts →
   concat [ parse (f x) ts' | (ts',x) ← px ts])
```

The notation:

$$[\ f\ x \mid x \leftarrow xs\ ]$$

means we are building a list of values f x, one for each value x in xs.

In other words;

$$\text{map } f \; xs = [f \; x \mid x \leftarrow xs]$$

Also remember that concat :: [[a]] → [a] simply flattens a list of lists into a single list

<u>Example.</u>

$$\underbrace{\text{item}}_{\text{Parser Char}} \gg\!= \underbrace{\text{produce}}_{\text{Char} \to \text{Parser Char}} \quad :: \text{Parser Char}$$

parse (item >>= produce) "hello"

= {...}

[("ello", 'h')]