

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

item :: Parser Char

~~item :: Parser~~

item = Parser ( $\lambda ts \rightarrow$  cases of

$[] \rightarrow []$

$(x:xs) \rightarrow [(xs, x)]$  )

REMEMBER

Parser  $x \cong \text{String} \rightarrow$

$[(\text{String}, x)]$

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

class Monad m where

return ::  $a \rightarrow m a$

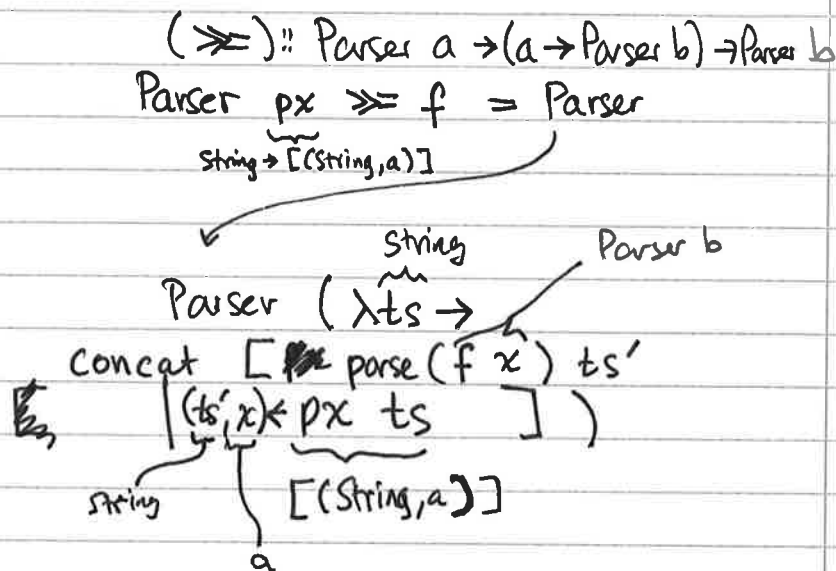
( $\gg$ ) ::  $m a \rightarrow (a \rightarrow m b) \rightarrow m b$

$\nwarrow$  pronounced "bind". Think of this as being a powerful " $;$ ".

To understand return for parsers, we are looking for something of type  $a \rightarrow \text{Parser } a$

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

For bind, we specialise to:



So, to sum up:

$$\begin{aligned}
 & (\gg) :: \text{Parser } a \rightarrow (a \rightarrow \text{Parser } b) \rightarrow \text{Parser } b \\
 & \text{Parser } px \gg f = \text{Parser } (\lambda ts \rightarrow \\
 & \quad \text{concat} [ \text{parse } (f \ x) \ ts' \mid (ts', x) \leftarrow px \ ts ])
 \end{aligned}$$

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;

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

Also remember that  $\text{concat} :: [[a]] \rightarrow [a]$   
simply flattens a list of lists into a single list

Example.

item  $\gg$  produce  $:: \text{Parser Char}$

$\text{Parser Char} \quad \text{Char} \rightarrow \text{Parser Char}$

$\text{parse (item } \gg \text{ produce) "hello"}$

$= \{ \dots \}$

$[(\text{"ello"}, 'h')]$