Parser

Monadic parsing approach - generates recursive descend parser

Parser a = String -> [(a, String)]

It takes String and produces a list of results, empty list = parsing failure

(+++) :: Parser a -> Parser a -> Parser a

Produces a parser that firstly tries to parse string with first parser and than with second

(many) :: Parser a -> Parser [a]

Takes a parser and produces a parser that can parse list of something

And other combinators

```
term :: Parser Term
term = abstraction +++ application +++ var
```

Where **abstraction**; **application**; **var** are corresponding parsers

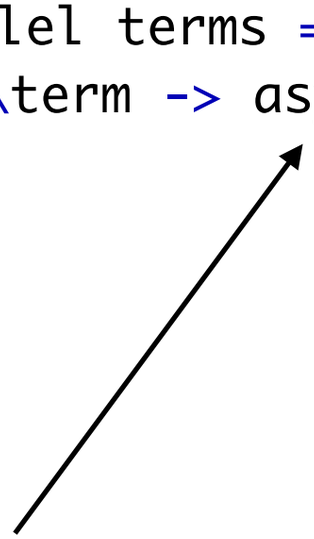
9

Compute in parallel

(…(**v**)**t**…)**t**n - **v** is a lambda-variable and **t** is a term

Could be thought as: [t1, t2, …] we can compute terms in parallel

```haskell
computeParallel :: [Term] -> IO [CTerm]
computeParallel terms =
  traverse (\term -> async $ krivineMachine term)
```



Spawns a thread that computes term

Parser

Monadic parsing approach - generates recursive descend parser

Parser a = String -> [(a, String)]
It takes String and produces a list of results, empty list = parsing failure
(+++) :: Parser a -> Parser a -> Parser a
Produces a parser that firstly tries to parse string with first parser and than with second
(many) :: Parser a -> Parser [a]
Takes a parser and produces a parser that can parse list of something
And other combinators

```
term :: Parser Term
term = abstraction +++ application +++ var
```

Where **abstraction**; **application**; **var** are corresponding parsers