

## Parser Combinators

## Today

What is a parser in Haskell

Parser combinators: build your own parser

- How to use them
- How they are defined

2

## Parsers for fixed chars and strings

```
token :: Eq a => a -> Parser a a

token '(' "(a+b)" =
  Ok('(', "a+b)")

tokenList :: Eq a => [a] -> Parser a [a]

tokenList "begin" "begin;end" =
  OK("begin", ";end")
```

3

## Identifiers

Grammar:

`ident ::= ('A' | ... | 'Z' | 'a' | ... | 'z')+`

Predefined in Haskell:

`isAlpha :: Char -> Bool`

4

## A do-it-yourself parser

```
parseIdent1 :: Parser Char String
parseIdent1 [] = Fail "Syntax error"
parseIdent1 cs = Ok(parseIdent cs)

parseIdent :: String -> (String,String)
parseIdent (c:cs) =
  if isAlpha c then let (id,r) = parseIdent cs
                       in (c:id,r)
  else ([],c:cs)
parseIdent [] = ([],[])
```

5

## Parsing categories of chars

```
spot :: (a -> Bool) -> Parser a a

spot isAlpha "xyz" = Ok('x', "yz")
spot isAlpha "++" = Fail "Syntax error"
```

6

## Iterating parsers

```
many :: Parser a b -> Parser a [b]

many (tokenList "->") "->->--" =
  Ok(["->", "->"], "--")
many (tokenList "->") "--" =
  Ok([], "--")
```

7

## Iterating parsers at least once

```
many1 :: Parser a b -> Parser a [b]

many1 (token '!') "!!!--" =
  Ok("!!!", "--")
many1 (token '!') "--" =
  Fail
```

8

## Parsing identifiers

```
parseIdent :: Parser Char String
parseIdent = many1(spot isAlpha)

parseIdent "abc--" =
  Ok("abc", "--")
parseIdent "--abc" =
  Fail ...
```

9

## Parsing integers

```
isDigit :: Char -> Bool

parseInt = many1(spot isDigit)

parseInt "12!" = Ok("12", "!")

How to produce Ok(12, "!") ??
```

10

## Transforming results

```
(>>>) ::
  Parser a b -> (b -> c) -> Parser a c

parseEvalInt = parseInt >>> evalInt

evalInt :: String -> Int

parseEvalInt "12!" = Ok(12, "!")
```

11

## Sequences and alternatives

```
(|||) :: Parser a b -> Parser a b ->
  Parser a b

(&&&) :: Parser a b -> Parser a c ->
  Parser a (b,c)

(spot isAlpha ||| spot isDigit) "lxy" =
  Ok('l', "xy")
(spot isAlpha &&& spot isDigit) "x12" =
  Ok(('x','1'), "2")
```

12

### Example: alphanumeric identifiers

Grammar: `ident2 ::= Alpha (Alpha | Digit)*`

```
parseIdent2 =  
  spot isAlpha &&&  
  many(spot isAlpha ||| spot isDigit)
```

```
parseIdent2 "x25z+" =  
  Ok(('x', "25z"), "+")
```

Better: `Ok("x25z", "+")`

13

### Alphanumeric identifiers (2)

```
parseIdent1 :: Parser Char String  
parseIdent1 = parseIdent2 >>>  
  (uncurry (:))
```

14

### Case study: parsing regular expressions

```
re ::= '0'  
  | String  
  | '(' re re ')'  
  | '(' re '|' re ')'  
  | '(' re ')**'
```

Auxiliary functions:

```
p1 -$ p2 = p1 &&& p2 >>> fst  
p -$ $ cs = p -$ tokenList cs  
p1 $- p2 = p1 &&& p2 >>> snd  
cs $$- p = tokenList cs $- p
```

15

### Building the abstract syntax tree

```
data RegExp a =  
  Epsilon  
  | Literal a  
  | Or (RegExp a) (RegExp a)  
  | Conc (RegExp a) (RegExp a)  
  | Repeat (RegExp a)
```

16