Monaadilised parserid

Süntaksanalüüs ja parserid

- Parseri ülesandeks on:
  - kontrollida kas sisendstring on süntaktiliselt korrektne;
  - konstrueerida sisendstringile vastav AST.
- Parserite soovitavad omadused:
  - BNF lähedane esitus;
  - parserite konstrueerimine olemasolevatest parseritest;
  - mittedetermineeritud grammatikate kasutamine;
  - kontekstist sõltuvate keelte äratundmine.

Monaadilised parserid

Parserite tüüp

$$\textbf{newtype}\ Parser\ a = P\ (String \to [(a, String)])$$
$$runP :: Parser\ a \to String \to [(a, String)]$$
$$runP\ (P\ p) = p$$

Parserite monaad

$$\begin{aligned}
&\textbf{instance}\ Monad\ Parser\ \textbf{where}\\
&\quad return\ a = P\ \$\ \backslash cs \to [(a, cs)]\\
&\quad p \gg\!= f \quad = P\ \$\ \backslash cs \to concat\ [runP\ (f\ a)\ cs'\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid (a, cs') \leftarrow runP\ p\ cs]
\end{aligned}$$

Monaadilised parserid

Primitiivparserid

```
instance MonadPlus Parser where
        mzero     = P $ \cs -> []
        mplus p q = P $ \cs -> runP p cs ++ runP q cs
(<|>)   :: Parser a -> Parser a -> Parser a
p <|> q = p `mplus` q
item    :: Parser Char
item    = P $ \cs -> [(head cs, tail cs) | not (null cs)]
```

Monaadilised parserid

Elementaarparserid

```
sat    :: (Char → Bool) → Parser Char
sat p  = do {c ← item; if p c then return c else mzero}
char   :: Char → Parser Char
char c = sat (c ==)
```

Näide

```
data Tree = Nil | Bin Tree Tree

parens :: Parser Tree
parens = do   char '('
              t1 ← parens
              char ')'
              t2 ← parens
              return (Bin t1 t2)
         <|> return Nil
```

Monaadilised parserid

Iteratsioon

$$
\begin{aligned}
&many &&:: Parser\ a \to Parser\ [a] \\
&many\ p &&= many1\ p <|> return\ [] \\
&many1 &&:: Parser\ a \to Parser\ [a] \\
&many1\ p &&= \mathbf{do}\ \{a \leftarrow p; as \leftarrow many\ p; return\ (a : as)\}
\end{aligned}
$$

Lihtsaid parsereid

Võtmesõnad

$$
\begin{aligned}
&\mathit{string} && :: \mathit{String} \to \mathit{Parser\ String} \\
&\mathit{string}\ "" && = \mathit{return}\ "" \\
&\mathit{string}\ (c : cs) && = \mathbf{do}\ \{\mathit{char}\ c; \mathit{string}\ cs; \mathit{return}\ (c : cs)\}
\end{aligned}
$$

Identifikaatorid

$$
\begin{aligned}
&\mathit{identifier} :: \mathit{Parser\ String} \\
&\mathit{identifier} = \mathbf{do}\ c \leftarrow \mathit{lower} \\
&\qquad\qquad\qquad\quad cs \leftarrow \mathit{many\ alphanum} \\
&\qquad\qquad\qquad\quad \mathit{return}\ (c : cs)
\end{aligned}
$$

Lihtsaid parsereid

Naturaalarvud

```
natural :: Parser Int
natural = do ds ← many1 digit
             return (foldl1 (\a b → 10 * a + b) ds)
digit   :: Parser Int
digit   = do c ← sat isDigit
             return (ord c − ord '0')
```

Täisarvud

```
integer :: Parser Int
integer =     do {char '-'; n ← natural; return (−n)}
          <|> natural
```

Lihtsaid parsereid

Reaalarvud

```
floating :: Parser Double
floating = do i ← integer
              f ← fraction <|> return 0
              return (fromIntegral i + f)

fraction :: Parser Double
fraction = do char '.'
              ds ← many1 digit
              return (foldr op 0 ds)
  where d `op` x = (x + fromIntegral d) / 10
```

Lihtsaid parsereid

Tühisümbolid

$$
\begin{aligned}
space &:: Parser\ String \\
space &= many\ (sat\ isSpace) \\
token &:: Parser\ a \to Parser\ a \\
token\ p &= \mathbf{do}\ \{a \leftarrow p; space; return\ a\} \\
keyc\ c &= token\ (char\ c) \\
keyw\ cs &= token\ (string\ cs) \\
ident &= token\ identifier \\
nat &= token\ natural \\
int &= token\ integer \\
float &= token\ floating
\end{aligned}
$$

Parserite transformaatorid

Sulud

$$pack :: Parser\ a \to Parser\ b \to Parser\ c \to Parser\ b$$
$$pack\ s1\ p\ s2 = \mathbf{do}\ \{s1; x \leftarrow p; s2; return\ x\}$$

Näide

$$paren\ p = pack\ (keyc\ '(')\ p\ (keyc\ ')')$$
$$brack\ p = pack\ (keyc\ '[')\ p\ (keyc\ ']')$$
$$block\ p\ = pack\ (keyw\ \texttt{"begin"})\ p\ (keyw\ \texttt{"end"})$$

Parserite transformaatorid

Eraldajaga jadad

$$
\begin{aligned}
&sepby &&:: Parser\ a \to Parser\ b \to Parser\ [a] \\
&p\ `sepby`\ sep &&= (p\ `sepby1`\ sep) <|> return\ [] \\
&sepby1 &&:: Parser\ a \to Parser\ b \to Parser\ [a] \\
&p\ `sepby1`\ sep &&= \mathbf{do}\ a \leftarrow p \\
&&&\qquad as \leftarrow many\ (sep \gg p) \\
&&&\qquad return\ (a : as)
\end{aligned}
$$

Näide

$$
\begin{aligned}
&commaList\ p = sepby\ p\ (keyw\ ",") \\
&semicList\ p\ \ = sepby\ p\ (keyw\ ";")
\end{aligned}
$$

Aritmeetilised avaldised

Grammatika

$$
\begin{aligned}
expr = {} & int \mid expr + expr \mid expr - expr \\
\mid {} & expr * expr \mid expr \ / \ expr \mid expr \ \hat{} \ expr \\
\mid {} & (expr)
\end{aligned}
$$

Abstraktne süntaksipuu

$$
\begin{aligned}
\mathbf{data}\ Expr = {} & Con\ Int \\
\mid {} & Expr :+: Expr \\
\mid {} & Expr :-: Expr \\
\mid {} & Expr :*: Expr \\
\mid {} & Expr :/: Expr \\
\mid {} & Expr :\hat{}: Expr
\end{aligned}
$$

Aritmeetilised avaldised

Parser ver. 0

$$
\begin{aligned}
expr0 = \ & \mathbf{do}\,\{e0 \leftarrow expr0; keyc\ \text{'+'}; \\
& \qquad e1 \leftarrow expr0; return\ (e0\ :+:\ e1)\} \\
<|> \ & \mathbf{do}\,\{e0 \leftarrow expr0; keyc\ \text{'-'}; \\
& \qquad e1 \leftarrow expr0; return\ (e0\ :-:\ e1)\} \\
<|> \ & \qquad \ldots \\
<|> \ & \mathbf{do}\,\{e0 \leftarrow expr0; keyc\ \text{'\^{}'}; \\
& \qquad e1 \leftarrow expr0; return\ (e0\ :\hat{}:\ e1)\} \\
<|> \ & \mathbf{do}\,\{i \leftarrow int; return\ (Con\ i)\} \\
<|> \ & paren\ expr0
\end{aligned}
$$

NB!

Ei tööta kuna grammatika on vasakrekursiivne!!

Aritmeetilised avaldised

Parser ver. 1

$$
\begin{aligned}
expr1 &= \mathbf{do} \quad a \leftarrow atom1 \\
&\qquad\quad op \leftarrow oper1 \\
&\qquad\quad e \leftarrow expr1 \\
&\qquad\quad return\ (a\ `op`\ e) \\
&\quad <|> atom1 \\
oper1 &= \quad (keyc\ \text{'+'} \gg return\ (:+:)) \\
&\quad <|> (keyc\ \text{'-'} \gg return\ (:-:)) \\
&\quad <|> (keyc\ \text{'*'} \gg return\ (:*:)) \\
&\quad <|> (keyc\ \text{'/'} \gg return\ (:/:)) \\
&\quad <|> (keyc\ \text{'\^{}'} \gg return\ (:\hat{}:)) \\
atom1 &= \mathbf{do} \quad \{i \leftarrow int; return\ (Con\ i)\} \\
&\quad <|> paren\ expr1
\end{aligned}
$$

Parserite transformaatorid

Eraldajaga jadad

```
chainl :: Parser a → Parser (a → a → a) → Parser a
chainl p s = do
        x  ← p
        ys ← many (do {op ← s; y ← p; return (op, y)})
        return (foldl (\a (op, y) → a `op` y) x ys)
chainr :: Parser a → Parser (a → a → a) → Parser a
chainr p s = do
        ys ← many (do {y ← p; op ← s; return (y, op)})
        x  ← p
        return (foldr (\(y, op) b → y `op` b) x ys)
```

Aritmeetilised avaldised

Parser ver. 2

```
expr2  :: Parser Expr
expr2  = chainl term2  (     (keyc '+' ≫ return (:+:))
                        <|> (keyc '-' ≫ return (:-:)))
term2  :: Parser Expr
term2  = chainl fact2  (     (keyc '*' ≫ return (:*:))
                        <|> (keyc '/' ≫ return (:/:)))
fact2  :: Parser Expr
fact2  = chainr atom2 (keyc '^' ≫ return (:^:))
atom2 :: Parser Expr
atom2 = do  {i ← int; return (Con i)}
        <|> paren expr2
```

Aritmeetilised avaldised

Aritmeetiliste avaldiste väärtustaja

```
expr3  :: Parser Int
expr3  = chainl term3   (   (keyc '+' >> return (+))
                        <|> (keyc '-' >> return (-)))
term3  :: Parser Int
term3  = chainl fact3   (   (keyc '*' >> return (*))
                        <|> (keyc '/' >> return div))
fact3  :: Parser Int
fact3  = chainr atom3 (keyc '^' >> return (^))
atom3 :: Parser Int
atom3 = int <|> paren expr3
```

Monaadilised parserid

Efektiivsust parandavaid kombinaatoreid

```
first      :: Parser a → Parser a
first p    = P $ \cs → case runP p cs of
                         []       → []
                         (x : xs) → [x]
(<|>)      :: Parser a → Parser a → Parser a
p <|> q = first (p `mplus` q)
```

Kogu sisendi parsimine

```
parse :: Parser a → String → a
parse p cs = case runP (first (space ≫ p)) cs of
                [(x, "")] → x
                _         → error "Parse error"
```