Парсер-комбинаторы и Parser Expression Grammars

Автор: Екатерина Вербицкая

Санкт-Петербургский государственный университет
Математико-механический факультет

8 декабря 2015г.

Долг: зачем нужен RNGLR

Долг: зачем нужен RNGLR

Для обработки неоднозначных грамматик. Часто нельзя выбрать заранее, какая из альтернатив "верна".

Долг: зачем нужен RNGLR

Для обработки неоднозначных грамматик. Часто нельзя выбрать заранее, какая из альтернатив "верна".

$$
\begin{aligned}
S &\rightarrow \mathit{if}\ b\ \mathit{then}\ S\ \mathit{else}\ S \\
S &\rightarrow \mathit{if}\ b\ \mathit{then}\ S \\
S &\rightarrow a
\end{aligned}
$$

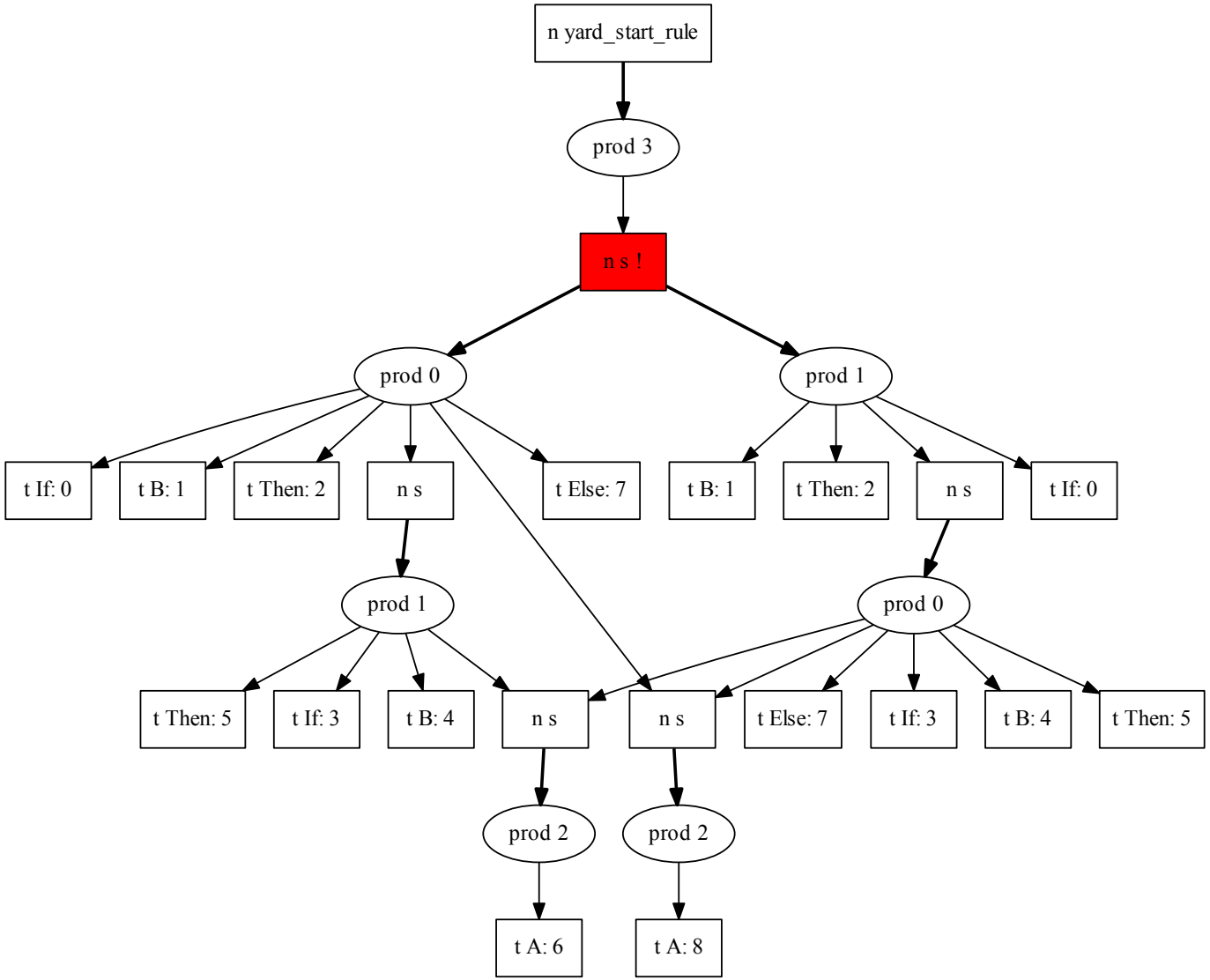
Долг: зачем нужен RNGLR

Для обработки неоднозначных грамматик. Часто нельзя выбрать заранее, какая из альтернатив "верна".

$$
\begin{aligned}
S &\rightarrow \mathit{if}\ b\ \mathit{then}\ S\ \mathit{else}\ S \\
S &\rightarrow \mathit{if}\ b\ \mathit{then}\ S \\
S &\rightarrow a
\end{aligned}
$$

2 дерева вывода для цепочки $\mathit{if}\ b\ \mathit{then}\ \mathit{if}\ b\ \mathit{then}\ a\ \mathit{else}\ a$

Лес разбора: объединяет 2 дерева



Парсер-комбинаторы: общая идея

- Написать примитивные парсеры (например, для обработки отдельных символов)
- Написать комбинаторы для составления из примитивных парсеров более сложных
  - Последовательности
  - Альтернативы
  - Повторение
  - Опциональный парсер
  - ...
- С использованием этого инструментария описать необходимый парсер

Парсер-комбинаторы: тип парсера

$$Parser \quad = \quad String \rightarrow Tree$$

Парсер-комбинаторы: тип парсера

$$
\begin{aligned}
Parser &= String \to Tree \\
Parser &= String \to (Tree, String)
\end{aligned}
$$

Парсер-комбинаторы: тип парсера

$$
\begin{aligned}
Parser &= String \to Tree \\
Parser &= String \to (Tree, String) \\
Parser &= String \to [(Tree, String)]
\end{aligned}
$$

Парсер-комбинаторы: тип парсера

$$
\begin{aligned}
Parser &= String \to Tree \\
Parser &= String \to (Tree, String) \\
Parser &= String \to [(Tree, String)]
\end{aligned}
$$

$$
Parser\ a = String \to [(a, String)]
$$

Примитивные парсеры

```
result :: a -> Parser a
result v = \inp -> [(v,inp)]

zero :: Parser a
zero = \inp -> []

item :: Parser Char
item = \inp -> case inp of
                 [] -> []
                 (x:xs) -> [(x,xs)]
```

Парсер-комбинаторы

```
bind :: Parser a -> (a -> Parser b) -> Parser b
p `bind` f = \inp -> concat [f v inp' | (v,inp') <- p inp]

p `seq` q = p `bind` \x ->
            q `bind` \y ->
            result (x,y)

sat :: (Char -> Bool) -> Parser Char
sat p = item `bind` \x ->
        if p x then result x else zero

plus :: Parser a -> Parser a -> Parser a
p `plus` q = \inp -> (p inp ++ q inp)
```

Маленькие полезные парсеры

```
char :: Char -> Parser Char
char x = sat (\y -> x == y)

digit :: Parser Char
digit = sat (\x -> '0' <= x && x <= '9')

lower :: Parser Char
lower = sat (\x -> 'a' <= x && x <= 'z')

upper :: Parser Char
upper = sat (\x -> 'A' <= x && x <= 'Z')
```

Менее маленькие полезные парсеры

```
letter :: Parser Char
letter = lower `plus` upper

alphanum :: Parser Char
alphanum = letter `plus` digit

word :: Parser String
word = neWord `plus` result ""
       where
         neWord = letter `bind` \x ->
                  word `bind` \xs ->
                  result (x:xs)

word "Yes!" = [("Yes","!"), ("Ye","s!"),
               ("Y","es!"), ("","Yes!")]
```

Монадические парсер-комбинаторы

```
class Monad m where
  result :: a -> m a
  bind :: m a -> (a -> m b) -> m b

instance Monad Parser where
  -- result :: a -> Parser a
  result v = \inp -> [(v,inp)]

  -- bind :: Parser a -> (a -> Parser b) -> Parser b
  p `bind` f = \inp -> concat [f v out | (v,out) <- p inp]
```

Монадические парсер-комбинаторы

```
class Monad m => MonadOPlus m where
  zero :: m a
  (++) :: m a -> m a -> m a

instance MonadOPlus Parser where
  -- zero :: Parser a
  zero = \inp -> []

  -- (++) :: Parser a -> Parser a -> Parser a
  p ++ q = \inp -> (p inp ++ q inp)
```

Другая форма записи

```
p1 `bind` \x1 ->
p2 `bind` \x2 ->
...
pn `bind` \xn ->
result (f x1 x2 ... xn)
```

```
[ f x1 x2 ... xn | x1 <- p1
                 , x2 <- p2
                 , ...
                 , xn <- pn ]
```

```
string :: String -> Parser String
string "" = [""]
string (x:xs) = [x:xs | _ <- char x, _ <- string xs]
```

Монадические парсер-комбинаторы

```
sat :: (Char -> Bool) -> Parser Char
sat p = [x | x <- item, p x]

many :: Parser a -> Parser [a]
many p = [x:xs | x <- p, xs <- many p] ++ [[]]
```

```
ident :: Parser String
ident = [x:xs | x <- lower, xs <- many alphanum]
```

Примеры парсеров

```
many1 :: Parser a -> Parser [a]
many1 p = [x:xs | x <- p, xs <- many p]

nat :: Parser Int
nat = [eval xs | xs <- many1 digit]
      where
        eval xs = foldl1 op [ord x - ord '0' | x <- xs]
        m `op` n = 10*m + n

int :: Parser Int
int = [-n | _ <- char '-', n <- nat] ++ nat

int :: Parser Int
int = [f n | f <- op, n <- nat]
      where
        op = [negate | _ <- char '-'] ++ [id]
```

Пример: список чисел

```
ints :: Parser [Int]
ints = [n:ns | _ <- char '['
             , n <- int
             , ns <- many [x | _ <- char ',', x <- int]
             , _ <- char ']']


sepby1 :: Parser a -> Parser b -> Parser [a]
p `sepby1` sep = [x:xs | x <- p
                       , xs <- many [y | _ <- sep, y <- p]]

ints = [ns | _ <- char '['
           , ns <- int `sepby1` char ','
           , _ <- char ']']
```

Список чисел: еще короче

```
bracket :: Parser a -> Parser b -> Parser c -> Parser b
bracket open p close = [x | _ <- open, x <- p, _ <- close]

ints = bracket (char '[')
               (int `sepby1` char ',')
               (char ']')
```

Пример: арифметические выражения

$$
\begin{array}{rcl}
\mathit{expr} & ::= & \mathit{expr}\ \mathit{addop}\ \mathit{factor} \mid \mathit{factor} \\
\mathit{addop} & ::= & + \mid - \\
\mathit{factor} & ::= & \mathit{nat} \mid (\ \mathit{expr}\ )
\end{array}
$$

```
expr :: Parser Int
addop :: Parser (Int -> Int -> Int)
factor :: Parser Int

expr = [f x y | x <- expr
              , f <- addop
              , y <- factor] ++ factor

addop = [(+) | _ <- char '+'] ++ [(-) | _ <- char '-']

factor = nat ++ bracket (char '(') expr (char ')')
```

Избавляемся от левой рекурсии

```
expr = [foldl (\x (f,y) -> f x y) x fys
           | x <- factor
           , fys <- many [(f,y) | f <- addop, y <- factor]]

addop = [(+) | _ <- char '+'] ++ [(-) | _ <- char '-']

factor = nat ++ bracket (char '(') expr (char ')')
```

Преимущества монадических парсер-комбинаторов

- Простота
- Гибкость
- Выразительность
- Возможность откатываться (backtracking)
- Лексический анализ не нужно выделять в отдельный шаг
- Можно считать семантику во время синтаксического анализа

Недостатки монадических парсер-комбинаторов

- Если использовать неграмотно, можно получить непредсказуемое время работы и легко исчерпать всю доступную память
  - Наличие общих префиксов у нескольких правил. Решение: факторизация грамматики
  - Вычисление промежуточных результатов, где не было надо. Решение: использование ленивости (например, $p +\!+\!+\, q = \mathit{first}(p +\!+ q)$)

Parser Expression Grammars

- PEG G — четверка (V, T, P, p_S), где
  - V — конечное множество нетерминалов
  - T — алфавит (конечное множество терминалов)
  - P — функция из V в выражения (parser expression)
  - p_S — стартовое выражение

- Parser expression
  - Пустая строка ε
  - Терминал a
  - Нетерминал A
  - Последовательность $p_1 p_2$, где p_1, p_2 — parser expression
  - Упорядоченный выбор p_1 / p_2, где p_1, p_2 — parser expression
  - 0-или-больше p^*, где p — parser expression
  - Предикат Не $!p$, где p — parser expression

Отношение PEG

$$G[p]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y, x')$$

- Выражение p парсит строку xy, съедая x и оставляя y, возвращая x' как результат
- Если справа *fail*, значит, распарсить строку не удалось

Операционная семантика PEG: пустая строка

$$\frac{}{G[\varepsilon]\ x \overset{\text{PEG}}{\rightsquigarrow} (x, \varepsilon)}$$

Операционная семантика PEG: терминал

$$\frac{}{G[a]\ ax \overset{\text{PEG}}{\rightsquigarrow} (x,\ a)}$$

Операционная семантика PEG: терминал

$$\frac{}{G[a]\ ax \overset{\text{PEG}}{\rightsquigarrow} (x,\ a)}$$

$$\frac{}{G[b]\ ax \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}\ ,\ b \neq a$$

Операционная семантика PEG: терминал

$$\frac{}{G[a]\ ax \overset{\text{PEG}}{\rightsquigarrow} (x,\ a)}$$

$$\frac{}{G[b]\ ax \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}\ ,\ b \neq a$$

$$\frac{}{G[a]\ \varepsilon \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Операционная семантика PEG: переменная

$$\frac{G[P(A)]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}{G[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, A[x'])}$$

Операционная семантика PEG: переменная

$$\frac{G[P(A)]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\ x')}{G[A]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\ A[x'])}$$

$$\frac{G[P(A)]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[A]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Операционная семантика PEG: последовательность

$$\frac{G[p_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (yz,\, x') \qquad G[p_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} (z,\, y')}{G[p_1\, p_2]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (z,\, x'y')}$$

Операционная семантика PEG: последовательность

$$\frac{G[p_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (yz,\, x') \qquad G[p_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} (z,\, y')}{G[p_1\, p_2]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (z,\, x'y')}$$

$$\frac{G[p_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x') \qquad G[p_2]\ y \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[p_1\, p_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Операционная семантика PEG: последовательность

$$\frac{G[p_1]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (yz,\, x') \qquad G[p_2]\ yz \overset{\text{PEG}}{\rightsquigarrow} (z,\, y')}{G[p_1\, p_2]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (z,\, x'y')}$$

$$\frac{G[p_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x') \qquad G[p_2]\ y \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[p_1\, p_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

$$\frac{G[p_1]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[p_1\, p_2]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Операционная семантика PEG: выбор

$$\frac{G[p_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}{G[p_1 \,/\, p_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}$$

Операционная семантика PEG: выбор

$$\frac{G[p_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}{G[p_1 \,/\, p_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}$$

$$\frac{G[p_1]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail} \qquad G[p_2]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[p_1 \,/\, p_2]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Операционная семантика PEG: выбор

$$\frac{G[p_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}{G[p_1 \,/\, p_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}$$

$$\frac{G[p_1]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail} \qquad G[p_2]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[p_1 \,/\, p_2]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

$$\frac{G[p_1]\ xy \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail} \qquad G[p_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}{G[p_1 \,/\, p_2]\ xy \overset{\text{PEG}}{\rightsquigarrow} (y,\, x')}$$

Операционная семантика PEG: предикат не

$$\frac{G[p]\ \ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[!p]\ \ x \overset{\text{PEG}}{\rightsquigarrow} (x, \varepsilon)}$$

Операционная семантика PEG: предикат не

$$\frac{G[p]\ \ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[!p]\ \ x \overset{\text{PEG}}{\rightsquigarrow} (x, \varepsilon)}$$

$$\frac{G[p]\ \ xy \overset{\text{PEG}}{\rightsquigarrow} (y, x')}{G[!p]\ \ xy \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Операционная семантика PEG: повторение

$$\frac{G[p]\ \ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[p^*]\ \ x \overset{\text{PEG}}{\rightsquigarrow} (x, \varepsilon)}$$

Операционная семантика PEG: повторение

$$\frac{G[p]\ x \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[p^*]\ x \overset{\text{PEG}}{\rightsquigarrow} (x, \varepsilon)}$$

$$\frac{G[p]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (yz, x') \qquad G[p^*]\ yz \overset{\text{PEG}}{\rightsquigarrow} (z, y')}{G[p^*]\ xyz \overset{\text{PEG}}{\rightsquigarrow} (z, x'y')}$$

Борьба с левой рекурсией: ограниченная левая рекурсия

- A^n имеет не более n леворекурсивных вызовов A, A^0 всегда завершается ошибкой

$$
\begin{array}{lcl}
E^0 & ::= & \mathit{fail} \\
E^1 & ::= & E^0 + n/n = \bot + n/n = n \\
E^2 & ::= & E^1 + n/n = n + n/n \\
E^3 & ::= & E^2 + n/n = (n + n/n) + n/n \\
 & \dots & \\
E^n & ::= & E^{n-1} + n/n
\end{array}
$$

Пример: разбираем с разными границами строку $n + n$

$$
\begin{aligned}
E^0(n + n) &= \mathit{fail} \\
E^1(n + n) &= (\bot + n/n)(n + n) = (n, +n) \\
E^2(n + n) &= (n + n/n)(n + n) = (n + n, \varepsilon)
\end{aligned}
$$

$$
E^3(n + n) = ((n + n/n) + n/n)(n + n) = (n, +n)
$$

При разборе входа с помощью E^3, во внутренней альтернативе $(n + n/n)$ первый вариант срабатывает (и поэтому второй вариант не будет рассматриваться вообще!). После этого пытаемся разобрать оставшийся ε с помощью оставшегося $+n$ и не можем. Остается только вторая альтернатива, поэтому матчится только один n.

Борьба с левой рекурсией

- Ищем значение n для каждого леворекурсивного нетерминала
- Подбирается такая граница, чтобы префикс, обработанный правилом, имел максимальную длину
- Промежуточные значения сохраняются в табличку L
  - $L[(A, x) \to X](B, y) = L(B, y)$, если $B \neq A$ или $y \neq x$
  - $L[(A, x) \to X](A, x) = X$

Обработка леворекурсивного нетерминала

$$\frac{(A, xyz) \notin \mathcal{L} \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto \texttt{fail}] \overset{\text{PEG}}{\rightsquigarrow} (yz, x') \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto (yz, x')] \overset{\text{INC}}{\rightsquigarrow} (z, (xy)')}{G[A]\ xyz\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} (z, A[(xy)'])}$$

Обработка леворекурсивного нетерминала

$$\frac{(A, xyz) \notin \mathcal{L} \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto \texttt{fail}] \overset{\text{PEG}}{\rightsquigarrow} (yz, x') \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto (yz, x')] \overset{\text{INC}}{\rightsquigarrow} (z, (xy)')}{G[A]\ xyz\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} (z, A[(xy)'])}$$

$$\frac{(A, x) \notin \mathcal{L} \quad G[P(A)]\ x\ \mathcal{L}[(A, x) \mapsto \texttt{fail}] \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[A]\ x\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Обработка леворекурсивного нетерминала

$$\frac{(A, xyz) \notin \mathcal{L} \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto \texttt{fail}] \overset{\text{PEG}}{\rightsquigarrow} (yz, x') \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto (yz, x')] \overset{\text{INC}}{\rightsquigarrow} (z, (xy)')}{G[A]\ xyz\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} (z, A[(xy)'])}$$

$$\frac{(A, x) \notin \mathcal{L} \quad G[P(A)]\ x\ \mathcal{L}[(A, x) \mapsto \texttt{fail}] \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[A]\ x\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

$$\frac{\mathcal{L}(A, xy) = \texttt{fail}}{G[A]\ xy\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

Обработка леворекурсивного нетерминала

$$\frac{(A, xyz) \notin \mathcal{L} \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto \texttt{fail}] \overset{\text{PEG}}{\rightsquigarrow} (yz, x') \quad G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto (yz, x')] \overset{\text{INC}}{\rightsquigarrow} (z, (xy)')}{G[A]\ xyz\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} (z, A[(xy)'])}$$

$$\frac{(A, x) \notin \mathcal{L} \quad G[P(A)]\ x\ \mathcal{L}[(A, x) \mapsto \texttt{fail}] \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[A]\ x\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

$$\frac{\mathcal{L}(A, xy) = \texttt{fail}}{G[A]\ xy\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}$$

$$\frac{\mathcal{L}(A, xy) = (y, x')}{G[A]\ xy\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} (y, A[x'])}$$

Семантика отношения INC

$$\frac{G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (yzw, x')] \overset{\text{PEG}}{\rightsquigarrow} (zw, (xy)') \quad G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (zw, (xy)')] \overset{\text{INC}}{\rightsquigarrow} (w, (xyz)')}{G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (yzw, x')] \overset{\text{INC}}{\rightsquigarrow} (w, (xyz)')}, \text{where } y \neq \varepsilon$$

Семантика отношения INC

$$\frac{G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (yzw, x')] \overset{\text{PEG}}{\rightsquigarrow} (zw, (xy)') \quad G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (zw, (xy)')] \overset{\text{INC}}{\rightsquigarrow} (w, (xyz)')}{G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (yzw, x')] \overset{\text{INC}}{\rightsquigarrow} (w, (xyz)')}, \text{where } y \neq \varepsilon$$

$$\frac{G[P(A)]\ x\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[P(A)]\ x\ \mathcal{L} \overset{\text{INC}}{\rightsquigarrow} \mathcal{L}(A, x)}$$

Семантика отношения INC

$$\frac{G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (yzw, x')] \overset{\text{PEG}}{\rightsquigarrow} (zw, (xy)') \quad G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (zw, (xy)')] \overset{\text{INC}}{\rightsquigarrow} (w, (xyz)')}{G[P(A)]\ xyzw\ \mathcal{L}[(A, xyzw) \mapsto (yzw, x')] \overset{\text{INC}}{\rightsquigarrow} (w, (xyz)')}, \text{where } y \neq \varepsilon$$

$$\frac{G[P(A)]\ x\ \mathcal{L} \overset{\text{PEG}}{\rightsquigarrow} \texttt{fail}}{G[P(A)]\ x\ \mathcal{L} \overset{\text{INC}}{\rightsquigarrow} \mathcal{L}(A, x)}$$

$$\frac{G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto (z, (xy)')] \overset{\text{PEG}}{\rightsquigarrow} (yz, x')}{G[P(A)]\ xyz\ \mathcal{L}[(A, xyz) \mapsto (z, (xy)')] \overset{\text{INC}}{\rightsquigarrow} (z, (xy)')}$$

Литература

- Monadic parser combinators:
  `http://www.cs.nott.ac.uk/~pszgmh/monparsing.pdf`
- Left recursion in Parsing Expression Grammars:
  `http://arxiv.org/pdf/1207.0443.pdf`
- Парсер-комбинаторная библиотека на F#:
  `https://github.com/pragmatrix/ScanRat`