Парсер-комбинаторы и левая рекурсия

продолжение

Автор: Екатерина Вербицкая

Лаборатория языковых инструментов JetBrains
Санкт-Петербургский государственный университет
Математико-механический факультет

9 ноября 2015г.

В предыдущей серии

Парсер-комбинатор: функция высшего порядка, которая конструирует парсер из набора других парсеров.

- Примитивные парсеры
- Комбинаторы для последовательности, выбора, повторения и т.д.

Монадические парсер-комбинаторы

```
-- 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]

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

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

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

- Если использовать неграмотно, можно получить непредсказуемое время работы и легко исчерпать всю доступную память
  - Есть набор практик, которые позволяют избежать проблем

Левая рекурсия в монадических парсер-комбинаторах

В явном виде приводит к зацикливанию. Есть комбинатор для частного случая:

```
chainl1 :: Parser a -> Parser (a -> a -> a) -> Parser a
p `chainl1` op = [foldl (\x (f,y) -> f x y) x fys
                  | x <- p
                  , fys <- many [(f,y) | f <- op, y <- p]]

expr = factor `chainl1` addop
addop = [(+) | _ <- char '+'] ++ [(-) | _ <- char '-']
factor = nat ++ bracket (char '(') expr (char ')')
```

Что можно парсить монадическими парсер-комбинаторами?

- Все контекстно-свободные + некоторое подмножество контекстно-зависимых
- Parsec: "It can parse context-sensitive, infinite look-ahead grammars"

Packrat-парсер

- Unlimited lookahed + backtracking
- Гарантирует линейное время работы на LL(k) и LR(k) грамматиках
- Использует запоминание

Борьба с левой рекурсией: Warth et al

- Первый раз, когда наткнулись на левую рекурсию, запомнить в таблице ошибку анализа — посадить семечко
- Откатиться во входном потоке в начало правила и применить правило еще раз — прорастить семечко

- Определение левой рекурсии:
  - В таблице наряду с вычисленным значением храним галочку: леворекурсивно ли это правило
  - Перед применением правила, выставляем галочку в false
  - Если в таблице нет вычисленного значения, значит попали в левую рекурсию: выставляем галочку в true, садим семечко

Warth et al: неявная рекурсия

- Храним стек вызовов правил
- При проращивании семечка проращиваем все вовлеченные в левую рекурсию правила

Проблемы с подходом Warth et al

Проблемы с ассоциативностью, если есть и левая, и правая рекурсия

```
E :: E - E | Num
```

"1 – 2 – 3"разберется как "1 – (2 – (3))". Однако если переписать грамматику следующим образом, ассоциативность будет правильной

```
E :: E - E (- E)? | Num
```

Частичное решение: ограничить правую рекурсию максимум одним шагом в глубину

Parser Expression Grammars

- Похоже на EBNF-форму, но выбор упорядочен
- Нет — неоднозначностям, да — правильному приоритету!
- Любая PEG может быть проанализирована packrat-парсером
- Класс принимаемых языков: не известен
  - Все LR(k), LL(k) языки
  - Какие-то еще КС-языки
  - Но! Нельзя разобрать $S \rightarrow x\,S\,x\,|x$
  - Зато! Можно описать не-КС язык $a^n\,b^n\,c^n$

```
S ::= &(A c) a + B !(a/b/c)
A ::= a A? b
B ::= b B? c
```

Литература

- Monadic Parser Combinators:
  http://www.cs.nott.ac.uk/~pszgmh/monparsing.pdf
- Packrat Parsing:
  http://www.brynosaurus.com/pub/lang/packrat-icfp02.pdf
- Packrat Parsers Can Support Left Recursion:
  http://www.vpri.org/pdf/tr2007002_packrat.pdf
- Проблемы с предыдущим подходом (Direct Left-Recursive Parsing Expression Grammars):
  http://tratt.net/laurie/research/pubs/papers/tratt_direct_left_recursive_parsing_expression_grammars.pdf