Ostap: синтаксическое расширение OCaml для создания парсер-комбинаторов с поддержкой левой рекурсии

Автор: Екатерина Вербицкая

Санкт-Петербургский государственный университет
Лаборатория языковых инструментов JetBrains

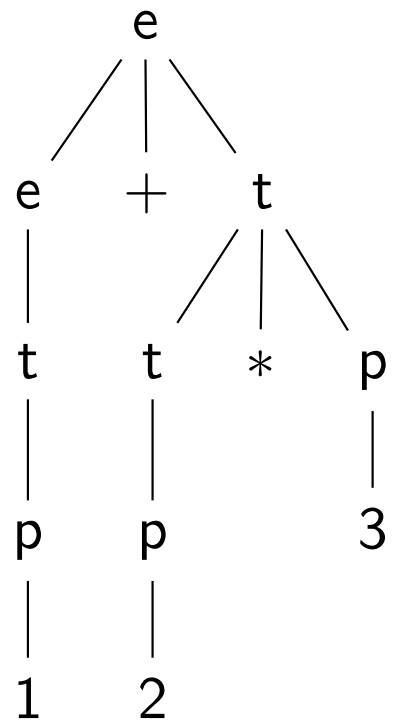
05 апреля 2017

Синтаксический анализ

Сопоставление последовательности лексем с грамматикой языка

$$1 + 2 * 3$$

$$\begin{array}{lcl} e & : & e + t \mid t \\ t & : & t * p \mid p \\ p & : & 0 \mid 1 \mid 2 \mid \ldots \mid 9 \end{array}$$



Парсер-комбинаторы

- Подход к реализации синтаксического анализа в парадигме функционального программирования
- Реализуют нисходящий разбор
- Синтаксический анализатор: функция высшего порядка
- Позволяет считать семантику “на лету”, без явного построения деревьев разбора
- Анализатор произвольной сложности можно получить путем комбинирования нескольких простых базовых парсеров
- Позволяют разбирать некоторые контекстно-зависимые языки

Тип парсер-комбинаторов

```
type ('a, 'e) tag =
  Parsed of 'a * 'e option
| Failed of 'e option

type ('s, 'r, 'e) result = ('r * 's, 'e) tag

and  ('s, 'r, 'e) parse  = 's -> ('s, 'r, 'e) result
```

Базовые парсер-комбинаторы

```
let empty s = Parsed (((), s), None)

let fail  s = Failed None

let lift  s = Parsed ((s, s), None)

let return x s = Parsed ((x, s), None)
```

Комбинатор последовательности

```
let seq x y s =
  match x s with
  | Parsed ((b, s'), err) ->
    (match y b s' with
    | Failed x -> Failed (join err x)
    | x -> x
    )
  | Failed x -> Failed x

let (|>) = seq
```

Комбинатор альтернативы

```
let alt x y s =
  match x s with
  | Failed x ->
    (match y s with
    | Failed y -> Failed (join x y)
    | Parsed (ok, err) -> Parsed (ok, join x err)
    )
  | x -> x

let (<|>) = alt
```

Пример парсера для языка $\{a^n \mid n \geq 0\}$

```
(* A : epsilon | 'a' A *)
let rec a s =
  alt empty
      (seq (terminal 'a') (fun _ -> a))
      s
```

Семантические действия

```
let map f p s =
  match p s with
  | Parsed ((b, s'), e) -> Parsed ((f b, s'), e)
  | x -> x
```

Вспомогательные парсер-комбинаторы

```
let opt p =
  alt
    (map (fun x -> Some x) p)
    (return None)

let rec many p =
  seq
    p
    (alt
      (fun h -> map (fun t -> h :: t) (many p))
      (return [])
    )
```

```
let a = many (terminal 'a')
```

Синтаксическое расширение

```
ostap (
  e : x:e "+" y:t {`Add (x, y)}
    | t;
  t : x:t "*" y:p {`Mul (x, y)}
    | p;
  p : (* parse integer *)
)
```

Парсеры высшего порядка: переиспользование кода

```
ostap (
  e : x:e "+" y:t {`Add (x, y)}
    | t;
  t : x:t "*" y:p {`Mul (x, y)}
    | p;
  p : (* parse integer *)
)
```

```
ostap (
  e : x:e "+" y:t {`Add (x, y)}
    | t;
  t : x:t "*" y:p {`Mul (x, y)}
    | p;
  p : (* parse double *)
)
```

Парсеры высшего порядка

```
ostap (
  e[p] : x:e[p] "+" y:t[p] {`Add (x, y)}
       | t[p];
  t[p] : x:t[p] "*" y:p    {`Mul (x, y)}
       | p;
)
```

```
ostap (
  e_integer: e[(* parse integer *)]
)
```

```
ostap (
  e_double : e[(* parse double  *)]
)
```

Парсер-комбинаторы и левая рекурсия

Являясь реализацией нисходящего анализа, парсер-комбинаторы не способны обрабатывать леворекурсивные правила

```
ostap (
  e[p] : e[p] "+" t[p] (* to apply e, *)
       | t[p];         (* one needs to apply e... *)
  t[p] : t[p] "*" p
       | p;
)
```

Удаление левой рекурсии значительно усложняет спецификацию языка и ухудшает композициональность анализаторов

Поддержка леворекурсивных анализаторов

- Frost R. A., Hafiz R., Callaghan P. Parser combinators for ambiguous leftrecursive grammars
  - Ограничение количества леворекурсивных вызовов длиной непрочитанной строки
- Warth A., Douglass J. R., Millstein T. D. Packrat parsers can support left recursion
  - Использование мемоизации для обеспечения завершаемости
- Требуют, чтобы парсер был первого порядка
- Izmaylova A., Afroozeh A., Tijs van der Storm (???). Practical, general parser combinators
  - Использование техники CPS для обеспечения завершаемости
- Фиксируют конкретную семантику

Поддержка левой рекурсии в PEG

- Medeiros S., Mascarenhas F., Ierusalimschy R. Left Recursion in Parsing Expression Grammars
- Динамический поиск наилучшего количества леворекурсивных вызовов
- Использует мемоизацию
- Поддерживает явную, неявную, взаимную рекурсию
- Требуют, чтобы парсер был первого порядка

Поддержка левой рекурсии в Ostap

- Используется идея Medeiros et al
- Специальный комбинатор `fix` для поддержки леворекурсивных парсеров высшего порядка

```
ostap (
  e[p] : e[p] "+" e[p]
       | p;
)
```

```
ostap (
  let e p s = fix (fun e -> ostap(e "+" e | p)) s
)
```

Заключение

- Представлена библиотека парсер-комбинаторов и синтаксическое расширение для языка OCaml
- Реализована поддержка леворекурсивных спецификаций синтаксических анализаторов
  - Позволяет использование парсеров высшего порядка
  - Сложность растет экспоненциально в зависимости от глубины вложенности рекурсии