PLC-2017

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

Автор: Екатерина Вербицкая

Санкт-Петербургский государственный университет
Лаборатория языковых инструментов JetBrains

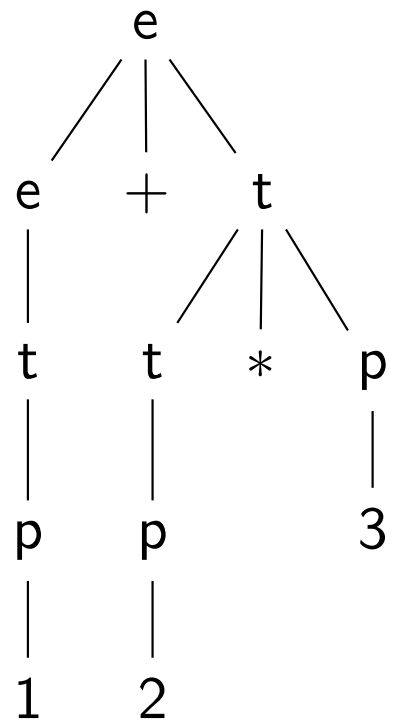
05 апреля 2017

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 парсеров

```
type α tag = Parsed of α | Failed

type (σ, ρ) result = (ρ * σ) tag

and  (σ, ρ) parse = σ →(σ, ρ) result
```

Базовые парсер-комбинаторы

```
let empty s = Parsed ((), s)

let fail  s = Failed

let return x s = Parsed (x, s)
```

Комбинатор последовательности

```
let seq x y s =
  match x s with
  | Parsed (b, s') → y b s'
  | Failed → Failed

let (▷) = seq
```

Комбинатор альтернативы

```
let alt x y s =
  match x s with
  | Failed → y s
  | x → x

let (◊) = alt
```

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

```
(* A : epsilon | 'a' A *)
let rec a s = empty ◊ (terminal 'a' ▷ fun _ → a) s
```

Семантические действия

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

Вспомогательные парсер-комбинаторы

```
let opt p = map (fun x → Some x) p ◊ return None

let rec many p =
  p ▷ (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

- Ограничение количества леворекурсивных вызовов длиной непрочитанной строки
  - Frost R. A., Hafiz R., Callaghan P. 2008
- Использование мемоизации для обеспечения завершаемости
  - Warth A., Douglass J. R., Millstein T. D. 2008
- Требуют, чтобы парсер был первого порядка

- Использование техники CPS для обеспечения завершаемости
  - Izmaylova A., Afroozeh A., van der Storm T. 2016 Practical, general parser combinators
- Фиксируют конкретную семантику

Поддержка левой рекурсии в PEG

- Medeiros S., Mascarenhas F., Ierusalimschy R. 2014
- 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
)
```

Комбинатор *fix*

```
let fix p s =
  let x' = ref None in
  let rec fix p s = p (fix p) s in
  let help cur prev =
    match cur, prev with
    | Failed, _ → prev
    | Parsed (_,s), Parsed (_,s') when s#pos < s'#pos → prev
    | _, _ → cur
  in
  let p x s =
    match !x' with
    | None → x' := Some (fun s → memo x s); help (p x s) (x s)
    | Some x → help (p x s) (x s)
  in
  fix 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