Tópicos em LP

Fabio Mascarenhas – 2018.1

http://www.dcc.ufrj.br/~fabiom/comp2

Exercício

- Modifique o scanner para operar em strings ou sequências de bytes como entrada ao invés de apenas strings

Parsers

- Um parser processa uma sequência de entrada, consumindo uma parte (ou toda a sequência) para produzir um resultado

- Um analisador léxico é um tipo de parser: a entrada é uma sequência de caracteres, e o analisador produz um token

  bytes

- Um analisador sintático é outro tipo: a entrada é uma sequência de tokens, e o analisador produz uma árvore sintática (abstrata ou não)

- Em um analisador sintático recursivo, cada parte do analisador também é um parser, cada uma consumindo uma parte da entrada e produzindo um resultado parcial; o analisador completo é feito através da composição dessas partes

Combinadores de parsing

- Combinadores de parsing são uma técnica para expressar parsers recursivos em uma linguagem onde funções são valores de primeira classe

- A ideia é construir parsers mais complexos a partir da composição de parsers mais simples, mas usando *combinadores* ao invés da composição sintática de um analisador recursivo tradicional

- Um parser é uma função que recebe a entrada e retorna um resultado e um sufixo dessa entrada

- Um combinador é uma função que recebe uma ou mais funções que descrevem parsers e as combina em um novo parser

Lista de resultados

- Nem sempre um parser é bem sucedido
- Uma determinada entrada também pode ter mais de uma análise possível
- Para representar essas duas possibilidades definimos que um parser retorna uma *lista de resultados* ao invés de um resultado só
- Cada elemento dessa lista é um par com o resultado em si e um sufixo da entrada
- Se a lista for vazia, o parser falhou

Um combinador simples

- O parser mais simples é aquele que reconhece um único item da entrada, dado algum *predicado*

- O combinador `class` retorna um desses parsers, dado o predicado (a *classe*) do item desejado:

```
local function class(p)
    return function (input, pos)
                local x = input:byte(pos)
                if x ~= nil and p(x) then
                    return { { x, pos+1 } }
                else
                    return { }
                end
            end
end
```

Bind

- O combinador `bind` junta um parser com uma função que recebe o resultado de um parser e retorna um novo parser:

```
local function bind(p, f)
    return function (input, pos)
        local res = p(input, pos)
        local nres = {}
        for i = 1, #res do
            local res = f(res[i][1])(input, res[i][2])
            for i = 1, #res do
                nres[#nres+1] = res[i]
            end
        end
        return nres
    end
end
```

Quiz

- O que acontece se o parser passado para `bind` falhar (retornar uma lista vazia de resultados)? E se o parser retornado por `f` falhar para algum sufixo da entrada?

Sequência

- O combinador unit produz um parser que não consome nada, apenas gera um resultado:

```
local function unit(x)
    return function (input, pos)
                return { { x, pos } }
            end
end
```

- Podemos juntar unit e bind para fazer um combinador que aplica dois parsers em sequência, juntando cada combinação de resultados em um par

```
local function seq(p1, p2)
  return bind(p1, function (res1)
    return bind(p2, function (res2)
      return unit(seq:new{ res1, res2 })
    end)
  end)
end
```