Compiladores II

Fabio Mascarenhas - 2014.2

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

Escolha LL(1)

- Outro tipo de escolha útil é a escolha guiada por determinado predicado aplicado ao primeiro item da entrada:

```
local function pchoice(pred, p1, p2)
  return function (input)
           local item = input:byte(1)
           if pred(item) then
               return p1(input)
           else
               return p2(input)
           end
         end
end
```

Dojo

- Reescreva o scanner para usar os combinadores que vimos, ao invés de acessar diretamente os itens da entrada

- Faça o scanner ignorar comentários; comentários começam com `--` e vão até o final da linha

Análise sintática

- Obviamente que combinadores de parsing não se restringem à análise léxica
- Ações semânticas são facilmente expressas com `bind`, mas recursão é um problema
- Uma regra gramatical não recursiva pode ser construída usando os combinadores que já temos, mas uma regra recursiva precisa ser expressa como uma função que constrói o parser:

```
local function ids()
    return seq(token("id"), token(","), ids()) ^ function (tup)
                                                     local i, _, is = tup()
                                                     return unit(is:add(i, 1))
                                                 end /
           token("id") ^ function (i) return unit(vector{i}) end
end
```

Combinadores de recursão

- Recursão em uma gramática é usada para *listas* e para *operações binárias*

- Podemos definir combinadores genéricos para esses dois tipos, e assim definir regras gramaticais sem precisar usar recursão explícita

```
local function listof(p, sep)
    return seq(p, poss(sep ^ function (_) return p end)) ^
                        function (tup)
                            local a, as = tup()
                            return unit(as:add(a, 1))
                        end
end
```

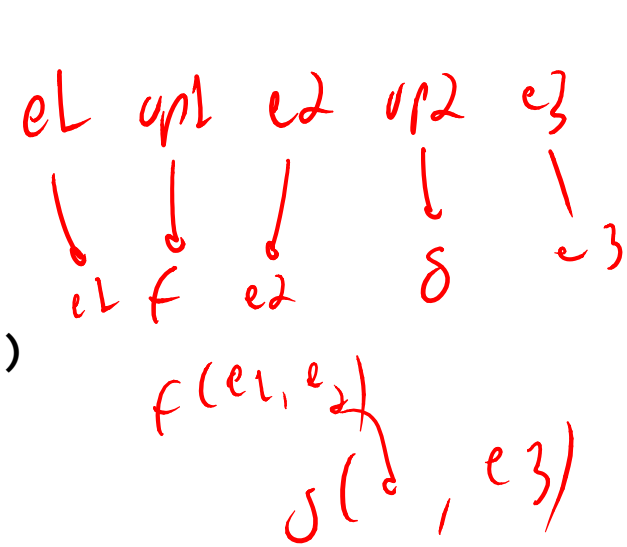
- O combinador `listof` reconhece uma lista de elementos com algum separador, jogando fora os resultados produzidos pelos separadores

ids = listof(token("id") / token(","))

exps = listof(exp, token(","))

Combinadores de recursão (2)

- Os combinadores chainl e chainr reconhecem expressões binárias associando à esquerda ou à direita



```
local function chainl(p, op)
    local function rest(e1)
        return seq(op, p) ^ function (tup)
                                local f, e2 = tup()
                                return rest(f(e1, e2))
                            end / unit(e1)
    end
    return p ^ rest
end

local function chainr(p, op)
    return p ^ function (e1)
                    return seq(op, chainr(p, op)) ^ function (tup)
                                                        local f, e2 = tup()
                                                        return unit(f(e1, e2))
                                                    end / unit(e1)
                end
end
```