Compiladores II

Fabio Mascarenhas - 2014.2

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

PEGs

- As gramáticas de expressões de parsing, ou PEGs (parsing expression grammars) são uma linguagem para especificar parsers determinísticos

- Ao contrário das gramáticas livres de contexto, PEGs têm um mapeamento natural para os combinadores que estamos usando

- Uma *expressão de parsing* pode ser a expressão vazia `''`, um *terminal* `'a'`, um *não-terminal* `A`, uma *sequência* `pq`, onde `p` e `q` são expressões de parsing, uma *escolha ordenada* `p/q`, uma *repetição* `p*`, ou um *predicado* `!p`

- A únicas expressões que não correspondem a combinadores que já usamos são `!p`, que é fácil definir como um combinador, e os não-terminais, que vamos a seguir

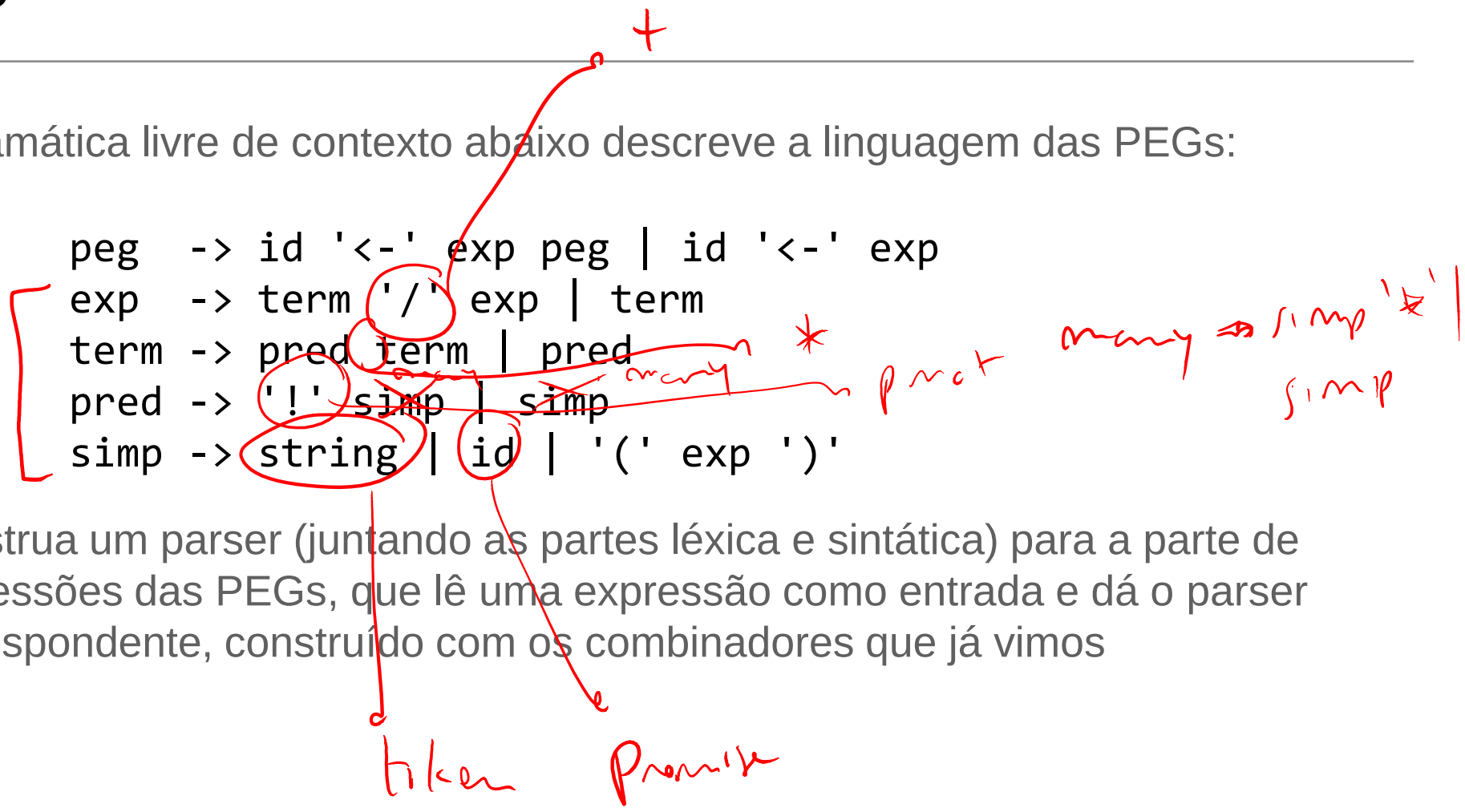
Não-terminais e gramáticas

- Uma PEG é um mapeamento de não-terminais para expressões de parsing
- Quando aplicamos o parser de uma PEG a uma entrada, fica implícito que qualquer não-terminal encontrado é resolvido no contexto dessa PEG: o efeito de aplicar um não-terminal é o efeito de aplicar a expressão de parsing correspondente
- Não-terminais dão o poder de *recursão* às PEGs, mas com duas restrições:
  - Todo não-terminal referenciado tem que ser definido
  - Não pode haver *recursão à esquerda* direta ou indireta

Dojo

- A gramática livre de contexto abaixo descreve a linguagem das PEGs:

```
peg  -> id '<-' exp peg | id '<-' exp
exp  -> term '/' exp | term
term -> pred term | pred
pred -> '!' simp | simp
simp -> string | id | '(' exp ')'
```



- Construa um parser (juntando as partes léxica e sintática) para a parte de expressões das PEGs, que lê uma expressão como entrada e dá o parser correspondente, construído com os combinadores que já vimos

Ações

- Uma maneira de introduzir ações semânticas em nossa linguagem de PEGs é ter uma sintaxe para um operador similar ao combinador `bind`, que passaria o resultado de uma expressão para uma função, definida fora da gramática:

```
term -> bind term | bind
bind -> pred '->' id | pred
```

- A alta precedência é proposital, para ficar parecido com a precedência do operador `^` que estamos usando para bind

- Se usamos `bind` para `->` então a função descrita por id pode afetar a análise, já que ela retorna um parser