

Compiladores — Folha laboratorial 3

Pedro Vasconcelos, DCC/FCUP

Outubro 2020

Gramáticas independentes de contexto

Exercício 1

- (a) Considere a gramática apresentada na aula teórica com símbolos terminais $\{a, b\}$, não-terminais $\{S, B\}$, símbolo inicial S e as produções seguintes.

$$\begin{array}{ll} S \rightarrow aSB & B \rightarrow Bb \\ S \rightarrow \varepsilon & B \rightarrow b \\ S \rightarrow B & \end{array}$$

Descreva a linguagem reconhecida por esta gramática numa frase. (Sugestões: Onde podem ocorrer as e bs ? Qual relação entre as suas contagens?)

- (b) Escreva uma gramática para a linguagem de palavras com o mesmo número de as e bs (em qualquer ordem)
- (c) Escreva uma gramática para a linguagem de parêntesis casados. Alguns exemplos:

| Palavras aceites | Palavras não aceites |
|-------------------------------|----------------------|
| ε (palavra vazia) | (|
| () |) |
| (()) |)(|
| ()() | ((|
| (())()) | (()) |

- (d) Escreva uma gramática para a linguagem da expressão regular $((ab^*a) | (ba^*b))$
- (e) Escreva uma gramática para a linguagem da expressão regular $((0|1)^+ \cdot "(0|1)^* | ((0|1)^* \cdot "(0|1)^+)$

Exercício 2

Considere uma gramática de programas sequências apresentada na aula teórica:

$$\begin{array}{ll} S \rightarrow S ; S & E \rightarrow \text{ident} \\ S \rightarrow \text{ident} = E & E \rightarrow \text{num} \\ S \rightarrow \text{ident} ++ & E \rightarrow E + E \\ & E \rightarrow (S , E) \end{array}$$

- (a) Mostre que a gramática acima é ambígua, i.e., encontre uma palavra com duas derivações correspondentes a árvores distintas. Consegue encontrar mais do que um exemplo?
- (b) Re-escreva a gramática de forma a eliminar a ambiguidade.

Exercício 3

Considere a gramática para expressões aritméticas fatorizada de forma a eliminar ambiguidade e recursão à esquerda:

$$\begin{array}{lll}
 E \rightarrow T E' & T \rightarrow F T' & \\
 E' \rightarrow + T E' & T' \rightarrow * F T' & F \rightarrow \text{num} \\
 E' \rightarrow - T E' & T' \rightarrow / F T' & F \rightarrow (E) \\
 E' \rightarrow \varepsilon & T' \rightarrow \varepsilon &
 \end{array}$$

Partindo do esqueleto de código Haskell no repositório *Git*, implemente um *parser* para esta gramática usando descida recursiva. O *parser* deve aceitar expressões bem formadas como $123*(45+6)$ ou $(1+2*3)/(4+5)$ e rejeitar expressões incorretas como $)12+53($ ou $(1++2*3)$.

Comece por estudar o analisador lexical escrito usando Alex (ficheiro `src/Lexer.x`). Pode usar `cabal build` e `cabal run` para compilar e executar alguns testes (ver o ficheiro `src/Tests.hs`).

Notas:

1. o comando `cabal` executa o Alex para gerar o `.hs` a partir do `.x`; não é necessário executar manualmente
2. não deve acrescentar o `Lexer.hs` ao repositório Git (uma vez que este será gerado automaticamente).

Exercício 4

Considerando de novo a gramática do Exercício 3:

- (a) Determine o valor de `NULLABLE` e dos conjuntos `FIRST` e `FOLLOW` para os não-terminais desta gramática. (Sugestão: comece por determinar as repostas por inspeção das produções e só depois use as equações apresentadas na aulas para verificar se a sua intuição está correta.)
- (b) Mostre que a gramática acima é $LL(1)$ calculado a tabela de *parsing* preditivo.