Compiladores (CC3001)
Aula 4: Gramáticas independentes de contexto

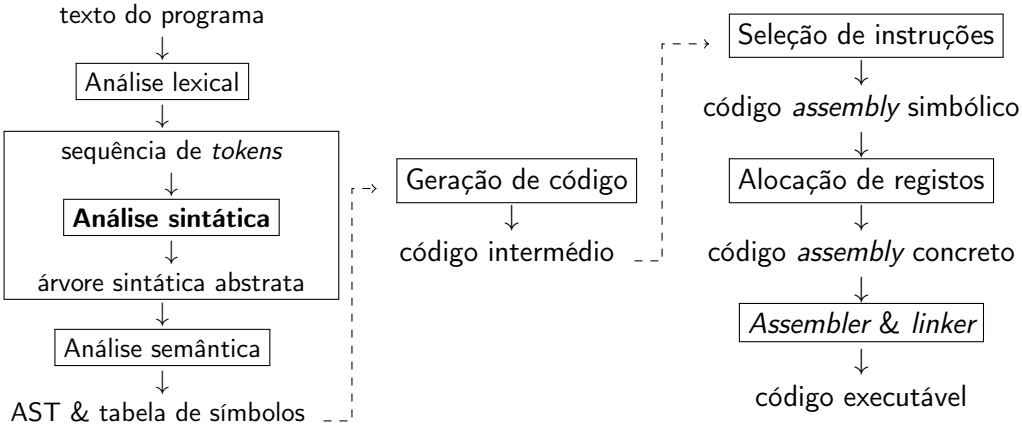
Mário Florido

DCC/FCUP

2024

[dcc] FC FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Re-lembrar: fases dum compilador



Análise sintática

Gramáticas independentes de contexto

Escrever gramáticas

sintaxe: estudo das regras e dos princípios que regem a organização dos constituintes das frases (...)

Dicionário Priberam da Língua Portuguesa

- Um programa sintaticamente bem formado respeita regras de estrutura, e.g.:
  - chavetas { } e parêntesis ( ) "casados"
  - operadores +, ∗, etc. com número correto de operandos;
  - instruções terminadas ou separadas corretamente (ponto-e-vírgula ;)
- Tal como na linguagem natural, um programa pode estar sintaticamente bem formado e mesmo assim não ter sentido
  Exemplo (Chomsky, 1957): "*Colorless green ideas sleep furiosly*"
- O analisador sintático (*parser*) constroi a árvore sintática a partir da sequência de *tokens* (ou reporta um erro)
- Ferramenta essencial: gramáticas independentes de contexto

Esta aula

Análise sintática

Gramáticas independentes de contexto

Escrever gramáticas

Uma gramática independente de contexto $G = (\Sigma, N, S, P)$ é definida por:

- Σ conjunto de símbolos *terminais*;
- N conjunto de símbolos *não-terminais*;
- $S \in N$ símbolo inicial;
- P conjunto de *produções* da forma $X \to \alpha$ tais que:
  - X é um não-terminal;
  - α é uma sequência (possivelmente vazia) de terminais ou não-terminais

Exemplo

Terminais:

$$\Sigma = \{a, b\}$$

Não-terminais:

$$N = \{S, B\}$$

Símbolo inicial:

$$S$$

Produções:

$$S \to aSB$$
$$S \to \varepsilon$$
$$S \to B$$
$$B \to Bb$$
$$B \to b$$

A relação de derivação $\Rightarrow$ substitui um não-terminal pelo lado direito de alguma produção.

Exemplo: dadas as produções

$$
\begin{aligned}
S &\to aSB && (1)\\
S &\to \varepsilon && (2)\\
S &\to B && (3)\\
B &\to Bb && (4)\\
B &\to b && (5)
\end{aligned}
$$

Anotando cada derivação com a produção usada:

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

- Começando com o símbolo inicial...
- substituimos não-terminais usando as produções...
- quando só restam terminais: obtemos uma *palavra* descrita pela gramática.

Para a gramática anterior G:

$$S \Rightarrow aSB \Rightarrow aaSBB \Rightarrow aaBB \Rightarrow aaBbB \Rightarrow aabbB \Rightarrow aabbb$$

Logo: a palavra $aabb \in L(G)$.

Mais formalmente, se $G = (\Sigma, N, S, P)$ então

$$L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$$

($\Rightarrow^*$ é o *fecho transitivo* da derivação.)

$$
\begin{array}{lll}
G: & S & \to aSB \\
 & S & \to \varepsilon \\
 & S & \to B \\
 & B & \to Bb \\
 & B & \to b
\end{array}
$$

Descrever a linguagem da gramática G numa frase.

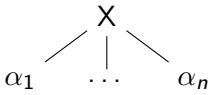
- Onde podem ocorrer letras a e b numa palavra aceite?
- Qual a relação entre o *número* de as e bs?

Podemos representar cada passo numa derivação como um nó numa árvore sintática.

Cada produção

$$X \to \alpha_1 \ldots \alpha_n$$

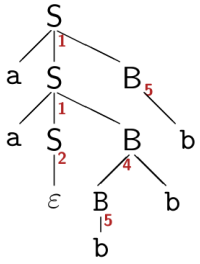
corresponde a um nó com n sub-árvores:



Exemplo: a derivação

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

corresponde à arvore:



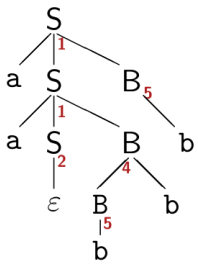
$$\begin{array}{lllll} G: & S & \to aSB & (1) \\ & S & \to \varepsilon & (2) \\ & S & \to B & (3) \\ & B & \to Bb & (4) \\ & B & \to b & (5) \end{array}$$

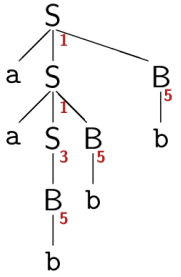
Podemos obter $S \Rightarrow^* aabbb$ de duas maneiras:

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb \quad (6)$$

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{3}{\Rightarrow} aaBBB \overset{5}{\Rightarrow} aabBB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb \quad (7)$$

A derivação (6) corresponde à árvore à esquerda; (7) corresponde à árvore à direita.





G :

$$\begin{aligned} S &\to aSB \quad (1)\\ S &\to \varepsilon \quad (2)\\ S &\to B \quad (3)\\ B &\to Bb \quad (4)\\ B &\to b \quad (5) \end{aligned}$$

Uma gramática diz ambígua se existem palavras que admitem derivações com àrvores distintas.

Logo, a gramática G é ambígua porque as derivações

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{3}{\Rightarrow} aaBBB \overset{5}{\Rightarrow} aabBB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

correspondem a árvores distintas.

Uma gramática diz ambígua se existem palavras que admitem derivações com àrvores distintas.

Logo, a gramática G é ambígua porque as derivações

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{2}{\Rightarrow} aaBB \overset{4}{\Rightarrow} aaBbB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

$$S \overset{1}{\Rightarrow} aSB \overset{1}{\Rightarrow} aaSBB \overset{3}{\Rightarrow} aaBBB \overset{5}{\Rightarrow} aabBB \overset{5}{\Rightarrow} aabbB \overset{5}{\Rightarrow} aabbb$$

correspondem a árvores distintas.

Notas:

- derivações diferentes podem corresponder à mesma árvore (diferindo apenas na ordem das substituições)
- para que a gramática seja ambígua devemos ter *árvores diferentes* e não apenas *derivações diferentes*

Porquê evitar ambiguidade?

- Como modelo matemático para *descrever* uma linguagem, uma gramática ambígua é perfeitamente válida
- Mas num compilador vamos usar a gramática para associar *significado* aos fragmentos de programa
- Nesse caso é importante que não existam interpretações divergentes
- As linguagens de programação são concebidas de forma que a gramática não seja ambígua (ou pelo menos com regras para resolver ambiguidades)

Esta aula

Análise sintática

Gramáticas independentes de contexto

Escrever gramáticas

Exemplo: expressões aritméticas

Símbolo não-terminal: E

Símbolos terminais (tokens): num + * ( )

Produções:

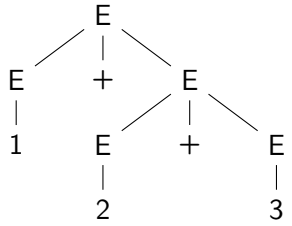
$$E \to E{+}E$$
$$E \to E{*}E$$
$$E \to \text{num}$$
$$E \to (E)$$

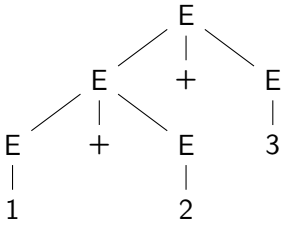
Podemos também abreviar as produções do mesmo não-terminal:

$$E \to E{+}E \mid E{*}E \mid \text{num} \mid (E)$$

- Esta gramática é *ambígua*
- Vamos ver duas expressões que admitem mais que uma árvore sintática

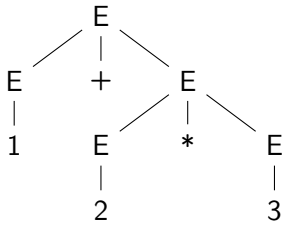
Duas árvores de derivação para 1+2+3:

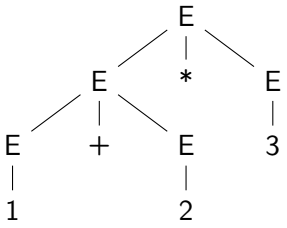




As duas árvores podem calcular o mesmo resultado — mas em qualquer caso mostram que a gramática é ambígua.

Duas árvores de derivação para 1+2*3:





Neste caso as duas árvores representam computações distintas!

Frequentemente podemos eliminar a ambiguidade re-escrevendo a gramática.

Para a gramática de expressões devemos fixar:

- a associatividade dos operadores
  - esquerda: 1+2+3 interpretado como $(1 + 2) + 3$
  - direita: 1+2+3 interpretado como $1 + (2 + 3)$
- a prioridade entre operadores + e $*$
  e.g. 1+2*3 interpretado como $1 + (2 \times 3)$ ou como $(1 + 2) \times 3$

Vamos re-escrever a gramática de forma fixar estas escolhas.

Eliminar ambiguidade (cont.)

$$
\begin{array}{lll}
E \to E + T & T \to T * F & F \to \text{num} \\
E \to T & T \to F & F \to (\ E\ )
\end{array}
$$

- Novos terminais e produções:

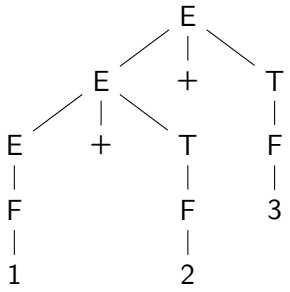
  expressões E são somas de *termos*;

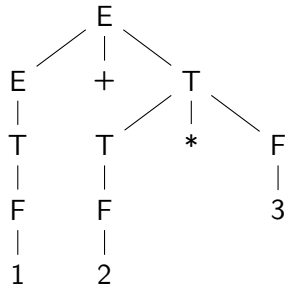
  termos T são produtos de *fatores*;

  fatores F são constantes ou expressões entre parêntesis.

- Produções de E e T com recursão à esquerda implicam associativatividade à esquerda para + e $*$

Na nova gramática 1+2+3 e para 1+2*3 admitem árvores únicas:





Símbolos não-terminais: S (*statements*) E (*expressions*)

Símbolos terminais (tokens): ident num = ( ) + , ; ++

Produções:

$S \to S\ ;\ S$ $\qquad$ $E \to \text{ident}$

$S \to \text{ident} = E$ $\qquad$ $E \to \text{num}$

$S \to \text{ident}\ ++$ $\qquad$ $E \to E + E$

Exemplo de sintaxe concreta (antes da análise lexical):

```
a = 17; b = 2
a = 0; (a++; b=a+5)
```

(Notar que aqui o ponto-vírgula é um *separador* de comandos — e não *terminador* como na linguagem C.)

Exercícios:

1. Mostrar que esta gramática é ambígua.
2. Re-escrever a gramática de forma a eliminar a ambiguidade.
   (Nota: não é apenas nas expressões!)

Muitas linguagens de programação permitem escrever *if/then* com *else* optional:

$$S \rightarrow \text{if } E \text{ then } S \text{ else } S$$
$$S \rightarrow \text{if } E \text{ then } S$$
$$S \rightarrow \textit{etc.}$$

Então

$$\text{if } e_1 \text{ then if } e_2 \text{ then } s_1 \text{ else } s_2$$

poderia ser interpretado de duas formas distintas:

$$\text{if } e_1 \text{ then } \{\text{if } e_2 \text{ then } s_1 \text{ else } s_2\} \quad (8)$$
$$\text{if } e_1 \text{ then } \{\text{if } e_2 \text{ then } s_1\} \text{ else } s_2 \quad (9)$$

Normalmente a opção (8) é preferida: associamos o *else* ao *if* mais próximo.

Podemos re-escrever a gramática para remover a ambiguidade introduzindo dois não-terminais M (*matched statements*) e U (*unmatched statements*).

$$
\begin{aligned}
S &\to M \\
S &\to U \\
\\
M &\to \text{if } E \text{ then } M \text{ else } M \\
M &\to \textit{etc.} \\
\\
U &\to \text{if } E \text{ then } S \\
U &\to \text{if } E \text{ then } M \text{ else } U
\end{aligned}
$$

Na prática: pode ser preferível não mudar a gramática e resolver a ambiguidade na implementação do analisador sintático (veremos mais à frente).