INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

Curso de Engenharia de Computação
ECM253 – Linguagens Formais, Autômatos e Compiladores

Gramáticas e Árvores de Análise Sintática

Slides da disciplina ECM253 – Linguagens Formais, Autômatos e Compiladores
Curso de Engenharia de Computação
Instituto Mauá de Tecnologia – Escola de Engenharia Mauá
Prof. Marco Antonio Furlan de Souza
<marco.furlan@maua.br>

Análise sintática

- **Aspectos gerais**
  - A **análise sintática** verifica a **sintaxe** ou **estrutura de um programa**;
  - A sintaxe da maioria das linguagens de programação segue as **regras gramaticais** de uma **gramática livre de contexto**;
  - A gramática é **importante** para o construtor de compiladores e para que desenvolvedores; aprendam a programar com a linguagem (mas ainda é necessário entender a semântica de suas construções);
  - Existem diversas linguagens de programação padronizadas: **C**, **C++**, **Java**, **Javascript**, para citar algumas.
- **Uma gramática é definida por regras**
  - **Regras gramaticais** são **similares** às,**expressões regulares** porém **incluem recursão**;
  - Uma forma conveniente de representar uma gramática é por meio de **árvore de análise sintática**.

O processo de análise sintática

- Das **marcas produzidas** pelo **analisador léxico**, produz-se a a **estrutura sintática** do programa:

$$\textit{seqüência de marcas} \xrightarrow{\text{analisador sintático}} \textit{árvore sintática}$$

- Em um compilador de uma **única passagem** não há necessidade de construir explicitamente a árvore sintática; caso contrário isso torna-se uma tarefa à parte;
- A **árvore sintática** é uma **estrutura de dados dinâmica** cujos **nós** são **registros** contendo **atributos** necessários ao restante do processo de compilação;
- O **tratamento de erros** é **mais complicado** na análise sintática que na análise léxica.

Gramáticas livres de contexto

- Uma **gramática livre de contexto** é a 4-upla $G = (T, N, P, S)$ onde:
  - T é um conjunto de símbolos denominados **terminais**;
  - N é um conjunto de símbolos denominados **não-terminais**, $T \cap N = \emptyset$;
  - P é um conjunto de **regras gramaticais** ou **produções** da forma $A \rightarrow \alpha$, sendo $A \in N$ e $\alpha \in (T \cup N)^*$;
  - O significado de $A \rightarrow \alpha$ é "A pode ser reescrito (ou substituído) como (por) α". Somente não-terminais aparecem no lado esquerdo das regras;
  - S é o **símbolo inicial**, $S \in N$;
  - O conjunto $(T \cup N)$ é denominado de conjunto de símbolos de G e o conjunto T é o **alfabeto da linguagem** gerada por G.
- É amplamente **utilizada** na **especificação** da **estrutura sintática** de linguagens de programação.

Gramáticas livres de contexto

- **Definições**
  - **Passo de derivação** sobre G
    - Possui a forma geral $\alpha A \gamma \Rightarrow \alpha\beta\gamma$, sendo $\alpha, \gamma \in (T \cup N)^*$ e $A \rightarrow \beta \in P$. A expressão $\alpha\beta\gamma$ é denominada de **forma sentencial**.
    - A relação $\alpha \Rightarrow^* \beta$ é denominado **fecho transitivo da derivação** $\Rightarrow$, implicando que deve haver uma seqüência com zero ou mais passos de derivação ($n > 0$):

$$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n, \text{ tal que } \alpha = \alpha_1 \text{ e } \beta = \alpha_n$$

    - Uma **derivação** sobre a gramática G tem a forma $S \Rightarrow^* w$, $w \in T^*$ (uma **sentença**), sendo S o símbolo inicial de G.
    - **A linguagem gerada por** G, $L(G)$ é definida pelo conjunto $L(G) = \{w \in T^* | S \Rightarrow^* w\}$.
    - Uma **derivação à esquerda** é aquela que em cada passo $\alpha A \gamma \Rightarrow \alpha\beta\gamma$ é tal que $\alpha \in T^*$. A cadeia é gerada da esquerda para a direita.
    - Uma **derivação à direita** é aquela que em cada passo $\alpha A \gamma \Rightarrow \alpha\beta\gamma$ é tal que $\gamma \in T^*$. A cadeia é gerada da direita para a esquerda.

Gramáticas livres de contexto

- **Exemplo de derivação**
  - **Considerando a gramática** $G = (T, N, P, S)$, onde:
    - $T = \{\texttt{numero}, \texttt{+}, \texttt{-}, *, \texttt{(}, \texttt{)}\}$ (considerar `numero` uma expressão regular que casa com números inteiros)
    - $N = \{E, O\}$ (E representa expressão; O representa operador)
    - $S = E$ (O símbolo de partida é E)
    - $P = \{\ E \rightarrow EOE,$
      $E \rightarrow (E),$
      $E \rightarrow \texttt{numero},$
      $O \rightarrow \texttt{+},$
      $O \rightarrow \texttt{-},$
      $O \rightarrow * \ \}$

Gramáticas livres de contexto

- **Exemplo de derivação**
  - A cadeia `(34-3)*42` é **derivada à direita** na gramática G assim:

$$
\begin{aligned}
E &\Rightarrow EOE && [E \to EOE] \\
&\Rightarrow EO\texttt{numero} && [E \to \texttt{numero}] \\
&\Rightarrow E * \texttt{numero} && [O \to *] \\
&\Rightarrow (E) * \texttt{numero} && [E \to (E)] \\
&\Rightarrow (EOE) * \texttt{numero} && [E \to EOE] \\
&\Rightarrow (EO\texttt{numero}) * \texttt{numero} && [E \to \texttt{numero}] \\
&\Rightarrow (E - \texttt{numero}) * \texttt{numero} && [O \to -] \\
&\Rightarrow (\texttt{numero} - \texttt{numero}) * \texttt{numero} && [E \to \texttt{numero}]
\end{aligned}
$$

Gramáticas livres de contexto

- **Notação BNF**
  - A notação **BNF** ou **Backus-Naur** é uma **metalinguagem** que permite especificar linguagens livres de contexto;
  - Elementos da notação:
    - O simbolo entre "⟨" e "⟩" é um símbolo **não-terminal**;
    - O símbolo escrito como `simbolo` é uma **expressão regular** (cadeia terminal);
    - O símbolo escrito entre "`'`" e "`'`" é um símbolo **constante** ou **literal**;
    - O **metasímbolo** "::=" indica que a **parte esquerda** da regra é **definida** (ou reescrita) como a parte **direita da regra**;
    - O **metasímbolo** "|" indica que uma regra possui uma definição alternativa ("ou").
  - **Exemplo**

    ⟨*exp*⟩ `::=` ⟨*exp*⟩ ⟨*op*⟩ ⟨*exp*⟩ `|` `'('`⟨*exp*⟩`')'` `|` `numero`

    ⟨*op*⟩ `::=` `'+'` `|` `'-'` `|` `'*'`

Gramáticas livres de contexto

- **Exemplos**

1. ⟨*E*⟩ `::= '(' ⟨E⟩ ')'|a`

   Gera a linguagem $L(G) = \{a, (a), ((a)), (((a)))\}$.

2. ⟨*E*⟩ `::= '('⟨E⟩')'`

   Essa gramática gera a linguagem $L(G) = \emptyset$. Por quê?

3. ⟨*E*⟩ `::= ⟨E⟩ + a | a`

   Essa gramática gera a linguagem $L(G) = \{a, a+a, a+a+a, \ldots\}$

4. ⟨*decl*⟩ `::= ⟨decl-if⟩ | ⟨outra⟩`

   ⟨*decl-if*⟩ `::= if '('⟨exp⟩')' ⟨decl⟩`
   `  | if '('⟨exp⟩')' ⟨decl⟩ else ⟨decl⟩`

   ⟨*exp*⟩ `::= '0' | '1'`

   Que cadeias essa gramática gera?

Gramáticas livres de contexto

- **Exemplos**

5. ⟨*A*⟩ `::= '('`⟨*A*⟩`')'`⟨*A*⟩ | ϵ

   Essa gramática gera cadeias de “parênteses balanceados”. Prove!

6. ⟨*decl*⟩ `::=` ⟨*decl-if*⟩ | ⟨*outra*⟩

   ⟨*decl-if*⟩ `::= if '('`⟨*exp*⟩`')'` ⟨*decl*⟩ ⟨*parte-else*⟩

   ⟨*parte-else*⟩ `::= else` ⟨*decl*⟩ | ϵ

   ⟨*exp*⟩ `::= '0' | '1'`

   Qual é a diferença do exemplo 4?

7. ⟨*decl-seq*⟩ `::=` ⟨*decl*⟩ `;` ⟨*decl-seq*⟩ | ⟨*decl*⟩ `;`

   ⟨*decl*⟩ `::= S` | ϵ

   A linguagem gerada por essa gramática é $L(G) = \{\epsilon\texttt{;}, \texttt{S;}, \texttt{S;S;}, \ldots\}$

Gramáticas livres de contexto

- **Notação EBNF**
  - **EBNF** = **Extended Backus-Naur Form**;
  - **Simbologia**(ISO/IEC 14977)
    - A **definição** de um **não terminal** é feito com “=”;
    - **Símbolos não terminais** são escritos normalmente em **letras minúsculas**;
    - **Repetições de zero ou mais** vezes do **elemento** x são escritos com **chaves**, assim `{x}`;
    - O símbolo **separador de definições** é “`|`”;
    - O **operador** “$*$” realiza repetição assim: `3*x` é o mesmo que `xxx`;
    - Para indicar que um **elemento** x é **opcional**, utilizam-se **colchetes**, assim `[x]`;
    - Uma **definição** termina com “`;`”;
    - **Agrupamentos** são escritos com auxílio de parênteses, “`(`” e “`)`”;
    - **Concatenação** é escrita com vírgula, “`,`”;
    - **Símbolos** ou **cadeias terminais** são escritos entre “ e ” ou entre ‘ e ’;
    - **Comentários** são escritos entre `(*` e `*)`;
    - **Sequencias especiais** (textos arbitrários) são escritas entre `?` e `?`.

Gramáticas livres de contexto

- **Notação EBNF**
  - **Exemplo**
    - **BNF**

⟨*expression*⟩ ::= ⟨*expression*⟩ '+' ⟨*term*⟩
| ⟨*expression*⟩ '-' ⟨*term*⟩
| ⟨*term*⟩

⟨*term*⟩ ::= ⟨*term*⟩ '*' ⟨*factor*⟩
| ⟨*term*⟩ '/' ⟨*factor*⟩
| ⟨*factor*⟩

⟨*factor*⟩ ::= `digit`
| '(⟨*expresion*⟩')'

    - **EBNF**

```
expression = term, {('+'|'-'), term};
term = factor, {('*'|'/'), factor};
factor = digit | '(', expression, ')';
```

Gramáticas livres de contexto

- **Descrição com diagramas sintáticos**
  - É uma **forma alternativa** ao **(E)BNF**, representando a **estrutura sintática** de uma linguagem na forma de um **grafo**;
  - Considerar **expressões aritméticas** simples em **BNF**:

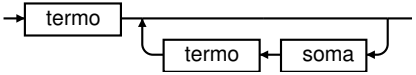
    ⟨*exp*⟩ ::= ⟨*exp*⟩ ⟨*soma*⟩ ⟨*termo*⟩ | ⟨*termo*⟩
    ⟨*soma*⟩ ::= '+' | '-'
    ⟨*termo*⟩ ::= ⟨*termo*⟩ ⟨*mult*⟩ ⟨*fator*⟩ | ⟨*fator*⟩
    ⟨*mult*⟩ ::= '*'
    ⟨*fator*⟩ ::= '(' ⟨*exp*⟩ ')' | num
  - Que, em **EBNF** é:

```
exp = termo,{soma,termo};
soma = '+'|'-';
termo = fator,{mult,termo};
mult = '*';
fator = '(',exp,')' | num;
```

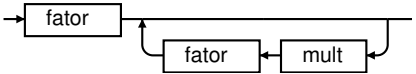
Gramáticas livres de contexto

- **Descrição com diagramas sintáticos**
  - Diagramas sintáticos são traduzidos diretamente de EBNF:

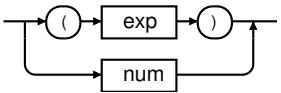
exp:



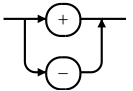
termo:



fator:



soma:



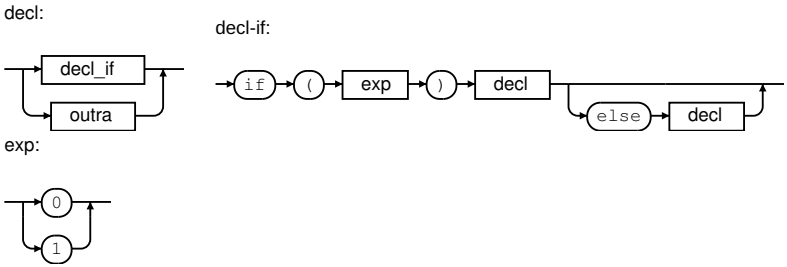
mult:

Gramáticas livres de contexto

- **Descrição com diagramas sintáticos**
  - Exemplo das **declarações** `if` **simplificadas** em EBNF:

```
decl = decl_if|outra;
decl_if = 'if','(',exp,')',decl,['else',decl];
exp = '0'|'1'
```

    - **Diagramas sintáticos correspondentes**



Árvores de análise sintática

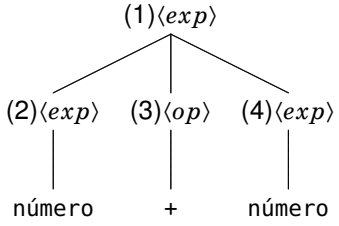
- **Definições**
  - Uma **árvore de análise sintática** sobre uma **gramática** G é uma árvore com raiz rotulada e com as seguintes propriedades:
    - **(1)** Cada **nó** é **rotulado** com um **terminal**, um **não-terminal** ou ϵ;
    - **(2)** O **nó-raiz** é **rotulado** com o **símbolo inicial** S;
    - **(3)** Cada **nó-folha** é **rotulado** com um **terminal** ou com ϵ;
    - **(4)** Cada **nó não-folha** é rotulado com um **não-terminal**;
    - **(5)** Se um **nó com rótulo** $A \in N$ tiver n filhos rotulados $X_1, X_2, \ldots, X_n$ (terminais ou não-terminais), então $A \rightarrow X_1\, X_2\, X_3\, \ldots\, X_n \in P$.
  - Na árvore de análise sintática, a **derivação à esquerda** corresponde um **percurso** em **pré-ordem** na árvore e a **derivação à direita** corresponde a um **percurso** em **pós-ordem reverso** na árvore;
  - Uma gramática G é **ambígua** de existir uma **cadeia** $w \in L(G)$ com **duas ou mais árvores** de análise sintática distintas.

Árvores de análise sintática

- **Exemplo de derivação** (gramática do slide 13)
  - **Derivação à esquerda**

$$
\begin{aligned}
exp &\Rightarrow exp\ op\ exp && (1)\\
&\Rightarrow \texttt{número}\ op\ exp && (2)\\
&\Rightarrow \texttt{número} + exp && (3)\\
&\Rightarrow \texttt{número} + \texttt{número} && (4)
\end{aligned}
$$

Corresponde ao **percorrimento pré-ordem** na árvore de análise sintática (análise **top-down**):

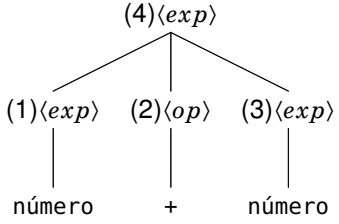


Árvores de análise sintática

- **Exemplo de derivação** (gramática do slide 13)
  - **Derivação à direita**

$$
\begin{aligned}
exp &\Rightarrow exp\ op\ exp && (1)\\
&\Rightarrow exp\ op\ \texttt{número} && (2)\\
&\Rightarrow exp + \texttt{número} && (3)\\
&\Rightarrow \texttt{número} + \texttt{número} && (4)
\end{aligned}
$$

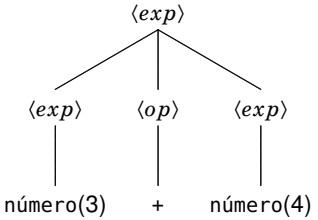
O **reverso** desta derivação corresponde ao **percorrimento pós-ordem** na árvore de análise sintática (análise **bottom-up**):



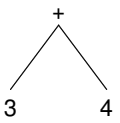
Árvores sintáticas abstratas

- **Definição**
  - São árvores que simplificam **a estrutura sintática**;
  - Para **transformar** uma **árvore de análise sintática** em uma **árvore sintática abstrata**, basta **mover** os símbolos **terminais** representando as **operações** e **comandos** para os **nós-raiz** das **subárvores**, deixando os **operandos** como seus nós filhos;
  - Por exemplo, a expressão `3+4` produz as árvores:

Árvore de análise sintática



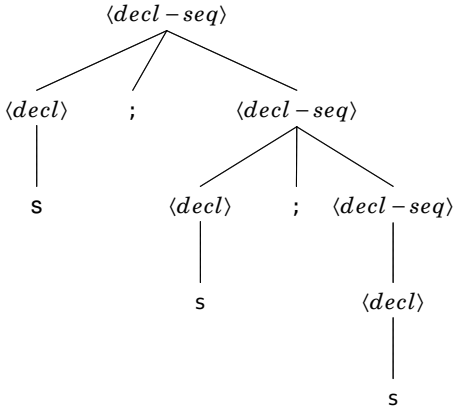
Árvore sintática abstrata



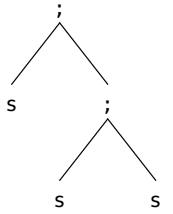
Árvores sintáticas abstratas

- **Exemplos de árvores sintáticas abstratas**

Árvore de análise sintática



Árvore sintática abstrata



Gramáticas ambíguas

- Uma gramática é **ambígua** se alguma sentença produzida por ela possui mais do que uma derivação (à direita ou à esquerda).
- **Exemplo**
  - A gramática a seguir é ambígua (teste com `34-3*42`):

    ⟨*exp*⟩ ::= ⟨*exp*⟩ ⟨*op*⟩ ⟨*exp*⟩ | (exp) | número

    ⟨*op*⟩ ::= + | - | *
  - Mas pode ser reescrita assim (adiciona precedência e associatividade):

    ⟨*exp*⟩ ::= ⟨*exp*⟩ ⟨*soma*⟩ ⟨*termo*⟩ | ⟨*termo*⟩

    ⟨*soma*⟩ ::= + | -

    ⟨*termo*⟩ ::= ⟨*termo*⟩ ⟨*mult*⟩ ⟨*fator*⟩ | ⟨*fator*⟩

    ⟨*mult*⟩ ::= *

    ⟨*fator*⟩ ::= (⟨*exp*⟩) | número

Gramáticas ambíguas

- **Exemplo**
  - Considerar a gramática:

    ⟨*decl*⟩ ::= ⟨*decl-if*⟩ | outra

    ⟨*decl-if*⟩ ::= if(⟨*exp*⟩) decl
    | if(⟨*exp*⟩) ⟨*decl*⟩ else ⟨*decl*⟩

    ⟨*exp*⟩ ::= 0|1
  - Verifique que ela é ambígua gerando árvores de derivações para a cadeia:

    ```
    if (0) if(1) outra else outra
    ```
  - Este é o problema do **else pendente**.

Referências bibliográficas

AHO, A. V.; SETHI, R.; LAM, M. S. **Compiladores: princípios, técnicas e ferramentas**. 2. ed. [s.l.] Pearson, 2007.

COOPER, K.; TORCZON, L. **Construindo compiladores**. 2. ed. Rio de Janeiro: Elsevier, 2014.

LOUDEN, K. C. **Compiladores: princípios e práticas**. [s.l.] Pioneira Thomson Learning, 2004.