# Compiladores

Prof. Marc Antonio Vieira de Queiroz

Ciência da Computação - UNIFIL
LAB 7
*marc.queiroz@unifil.br*

14/05/2014

# Roteiro I

# Verificando a linguagem gerada por uma gramática I

Embora seja raro que programadores testem a gramática para a linguagem reconhecida é interessante saber para um conjunto de produções geram uma linguagem particular.

Considere a seguinte gramática:

$$S \rightarrow ( \ S \ ) \ S \ \mid \ \epsilon$$

Figura: 1.

# Linguagens Livre de Contexto comparadas a Expressões Regulares I

Gramáticas apresentam uma notação mais poderosa que expressões regulares.

Toda construção que pode ser escrita por uma expressão regular pode se descrita por uma gramática, mas não vice-versa. Alternativamente, toda linguagem regular é uma linguagem livre de contexto, mas não vice-versa. Por exemplo: A expressão regular (a—b)*aab e a gramática abaixo descrevem a mesma linguagem, o conjunto de strings começando por a s e b s e terminando em abb.

# Linguagens Livre de Contexto comparadas a Expressões Regulares II

$$
\begin{array}{lcl}
A_0 & \rightarrow & aA_0 \mid bA_0 \mid aA_1 \\
A_1 & \rightarrow & bA_2 \\
A_2 & \rightarrow & bA_3 \\
A_3 & \rightarrow & \epsilon
\end{array}
$$

Figura: 2.

1. Para cada estado i do NFA, crie um não terminal $A_i$.
2. Se o estado i tiver uma transição para o estado j para entrada a, adicione a produção $A_i \rightarrow aA_j$. Se o estado i for para o estado j na entrada $\epsilon$, adicione a produção $A_i \rightarrow A_j$.
3. Se $A_i$ for um estado de aceitação então $A_i \rightarrow \epsilon$.
4. Se i for um estado inicial, faça $A_i$ ser o símbolo inicial da gramática.

# Exercícios seção 4.2

Exercícios da seção 4.2 do livro Compiladores: Princípios, Técnicas e Ferramentos. Autor Aho.

Disponível no portal do aluno da Unifil.

# Escrevendo uma gramática

As gramáticas são capazes de descrever a maioria, mas não todas, das sintaxes de linguagens de programação. Para começar, o requisito de que os identificadores devem ser declarados antes de ser utilizados, não podem ser descritos por uma linguagem livre de contexto.

# Análise Léxica versus Análise Sintática I

Como visto anteriormente tudo que pode ser descrito por uma expressão regular pode ser descrita por uma gramática.

Por que utilizar expressões regulares para definir a análise léxica de uma linguagem?

- Separar a estrutura sintática de uma linguagem em partes léxicas e não léxicas permite a modularização do front end do compilador em dois componentes de tamanho gerenciável.
- As regras do analisador léxica são relativamente simples, e para descrevê-las não é necessário mais do que gramáticas.
- Expressões regulares são geralmente mais concisas e fáceis de entender tokens do que as gramáticas.

# Análise Léxica versus Análise Sintática II

- Analisadores léxicos mais eficientes podem ser construídos automaticamente a partir de expressões regulares do que de gramáticas arbitrárias.

# Eliminando ambiguidade:

Algumas vezes a gramática ambígua pode ser reescrita para eliminar a ambiguidade.
Exemplo:

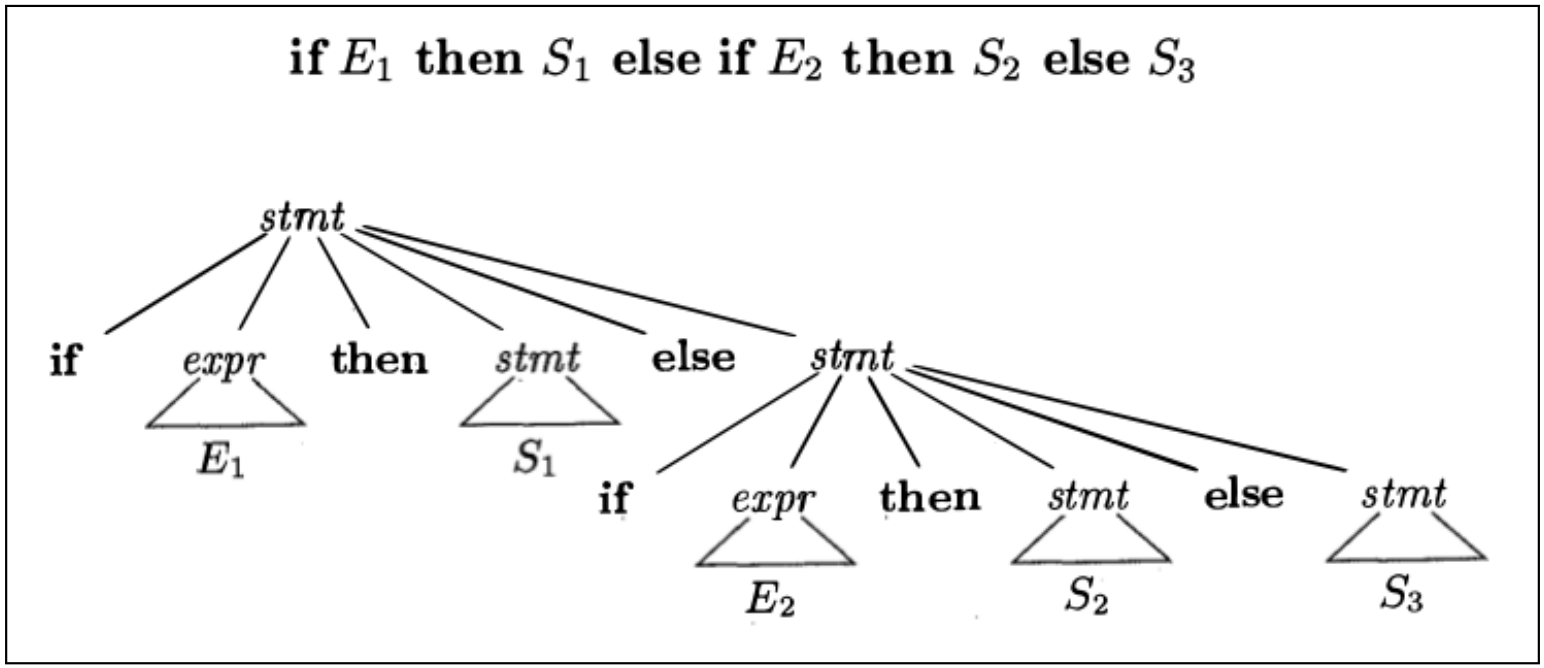


Figura: 3.

Other significa qualquer outro comando (statement).

# Exemplo de ambiguidade I

**if** $E_1$ **then if** $E_2$ **then** $S_1$ **else** $S_2$

Figura: 4.

# Exemplo de ambiguidade

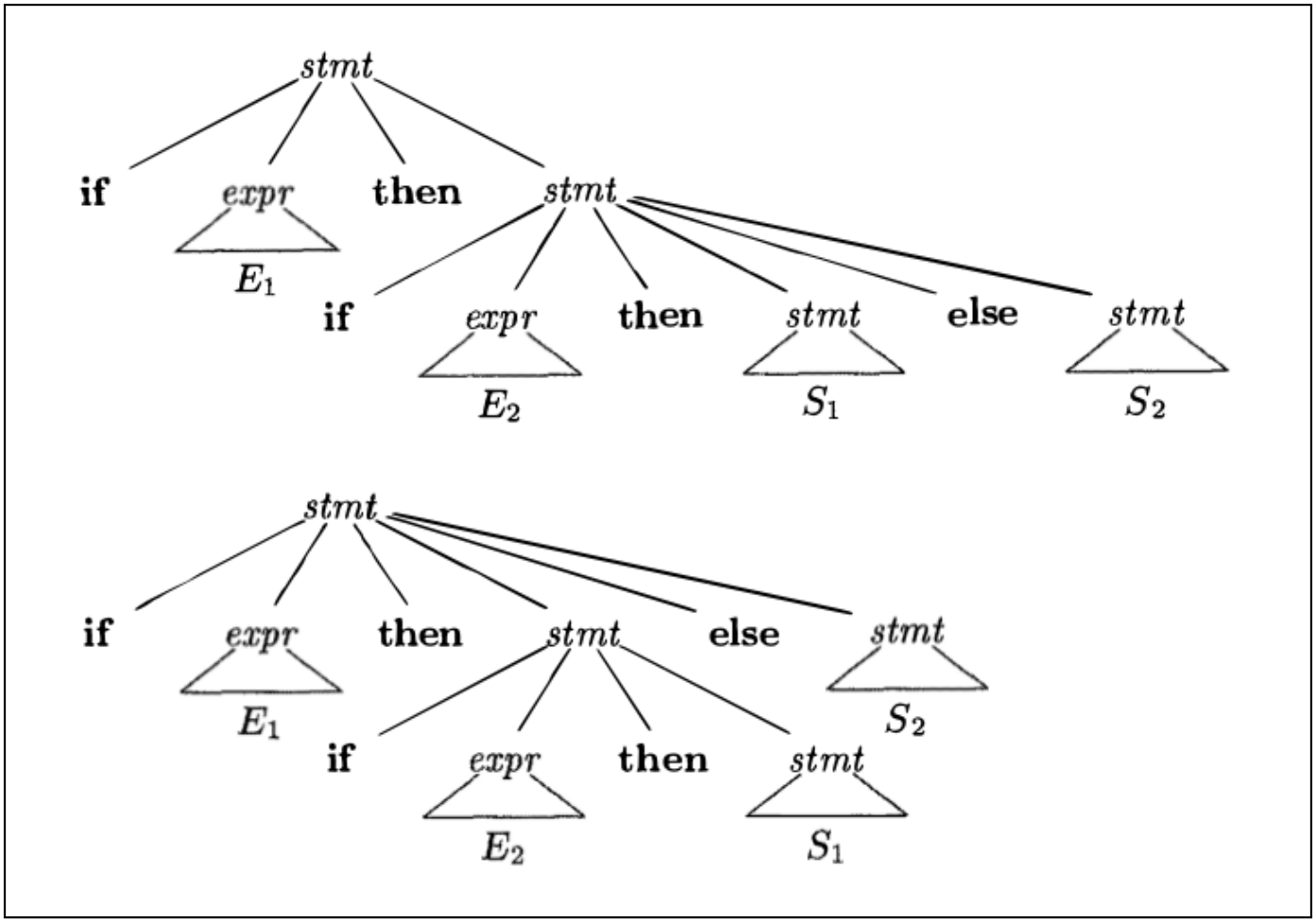


Figura: 5.

# Exemplo de gramática

Exemplo de gramática não ambígua para os comandos if-then-else

| | | |
|---|---|---|
| *stmt* | → | *matched_stmt* |
| | \| | *open_stmt* |
| *matched_stmt* | → | **if** *expr* **then** *matched_stmt* **else** *matched_stmt* |
| | \| | **other** |
| *open_stmt* | → | **if** *expr* **then** *stmt* |
| | \| | **if** *expr* **then** *matched_stmt* **else** *open_stmt* |

Figura: 6.