Linguagens Formais e Autômatos

Aula 14 - Ambiguidade

Prof. Dr. Daniel Lucrédio
Departamento de Computação / UFSCar
Última revisão: ago/2015

Referências bibliográficas

- **Introdução à teoria dos autômatos, linguagens e computação / John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman** ; tradução da 2.ed. original de Vandenberg D. de Souza. - Rio de Janeiro : Elsevier, 2002 (Tradução de: Introduction to automata theory, languages, and computation - ISBN 85-352-1072-5)
  - Capítulo 5 - Seção 5.4
- **Introdução à teoria da computação / Michael Sipser** ; tradução técnica Ruy José Guerra Barretto de Queiroz ; revisão técnica Newton José Vieira. -- São Paulo : Thomson Learning, 2007 (Título original : Introduction to the theory of computation. "Tradução da segunda edição norte-americana" - ISBN 978-85-221-0499-4)
  - Capítulo 2 - Seção 2.1

Ambiguidade

- Considere as seguintes frases (verídicas), extraídas de um sistema de pedidos de um almoxarifado de um banco
  - “Armário para funcionário de aço”
  - “Cadeira para gerente sem braços”
- Quem é de aço? O armário ou funcionário?
- Quem não tem braços? A cadeira ou o gerente?
- O problema é a ambiguidade

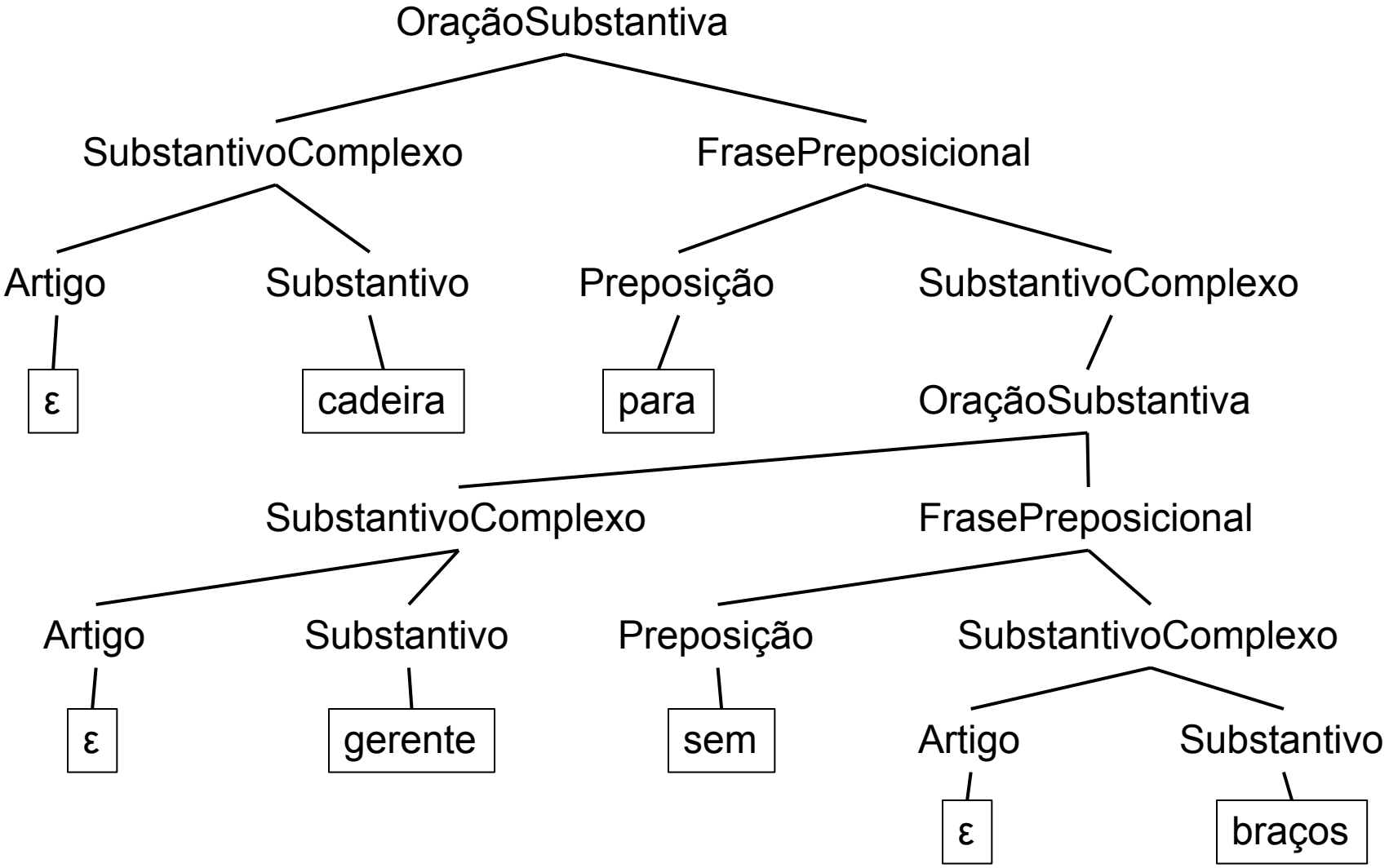
Ambiguidade

- Gramática da língua portuguesa (trecho)

```
OraçãoSubstantiva → SubstantivoComplexo
                  |  SubstantivoComplexo FrasePreposicional ;
FraseVerbal → VerboComplexo
                  |   VerboComplexo FrasePreposicional ;
FrasePreposicional → Preposição SubstantivoComplexo ;
SubstantivoComplexo → Artigo Substantivo | OraçãoSubstantiva
  ;
VerboComplexo → Verbo | Verbo OraçãoSubstantiva ;
Artigo → o | a | um | uma | ε ;
Substantivo → armário | funcionário | gerente | cadeira |
  braços | aço ;
Verbo → gosta | brinca | olha ;
Preposição → para | com | de | sem ;
```

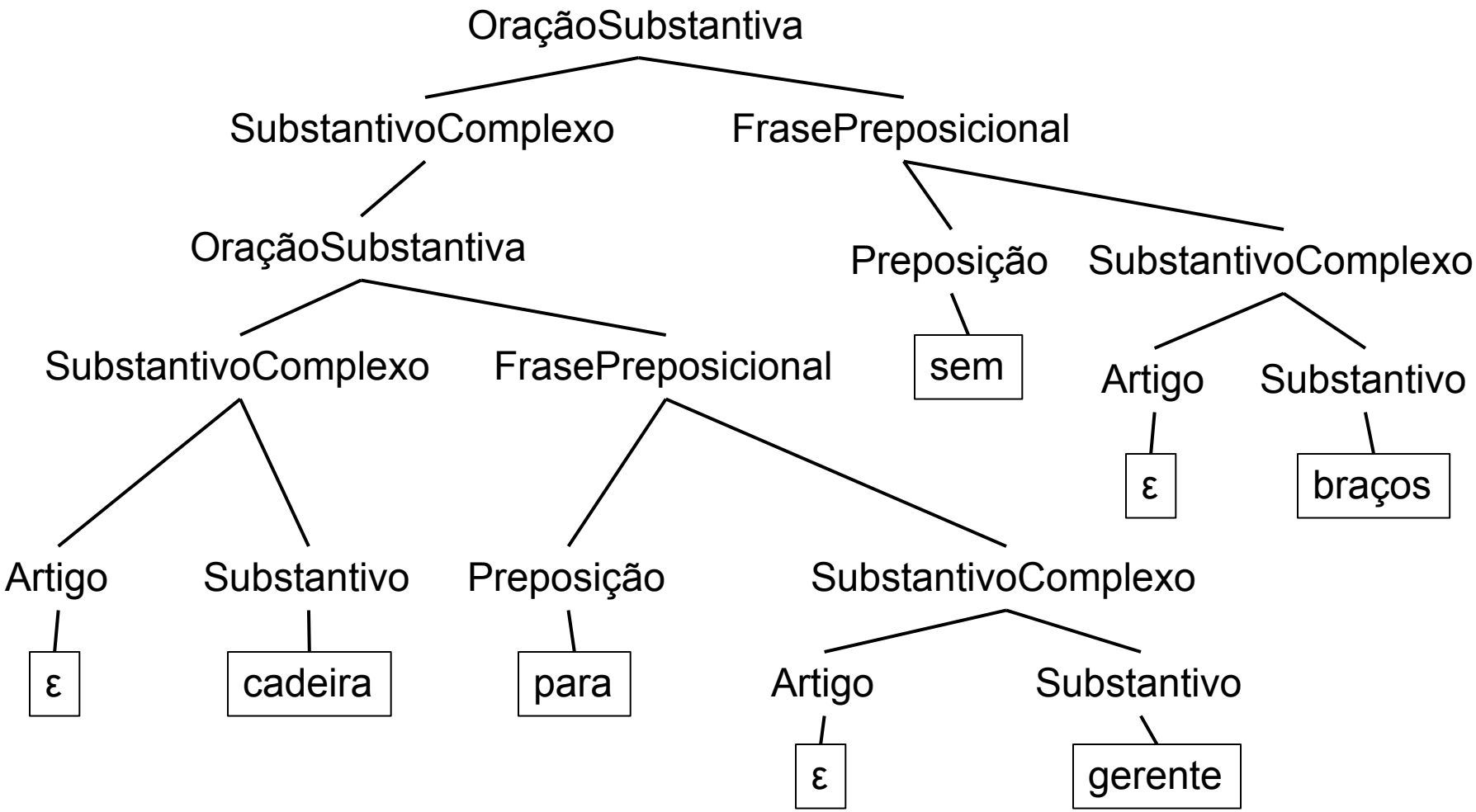
Ambiguidade

- Análise sintática da frase (1): cadeira para gerente sem braços



Ambiguidade

- Análise sintática da frase (2): cadeira para gerente sem braços



Ambiguidade

- Outro exemplo, gramática à direita
  - Encontre derivações mais à esquerda para a cadeia a + b*a
- Respostas:
- E ⇒ E + E ⇒ I + E ⇒ a + E ⇒ a + E * E ⇒ a + I * E ⇒ a + b * E ⇒ a + b * I ⇒ a + b * a
- E ⇒ E * E ⇒ E + E * E ⇒ I + E * E ⇒ a + E * E ⇒ a + I * E ⇒ a + b * E ⇒ a + b * I ⇒ a + b * a

E → I
E → E + E
E → E * E
E → (E)
I → a
I → b
I → Ia
I → Ib
I → I0
I → I1

Ambiguidade

- A diferença entre as árvores e as derivações mais à esquerda é significativa
  - Dependendo de qual árvore usar, o gerente pode ficar sem braços
    - Cadeira para ___
    - ___ sem braços
  - Dependendo de qual derivação à esquerda usar, a adição pode ocorrer antes da multiplicação
    - a + ___
    - ___ * a

Ambiguidade

- Gramáticas são usadas para dar estrutura para programas, documentos, etc
  - Supõe-se que essa estrutura é única
  - Caso não seja, podem ocorrer problemas
- Nem toda gramática fornece estruturas únicas
  - Ambiguidade
  - Algumas vezes é possível reprojetar a gramática para eliminar a ambiguidade
  - Em outras vezes, isso é impossível
    - Ou seja, existem linguagens “inerentemente ambíguas”
    - Isto é: toda gramática para esta linguagem será fatalmente ambígua

Ambiguidade

- O que caracteriza ambiguidade
  - A existência de duas ou mais árvores de análise sintática para pelo menos uma cadeia da linguagem
- Formalmente:
  - Uma CFG G = (V,T,P,S) é ambígua se existe pelo menos uma cadeia w em T* para o qual podemos encontrar duas árvores de análise sintática diferentes, cada qual com uma raiz identificada como S e um resultado w.
  - Se TODAS as cadeias tiverem no máximo uma árvore de análise sintática, a gramática é não-ambígua

Ambiguidade

- Também pode-se pensar na ambiguidade em termos de derivações
- Teorema: Para cada gramática G = (V,T,P,S) e cadeia w em T*, w tem duas árvores de análise sintática distintas se e somente se w tem duas derivações mais à esquerda distintas a partir de S
  - Corolário: Se para uma gramática G = (V,T,P,S), e uma cadeia w em T*, for possível encontrar duas derivações mais à esquerda distintas, G é ambígua
- O mesmo vale para derivações mais à direita

Exercícios

- Prove que a seguinte gramática é ambígua
  - $S \rightarrow aS \mid aSbS \mid \varepsilon$
- Sugestão: mostre que a cadeia aab tem duas:
  - Árvores de análise sintática, derivações mais à esquerda ou derivações mais à direita
- Resposta (usando derivações mais à esquerda):
  - $S \rightarrow aS \rightarrow aaSbS \rightarrow aabS \rightarrow aab$
  - $S \rightarrow aSbS \rightarrow aaSbS \rightarrow aabS \rightarrow aab$

Exercícios

- Prove que a seguinte gramática é ambígua:
  - $S \rightarrow aSbS \mid aS \mid \varepsilon$
- Sugestão: mostre que a cadeia aab tem duas:
  - Árvores de análise sintática, derivações mais à esquerda ou derivações mais à direita
- Resposta (usando derivações mais à direita):
  - $S \rightarrow aSbS \rightarrow aSb \rightarrow aaSb \rightarrow aab$
  - $S \rightarrow aS \rightarrow aaSbS \rightarrow aaSb \rightarrow aab$

Exercícios

- Considere a seguinte gramática:
  - S → ε | SS | “if” C “then” S “else” S | “if” C “then” S
  - S → “System.out.print(” STRING “)”
  - C → “(i <= 1)”
  - STRING → “”[^””]*””
- A gramática é ambígua?
- Sugestão, analise a seguinte cadeia:

```
System.out.print('Teste')
if (i <= 1) then
   System.out.print('Alo Mundo')
   if (i< = 1) then
      System.out.print('Adeus Mundo')
   else
      System.out.print('Alo de novo')
System.out.print('Fim do programa')
```

Ambiguidade

- Eliminando a ambiguidade
  - Problemas
- Primeiro: saber se uma gramática é ambígua é um problema indecidível, ou seja, descobrir que uma gramática é ambígua depende de análise, exemplos e um pouco de sorte!
- Segundo: existem linguagens inerentemente ambíguas, ou seja, TODA CFG será ambígua
- Terceiro: mesmo para uma linguagem que não é inerentemente ambígua, não existe um algoritmo para remover a ambiguidade

Ambiguidade

- Existem algumas técnicas bem conhecidas, para alguns casos de ambiguidade
- Primeira técnica: forçar a precedência de terminais introduzindo novas regras
- Segunda técnica: modificar ligeiramente a linguagem
- Terceira técnica: forçar a precedência de terminais diretamente no analisador

Eliminando ambiguidade

- Forçando a precedência modificando-se as regras:
  - No exemplo à direita, há duas causas para ambiguidade:
    - Terminais “+” e “*” tem a mesma precedência
    - Terminais idênticos podem se agrupar a partir da esquerda ou direita
- Para resolver, vamos forçar a precedência e associatividade

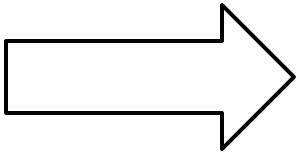
```
E → I
E → E + E
E → E * E
E → (E)
I → a
I → b
I → Ia
I → Ib
I → I0
I → I1
```

Eliminando a ambiguidade

- Vamos pensar em expressões aritméticas em termos de:
- Fatores
  - Elementos indivisíveis, ou seja, um fator não pode ser “quebrado” por um * ou +
  - Neste caso: identificadores e expressões entre parênteses
- Termos
  - Elementos compostos de fatores multiplicados. Ou seja, um termo é uma multiplicação de um ou mais fatores, que não pode ser “quebrada” por um +
- Expressões
  - Elementos compostos de termos somados. Ou seja, uma expressão é a soma de um ou mais termos

Eliminando ambiguidade

```
E → I
E → E + E
E → E * E
E → (E)
I → a
I → b
I → Ia
I → Ib
I → I0
I → I1
```



```
E → T | E + T
T → F | T * F
F → I | (E)
I → a
I → b
I → Ia
I → Ib
I → I0
I → I1
```

Eliminando ambiguidade

- Faça o teste agora, para a cadeia a + b * a (com derivações mais à esquerda)

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow$

$F + T \Rightarrow I + T \Rightarrow a + T \Rightarrow$

$a + T * F \Rightarrow a + F * F \Rightarrow$

$a + I * F \Rightarrow a + b * F \Rightarrow$

$a + b * I \Rightarrow a + b * a$

$E \rightarrow T \mid E + T$

$T \rightarrow F \mid T * F$

$F \rightarrow I \mid (E)$

$I \rightarrow a$

$I \rightarrow b$

$I \rightarrow Ia$

$I \rightarrow Ib$

$I \rightarrow I0$

$I \rightarrow I1$

Eliminando ambiguidade

- Segunda técnica: modificando ligeiramente a linguagem

```
S → ε | SS | “if” C “then” S “else” S “endif” | if” C “then” S “endif”
S → “System.out.print(” STRING “)”
C → “(i <= 1)”
STRING → “’”[^””’”]*””’”
```

- Veja o resultado

```
System.out.print(‘Teste’)
if (i <= 1) then
   System.out.print(‘Alo Mundo’)
   if (i< = 1) then
      System.out.print(‘Adeus Mundo’)
   else
      System.out.print(‘Alo de novo’)
   endif
endif
System.out.print(‘Fim do programa’)
```

Eliminando ambiguidade

- Terceira técnica: forçar a precedência de terminais diretamente no analisador

Gramática
(com ambiguidade)

```
E → I
E → E + E
E → E * E
E → (E)
I → a
I → b
I → Ia
I → Ib
I → I0
I → I1
```

+

Regras de precedência
e associatividade

```
%left '+'
%left '*'
```

Exemplo no analisador YACC

Eliminando ambiguidade

- Sempre que houver um ponto de dúvida (ambiguidade), o YACC resolve olhando a precedência e associatividade
- Ex: a + b * a (no YACC)
  - Após ler o caractere “a”, o YACC faz a inferência usando as regras I → a e E → I, resultando em E + b * a
  - Após ler o caractere “+”, não há inferência possível
  - Após ler o caractere “b”, o YACC faz a inferência usando as regras I → b e E → I, resultando em E + E * a
  - Neste ponto, ocorre um conflito (decorrente da ambiguidade da gramática)
    - O YACC pode:
      - Fazer a inferência, reduzindo E + E para E, resultando em E * a
      - Continuar a leitura, lendo o caractere “*”, para só depois fazer a inferência

Eliminando a ambiguidade

- Como resolver esse conflito?
  - Através da precedência e associatividade dos terminais
  - No exemplo abaixo, * tem precedência sobre o +
  - E ambos são associativos à esquerda

```
%left '+'
%left '*'
```

- Ou seja, no momento o YACC está na seguinte configuração
  - E + E <YACC está aqui> * a
- Ele então olha para o terminal mais à direita do seu lado esquerdo (+), e o terminal mais à esquerda do seu lado direito (*)
  - Neste caso, * tem precedência sobre +, então a decisão é não inferir neste momento, e sim continuar lendo:
  - E + E * <YACC> a → E + E * a <YACC> → E + E * E → E + E → E

Ambiguidade inerente

- Algumas linguagens são inerentemente ambíguas
- Ex: L = $\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$
- Ex: L = $\{a^i b^j c^k \mid i = j \text{ ou } j = k\}$
- Demonstração é complexa
  - Envolve analisar profundamente a linguagem e a característica de suas cadeias
- Para estes casos, é impossível remover a ambiguidade

Fim

Aula 14 - Ambiguidade