UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br
Departamento de Ciências de Computação

SCC-505 - Capítulo 2
Linguagens Livres de Contexto e Autômatos de Pilha (versão 2)

João Luís Garcia Rosa[1]

[1]Departamento de Ciências de Computação
Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo
http://www.icmc.usp.br/~joaoluis

2010

Sumário

Sumário

Definição

- **Definição**: Uma gramática G é **livre de contexto** (GLC) se v é apenas um símbolo não terminal para toda produção $v \rightarrow w$ em P ($v \in V$ e $w \in (\Sigma \cup V)^*$). Uma linguagem L sobre algum alfabeto terminal Σ é livre de contexto (LLC) se pode ser gerada por uma GLC.
- Então a linguagem dos palíndromos, a linguagem dos parênteses casados e a linguagem construída de cadeias de números iguais de *a*'s e *b*'s são todas livres de contexto, porque em todas foi mostrada uma GLC.

Exemplo

- **Exemplo**: A seguinte GLC gera todas as cadeias sobre o alfabeto terminal $\Sigma = \{0, 1\}$ com um número igual de 0's e 1's.

$$\Sigma = \{0, 1\}$$
$$V = \{S, A, B\}$$

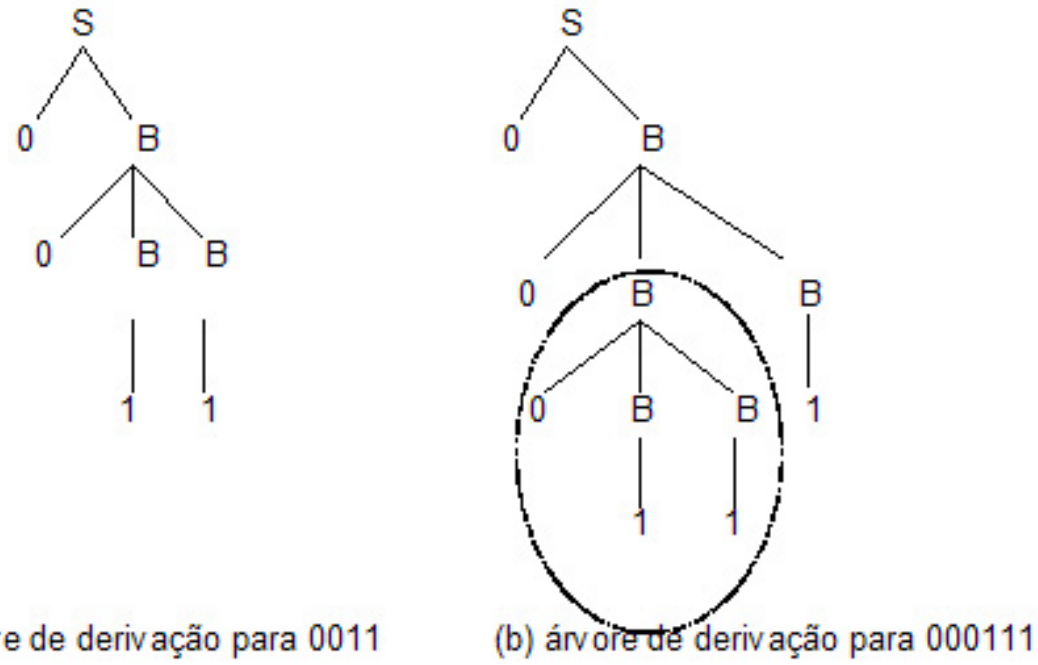
P compreende as seguintes produções:

$$S \rightarrow 0B|1A$$
$$A \rightarrow 0|0S|1AA$$
$$B \rightarrow 1|1S|0BB$$

As derivações livres de contexto têm uma representação em árvore muito útil e elegante. Por exemplo, a derivação das cadeias 0011 e 000111 usando a gramática do Exemplo acima é mostrada na próxima figura.

Sumário

Árvore de derivação



(a) árvore de derivação para 0011 (b) árvore de derivação para 000111

- A Figura (a) representa as derivações:
  - $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 001B \Rightarrow 0011$: **derivação mais a esquerda**
  - $S \Rightarrow 0B \Rightarrow 00BB \Rightarrow 00B1 \Rightarrow 0011$: **derivação mais a direita**.

Árvore de derivação

- Se uma cadeia pode ser derivada legalmente por uma GLC, então pode-se descrever esta derivação por uma árvore T com as seguintes propriedades:
  1. A raiz é rotulada com o símbolo inicial S;
  2. Todo nó que não é uma folha é rotulado com uma variável - um símbolo de V;
  3. Todo nó que é uma folha é rotulado com um terminal - um símbolo de Σ (ou possivelmente com λ);
  4. Se o nó N é rotulado com um A, e N tem k descendentes diretos $N_1, ..., N_k$, rotulados com símbolos $A_1, ..., A_k$, respectivamente, então existe uma produção da gramática da forma $A \rightarrow A_1 A_2 ... A_k$;
  5. Uma expressão derivada por alguma derivação pode ser obtida pela leitura das folhas da árvore associada com esta derivação, da esquerda para a direita.

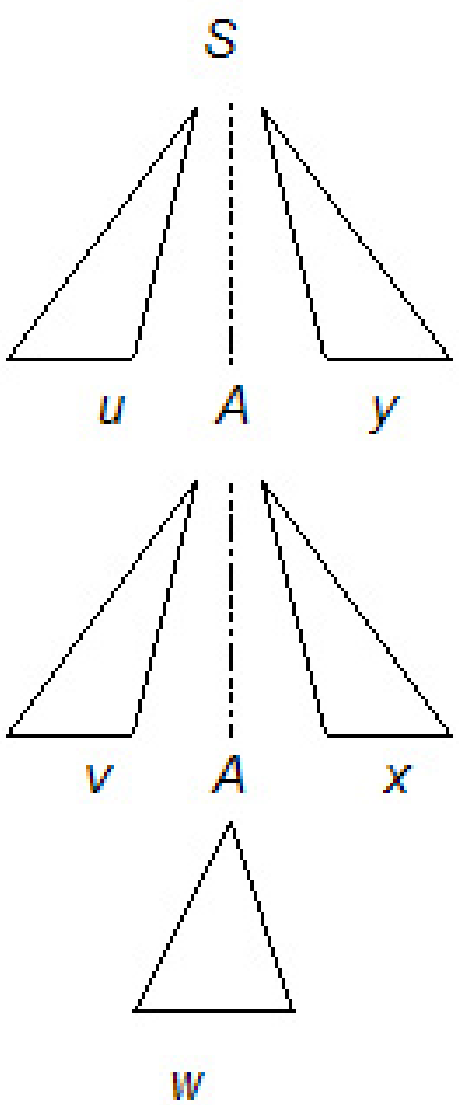
Árvore de derivação

- Dada uma árvore para uma derivação livre de contexto, define-se o **comprimento** de um caminho da raiz à folha como sendo o número de não terminais neste caminho. A altura de uma árvore é o comprimento de seu caminho mais longo. (Logo, na Figura 2 (a), a altura da árvore é 3.)
- Considere a sub-árvore assinalada na Figura 2 (b). É uma árvore-B legal; isto é, é uma árvore de derivação legal usando a gramática do Exemplo 3 exceto que a raiz é um B e não um S.
- Agora se for considerada qualquer árvore de derivação legal para uma cadeia nesta linguagem e substituída qualquer sub-árvore-B nela com a sub-árvore assinalada, obtém-se uma outra árvore de derivação legal.
- Este é o significado de “livre de contexto” do ponto de vista de representações de árvore.

Árvore de derivação

- Vai-se mostrar agora como a aplicação sistemática deste princípio de substituição de sub-árvores pode ser usado para estabelecer um resultado chave sobre a estrutura das linguagens livres de contexto.
- Suponha que haja uma árvore de derivação T para uma cadeia z de terminais gerada por alguma gramática G, e suponha depois que o símbolo não terminal A aparece duas vezes em algum caminho, como mostrado na Figura 6, onde $z = uvwxy$.
- Aqui a árvore-A inferior deriva a cadeia terminal w, e a árvore-A superior deriva a cadeia vwx.
- Como a gramática é livre de contexto, a substituição da árvore-A superior pela árvore-A inferior não afeta a legalidade da derivação.
- A nova árvore deriva a cadeia uwy.

Árvore de derivação



Árvore de derivação

- Por outro lado, se for substituída a árvore-A inferior pela árvore-A superior obtem-se uma árvore de derivação legal para a cadeia $uvvwxxy$, que pode-se escrever como uv^2wx^2y.
- Esta substituição superior-para-inferior pode ser repetida qualquer número finito de vezes, obtendo-se o conjunto de cadeias $\{uv^nwx^ny | n \geq 0\}$.
- Toda LLC infinita deve conter infinitos subconjuntos de cadeias desta forma geral.

Lema do Bombeamento para LLC

- **Lema do Bombeamento para Linguagens Livres de Contexto**. (Também conhecido como Teorema *uvwxy*). Seja L uma LLC. Então existem constantes p e q dependentes de L apenas, tal que se $z \in L$ com $|z| > p$, então z pode ser escrito como $uvwxy$ de tal forma que
  1. $|vwx| \leq q$;
  2. no máximo um (v ou x) está vazio; e
  3. para todo $i \geq 0$, as cadeias $uv^i wx^i y \in L$.
- Note o seguinte:
  1. Dada uma linguagem gerada por uma gramática que não é livre de contexto, não se pode deduzir imediatamente que ela também não é gerada por uma GLC.
  2. Mas se uma linguagem infinita não obedece o lema do bombeamento para linguagens livres de contexto, ela não pode ser gerada por uma GLC.

Sumário

Forma Normal de Greibach

- **Definição**: Diz-se que uma GLC está na **forma normal de Greibach** (FNG) se toda produção for da forma

$$A \rightarrow bW$$

onde $A \in V$, $b \in \Sigma$ e $W \in V^*$.

- **Teorema**: Seja G qualquer GLC. Então existe uma gramática equivalente G', isto é, $L(G) = L(G')$, na forma normal de Greibach.

Sumário

A Pilha como Processador de Linguagem

- Nesta seção vai-se estabelecer uma das equivalências mais importantes na ciência da computação teórica: prova-se que uma linguagem é livre de contexto se e somente se algum autômato de pilha possa aceitar a linguagem de uma forma precisa. Este resultado tem importância prática e teórica, porque um armazém de pilha é a base para muitos algoritmos usados no *parsing* de linguagens livres de contexto.
- Uma pilha, familiar em ciência de computação, é uma estrutura *last-in first-out* com uma operação “push” que adiciona à pilha e uma operação “pop” que remove o elemento do topo da pilha, se existir.

A Pilha como Processador de Linguagem

- A noção "pura" de uma pilha, descrita informalmente acima, aumentada pela noção de transição de estado não determinístico, provê um modelo de autômato completo para linguagens livres de contexto. Entretanto, certa estrutura adicional é também conveniente, e os próximos três exemplos mostram porque isto é assim.
- **Exemplo**: Considere a linguagem

$$\{w | w \in (a+b)^*, \text{ } w \text{ tem um número igual de } a\text{'s e } b\text{'s}\}.$$

- Esta linguagem é informalmente aceitável por uma pilha usando o seguinte algoritmo. Inicialmente, a pilha está vazia. Percorra a cadeia w da esquerda para a direita, e realize as seguintes operações, baseado no símbolo corrente no topo da pilha.

A Pilha como Processador de Linguagem

- Algoritmo:
  1. se a pilha estiver vazia e o símbolo corrente de w for um a, ponha A na pilha;
  2. se a pilha estiver vazia e o símbolo corrente de w for um b, ponha B na pilha;
  3. se o símbolo no topo da pilha for um A e o símbolo corrente de w for um a, coloque (*push*) um outro A na pilha;
  4. se o símbolo no topo da pilha for um B e o símbolo corrente de w for um b, coloque (*push*) um B na pilha;
  5. se o símbolo no topo da pilha for um A e o símbolo corrente de w for um b, retire (*pop*) da pilha;
  6. se o símbolo no topo da pilha for um B e o símbolo corrente de w for um a, retire (*pop*) da pilha.
- Uma cadeia w tem um número igual de a's e b's se e somente se, depois de processar w, a pilha estiver **vazia**.

A Pilha como Processador de Linguagem

- Segue um "programa" de pilha para o algoritmo anterior.
- Z_0 denota o fundo da pilha.
- A notação $\langle x, D, v \rangle$ significa "se x é o próximo símbolo da cadeia de entrada w e D é o símbolo no topo da pilha, então **substitua** D pela cadeia v."
- Pilha = cadeia, topo da pilha é o símbolo mais a esquerda.
- Então a descrição informal precedente pode ser rescrita como se segue:
  1. $\langle a, Z_0, AZ_0 \rangle$
  2. $\langle b, Z_0, BZ_0 \rangle$
  3. $\langle a, A, AA \rangle$
  4. $\langle b, B, BB \rangle$
  5. $\langle a, B, \lambda \rangle$
  6. $\langle b, A, \lambda \rangle$
  7. $\langle \lambda, Z_0, \lambda \rangle$: "regra-$\lambda$": quando não houver entrada, apaga Z_0

Sumário

O Autômato de Pilha

- **Exemplo**. Considere a linguagem $\{wcw^R \mid w \in (a+b)^*\}$.
- Esta linguagem é reconhecível por um autômato do tipo do último exemplo, mas um jeito mais conveniente de reconhecê-la é aumentar a maquinaria da pilha com uma estrutura de estados finitos consistindo de dois estados, um chamado "push" para processar a primeira metade da cadeia, o outro, chamado "pop", para a segunda metade.
- A entrada c engatilha a transição de "push" para "pop".
- Vai-se usar agora a notação $\langle q, x, D, q', v \rangle$ para significar "se a máquina de estados estiver no estado q, x for o próximo símbolo da cadeia de entrada e D for o símbolo no topo da pilha, então a máquina de estados pode mudar para o estado q' e substituir D pela cadeia v."

O Autômato de Pilha

- Deve-se usar o símbolo Z para um símbolo de pilha arbitrário.
- Então a instrução $\langle push, a, Z, push, AZ \rangle$ é o resumo para as três instruções:
  - $\langle push, a, Z_0, push, AZ_0 \rangle$
  - $\langle push, a, A, push, AA \rangle$
  - $\langle push, a, B, push, AB \rangle$
- Usando esta notação aumentada, pode-se descrever o **autômato de pilha** (APN) como:
  1. $\langle push, a, Z, push, AZ \rangle$
  2. $\langle push, b, Z, push, BZ \rangle$
  3. $\langle push, c, Z, pop, Z \rangle$
  4. $\langle pop, a, A, pop, \lambda \rangle$
  5. $\langle pop, b, B, pop, \lambda \rangle$
  6. $\langle pop, \lambda, Z_0, pop, \lambda \rangle$

O Autômato de Pilha

- Note que o APN irá parar (não terá instruções para seguir) se ele ler um *a* enquanto *B* estiver no topo da pilha, no estado *pop*.
- A seguinte tabela mostra as configurações de pilha sucessivas nos segmentos de cadeia sucessivos para entrada *abbcbba*:

cadeia	pilha
abbcbba	Z_0
bbcbba	AZ_0
bcbba	BAZ_0
cbba	$BBAZ_0$
bba	$BBAZ_0$
ba	BAZ_0
a	AZ_0
	Z_0

O Autômato de Pilha

- **Definição**: Um **autômato de pilha** (APN) (não determinístico) é uma séptupla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, onde:
  1. Q é um conjunto finito de estados;
  2. Σ é um conjunto finito de símbolos de entrada;
  3. Γ é um conjunto finito de símbolos de pilha;
  4. δ é o conjunto de transições $\langle q, x, Z, q', \sigma \rangle$ que também se escreve na notação $(q', \sigma) \in \delta(q, x, Z)$ tal que se possa considerar δ como uma função de transição:

     $$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \text{subconjuntos finitos de } Q \times \Gamma^*$$
  5. q_0 é o estado inicial;
  6. Z_0 é o símbolo de pilha inicial;
  7. $F \subset Q$ é um conjunto de estados de aceitação.

Descrição Instantânea

- Dada esta maquinaria, pode-se falar sobre uma **descrição instantânea** (DI) de um APN M:
  - Uma DI para um APN é uma tripla (q, w, σ) onde q é um estado, $w = x_1 x_2 ... x_n$ é uma cadeia de símbolos de entrada ainda a ser lidos com o APN correntemente lendo x_1, e $\sigma = Z_1 Z_2 ... Z_m$ é a cadeia de símbolos na pilha com Z_1 no topo e Z_m no fundo.
- As transições entre DI's mapeiam DIs para DIs não-deterministicamente de duas formas: Se M estiver no estado q com o símbolo de entrada corrente x e elemento de pilha corrente A, então
  1. Se $\langle q, x, A, q', A_1 ... A_k \rangle$ para $x \in \Sigma$ é uma quíntupla do APN, então M pode (não determinismo!) causar a seguinte transição entre DI's: $(q, xw, A\sigma) \Rightarrow (q', w, A_1 ... A_k \sigma)$;
  2. Se $\langle q, \lambda, A, q', A_1 ... A_k \sigma \rangle$ é uma quíntupla do APN, então sua execução pode causar a transição entre DI's: $(q, xw, A\sigma) \Rightarrow (q', xw, A_1 ... A_k \sigma)$.

Aceitação pela Pilha Vazia e pelo Estado Final

- **Definição**: Define-se a **linguagem aceita pela pilha vazia** por um APN M como

  $$T(M) = \{w | (q_0, w, Z_0) \Rightarrow^* (q, \lambda, \lambda), \text{ para qualquer } q \in Q\}.$$

- **Definição**: Seja M um APN. Define-se o **conjunto de cadeias aceitas pelo estado final** por M como

  $$F(M) = \{w | (q_0, w, Z_0) \Rightarrow^* (q_a, \lambda, \sigma), \text{ para algum } q_a \in F \text{ e } \sigma \in \Gamma^*\}.$$

- **Teorema**: Uma linguagem L é APN aceitável pelo estado final se e somente se ela for APN aceitável pela pilha vazia.

Autômato de Pilha

- **Exemplo**: Considere a gramática para $\{a^n b^n \mid n \geq 1\}$ na forma normal de Greibach usando produções

$$P = \{S \to aSB \mid aB,\ B \to b\}$$

Define-se um APN *M não determinístico* de um estado com as transições

$$\langle a, S, SB\rangle,\ \langle a, S, B\rangle,\ \langle b, B, \lambda\rangle$$

Claramente, na leitura da cadeia de entrada $a^n b^n$ tem-se os resultados intermediários possíveis

1. $(\star, a^n b^n, S) \Rightarrow^* (\star, a^{n-j} b^n, SB^j)$
2. $(\star, a^n b^n, S) \Rightarrow^* (\star, a^{n-k} b^n, B^k)\ (k > 1)$
3. $(\star, a^n b^n, S) \Rightarrow^* (\star, b^n, B^n)$

- A derivação (1) é um estágio intermediário para derivar uma DI da forma (2) ou (3). Mas (2) é um *dead end* - nenhuma transição é aplicável - se $n > k$, enquanto (3) está no caminho da derivação completa
  - $(\star, a^n b^n, S) \Rightarrow^* (\star, \lambda, \lambda)$.

Sumário

O Teorema da Equivalência

- O teorema da equivalência estabelece que as linguagens aceitas por APNs são precisamente as linguagens livres de contexto.
- **Teorema**: Seja $L = L(G)$ para alguma GLC G. Então L é aceita por algum APN (em geral, não determinístico). Ou seja, $L = T(M)$ para algum APN.

O Teorema da Equivalência

- **Exemplo**: Considere as seguintes produções de uma gramática na forma normal de Greibach para a linguagem dos parênteses casados:

$$S \to (L \mid \lambda,\ L \to (LL \mid)$$

- O seguinte é uma derivação mais a esquerda da cadeia (()()) nesta gramática.

$$S \Rightarrow (L \Rightarrow ((LL \Rightarrow (()L \Rightarrow (()(LL \Rightarrow (()()L \Rightarrow (()())$$

- Dá-se o APN associado (de um estado) a seguir, com o símbolo S designando o fundo da pilha.

$$\langle (, S, L\rangle,\ \langle \lambda, S, \lambda\rangle,\ \langle (, L, LL\rangle,\ \langle), L, \lambda\rangle$$

- Para APNs de um estado pode-se reverter o método do resultado prévio para achar uma gramática livre de contexto que gere a linguagem aceita pelo APN.

O Teorema da Equivalência

- **Exemplo**: Considere o Exemplo 5, um APN de um estado que aceita a linguagem de números iguais de a's e b's. As produções da gramática associada (substituindo Z_0 por Z) são:

$$Z \rightarrow aAZ \mid bBZ \mid \lambda,\ A \rightarrow aAA \mid b,\ B \rightarrow bBB \mid a$$

  Vai-se resumir este processo estabelecendo o seguinte resultado:
- **Teorema**: Seja M um APN de um estado. Então existe uma GLC G tal que $L(G) = T(M)$.
  - Para formar a gramática, converta triplas da forma $\langle a, B, \sigma \rangle$ para $\sigma \in \Gamma^*$ em produções da forma $B \rightarrow a\sigma$. Converta triplas da forma $\langle \lambda, B, \sigma \rangle$ em produções da forma $B \rightarrow \sigma$.

O Teorema da Equivalência

- Por causa do Teorema 4, precisa-se apenas provar que qualquer APN pode ser simulado por um APN de um estado a fim de estabelecer a equivalência completa de linguagens livres de contexto e a classe de linguagens aceitas pelos APNs. O próximo teorema estabelece este resultado.
- **Teorema**: Seja M um APN. Então existe um APN de um estado M' tal que $T(M) = T(M')$.
- Mostrou-se no capítulo 1 que todo autômato finito não determinístico (AFN) M é equivalente a um determinístico (AFD), M', isto é, $T(M) = T(M')$.
- No caso dos APN's, esta equivalência também é verdadeira?
- A resposta é **não**: existem LLCs que não podem ser aceitas por APNs determinísticos.

Sumário

Linguagens de Programação

- Como já foi visto, as linguagens livres de contexto são importantes para a ciência da computação porque elas representam um mecanismo razoavelmente adequado para especificar a sintaxe das linguagens de programação (LP).
- Seja o seguinte exemplo, as construções de programação **if-then** e **if-then-else** estão presentes em muitas linguagens de programação. Como uma primeira aproximação, considere as seguintes produções de uma gramática:
  - $S \rightarrow \mathit{if}\ C\ \mathit{then}\ S\ \mathit{else}\ S \mid \mathit{if}\ C\ \mathit{then}\ S \mid a \mid b$
  - $C \rightarrow p \mid q$
- Aqui, S é um (comando) não terminal, C é um (condicional) não terminal, a e b são (comandos) terminais, p e q são (condições) terminais e *if*, *then* e *else* são (palavras reservadas) terminais.

Linguagens de Programação

- Existem problemas com esta gramática. Ela gera a linguagem pretendida, mas de forma **ambígua**. Em particular,

  $$\textit{if } p \textit{ then if } q \textit{ then } a \textit{ else } b$$

  pode ser gerado de duas formas (vide figura 4 a e b), correspondendo às duas interpretações diferentes do comando:

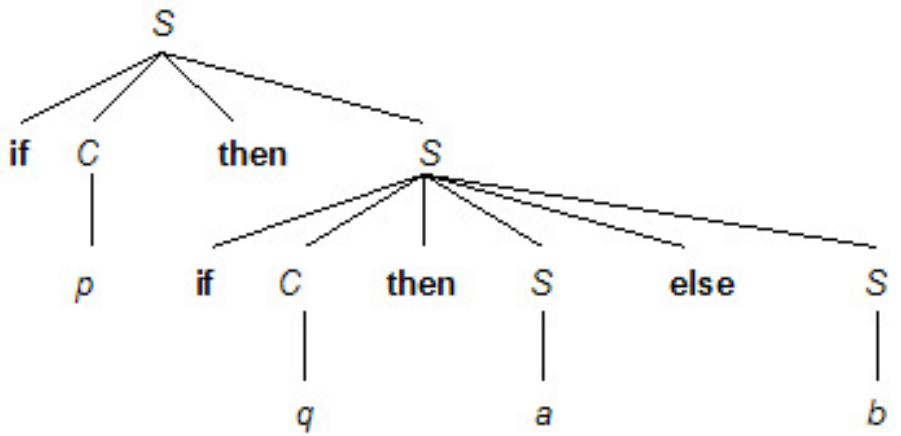
  $$\textit{if } p \textit{ then } (\textit{if } q \textit{ then } a \textit{ else } b)$$

  e

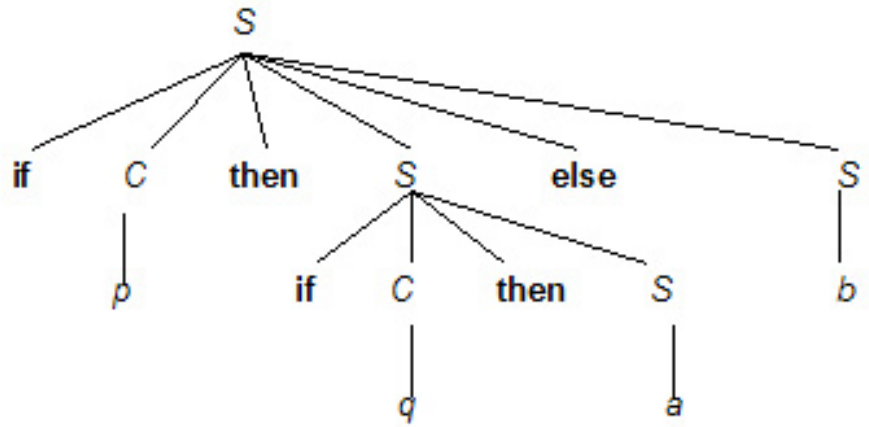
  $$\textit{if } p \textit{ then } (\textit{if } q \textit{ then } a) \textit{ else } b.$$

- Do ponto de vista da programação, um jeito padrão de interpretar tais construções é associar cada comando **else** com o **if** mais próximo.

Linguagens de Programação



(a)



(b)

Gramática Ambígua

- As gramáticas não ambíguas são essenciais para uma especificação sintática bem definida, de outra forma, um compilador poderia traduzir um programa de formas diferentes gerando resultados completamente diferentes.
- Logo, o estudo da ambiguidade é um aspecto prático e teórico importante da teoria das linguagens formais.
- **Definição**: Uma GLC G é **ambígua** se alguma cadeia $w \in L(G)$ tem duas árvores de derivação distintas. Alternativamente, G é ambígua se alguma cadeia em $L(G)$ tem duas derivações mais a esquerda (ou mais a direita) distintas. Uma gramática é não ambígua se ela não for ambígua e uma linguagem L é **inerentemente ambígua** se toda gramática para L for ambígua.

Bibliografia I

[1] Hopcroft, J. E., Ullman, J. D.
Formal Languages and Their Relation to Automata.
Addison-Wesley Publishing Company, 1969.

[2] Hopcroft, J. E., Ullman, J. D. e Motwani, R.
Introdução à Teoria de Autômatos, Linguagens e Computação.
Tradução da segunda edição americana. Editora Campus, 2003.

[3] JFLAP Version 6.0.
Ferramenta para Diagrama de Estados.
`www.jflap.org`.

Bibliografia II

[4] Moll, R. N., Arbib, M. A., and Kfoury, A. J.
An Introduction to Formal Language Theory.
Springer-Verlag, 1988.

[5] Rosa, J. L. G.
Linguagens Formais e Autômatos.
Editora LTC, 2010.