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

SCC-205 - Capítulo 2
Linguagens Livres de Contexto e Autômatos de Pilha

João Luís Garcia Rosa[1]

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

2009

Sumário

Sumário

Definição

- **Definição**: Uma gramática G é **livre de contexto** (GLC) se v é apenas um símbolo não terminal para toda produção $v \to w$ 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.

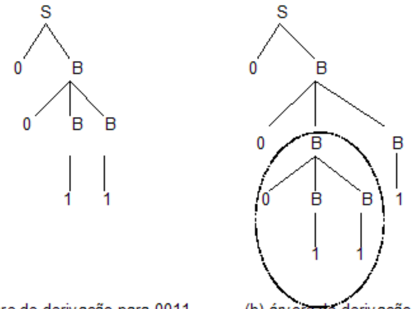
$$\Sigma = \{0, 1\}$$
$$V = \{S, A, B\}$$

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.

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 $A \rightarrow A_1A_2...A_k$;
  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.

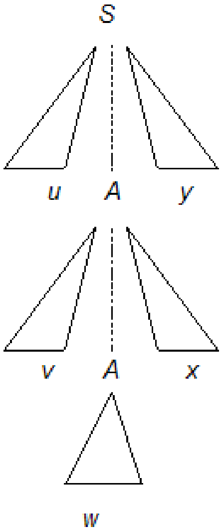
Árvore de derivação

- 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 4 (a), a altura da árvore é 3.)
- Considere a sub-árvore assinalada na Figura 4 (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 8, 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.

Sumário

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 $uv^iwx^iy \in L$.
- 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 Chomsky

- Na matemática, uma forma normal é uma forma padrão de ver alguns objetos matemáticos. Ou seja, a fração $\frac{1}{2}$ é uma **forma normal** para $\frac{5}{10}$, $\frac{3}{6}$ e $\frac{100}{200}$.
- Formas normais desempenham um papel importante na teoria da linguagem, e nesta seção serão discutidos muitos resultados importantes de forma normal para gramáticas livres de contexto.
- **Definição**: Diz-se que uma GLC $G = (\Sigma, V, S, P)$ está na **forma normal de Chomsky** (FNC) se cada produção de P estiver em uma das seguintes formas:
  1. $S \rightarrow \lambda$,
  2. $A \rightarrow BC$, onde $A, B, C \in V$,
  3. $A \rightarrow a$, onde $A \in V$ e $a \in \Sigma$.
- Além disso, se $S \rightarrow \lambda$ estiver em P, então $B, C \in V - \{S\}$ na forma (2).

Forma Normal de Chomsky

- **Teorema**: Seja G uma GLC arbitrária. Então, existe uma gramática equivalente G' na forma normal de Chomsky.
- **Exemplo**: Dada a gramática
  - $S \rightarrow ABC$
  - $C \rightarrow BaB|c$
  - $B \rightarrow b|bb$
  - $A \rightarrow a$

  Converte-se para a forma normal de Chomsky primeiro convertendo terminais do lado direito para variáveis e adicionando produções apropriadas:
  - $S \rightarrow ABC$
  - $C \rightarrow BA_1B|c$
  - $A_1 \rightarrow a$
  - $B \rightarrow b|B_1B_1$
  - $B_1 \rightarrow b$
  - $A \rightarrow a$

Forma Normal de Chomsky

- Finalmente separam-se os lados direitos com mais de duas variáveis:
  - $S \rightarrow AD$
  - $D \rightarrow BC$
  - $C \rightarrow BE|c$
  - $E \rightarrow A_1B$
  - $A_1 \rightarrow a$
  - $B \rightarrow b|B_1B_1$
  - $B_1 \rightarrow b$
  - $A \rightarrow a$

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 $W \in V^*$.
- **Lema**: Seja G uma GLC. Então existe uma gramática equivalente G', $L(G) = L(G')$, que não tem produções recursivas a esquerda, isto é, nenhuma produção da forma $A \rightarrow Av$, onde $A \in V$ e $v \in (\Sigma \cup V)^*$ (*Forma Normal sem Recursão à Esquerda*).

Forma Normal sem Recursão à Esquerda

- As gramáticas livres de contexto (GLC) podem ter produções recursivas à esquerda, como em:

  $$A \longrightarrow Av|w$$

  Estas produções derivam a cadeia wv^*, que também é derivada por:

  $$A \longrightarrow wB|w$$
  $$B \longrightarrow vB|v$$

  sem recursões à esquerda.

- Para eliminar recursões à esquerda em geral, substitua

  $$A \longrightarrow Av_1|Av_2|...|Av_n|w_1|...|w_m$$

  que gera a expressão regular $(w_1 + ... + w_m)(v_1 + ... + v_n)^*$, por:

  $$A \longrightarrow w_1|...|w_m|w_1B|...|w_mB$$
  $$B \longrightarrow v_1|...|v_n|v_1B|...|v_nB$$

Forma Normal de Greibach

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

- **PROVA**: Fixa-se a ordem das variáveis na gramática $G : V = \{A_1, A_2, ..., A_n\}$. G é *crescente* se toda produção de G for da forma
  1. $A_j \rightarrow A_k v$, com $j < k$; ou
  2. $A_j \rightarrow av$, com $a \in \Sigma$.

Forma Normal de Greibach

- Considerando que G tem produções ou da forma

$$A \rightarrow A_{i1} \ldots A_{in}, \text{ com } A_i's \in V \text{ ou}$$
$$A \rightarrow b, \text{ com } b \in \Sigma.$$

  - Se todas as produções de A_1 são crescentes, continua-se em A_2.
  - De outra forma, há uma regra A_1 recursiva à esquerda que é substituída como estabelecido em 6.
  - Daí vai-se para as produções A_2, substituindo por A_1 quando é encontrada uma produção da forma $A_2 \rightarrow A_1 x$, e então eliminam-se recursões à esquerda em A_2, se houver.
  - E assim por diante até A_n.
  - Então faz-se a substituição de volta de cima para baixo pondo G na FNG.
  - Finalmente, aplica-se a substituição de novo, às novas variáveis introduzidas quando a recursão à esquerda foi eliminada.

Forma Normal de Greibach: Um Exemplo

- Exemplo:
  - $A_1 \rightarrow A_2A_2|0$: é crescente
  - $A_2 \rightarrow A_1A_2|1$: não é crescente
- Como as produções A_1 são crescentes, não se mexe nelas. Deve-se mexer nas produções A_2, substituindo A_1 pelas suas produções:
  - $A_1 \rightarrow A_2A_2|0$
  - $A_2 \rightarrow A_2A_2A_2|0A_2|1$
- Observe que agora, a primeira produção A_2 tem recursão à esquerda. Seja $v = A_2A_2$, $w_1 = 0A_2$ e $w_2 = 1$, já que $A \rightarrow Av|w$ deve ser transformado em $A \rightarrow wB|w$ e $B \rightarrow vB|v$ (conforme procedimento de eliminação em 6), vem:
  - $A_1 \rightarrow A_2A_2|0$
  - $A_2 \rightarrow 0A_2|1|0A_2B|1B$
  - $B \rightarrow A_2A_2|A_2A_2B$

Forma Normal de Greibach: Um Exemplo

- Agora tanto as produções A_1 quanto as produções A_2 são crescentes. Deve-se colocar as produções na FNG, fazendo aparecer um símbolo terminal à esquerda do lado direito de cada produção:
  - $A_1 \longrightarrow 0A_2A_2|0A_2BA_2|1A_2|1BA_2|0$
  - $A_2 \longrightarrow 0A_2|0A_2B|1|1B$
- E em B deve-se eliminar variáveis na posição mais à esquerda do lado direito:
  - $A_1 \longrightarrow 0A_2A_2|0A_2BA_2|1A_2|1BA_2|0$
  - $A_2 \longrightarrow 0A_2|0A_2B|1|1B$
  - $B \longrightarrow$ $0A_2A_2|0A_2BA_2|1A_2|1BA_2|0A_2A_2B|0A_2BA_2B|1A_2B|1BA_2B$

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

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

- 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 $\{wcw^R | w \in (a+b)^*\}$.
- 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, $w = x_1x_2...x_n$ é uma cadeia de símbolos de entrada ainda a ser lidos com o APN correntemente lendo x_1, e $\sigma = Z_1Z_2...Z_m$ 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 $\langle q, \lambda, A, q', A_1...A_k \rangle$ 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

  $T(M) = \{w | (q_0, w, Z_0) \Rightarrow^* (q, \lambda, \lambda)$, para qualquer $q \in Q\}$.
- **Definição**: Seja M um APN. Define-se o **conjunto de cadeias aceitas pelo estado final** por M como

  $F(M) = \{w | (q_0, w, Z_0) \Rightarrow^* (q_a, \lambda, \sigma)$, para algum $q_a \in F$ 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 Generalizado

- **Definição**: Um **autômato de pilha generalizado** é um APN que se comporta de acordo com a descrição de máquina da Definição 5 exceto que podem existir transições em δ as quais lêem uma cadeia de símbolos de pilha (ao invés de exatamente um símbolo). Então, podem existir finitamente muitas quíntuplas da forma

$$\langle q, a, B_1 ... B_k, q', C_1 ... C_n \rangle$$

- **Proposição**: Se uma linguagem L é aceita por um APN generalizado, então L é aceita por um APN.

Autômato de Pilha

- Provar-se-á que as linguagens livres de contexto são exatamente as linguagens aceitas por autômatos de pilha.
- Metade deste resultado, que mostra que toda linguagem livre de contexto é aceita por um autômato de pilha, é baseada na forma normal de Greibach para uma gramática livre de contexto.
- Para motivar esta "simulação" de gramáticas livres de contexto por autômatos de pilha, primeiro mostra-se como o AFN derivado de uma gramática linear a direita pode ser visto como um APN de um estado.

Autômato de Pilha

- **Observação**: Dado um AFN $M = (Q, \Sigma, \delta, Q_0, F)$, forme o APN de um estado $M^{\circledast}$. Usa-se o símbolo $\star$ para denotar o estado único de $M^{\circledast}$. Os símbolos da pilha de $M^{\circledast}$ são os estados, Q, de M. Então pode-se escrever $M^{\circledast}$ como:

$$M^{\circledast} = (\{\star\}, \Sigma, Q, \delta^{\circledast}, \star, q_0, \{\star\})$$

  - As transições $\delta^{\circledast}$ de $M^{\circledast}$ imitam δ de acordo com a fórmula "trate o símbolo no topo da pilha de $M^{\circledast}$ como o estado de M," tal que $(\star, q') \in \delta^{\circledast}(\star, x, q)$ sse $q' \in \delta(q, x)$.
  - É então claro que $(\star, w, q_0) \Rightarrow^* (\star, \lambda, q)$ sse $q \in \delta^*(q_0, w)$.
  - Se adicionar-se a $\delta^{\circledast}$ as regras $(\star, \lambda) \in \delta^{\circledast}(\star, x, q)$ sse $\delta(q, x) \in F$
  - deduz-se que $(\star, w, q_0) \Rightarrow^* (\star, \lambda, \lambda)$ sse $\delta^*(q_0, w) \in F$.
  - Então $T(M^{\circledast}) = T(M)$.

Autômato de Pilha

- Em termos de gramática linear a direita, isto diz que a regra $A \to bB$ fornece a quíntupla $\langle \star, b, A, \star, B \rangle$ (que agora será abreviada para $\langle b, A, B \rangle$ como no Exemplo 5),
- enquanto a regra $A \to b$ fornece a quíntupla $\langle \star, b, A, \star, \lambda \rangle$.
- Em outras palavras, agora usando S no lugar de q_0, a derivação $S \Rightarrow^* w_1 A \Rightarrow w_1 bB \Rightarrow^* w_1 b w_2 = w$ é imitada pelo APN passando através das DIs

$$(\star, w, S) \Rightarrow^* (\star, bw_2, A) \Rightarrow (\star, w_2, B) \Rightarrow^* (\star, \lambda, \lambda)$$

- Informalmente, isto diz que a cadeia de entrada processada parcialmente contém a porção da cadeia original w que não é lida e o total da cadeia é aceita sse a variável na pilha pode derivar esta cadeia.

Autômato de Pilha

- **Exemplo**: Considere a gramática para $\{a^n b^n | n \geq 1\}$ na forma normal de Greibach usando produções
$$P = \{S \rightarrow aSB|aB, B \rightarrow b\}$$
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.
    - Sem perda de generalidade, assuma que G está na forma normal de Greibach. Deve-se fornecer um autômato para $L(G)$ na forma de um APN de um estado M. Como M tem apenas um estado $\star$ pode-se novamente abreviar quíntuplas $\langle \star, x, z, \star, w \rangle$ para a forma $\langle x, z, w \rangle$. Então associe qualquer regra da forma $A \rightarrow bCDE$ com a instrução APN $\langle b, A, CDE \rangle$. Qualquer regra da forma $A \rightarrow b$ leva à transição $\langle b, A, \lambda \rangle$. Regras-lambda tornam-se movimentos com nenhuma entrada: $A \rightarrow \lambda$ corresponde a $\langle \lambda, A, \lambda \rangle$.

O Teorema da Equivalência

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

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

- 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:

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

  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.
- 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:
  - $S \rightarrow if\ C\ then\ S\ else\ S \mid if\ C\ then\ S \mid a \mid b$
  - $C \rightarrow p \mid q$
- 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,

  $$if\ \ p\ then\ if\ \ q\ then\ a\ else\ b$$

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

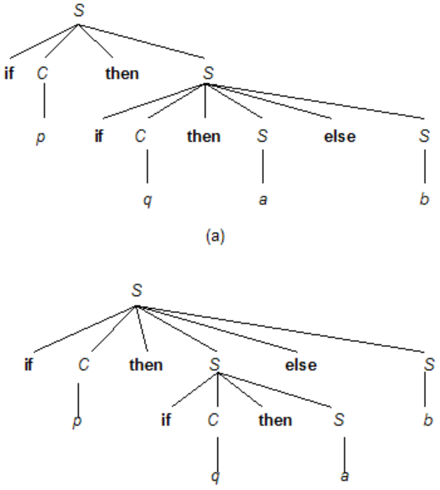
  $$if\ \ p\ then\ (if\ \ q\ then\ a\ else\ b)$$

  e

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

- 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.

Sumário

Parsing

- O exemplo do **if-then-else** ilustra como as gramáticas podem ser usadas para gerar construções de linguagens de programação.
- Dada uma cadeia (um texto de programa) e uma linguagem (de programação) especificada por alguma gramática, deve ser possível construir uma árvore de derivação para a cadeia rapidamente e facilmente se a cadeia pertencer à linguagem, ou relatar um erro se a cadeia não for bem formada.
- O problema de fazer derivação de cadeia *backward* - dada uma cadeia, recuperar sua derivação - é conhecido como o **problema do parsing para LLCs**.
- Sua solução satisfatória é um dos pontos mais importantes no desenvolvimento da ciência da computação.

Parsing bottom-up e *top-down*

- No momento, interessa uma visão simplificada do *parsing*: dada uma cadeia w, como se pode dizer se w é legalmente gerável por uma determinada GLC G?
- Existem duas estratégias básicas para resolver este problema.
- Uma estratégia, o parsing **bottom-up**, considera a construção de uma árvore de *parse* para w, tomando por hipótese uma árvore de derivação para w começando com o fundo (*bottom*) - as folhas da árvore - e trabalhando para cima na árvore até a raiz.
- A segunda estratégia básica, o parsing **top-down**, trabalha de outra forma, tomando por hipótese o topo de uma árvore de derivação, começando primeiro com a raiz.

Parsing bottom-up

- **Exemplo**: Fazer o *parsing* de expressões aritméticas de baixo para cima. Neste exemplo mostra-se uma gramática para um fragmento da linguagem de expressões aritméticas (G_{bu}) e explica-se um algoritmo simples para fazer o *parsing* destas expressões “bottom-up.” O símbolo inicial para a gramática G_{bu} é E.
  1. $E \longrightarrow E + T$
  2. $E \longrightarrow T$
  3. $T \longrightarrow T * F$
  4. $T \longrightarrow F$
  5. $F \longrightarrow (E)$
  6. $F \longrightarrow 2$

Parsing bottom-up

- Agora suponha que se deseja decidir se a cadeia
$$2 + 2 * 2$$
é gerada pela gramática G_{bu}. A abordagem é rodar a derivação *backward*, simulando uma derivação mais a direita como se processa a cadeia da esquerda para a direita. Os primeiros cinco passos do processo:
  1. $2 + 2 * 2 \Leftarrow F + 2 * 2$ (reverso 6)
  2. $\Leftarrow T + 2 * 2$ (reverso 4)
  3. $\Leftarrow E + 2 * 2$ (reverso 2)
  4. $\Leftarrow E + F * 2$ (reverso 6)
  5. $\Leftarrow E + T * 2$ (reverso 4)

  E agora está-se num ponto crucial no *parsing*. Três caminhos são possíveis:
  1. converter $E + T$ em E, deixando $E * 2$;
  2. converter T em E, deixando $E + E * 2$;
  3. converter 2 em F, deixando $E + T * F$.

Parsing bottom-up

- As primeiras duas escolhas não são viáveis - elas levam a “dead ends,” a partir das quais não há *parsing* bem sucedido possível. Escolha (3) que levará a um *parsing* bem sucedido, através de
  1. converter $T * F$ a T;
  2. converter $E + T$ a E, o símbolo inicial da gramática.
- Este exemplo ilustra o **não determinismo** possível no processo de *parsing*. O projeto de *parsers* eficientes - *parsers* que limitam, ou eliminam completamente, o tipo de não determinismo que se viu neste exemplo - é o maior componente da área da ciência da computação conhecido como **análise sintática**.

Parsing top-down

- **Exemplo**: o *parsing* “top-down.” Linguagem de programas-*while* (fragmento do C):

```
{
 y = 0;
 while (x != y)
   y++;
}
```

  Este programa estabelece a valor de y igual ao de x, isto é, o programa realiza o comando de atribuição $y = x$.
- Vai-se agora mostrar uma gramática simples (G_{td}) para a sintaxe desta linguagem:
  - $\Sigma = \{\{, \}, --, ++, =, != , while, ;, (,), 0, x, y\}$
  - $V = \{C, S, S_1, S_2, A, W, U, T, N\}$
  - Símbolo Inicial = C

Parsing top-down

- G_{td}:
  - Produções =
    1. $C \rightarrow \{S_1\}$ (C para comando composto)
    2. $S_1 \rightarrow S\ S_2$
    3. $S_2 \rightarrow S_1 | \lambda$
    4. $S \rightarrow A \mid W \mid C$
    5. $A \rightarrow U = N; | U\ T;$ (A para comando de atribuição)
    6. $T \rightarrow - - \mid + +$
    7. $W \rightarrow$ *while* $(U\ != \ U)\ S$ (W para comando *while*)
    8. $U \rightarrow x \mid y$
    9. $N \rightarrow 0$
- O *parsing top-down* é completamente determinístico: nenhum *backtracking* é necessário, e em todo estágio pode-se predizer sem dificuldade qual produção é apropriada, dado o símbolo do texto corrente. A técnica "top-down" é chamada de *parsing LL*.

Sumário

GLC e a Língua Natural

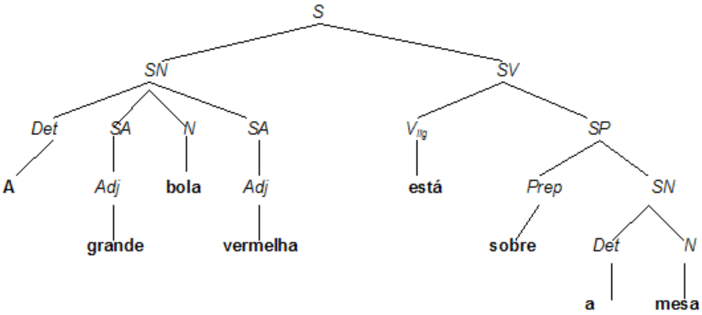
- Já foi dito que a teoria moderna das gramáticas formais segue em parte das teorias da língua natural que foram desenvolvidas pelo linguista Noam Chomsky.
- Vai-se descrever agora brevemente como as gramáticas formais podem ser usadas para descrever língua natural.
  - As categorias da língua natural são associadas a símbolos não terminais de uma gramática:
    - S: Sentença
    - SN: Sintagma Nominal
    - SV: Sintagma Verbal
    - Adj: Adjetivo
    - Det: Determinante
    - V: Verbo
    - N: Substantivo
    - $Prep$: Preposição
    - SA: Sintagma Adjetival
    - SP: Sintagma Preposicional

GLC e a Língua Natural

- Pode-se escrever agora uma GLC que descreve a estrutura sintática de um pequeno subconjunto do Português:
  - $S \rightarrow SN\ SV$
  - $SN \rightarrow Det\ N \mid Det\ SA\ N \mid Det\ N\ SA \mid Det\ SA\ N\ SA$
  - $SA \rightarrow Adj\ SA \mid Adj$
  - $SV \rightarrow Vlig\ SP$
  - $SP \rightarrow Prep\ SN$
- É claro que as palavras do Português constituem os símbolos terminais desta gramática, e portanto deve-se adicionar produções da forma:
  - $Vlig \rightarrow$ é | está
  - $Det \rightarrow$ o | a | os | as | um
  - $Prep \rightarrow$ em | sobre | para
- etc. Dada esta gramática pode-se derivar sentenças como "A grande bola vermelha está sobre a mesa" (figura 4).

GLC e a Língua Natural

Figure: Árvore de derivação para "A grande bola vermelha está sobre a mesa."



GLC e a Língua Natural

- Um problema imediato com esta gramática é a concordância de caso.
- A gramática gera sentenças do tipo "As grande bola vermelhas estão sobre as mesa."
- Deve-se portanto, incluir as concordâncias de gênero e número.
- Vários problemas linguísticos surgirão a partir desta gramática simples.
- Distorções não livres de contexto.
- Chomsky argumenta que uma língua natural não pode ser gerada por uma GLC.
- Há a necessidade de algo mais complexo.

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.
SCE-0185 - Teoria da Computação e Linguagens Formais.
Slides. Ciências de Computação. Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo, 2008.