Capítulo 3: Gramáticas

Já vimos que procedimentos podem ser usados para definir linguagens de duas maneiras essenciais: como *geradores*, procedimentos que enumeram os elementos da linguagem, e como *reconhecedores* (ou *aceitadores*), procedimentos que indicam quando uma sequência faz parte da linguagem. O tipo mais comum de gerador é a *gramática*. A idéia original de gramática vem do estudo de linguagens naturais, e as definições que apresentaremos aqui são essencialmente devidas ao linguista Noam Chomsky.

Fundamentalmente, uma gramática é composta por *regras de produção*, ou *regras de re-escrita*, através das quais é possível obter todos os elementos da linguagem a partir de um símbolo inicial, usando as regras para re-escrever (produzir) os elementos. Formalmente, definimos uma *gramática* G como sendo uma construção $<N, \Sigma, P, S>$, onde

- N é um alfabeto de símbolos auxiliares, chamados de símbolos não terminais, ou, simplesmente, de *nãoterminais*.
- Σ é o alfabeto no qual a linguagem é definida, cujos elementos são os símbolos terminais, ou, simplesmente, *terminais*.
- P é o conjunto de regras de re-escrita, chamadas simplesmente de *regras* ou *produções*.
- S é o *símbolo inicial*.

Vamos definir o *vocabulário* de G, como sendo $V = N \cup \Sigma$, o alfabeto composto pelos símbolos terminais e não terminais. O conjunto de regras P é uma relação binária no conjunto V^* de cadeias de símbolos quaisquer (terminais ou nãoterminais), isto é, $P \subseteq V^* \times V^*$, correspondendo cada regra individual a um par de cadeias (α, β). Entretanto, em vez de $(\alpha, \beta) \in P$, a notação habitual para a regra que permite a re-escrita de α como β é simplesmente $\alpha \rightarrow \beta$. Além disso, reunimos regras com o mesmo lado esquerdo α, tais como $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, …, $\alpha \rightarrow \beta_n$, na abreviação

$$\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n.$$

Nota: Em geral se procura usar as seguintes convenções:

letras	representam
S, A, B, C, …	nãoterminais
a, *b*, *c*, …	terminais
X, Y, Z, …	símbolos quaisquer
α, β, γ, …	cadeias quaisquer
x, y, z, …	cadeias de terminais
V, N, Σ, Γ, Δ, …	alfabetos

Como acontece com toda convenção, haverá alguns casos em que esta não será seguida.

□

Nota: Alguns autores preferem definir P como um subconjunto de $V^* N V^* \times V^*$, ou seja, exigem que, numa regra $\alpha \rightarrow \beta$, o lado esquerdo α contenha pelo menos um símbolo nãoterminal. A diferença entre as duas definições é irrelevante.

Exemplo: Vamos definir uma gramática

$$G = < N, \Sigma, P, S > = < \{ S \}, \{ 0, 1 \}, \{ (S, 0\ S\ 1), (S, \varepsilon) \}, S >$$

onde $N = \{ S \}$ é o conjunto de nãoterminais, $\Sigma = \{ 0, 1 \}$ é o conjunto de terminais, e $P = \{ (S, 0\ S\ 1), (S, \varepsilon) \} = \{ S \rightarrow 0S1, S \rightarrow \varepsilon \} = \{ S \rightarrow 0S1 \mid \varepsilon \}$ é o conjunto de regras.

Para mostrar que a cadeia 000111 faz parte da linguagem associada à gramática, seguimos, a partir de S, os seguintes passos intermediários:

S, 0S1, 00S11, 000S111, 000111.

Assim, por três vezes S é substituído por 0S1, e finalmente, S é substituído pela sequência vazia ε. Como veremos a seguir, aplicar uma regra $\alpha \rightarrow \varepsilon$ é equivalente a simplesmente remover α.

□

Relação $\Rightarrow$ (deriva em um passo). Podemos definir a aplicação de uma regra através de uma relação: dizemos que $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ se e somente se $\alpha \rightarrow \beta$. Caso seja necessário distinguir entre várias gramáticas, podemos usar o símbolo $\Rightarrow_G$ para indicar explicitamente qual a gramática G utilizada.

Quando consideramos vários passos de uma derivação, podemos usar as definições de operações com relações vistas no capítulo 0, e indicar $\Rightarrow^i$ (*deriva em i passos*), $\Rightarrow^*$ (*deriva em zero ou mais passos*, ou, simplesmente, *deriva*), $\Rightarrow^+$ (*deriva em um ou mais passos*).

Uma sequência α derivável de S, (tal que $S \Rightarrow^* \alpha$) é chamada uma *forma sentencial* da gramática considerada.

Exemplo: Em referência à gramática do exemplo anterior, podemos escrever

$$S \Rightarrow 0S1,\ 0S1 \Rightarrow 00S11,\ 00S11 \Rightarrow 000S111,\ 000S111 \Rightarrow 000111$$

ou, de forma mais compacta,

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111$$

Podemos também escrever $S \Rightarrow^4 000111$, e $S \Rightarrow^* 000111$. As cadeias S, 0S1, 00S11, 000S111, e 000111 são formas sentenciais da gramática. Dessas, a mais importante é 000111, composta *exclusivamente por terminais*, que fará parte da *linguagem da gramática*, de acordo com a próxima definição.

Linguagem de uma gramática. Definimos a linguagem da gramática G = <N, Σ, P, S> por

$$L(G) = \{ x \in \Sigma^* \mid S \Rightarrow^* x \}.$$

Ou seja, a linguagem da gramática G é constituída pelas sequências x, que são compostas apenas de símbolos terminais e que podem ser obtidas em um número arbitrário de passos de derivação a partir do símbolo inicial S, usando as regras de P.

Exemplo: Seja G a gramática do exemplo anterior. Vamos mostrar que

$$L(G) = \{ 0^i1^i \mid i \in \mathbf{Nat} \}.$$

Para mostrar a igualdade dos dois conjuntos é necessário mostrar a *"dupla inclusão"* , que no caso se reduz a mostrar *(a)* e *(b)* a seguir:

(a) cada sequência da forma 0^i1^i pode ser derivada a partir de S, ou seja, $L(G) \supseteq \{ 0^i1^i \mid i \in \mathbf{Nat} \}$.

Basta observar que uma derivação a partir de S, em que se utiliza i vezes a regra $S \rightarrow 0S1$ e uma vez a regra $S \rightarrow \varepsilon$, gera exatamente $0^i\ 1^i$.

(b) uma sequência de terminais derivada de S tem a forma 0^i1^i, ou seja, $L(G) \subseteq \{ 0^i1^i \mid i \in \mathbf{Nat} \}$.

Basta notar que a partir de S, usando as duas regras de todas as maneiras possíveis, só é possível gerar sequências (formas sentenciais) da forma 0^i1^i , e da forma 0^iS1^i , e que cadeias da segunda forma não podem fazer parte da linguagem, por conter um símbolo não terminal.

□

Gramáticas equivalentes. Duas gramáticas G_1 e G_2 são chamadas de equivalentes, se ambas definem a mesma linguagem: $L(G_1) = L(G_2)$.

Exemplo: As duas gramáticas abaixo são equivalentes:

$$G_1 = < \{S, T\}, \{a, b\}, \{S \rightarrow aTa \mid bTb, T \rightarrow aT \mid bT \mid \varepsilon\}, S>$$

$$G_2 = < \{S, A, B\}, \{a, b\}, \{S \rightarrow aA \mid bB, A \rightarrow aA \mid bA \mid a, B \rightarrow aB \mid bB \mid b\}, S>$$

Para mostrar a equivalência destas duas gramáticas, basta apenas mostrar que, de forma diferente, ambas geram *exatamente* as cadeias cujo primeiro símbolo é igual ao último. Isso pode ser feito observando que:

- em G_1, o primeiro e o último símbolos são introduzidos na mesma regra;
- em G_2, os nãoterminais A e B *"lembram"* qual *"foi"* o primeiro símbolo, e determinam qual deve ser o último símbolo.

A demonstração completa deve ser feita por indução, e fica como exercício para o leitor interessado mostrar que

$$L(G_1) = L(G_2) = \{ cxc \mid c \in \{a,b\} \text{ e } x \in \{a,b\}^* \}.$$

Por exemplo, a cadeia aabba é da linguagem, e pode ser derivada como a seguir:

em G_1 : $S \Rightarrow aTa \Rightarrow aaTa \Rightarrow aabTa \Rightarrow aabbTa \Rightarrow aabba$

em G_2 : $S \Rightarrow aA \Rightarrow aaA \Rightarrow aabA \Rightarrow aabbA \Rightarrow aabba$

□

A hierarquia das gramáticas. Chomsky definiu quatro *"tipos"* de gramáticas, 0, 1, 2 e 3, que formam uma hierarquia: cada gramática de um tipo é também dos tipos menores. Nossa definição não é exatamente a mesma usada por Chomsky, mas é praticamente equivalente.

Gramáticas tipo 0 (*gramáticas sem restrição*). Exatamente as gramáticas vistas na definição anterior. As regras de uma gramática tipo 0 são regras da forma $\alpha \rightarrow \beta$, com α e β quaisquer.

Gramáticas tipo 1 (*gramáticas sensíveis ao contexto*, ou *gsc*). As gramáticas tipo 1 são as gramáticas com regras da forma $\alpha \rightarrow \beta$, em que se exige $| \alpha | \leq | \beta |$; é entretanto permitida uma regra que viola esta restrição: uma gramática tipo 1 pode ter a regra $S \rightarrow \varepsilon$, se S não aparece do lado direito de nenhuma regra.

Gramáticas tipo 2 (*gramáticas livres de contexto*, ou *glc*). As gramáticas tipo 2 são as gramáticas com regras da forma $A \rightarrow \beta$, onde A é um símbolo não terminal, e β é uma sequência qualquer de V*, possivelmente vazia.

Gramáticas tipo 3 (gramáticas regulares). As gramáticas tipo 3 só podem ter regras dos três tipos descritos a seguir:

$A \rightarrow a\ B$,	onde A e B são nãoterminais, e *a* é um terminal;
$A \rightarrow a$,	onde A é um nãoterminal, e *a* é um terminal;
$A \rightarrow \varepsilon$,	onde A é um nãoterminal.

Se uma linguagem tem uma gramática tipo 0, ela é uma *linguagem tipo 0*; se tem uma gramática tipo 1, ela é uma *linguagem tipo 1*, ou uma *linguagem sensível ao contexto* (*lsc*); se tem uma gramática tipo 2, ela é uma *linguagem tipo 2*, ou uma *linguagem livre de contexto* (*llc*); se tem uma gramática tipo 3, ela é uma *linguagem tipo 3*, ou uma *linguagem regular*.

Observações:

- As gramáticas tipo 3 são chamadas regulares pela simplicidade da estrutura de suas linguagens, garantida pelos rígidos formatos de suas regras.
- As gramáticas tipo 2 são chamadas de livres de contexto porque uma regra $A \rightarrow \beta$ indica que o nãoterminal A, *independentemente do contexto* em que estiver inserido, pode ser substituído por β.
- Finalmente, as gramáticas tipo 1 são chamadas de sensíveis ao contexto por permitirem regras da forma $\alpha\ A\ \gamma \rightarrow \alpha\ \beta\ \gamma$: A pode ser reescrito como β, *dependendo do contexto* em que A aparece (α à esquerda, γ à direita).
- O caso especial da regra $S \rightarrow \varepsilon$, nas gramáticas sensíveis ao contexto tem uma única finalidade: permitir que a cadeia vazia ε pertença a algumas linguagens sensíveis ao contexto. Com efeito, a aplicação de uma regra $\alpha \rightarrow \beta$ em que $| \alpha | \leq | \beta |$ não pode diminuir o comprimento da sequência à qual é aplicada, porque temos sempre $| \gamma\ \alpha\ \delta | \leq | \gamma\ \beta\ \delta |$. Como $| S | = 1$, e $| \varepsilon | = 0$, há necessidade de alguma regra que permita a diminuição do comprimento, para que uma derivação $S \Rightarrow^* \varepsilon$ seja possível, em algumas gramáticas sensíveis ao contexto.
- *As definições de gramáticas acima <u>não</u> formam exatamente uma hierarquia*. Certamente, uma gramática tipo 3 é sempre tipo 2, e uma gramática tipo 1 é

sempre tipo 0, mas nem todas as gramáticas tipo 2 são tipo 1. Isso acontece porque as regras da forma $A \rightarrow \varepsilon$ não satisfazem a restrição de comprimento, pois $|\ A\ | > |\ \varepsilon\ |$, já que $1 > 0$. Na última seção deste capítulo, entretanto, mostraremos um resultado que é suficiente para nossos propósitos: qualquer glc pode ser transformada, através de um algoritmo relativamente simples, em uma gramática que satisfaz simultaneamente as definições de glc e de gsc, e que, além disso, é equivalente à gramática original. Com esse resultado, vemos que, ao contrário das classes de gramáticas, as classes de linguagens definidas acima formam uma hierarquia. Ou seja, toda linguagem regular é uma llc, toda llc é uma lsc, e toda lsc é uma linguagem tipo 0.

Seguem-se alguns exemplos de gramáticas e linguagens de diversas classes. Naturalmente, é mais fácil mostrar que uma gramática pertence a uma certa classe, do que mostrar o resultado oposto: num caso, basta exibir uma gramática apropriada; no outro, é necessário mostrar que nenhuma gramática é apropriada. Algumas técnicas para obter esses resultados negativos serão apresentadas posteriormente, durante o decorrer do curso.

Exemplo: A linguagem $L = \{ cxc \mid c \in \{a, b\} \text{ e } x \in \{a, b\}^* \}$ do exemplo anterior é livre de contexto, uma vez que G_1 é uma glc. Por outro lado, como G_2 é uma gramática regular, L também é regular. Como toda linguagem regular é livre de contexto, em geral não se faz referência ao fato de que a linguagem é livre de contexto.

□

Exemplo: A linguagem $L = \{ xx^R \mid x \in \{0,1\}^* \}$ é livre de contexto. (A notação x^R indica a sequência *reversa* ou *refletida* de x: se $x = a_1\ a_2\ \ldots a_n$, $x^R = a_n\ \ldots a_2\ a_1$.) Basta considerar a gramática com regras

$$S \rightarrow 0S0 \mid 1S1 \mid \varepsilon$$

onde S é o único nãoterminal, e portanto é também o nãoterminal inicial. Para derivar 00100100, por exemplo, teríamos:

$$S \Rightarrow 0S0 \Rightarrow 00S00 \Rightarrow 001S100 \Rightarrow 0010S0100 \Rightarrow 00100100$$

A linguagem L *não* é regular, mas ainda não dispomos dos mecanismos apropriados para provar este fato.

□

Exemplo: A linguagem $L = \{ x\ x \mid x \in \{0,1\} \}$ é sensível ao contexto. Uma gramática para esta linguagem tem as regras indicadas a seguir:

1. $S \rightarrow 0ZS$	5. $Z0 \rightarrow 0Z$	9. $ZZ' \rightarrow Z'0$	13. $Z' \rightarrow 0$
2. $S \rightarrow 1US$	6. $Z1 \rightarrow 1Z$	10. $ZU' \rightarrow Z'1$	14. $U' \rightarrow 1$
3. $S \rightarrow 0Z'$	7. $U0 \rightarrow 0U$	11. $UZ' \rightarrow U'0$	
4. $S \rightarrow 1U'$	8. $U1 \rightarrow 1U$	12. $UU' \rightarrow U1$	

A utilização das regras é explicada a seguir, usando-se como exemplo a derivação de 00100010, na qual as regras estão indicadas como índices inferiores.

- *1 a 4:* regras usadas para gerar os símbolos das duas metades de xx. Os símbolos da segunda metade são indicados por Z (zero: 0) e U (um: 1). U' ou Z' indica o último símbolo da segunda metade.

$S \Rightarrow_1 0ZS \Rightarrow_1 0Z0ZS \Rightarrow_2 0Z0Z1US \Rightarrow_3 0Z0Z1U0Z'$

- *5 a 8:* regras usadas para levar todos os símbolos da segunda metade para depois dos símbolos da primeira metade:

  $0Z0Z1U0Z' \Rightarrow_7 0Z0Z10UZ' \Rightarrow_6 0Z01Z0UZ' \Rightarrow_5 0Z010ZUZ' \Rightarrow_5 00Z10ZUZ' \Rightarrow_6 001Z0ZUZ' \Rightarrow_5 0010ZZUZ'$

- *9 a 12:* regras usadas para converter os símbolos da segunda metade. O símbolo Z' (ou U') indica o último símbolo ainda não convertido.

  $0010ZZUZ' \Rightarrow_{11} 0010ZZU'0 \Rightarrow_{10} 0010ZZ'10 \Rightarrow_9 0010Z'010$

- *13 a 14:* regras usadas para converter o primeiro símbolo da segunda metade

  $0010Z'010 \Rightarrow_{13} 00100010$

Pela definição de linguagem de uma gramática, só nos interessam as sequências de terminais que são obtidas a partir do símbolo inicial S, usando as regras de produção da gramática. Por essa razão, a ordem em que as regras são aplicadas não é importante. Uma outra derivação possível para 00100010, em que as mesmas regras são usadas em uma ordem diferente é

$S \Rightarrow_1 0ZS \Rightarrow_1 0Z0ZS_5 \Rightarrow 00ZZS \Rightarrow_2 00ZZ1US \Rightarrow_6 00Z1ZUS \Rightarrow_6 001ZZUS \Rightarrow_3 001ZZU0Z' \Rightarrow_7 001ZZ0UZ' \Rightarrow_5 001Z0ZUZ' \Rightarrow_5 0010ZZUZ' \Rightarrow_{11} 0010ZZU'0 \Rightarrow_{10} 0010ZZ'10 \Rightarrow_9 0010Z'010 \Rightarrow_{13} 00100010$

Outro fato sem importância é a existência de "becos sem saída", isto é, sequências deriváveis de S que não levam a nenhuma sequência de terminais, não contribuindo, portanto, para a linguagem. Por exemplo, considere a derivação:

$S \Rightarrow_1 0ZS \Rightarrow_1 0Z0ZS \Rightarrow_2 0Z0Z1US \Rightarrow_3 0Z0Z1U0Z' \Rightarrow_{13} 0Z0Z1U00$

Note que a regra 13 foi aplicada "precocemente", de forma que as ocorrências restantes dos símbolos Z e U *não podem mais ser convertidas* em ocorrências de 0 e 1. Entretanto, como nenhuma sequência de terminais pode ser obtida a partir de 0Z0Z1U00, o fato de que 0Z0Z1U00 pode ser derivada a partir de S não introduz na linguagem nenhum elemento indevido. (0Z0Z1U00 é apenas uma forma sentencial — inútil — da gramática.)

A linguagem L *não* é livre de contexto, mas ainda não dispomos dos mecanismos para provar este fato.

□

Uma das principais motivações da definição de gsc vem do teorema abaixo: a restrição nos comprimentos dos lados esquerdo e direito das regras tem exatamente a finalidade de garantir a recursividade das linguagens sensíveis ao contexto. Entretanto, observamos que a recíproca do teorema não é verdadeira, ou seja existem linguagens recursivas que não são sensíveis ao contexto. Este último fato será demonstrado posteriormente.

Teorema: Toda lsc é um conjunto recursivo.

Dem.: Sejam G = <N, Σ, P, S> uma gsc, e L = L(G). Devemos, pela definição de conjunto recursivo, mostrar que existe um algoritmo que, quando recebe como entrada

uma cadeia arbitrária $x \in \Sigma^*$, indica se é ou não verdade que $x \in L$, parando e emitindo respectivamente *"sim"* ou *"não"*.

Sabemos que uma gsc pode ter uma regra $S \rightarrow \varepsilon$, se S não aparecer à direita em nenhuma regra. Para cada uma das outras regras, o comprimento do lado esquerdo não pode ser maior que o do lado direito, e cada aplicação da regra pode manter ou aumentar o comprimento, mas nunca diminuí-lo. Portanto, para considerar todas as derivações que podem levar a uma sequência x, basta considerar as formas sentenciais de comprimento menor ou igual ao de x: se $x \neq \varepsilon$, $S \Rightarrow^* \gamma \Rightarrow^* x$ implica que $1 \leq |\gamma| \leq |x|$. Esta idéia está aplicada no algoritmo a seguir:

1. Se $x \neq \varepsilon$, vá para 4.
2. Se P contém uma regra $S \rightarrow \varepsilon$, pare e diga *"sim"*.
3. Pare e diga *"não"*.
4. Faça $X = \{ S \}$, e $Y = \emptyset$.
5. Se $x \in X$, pare e diga *"sim"*
6. Se $X = \emptyset$, pare e diga *"não"*
7. Escolha uma cadeia qualquer α em X, retire α de X, e acrescente α a Y
8. Para cada β tal que $\alpha \Rightarrow \beta$, se $\beta \notin Y$, e se $|\beta| \leq |x|$, acrescente β a X.
9. vá para 5

O tratamento da entrada $x = \varepsilon$ se resume a verificar se existe a regra correspondente $S \rightarrow \varepsilon$; para as demais entradas, todas as possibilidades são consideradas, e uma forma sentencial só é eliminada se já foi considerada antes (pertence a Y), ou se é longa demais (comprimento maior que o de x). Durante a execução do algoritmo, X guarda as formas sentenciais que ainda devem ser consideradas. Note que se trata de um algoritmo: em cada execução do passo 7, é escolhida uma cadeia α de comprimento menor ou igual ao de x, e não são permitidas repetições.

□

Exemplo: Considere a gsc G com nãoterminais S e T, terminais a, b e c, símbolo inicial S e regras

$S \rightarrow aSa \mid bSb \mid T$

$T \rightarrow cT \mid c$

(Note que G é *também* uma glc.) Seja x = aabbb. Para determinar se $x \in L(G)$, são consideradas as formas sentenciais abaixo, que compõem o conjunto Y, ao final da execução:

{ S, aSa, bSb, T, aaSaa, abSba, aTa, baSab, bbSbb, bTb, cT, c, aaTaa, abTba,
acTa, aca, baTab, bbTbb, bcTb, bcb, ccT, cc, aacaa, abcba, accTa, acca,
bacab, bbcbb, bccTb, bccb, cccT, ccc, accca, bcccb, ccccT, cccc, ccccc }

Fica como exercício a execução passo a passo do algoritmo, mostrando que $x \notin L(G)$.

□

Teorema: Toda linguagem tipo 0 é um conjunto recursivamente enumerável.

Dem.: Seja $G = \langle N, \Sigma, P, S \rangle$ uma gramática (tipo 0). O procedimento abaixo enumera os elementos de L(G).

1. Faça X = { S }
2. Emita todos os elementos $\alpha \in X \cap \Sigma^*$.
3. Faça Y = { $\beta \mid \alpha \in X$ e $\alpha \Rightarrow \beta$ }
4. Faça X = Y
5. vá para 2.

A cada iteração do procedimento acima, X contém todas as sequências deriváveis de S em um passo adicional. Assim, se $S \Rightarrow^* x$ em n passos, em n passos o procedimento emitirá x.

□

Outros resultados sobre as classes de linguagens ainda dependem de resultados que serão apresentados posteriormente. Em particular, mostraremos posteriormente que todo conjunto r. e. é uma linguagem tipo 0, e apresentaremos os exemplos que mostram que as demais inclusões entre as classes de linguagens são próprias.

Gramáticas livres de contexto, gramáticas sensíveis ao contexto, e regras com lado direito vazio. Retomamos aqui o problema das regras com lado direito vazio em glc's e gsc's. Como já vimos anteriormente, a definição de glc permite indiscriminadamente a presença de regras $A \rightarrow \varepsilon$, onde A é um nãoterminal; essas regras podem violar a restrição da definição de gsc, uma vez que a única regra permitida com lado direito vazio é uma regra $S \rightarrow \varepsilon$, desde que o símbolo inicial S não apareça à direita em nenhuma regra.

Dessa maneira, não é imediato que toda llc é uma lsc, uma vez que uma glc pode não satisfazer a definição de gsc.

Esta seção tem como principal finalidade mostrar que *cada glc pode ser transformada em outra glc equivalente, que satisfaz a definição de gsc*. Como corolário desse fato, temos que *toda llc é também uma lsc*.

Nãoterminais anuláveis. Dizemos que um nãoterminal A de uma glc G é anulável se é possível derivar a cadeia vazia ε a partir de A, ou seja, se $A \Rightarrow^* \varepsilon$.

Lema: Dada uma gramática livre de contexto G = <N, Σ, P, S>, o algoritmo a seguir determina quais são os nãoterminais anuláveis de G, isto é, quais são os nãoterminais A a partir dos quais pode ser derivada a cadeia vazia ε.

Dem.: Considere o algoritmo:

1. Faça X = { $A \in N \mid A \rightarrow \varepsilon \in P$ }
2. Repita o passo 3 até que nenhum elemento novo possa ser acrescentado a X:
3. Para cada regra $B \rightarrow X_1 X_2 \ldots X_n$, se todos os símbolos $X_1, X_2, \ldots X_n$ pertencem a X, acrescente B a X.

Claramente, ao final da execução do algoritmo, X contém exatamente os nãoterminais anuláveis da gramática considerada.

□

Teorema: Para qualquer gramática livre de contexto G existe uma gramática livre de contexto G', equivalente a G, que satisfaz a definição de gramática sensível ao contexto.

Dem.: Seja G = <N, Σ, P, S>. Execute o algoritmo anterior para determinar quais os nãoterminais anuláveis de G. Seja S' um símbolo novo, não pertencente a N ou a Σ.

Defina a gramática G' = <N ∪ {S'}, Σ, P', S'>, sendo P' como obtido como descrito a seguir:

1. Inicialmente, faça P' = P.
2. Repita o passo 3 enquanto for possível acrescentar regras novas a P'.
3. Para cada regra $A \rightarrow \alpha B \beta$ de P', se B é anulável, acrescente $A \rightarrow \alpha \beta$ a P'.
4. Retire de P' todas as regras com lado direito vazio.
5. Acrescente a P' a regra $S' \rightarrow S$.
6. Se S é anulável, acrescente a P', além da regra acima, a regra $S' \rightarrow \varepsilon$.
   (Se acrescentada, esta será a única regra com lado direito vazio.)

Claramente, a gramática G' satisfaz as definições vistas de glc e de gsc. Note que, como S' é um símbolo novo, não ocorre à direita em nenhuma das regras de G'.

Resta assim provar que G e G' são equivalentes. Para isto, observamos que o uso das regras-ε em G corresponde ao uso de regras mais curtas em G': em vez de usar uma regra de G, que introduz (ocorrências de) nãoterminais anuláveis, e depois usar as regras necessárias para a transformação desses nãoterminais anuláveis em ε, usamos de saída uma regra mais curta de G', em que as correspondentes ocorrências desses nãoterminais anuláveis não aparecem (ver passo 3 do algoritmo acima). Na outra direção, similarmente, o uso de uma regra mais curta pode ser substituído pelo uso da regra mais longa de que se originou, seguida pelo uso das regras necessárias para transformar os nãoterminais que se deseja anular na sequência vazia.

□

Exemplo: Considere a gramática G, com o conjunto de regras P a seguir:

$S \rightarrow ABC \mid ABD$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow Bb \mid AC$
$C \rightarrow CC \mid c \mid \varepsilon$
$D \rightarrow d$

Para a aplicação do algoritmo acima, temos sucessivamente os seguintes valores para o conjunto X:

{ A, C }	regras $A \rightarrow \varepsilon$ e $C \rightarrow \varepsilon$
{ A, B, C }	regra $B \rightarrow AC$
{ A, B, C, S }	regra $S \rightarrow ABC$

até que X atinge seu valor final: X = { S, A, B, C }.

Na construção do conjunto P' de regras da gramática G', temos inicialmente P' = P:

$S \rightarrow ABC \mid ABD$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow Bb \mid AC$
$C \rightarrow CC \mid c \mid \varepsilon$
$D \rightarrow d$

Retirando as ocorrências de nãoterminais anuláveis, as seguintes regras são acrescentadas:

$S \rightarrow BC \mid AC \mid AB \mid BD \mid AD \mid C \mid B \mid A \mid D \mid \varepsilon$
$A \rightarrow a$
$B \rightarrow b$
$B \rightarrow A \mid C \mid \varepsilon$
$C \rightarrow C$

As regras com lado direito vazio são então retiradas, as regras $S' \rightarrow S$, e $S' \rightarrow \varepsilon$ são acrescentadas, já que $S \in X$. Ao final, temos as seguintes regras em P':

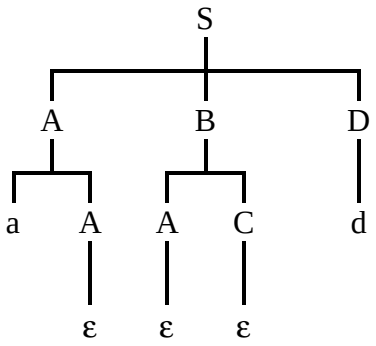
$S' \rightarrow S \mid \varepsilon$
$S \rightarrow ABC \mid ABD \mid BC \mid AC \mid AB \mid BD \mid AD \mid C \mid B \mid A \mid D$
$A \rightarrow aA \mid a$
$B \rightarrow Bb \mid AC \mid b \mid A \mid C$
$C \rightarrow CC \mid c \mid C$
$D \rightarrow d$

(A regra $C \rightarrow C$ é obviamente inútil.)

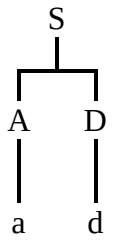
Podemos agora comparar uma derivação em G com a correspondente derivação em G'. Considere a seguinte derivação em G:

$S \Rightarrow ABD \Rightarrow aABD \Rightarrow aBD \Rightarrow aACD \Rightarrow aCD \Rightarrow aD \Rightarrow ad$

A aplicação de $S \rightarrow ABD$ introduz uma ocorrência de B que é depois anulada; o mesmo acontece com a aplicação da regra $A \rightarrow aA$. As regras correspondentes em G' são, portanto, $S \rightarrow AD$ e $A \rightarrow a$. As duas derivações correspondentes, em G e G', podem ser representadas pelas "árvores de derivação" a seguir.



árvore de derivação em G



árvore de derivação em G'

□

Corolário: Toda glc cuja linguagem não contém a sequência vazia pode ser transformada em uma glc equivalente que não tem regras com lado direito vazio.

Dem.: Se o símbolo inicial da gramática não é anulável, a construção vista na demonstração do teorema acima nos leva a uma gramática sem nenhuma regra-ε.

□

Exemplo: Seja a gramática G com regras

$E \rightarrow T\ E'$
$T \rightarrow F\ T'$
$F \rightarrow (\ E\ ) \mid a$
$E' \rightarrow +\ T\ E' \mid \varepsilon$
$T' \rightarrow *\ F\ T' \mid \varepsilon$

De acordo com o algoritmo visto na demonstração do teorema anterior, o conjunto dos nãoterminais anuláveis é X = {E', T'}. Assim, a nova gramática G' tem regras

$S' \rightarrow E$
$E \rightarrow T\ E' \mid T$
$T \rightarrow F\ T' \mid F$
$F \rightarrow (\ E\ ) \mid a$
$E' \rightarrow +\ T\ E' \mid +\ T$
$T' \rightarrow *\ F\ T' \mid *\ F$

A conversão de gramáticas livres de contexto em gramáticas sem regras vazias perdeu parte de sua importância, porque os algoritmos de análise sintática usados em compiladores modernos (ao contrário de algoritmos mais antigos) não fazem restrições ao uso de regras com lado direito vazio. Portanto, regras com lado direito vazio podem ser usadas sem problemas nas gramáticas de linguagens de programação, usadas para projeto e implementação de compiladores.

(*revisão de 27fev97)*