Manipulação de Gramáticas e Formas Normais

O texto apresentado neste documento é uma adaptação de:

Newton Vieira, 2006. Introdução aos fundamentos da computação: Linguagens e máquinas. CENGAGE Learning.

Thomas A. Sudkamp. 1997. Languages and machines: an introduction to the theory of computer science. Addison-Wesley Longman Publishing Co., Inc., USA.

Uma mesma linguagem pode ser gerada por inúmeras gramáticas. Algumas gramáticas podem ser mais adequadas do que outras, dependendo do contexto para o qual elas foram projetadas. Assim, é importante saber algumas técnicas de manipulação de GLC's de forma a obter GLC's equivalentes, mas com a presença ou ausência de certa(s) característica(s) relevantes para certa aplicação. Em particular, existem algumas formas normais de GLC's que são apropriadas em diversas situações, como, por exemplo, quando se pretende mostrar que uma certa propriedade vale para todas as linguagens geradas por GLC's.

Nas seções seguintes, vamos estudar técnicas que permitem transformar uma GLC G em outra GLC G' tal que L(G') = L(G) (sejam equivalentes, ou seja, gerem a mesma linguagem) e que possamos garantir que G' tenha certas propriedades desejáveis.

1. Eliminação de Símbolos Inúteis

Em gramáticas grandes, como gramáticas de linguagens de programação, pode acontecer de se esquecer de definir as regras relativas a uma variável; ou então, uma variável, apesar de ter suas regras já definidas, pode não ter sido utilizada ainda na formação de novas regras. Ambos os tipos de variáveis inúteis podem ser detectados. Após a detecção das variáveis inúteis, pode-se acrescentar novas regras para prever a definição ou uso das variáveis. Ou então, caso uma variável seja efetivamente inútil, deve-se eliminar todas as regras que possuem alguma ocorrência da mesma, e isso pode fazer com que até alguns símbolos terminais fiquem inúteis.

Definição 1.1

Seja uma GLC G = (V, Σ, R, P). Uma variável X ∈ V é dita ser uma variável **útil** se, e somente se, existem u, v ∈ (V U Σ) * tais que:

$$P \stackrel{*}{\Rightarrow} uXv \text{ e existe } w \in \Sigma^* \text{ tal que } uXv \stackrel{*}{\Rightarrow} w.$$

A primeira parte ($P \overset{*}{\Rightarrow} uXv$) exige que a variável X esteja em alguma forma sentencial produzida a partir do símbolo de partida P. Ou seja, se não é possível produzir X iniciando derivação com o símbolo de partida, então X não é uma variável útil para a CLG.

A segunda parte ($uXv \overset{*}{\Rightarrow} w$) exige que a partir de X seja produzida alguma sentença da linguagem, ou seja, palavras contendo apenas símbolos terminais. Se a partir de X só geradas palavras que sempre contêm não terminais, então X não é uma variável útil para a CLG, porque não contribui com geração de palavras da linguagem.

Exemplo 1.1

Seja a gramática ({P,A,B,C}, {a, b, c}, R, P), onde R contém as regras:

$P \rightarrow AB \mid \mathtt{a}$

$B \rightarrow \mathtt{b}$

$C \rightarrow \mathtt{c}$

- C é inútil: não existem u e v tais que $P \overset{*}{\Rightarrow} uCv$;
- A é inútil: não existe $w \in \Sigma^*$ tal que $A \overset{*}{\Rightarrow} w$;
- B é inútil: $P \overset{*}{\Rightarrow} uBv$ apenas para $u = A$ e $v = \lambda$, e não existe $w \in \Sigma^*$ tal que $AB \overset{*}{\Rightarrow} w$.

Um caso interessante neste exemplo é a variável B. Ela pode ser alcançada com derivações a partir do símbolo de partida, e permite que seja gerada ao menos uma palavra (“b”) contendo apenas símbolos terminais. Mas A única forma de alcançar B com derivações a partir do símbolo de partida é na forma sentencial AB, e é impossível transformar essa forma sentencial em uma sentença (palavra só com terminais). Logo B também não é útil para a GLC.

Eliminando-se as variáveis que não são úteis e todas as regras que as referenciam, além dos terminais b e c que não ocorrem em nenhuma regra associada às variáveis úteis, tem-se a gramática equivalente ({P}, {a}, {P → a}, P). A única palavra da linguagem dessa GLC é “a”. ❑

A seguir, usaremos a notação $\Rightarrow_G$, sendo G uma gramática, para indicar que a derivação está sendo aplicada com relação à gramática G.

Seja G = (V, Σ, R, P) tal que L(G) ≠ ∅. Uma GLC G’ equivalente a G, sem variáveis inúteis, pode ser construída em dois passos, como descrito a seguir.

PASSO 1:

Obtenha G’ = (V’, Σ, R’, P), onde:

- $V' = \{X \in V \mid X \overset{*}{\Rightarrow}_G w \text{ e } w \in \Sigma^*\}$, e
- $R' = \{r \in R \mid r \text{ não contém símbolo de } V - V'\}$;

Os itens acima indicam que primeiro deve-se determinar quais são as variáveis V' a partir das quais é possível derivar palavras contendo apenas símbolos terminais, que poderão então fazer parte da linguagem da GLC. Em seguida, o conjunto R' será construído contendo apenas as regras que não referenciam varáveis que tenham sido eliminadas, ou seja, as do conjunto V-V'.

O algoritmo abaixo pode ser usado para auxiliar a execução do Passo 1, permitindo definir automaticamente o conjunto $\{X \in V \mid X \stackrel{*}{\Rightarrow} w \text{ e } w \in \Sigma^*\}$, que são as variáveis a partir das quais é possível derivar palavras contendo apenas símbolos terminais.

Entrada: uma GLC $G = (V, \Sigma, R, P)$.
Saída: $\mathcal{I}_1 = \{X \in V \mid X \stackrel{*}{\Rightarrow} w \text{ e } w \in \Sigma^*\}$.
$\mathcal{I}_1 \leftarrow \emptyset$;
repita
    $\mathcal{N} \leftarrow \{Y \notin \mathcal{I}_1 \mid Y \rightarrow z \in R \text{ e } z \in (\mathcal{I}_1 \cup \Sigma)^*\}$;
    $\mathcal{I}_1 \leftarrow \mathcal{I}_1 \cup \mathcal{N}$
até $\mathcal{N} = \emptyset$;
retorne $\mathcal{I}_1$.

A cada iteração do algoritmo, tenta-se acrescentar ao conjunto de resposta uma nova variável que tenha ao menos uma regra associada a ela cujo lado direito seja formado apenas por terminais ou por variáveis que já pertençam ao conjunto resposta (e assim já sabemos que podem gerar sequências só com terminais). Quando não for mais possível acrescentar novas variáveis ao conjunto resposta, o processamento se encerra. Pode-se facilmente ver que o término em tempo finito é garantido, pois a cada iteração, no mínimo uma nova variável deve ser inserida ao conjunto resposta, e o número de variáveis de uma GLC é finito.

Antes de apresentar o método completo para eliminação de variáveis inúteis (incluindo o Passo 2 discutido posteriormente), vamos apresentar um exemplo de aplicação do Passo 1.

Exemplo 1.2

Seja a gramática G = ({A,B,C,D,E, F}, {0,1}, R, A}), onde R contém as regras:

$A \rightarrow ABC \mid AEF \mid BD$

$B \rightarrow B0 \mid 0$

$C \rightarrow 0C \mid EB$

$D \rightarrow 1D \mid 1$

$E \rightarrow BE$

$F \rightarrow 1F1 \mid 1$

Aplicando-se o Passo 1, teremos a seguinte sequência para a formação do conjunto de resposta:

iteração	conjunto resposta	ação
0	{ }	início do algoritmo
1	{ B, D, F }	acrescentou B, D, F porque: B → 0; D → 1; F → 1
2	{ B, D, F, A }	acrescentou A porque: A → BD
3	{ B, D, F, A }	parar porque não é possível inserir outras variáveis

Observe que, na iteração 2, a variável A é inserida porque tem regra cujo lado direito é formado por BD, ou seja, uma sequência formada de 2 não terminais em que ambos já estão no conjunto resposta. Na iteração 3, identifica-se que as variáveis que sobravam eram C e E, e nenhuma delas poderia satisfazer o critério para inserção no conjunto resposta, assim a iteração é interrompida.

Temos então V' = {A, B, D, F}. A gramática G' deve ser formada eliminando todas as regras que contêm as variáveis inúteis C e E, seja do lado esquerdo ou do lado direito:

$$A \to BD$$

$$B \to B0 \mid 0$$

$$D \to 1D \mid 1$$

$$F \to 1F1 \mid 1$$

❑

O Passo 1 é usado para construir uma gramática equivalente a GLC dada, mas contendo apenas variáveis que possam efetivamente gerar palavras da linguagem, ou seja, contendo apenas símbolos terminais.

O Passo 2, descrito a seguir, tem como objetivo eliminar as variáveis que não são alcançadas em derivações a partir do símbolo de partida.

PASSO 2:

Obtenha G'' = (V'', Σ, R'', P), onde:

- $V'' = \{X \in V' \mid P \overset{*}{\Rightarrow}_{G'} uXv\}$, e
- $R'' = \{r \in R' \mid r \text{ não contém símbolo de } V' - V''\}$;

Os itens acima são aplicados à gramática G' = (V', Σ, R', P) obtida como resultado do processamento do Passo 1. Indicam que deve-se determinar quais são as variáveis V'' que podem ser alcançadas por derivações começando com o símbolo de partida. Em seguida, o conjunto R'' será construído contendo apenas as varáveis que têm essa propriedade desejada, eliminando as variáveis identificadas como inúteis por esse critério.

O algoritmo abaixo pode ser usado para auxiliar a execução do Passo 2, permitindo definir automaticamente o conjunto $\{X \in V \mid P \overset{*}{\Rightarrow} uXv\}$, que são as variáveis alcançáveis por derivações iniciando com o símbolo de partida.

Entrada: uma GLC $G = (V, \Sigma, R, P)$.
Saída: $\mathcal{I}_2 = \{X \in V \mid P \overset{*}{\Rightarrow} uXv\}$.
$\mathcal{I}_2 \leftarrow \emptyset$; $\mathcal{N} \leftarrow \{P\}$;
repita
    $\mathcal{I}_2 \leftarrow \mathcal{I}_2 \cup \mathcal{N}$;
    $\mathcal{N} \leftarrow \{Y \notin \mathcal{I}_2 \mid X \to uYv \text{ para algum } X \in \mathcal{N}\}$
até $\mathcal{N} = \emptyset$;
retorne $\mathcal{I}_2$.

A cada iteração do algoritmo, tenta-se acrescentar ao conjunto de resposta uma nova variável que seja alcançada por alguma das variáveis já presentes no conjunto resposta, usando uma

derivação de comprimento 1. Isso equivale a coletar novas variáveis que estejam do lado direito de alguma regra cuja variável do lado esquerdo já esteja no conjunto resposta. Quando não for mais possível acrescentar novas variáveis ao conjunto resposta, o processamento se encerra. Pode-se facilmente ver que o término em tempo finito é garantido, pois a cada iteração, no mínimo uma nova variável deve ser inserida ao conjunto resposta, e o número de variáveis de uma GLC é finito.

Exemplo 1.3

Seja a gramática G' = ({A,B,D, F}, {0,1}, R, A}), onde R contém as regras:

$$A \to BD$$

$$B \to B0 \mid 0$$

$$D \to 1D \mid 1$$

$$F \to 1F1 \mid 1$$

Essa gramática foi obtida no Exemplo 1.2 com o processamento do Passo 1 sobre uma GCL G.

Aplicando-se o algoritmo do Passo 2, teremos a seguinte sequência para a formação do conjunto de resposta:

iteração	conjunto resposta	ação
0	{ }	início do algoritmo
1	{ A }	acrescentou símbolo de partida no início da primeira iteração
2	{ A, B, D }	acrescentou B, D porque: A → BD
3	{ A, B, D }	parar porque não é possível inserir outras variáveis

Temos então V'' = {A, B, D}. A gramática G'' será:

$$A \to BD$$

$$B \to B0 \mid 0$$

$$D \to 1D \mid 1$$

❑

2. Trocar regras de uma GLC por outras regras

A assunto discutido nesta seção não é exatamente uma propriedade que se deseja impor a uma GLC, mas sim um expediente que garante trocar regras por outras regras, sem alterar a linguagem que é gerada pela gramática. Essa abordagem será usada em outras transformações de gramáticas, posteriormente.

O teorema a seguir indica como isso irá funcionar.

Teorema 2.1

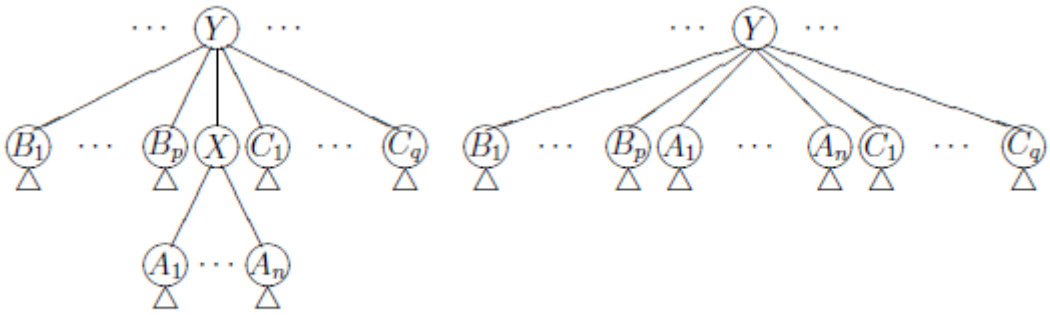
Seja uma GLC G = (V, Σ, R, P). Seja X→ w ∈ R, **onde X não pode ser o símbolo de partida**. Seja uma GLC G' = (V, Σ, R', P) onde R' é obtido assim:

(1) *para cada regra* $Y \to z \in R$, *para cada forma de escrever* z *como* $x_1 X x_2 \dots X x_{n+1}$, *onde* $n \geq 0$ *e* $x_i \in (V \cup \Sigma)^*$, *coloque a regra* $Y \to x_1 w x_2 \dots w x_{n+1}$ *em* R';

(2) *retire* $X \to w$ *de* R'.

Então G' é equivalente a G.

Não será exibida uma prova formal do Teorema 2.1, mas pode-se observar que a equivalência da linguagem é garantida porque, quando uma regra é eliminada, insere-se outras regras modificando outras regras, onde é inserido o lado direto da regra eliminada, mantendo a possibilidade de gerar as mesmas derivações. Observe um esquema representando essa ação em uma árvore de derivação:



(a) Antes. (b) Depois.

Exemplo 2.1

Seja a gramática G = ({P,A,B}, {a, b, c}, R, P), onde R contém as regras:

$P \to ABA$

$A \to \mathtt{a}A \mid \mathtt{a}$

$B \to \mathtt{b}B\mathtt{c} \mid \lambda$

Vamos eliminar (escolha arbitrária) a regra A → a . Usando o Teorema 2.1, vamos construir uma GLC G' equivalente a G, sem a regra que decidimos eliminar.

A regra P → ABA dará origem a quatro regras:

$P \to ABA \mid AB\mathtt{a} \mid \mathtt{a}BA \mid \mathtt{a}B\mathtt{a}$

e a regra A → aA dará origem a duas regras:

$A \to \mathtt{a}A \mid \mathtt{aa}$

Neste ponto, cabe uma explicação de como o Teorema 2.1 realmente funciona. A regra que decidimos eliminar é $A \rightarrow a$. Então temos que identificar todos os pontos nos lados direitos de outras regras em que a variável A aparece, e trocar pelo lado direito da regra eliminada ("a"), **gerando todas as combinações possíveis**. No caso da regra $P \rightarrow ABA$, a variável A aparece em 2 pontos do lado direito, e cada uma das ocorrências pode ficar com a variável A original ou trocá-la pelo lado direito da regra eliminada ("a"). No caso da regra $A \rightarrow aA$, a variável A aparece em 1 ponto do lado direito, podendo ficar com a variável A original ou trocá-la pelo lado direito da regra eliminada ("a"). Esse procedimento de considerar todas as combinações possíveis é importante porque a variável A pode ser trocada por "a" em alguma derivação, mas também pode ser trocada por outras subpalavras, se houver outras regras para A (exemplo: regra $A \rightarrow aA$). Manter as ocorrências da variável A nos lados direitos garante que A poderá ser eventualmente trocada por essas outras subpalavras.

Assim, no caso deste exemplo, a gramática resultante é G' = ({P,A,B}, {a, b, c}, R', P), onde R' contém as regras:

$$P \rightarrow ABA \mid AB\mathtt{a} \mid \mathtt{a}BA \mid \mathtt{a}B\mathtt{a}$$

$$A \rightarrow \mathtt{a}A \mid \mathtt{aa}$$

$$B \rightarrow \mathtt{b}B\mathtt{c} \mid \lambda$$

❑

No Exemplo 2.1, que o número de regras após o processamento com a eliminação da regra $A \rightarrow a$ aumentou, mas as derivações propiciadas são mais curtas. Por exemplo, "aa" tem a seguinte derivação na gramática original G:

$$\begin{aligned} P &\Rightarrow ABA && (\text{regra } P \rightarrow ABA) \\ &\Rightarrow \mathtt{a}BA && (\text{regra } A \rightarrow \mathtt{a}) \\ &\Rightarrow \mathtt{a}A && (\text{regra } B \rightarrow \lambda) \\ &\Rightarrow \mathtt{aa} && (\text{regra } A \rightarrow \mathtt{a}). \end{aligned}$$

E a mesma palavra tem a seguinte derivação em G':

$$\begin{aligned} P &\Rightarrow \mathtt{a}B\mathtt{a} && (\text{regra } P \rightarrow \mathtt{a}B\mathtt{a}) \\ &\Rightarrow \mathtt{aa} && (\text{regra } B \rightarrow \lambda). \end{aligned}$$

3. Eliminação de regras λ

A técnica apresentada na Seção 2 permite que seja eliminada qualquer regra λ de uma gramática, ou seja, qualquer regra do tipo $A \rightarrow \lambda$, com exceção da regra $P \rightarrow \lambda$ onde P é o símbolo de partida.

Mas, ao se eliminar uma regra λ usando-se o método da Seção 2, pode-se criar outras regras λ. Os passos abaixo representam uma técnica para eliminar todas as regras λ, preservando ou criando a regra $P \rightarrow \lambda$ (apenas no caso em que $\lambda \in L(G)$):

1. Determinar quais são as variáveis anuláveis, ou seja, quais variáveis podem gerar λ com uma ou mais derivações ($A \overset{*}{\Rightarrow} \lambda$).
2. Trocar todas as ocorrências das variáveis anuláveis em lados de direitos das regras por λ , seguindo o método da Seção 2 (considerar todas as combinações).
3. Eliminar todas as regras λ da gramática, com exceção da regra $P \rightarrow \lambda$, se houver.

O Passo 1 descrito acima consiste em determinar as variáveis anuláveis. O algoritmo a seguir pode ajudar nessa tarefa.

Entrada: uma GLC $G = (V, \Sigma, R, P)$;
Saída: $\{X \in V \mid X \overset{*}{\Rightarrow} \lambda\}$.
$\mathcal{A} \leftarrow \emptyset$;
repita
    $\mathcal{N} \leftarrow \{Y \notin \mathcal{A} \mid Y \rightarrow z \in R \text{ e } z \in \mathcal{A}^*\}$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{N}$
até $\mathcal{N} = \emptyset$;
retorne $\mathcal{A}$.

A cada iteração, são acrescentadas no conjunto resposta as variáveis que tenham alguma regra cujo lado direito seja λ ou formado por uma combinação de não terminais que já estejam no próprio conjunto resposta. Quando não for mais possível acrescentar novas variáveis ao conjunto resposta, o processamento se encerra. Pode-se facilmente ver que o término em tempo finito é garantido, pois a cada iteração, no mínimo uma nova variável deve ser inserida ao conjunto resposta, e o número de variáveis de uma GLC é finito.

Exemplo 3.1

Seja a gramática G = ({P,A,B}, {a, b, c}, R, P), onde R contém as regras:

$$P \rightarrow APB \mid C$$

$$A \rightarrow A\mathtt{aa}A \mid \lambda$$

$$B \rightarrow BB\mathtt{b} \mid C$$

$$C \rightarrow \mathtt{c}C \mid \lambda$$

Aplicando o algoritmo para encontrar as variáveis anuláveis:

iteração	conjunto resposta	ação
0	{ }	início do algoritmo
1	{ A, C }	acrescentou A, C porque: $A \rightarrow \lambda$; $C \rightarrow \lambda$
2	{ A, C, P, B }	acrescentou B, P porque: $P \rightarrow C$; $B \rightarrow C$
3	{ A, C, B, P }	parar porque não é possível inserir outras variáveis

O conjunto de variáveis anuláveis, neste caso, é todo o conjunto de não terminais da gramática. Em seguida, aplicamos a mesma técnica apresentada na Seção 2, usando as variáveis anuláveis e considerando o lado direito da regra como λ . Ou seja, construir

novas regras onde as ocorrências das variáveis anuláveis da lado direito são substituídas por λ , considerando todas as possíveis combinações. Todas as regras λ devem ser eliminadas, com exceção de $P \rightarrow \lambda$ (se existir). É fácil observar que neste caso teremos uma regra $P \rightarrow \lambda$ na gramática resultante, pois P é uma das variáveis anuláveis. O resultado será:

$$P \rightarrow APB \mid AP \mid AB \mid PB \mid A \mid B \mid C \mid \lambda$$

$$A \rightarrow A\mathtt{aa}A \mid \mathtt{aa}A \mid A\mathtt{aa} \mid \mathtt{aa}$$

$$B \rightarrow BB\mathtt{b} \mid B\mathtt{b} \mid \mathtt{b} \mid C$$

$$C \rightarrow \mathtt{c}C \mid \mathtt{c}$$

Observe como, na regra P → APB, por exemplo, foram consideradas todas as combinações possíveis para substituir as variáveis A, P e B por λ (somente porque essas 3 variáveis foram identificadas como anuláveis). No resultado das substituições, P → P foi descartada por motivos óbvios. O mesmo processo foi aplicado às regras A→ AaaA e B→ BBb. Todas as regras λ foram eliminadas, mantendo apenas P → λ (somente porque a própria variável de partida foi identificada como variável anulável). ❑

4. Eliminação de regras unitárias

Regras unitárias são regras da forma X → Y , onde X e Y são variáveis. Essas regras na realidade não geram derivações muito úteis, pois simplesmente trocam uma variável por outra. Vamos ver como é possível eliminá-las, sem alterar a linguagem gerada pela gramática.

Seja uma gramática G = (V, Σ, R, P). Diz-se que uma variável Z ∈ V é encadeada a uma variável X ∈ V se Z = X ou se existe uma sequência X → Y1, Y1 → Y2, . . . , Yn → Z de regras em R, n ≥ 0; no caso em que n = 0, tem-se apenas a regra X → Z. Ao conjunto de todas as variáveis encadeadas a X é dado o nome de enc(X). Note que X ∈ enc(X).
O algoritmo a seguir calcula tal conjunto.

Entrada: (1) uma GLC $G = (V, \Sigma, R, P)$, e
(2) uma variável $X \in V$.
Saída: $enc(X)$.
$\mathcal{U} \leftarrow \emptyset$; $\mathcal{N} \leftarrow \{X\}$;
repita
$\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{N}$;
$\mathcal{N} \leftarrow \{Y \notin \mathcal{U} \mid Z \rightarrow Y \in R \text{ para algum } Z \in \mathcal{N}\}$
até $\mathcal{N} = \emptyset$
retorne $\mathcal{U}$.

A cada iteração, são acrescentadas no conjunto resposta enc(X) as variáveis que sejam encadeadas a alguma outra variável que já esteja no próprio conjunto. Quando não for mais possível acrescentar novas variáveis, o processamento se encerra. Pode-se facilmente ver que o término em tempo finito é garantido, pois a cada iteração, no mínimo uma nova variável deve ser inserida ao conjunto resposta, e o número de variáveis de uma GLC é finito.

Usando os conjuntos enc(X), para X ∈ V, é simples construir uma gramática equivalente a uma GLC qualquer, eliminando as regras unitárias. Se G = (V, Σ, R, P) então uma gramática equivalente a G sem regras unitárias é G' = (V, Σ, R', P), onde

$$R' = \{X \to w \mid Y \in enc(X), Y \to w \in R \text{ e } w \notin V\}$$

A fórmula acima indica que a GLC G' não terá regras unitárias (não permite que w seja apenas uma variável). Para compensar a eliminação dessas regras, são acrescentadas novas regras: para todas as variáveis de enc(X), regras com lado direito igual às regras de X devem ser acrescentadas. Esse processo deve ficar claro com o exemplo a seguir.

Exemplo 4.1

Seja a gramática G usada para representar expressões aritméticas, com as regras abaixo:

$$E \to E{+}T \mid T$$

$$T \to T{*}F \mid F$$

$$F \to (E) \mid \mathtt{t}$$

Os símbolos terminais são +, *, t, (,).

Aplicando o algoritmo para encontrar enc(E):

iteração	conjunto resposta	ação
0	{ }	início do algoritmo
1	{ E }	acrescentou E no início da primeira iteração
2	{ E, T }	acrescentou T, porque: E → T
3	{ E, T, F }	acrescentou F, porque: T→ F
4	{ E, T, F }	parar porque não é possível inserir outras variáveis

Temos então enc(E) = { E, T, F }, que significa que a partir de E podem ser produzidas essas subpalavras formadas por apenas uma variável.

Se o mesmo algoritmo for aplicado a T e F, teremos:

enc(T) = { T, F }

enc (F) = { F }

Uma gramática equivalente, sem regras unitárias, pode ser construída usando o método descrito nesta seção, eliminando as regras unitárias e acrescentando em todas as variáveis de enc(X), regras com lado direito igual às regras de X. A nova gramática terá então as seguintes regras:

$E \rightarrow E{+}T \mid T{*}F \mid (E) \mid \mathtt{t}$

$T \rightarrow T{*}F \mid (E) \mid \mathtt{t}$

$F \rightarrow (E) \mid \mathtt{t}$ ❑

5. Símbolo de partida não recursivo

Esta transformação é muito simples, e busca apenas garantir que o símbolo de partida de uma GLC não seja recursivo, ou seja, não seja usado em lado direito de nenhuma regra.

Seja uma gramática G = (V, Σ, R, P) em que o símbolo de partida P é usado recursivamente, direta ou indiretamente, ou seja, P aparece no lado direito de alguma regra do conjunto R. A gramática G' = (V' U {P'}, Σ, R U {P'→ P}, P') é equivalente a G, e em G' o símbolo de partida não é recursivo.

Exemplo 5.1

Vamos utilizar novamente a gramática G que representa expressões aritméticas, com as regras abaixo:

E→ E + T | T

T → T * F | F

F → (E) | t

Em G, o símbolo de partida E é claramente diretamente recursivo (aparece do lado direito de uma regra do próprio símbolo E) e também indiretamente recursivo (aparece ainda do lado direito de uma regra do símbolo F).

A gramática G' a seguir é equivalente a G e não tem símbolo de partida recursivo.

E' → E

E→ E + T | T

T → T * F | F

F → (E) | t ❑

A gramática G' do Exemplo 5.1 não tem símbolo de partida recursivo, mas observe que essa transformação fez com que G' passasse a ter uma nova regra unitária. Neste caso, a gramática original G já tinha outras regras unitárias. Mas se G não tivesse regras unitárias, mesmo assim G' teria uma nova regra unitária introduzida pela transformação apresentada nesta seção.

6. Eliminar variável do lado direito de uma regra

Esta transformação permite eliminar o uso de uma variável, ao custo de muitas vezes aumentar o número total de regras de uma gramática.

Seja uma GLC G = (V, Σ, R, P) tal que $X \rightarrow uYv \in R$, onde $Y \in V$ e $Y \neq X$. Considere que **todas** a regras de Y em R sejam $Y \rightarrow w_1 \mid w_2 \mid \ldots \mid w_n$. Construa G' = (V, Σ, R', P) onde:

$$R' = (R - \{X \rightarrow uYv\}) \cup \{X \rightarrow uw_1v \mid uw_2v \mid \ldots \mid uw_nv\}$$

Então G' é equivalente a G.

O método acima consiste em criar novas regras substituindo uma variável Y que aparece no lado direito de uma regra de X por cada um dos lados direitos das regras de Y.

Exemplo 6.1

Vamos utilizar novamente a gramática G que representa expressões aritméticas, com as regras abaixo:

E→ E + T | T

T → T * F | F

F → (E) | t

Vamos escolher, arbitrariamente, eliminar o uso da variável F no lado direito das regras da variável T. Para isso, temos que substituir F por cada lado direito de suas regras.

A regra T → T * F deve ser substituída pelas regras T → T * (E) e T → T * t .

A regra T → F deve ser substituída pelas regras T → (E) e T → T * t .

A gramática G' será:

E→ E + T | T

T → T * (E) | T * t | (E) | t

F → (E) | t ❑

A gramática G' construída no Exemplo 6.1 é equivalente a G, sem usar a variável F no lado direito das regras de T. O número de regras total aumentou, comparando G e G', de 6 para 8. Outra observação interessante, neste caso, é que a variável F tornou-se inútil e poderia ser eliminada da gramática.

7. Eliminar recursividade à esquerda

Uma regra é recursiva à esquerda quando um mesmo símbolo não terminal aparece do lado esquerdo e como primeiro símbolo do lado direito de uma regra. Exemplo:

E→ E + T | T

Na regra acima, o símbolo E é recursivo à esquerda. Algumas abordagens para construção automática de analisadores sintáticos (programas que verificam se uma palavra pertence à linguagem de uma gramática) não podem ser usadas para gramáticas que tenham regras com recursividade à esquerda.

Nesta seção, vamos estudar um método que, dada uma GLC que contenha regras recursivas à esquerda, permite construir uma gramática equivalente sem recursividade à esquerda.

Sejam as seguintes todas as regras e uma variável X em uma GLC G:

$$X \to Xy_1 \mid Xy_2 \mid \ldots \mid Xy_n \mid w_1 \mid w_2 \mid \ldots \mid w_k$$

onde nenhum w_i começa com X. Obviamente, se k=0, todas as regras de X terão variáveis X do seu lado direito e então X será uma variável inútil, pois não poderá produzir palavras contendo apenas terminais. Supondo então que k não seja 0, as palavras produzidas por aplicações das regras de X, recursivamente, terão o seguinte formato:

- começam com algum w_i, para i = 1, .., k;
- têm em seguida 0 ou mais ocorrências de algum y_j, para j = 1, ..., n.

Esse formato de palavras poderia ser produzido também usando recursividade à direita, em vez de recursividade à esquerda, trocando as regras de X por:

$$X \to w_1 \mid w_2 \mid \ldots \mid w_k \mid w_1Z \mid w_2Z \mid \ldots \mid w_kZ$$

$$Z \to y_1 \mid y_2 \mid \ldots \mid y_n \mid y_1Z \mid y_2Z \mid \ldots \mid y_nZ$$

onde Z é uma variável nova.

Exemplo 7.1

Seja a gramática G = ({E}, {t,+, _, (,)}, R, E), sendo R dado por:

$$E \to E{+}E \mid E{*}E \mid (E) \mid \mathtt{t}$$

A variável E é recursiva à esquerda. Usando o método apresentado nesta seção, pode-se substituir as regras de E por:

$$E \to (E) \mid \mathtt{t} \mid (E)Z \mid \mathtt{t}Z$$

$$Z \to {+}E \mid {*}E \mid {+}EZ \mid {*}EZ$$

Nas transformações acima, aplicando o método para eliminar recursividade à esquerda, os componentes “y” das regras de E são “+E” e “*E”, e os componentes “w” são “(E)” e “t”. ❑

8. Formas Normais

Formas normais são restrições rígidas na forma das produções de gramáticas, mas que não reduzem o poder de geração de Gramáticas Livres de Contexto. Os objetivos de impor essas restrições pode ser permitir/facilitar uso das gramáticas por reconhecedores de linguagens automáticos ou facilitar a prova de teoremas.

Vamos estudar duas formas normais:

1. Forma Normal de Chomsky
2. Forma Normal de Greibach

Antes de apresentar essas duas formas normais, vamos retornar a um assunto que estudamos anteriormente: as gramáticas regulares. Lembramos que uma gramática é dita regular se for gramática (V, Σ ,R, P), em que cada regra tem uma das formas:

- $X \rightarrow a$
- $X \rightarrow aY$
- $X \rightarrow \lambda$

onde $X, Y \in V$ e $a \in \Sigma$.

Essas são restrições impostas para o formato das regras, mas não dizemos que isso constitui uma forma normal. Conforme afirmamos anteriormente, ainda sem prova formal, as restrições impostas pelas gramáticas regulares efetivamente diminuem o poder de geração das gramáticas livres de contexto. Por exemplo, seja G a gramática representada simplificadamente pelas regras a seguir:

$S \rightarrow aSb \mid \lambda$

A linguagem aceita por G é $\{ a^n b^n$, para $n \geq 0 \}$. A gramática G é livre de contexto, assim a linguagem gerada é livre de contexto. A gramática G não é regular, pois a regra $S \rightarrow aSb$ não satisfaz as restrições de gramáticas regulares. Na realidade, é impossível construir uma gramática regular que gere a linguagem $\{a^n b^n$, para $n \geq 0\}$. Essa é uma das linguagens livres de contexto que não é regular. Mais tarde, vamos demonstrar isso formalmente e apresentar um método que facilita a demonstração de que uma linguagem não é regular.

As abordagens que veremos a seguir restringem o formato das regras de gramáticas livres de contexto mas não diminuem seu poder de geração de palavras.

8.1. Forma Normal de Chomsky

Definição 8.1.1

Uma GLC G = (V, Σ, R, P) é dita estar na forma normal de Chomsky (FNC) se todas as suas regras estão nas formas:

- $P \rightarrow \lambda \ se \ \lambda \in L(G);$
- $X \rightarrow YZ \ para \ Y, Z \in V;$
- $X \rightarrow a \ para \ a \in \Sigma.$

Além disso, o símbolo de partida não pode ser recursivo, isto é, não pode aparecer no lado direito de nenhuma regra de produção.

A FNC exige que as regras de uma GLC tenham do seu lado direito a palavra vazia ou um único símbolo terminal ou exatamente dois símbolos não terminais (exceto o símbolo de partida). Veremos em seguida um método que, dada uma GLC G qualquer, permite construir uma GLC G' equivalente a G tal que G' esteja na Forma Normal de Chomsky.

Inicialmente, deve-se aplicar as seguintes transformações na GLC original, se necessárias, e nesta ordem apresentada:

1. Tornar o símbolo inicial não recursivo (Seção 5).
2. Eliminar regras λ (Seção 3).
3. Eliminar regras unitárias (Seção 4).
4. Eliminar símbolos inúteis (Seção 1).

Se as transformações acima forem aplicadas a uma gramática e o resultado for uma GLC G = (V, Σ, R, P), então podemos afirmar com certeza que as regras de G estarão na forma:

- $P \rightarrow \lambda \ se \ \lambda \in L(G);$
- $X \rightarrow a \ para \ a \in \Sigma;$
- $X \rightarrow w \ para \ |w| \geq 2.$

Além disso, o símbolo de partida de G não será recursivo. Podemos ver que apenas o terceiro tipo de formato apresentado acima pode não estar atendendo exatamente a Forma Normal de Chomsky. Vamos então descrever um método para executar outras transformações que finalmente irão produzir uma GLC equivalente, na Forma Normal de Chomsky.

O procedimento que vamos propor é dividido em 2 passos:

1. Modificar cada regra $X \rightarrow w$, $|w| \geq 2$, se necessário, de forma que ela fique contendo apenas variáveis. Para isto, substituir cada ocorrência de cada $a \in \Sigma$ que ocorra em w por uma variável, da seguinte forma: se existe uma regra da forma $Y \rightarrow a$ e esta é a única regra Y , substituir as ocorrências de a por Y em w; caso contrário, criar uma regra $Y \rightarrow a$, onde Y é uma variável nova, e substituir as ocorrências de a por Y em w.

2. Substituir cada regra $X \rightarrow Y_1Y_2 \ldots Y_n$, $n \geq 3$, onde cada Y_i é uma variável, pelo conjunto das regras: $X \rightarrow Y_1Z_1$, $Z_1 \rightarrow Y_2Z_2$, $\ldots$, $Z_{n-2} \rightarrow Y_{n-1}Y_n$, onde $Z_1, Z_2, \ldots, Z_{n-2}$ são variáveis novas.

Exemplo 8.1.1

Seja a gramática G com as seguintes regras:

$S \rightarrow aABC \mid a$

$A \rightarrow aA \mid a$

$B \rightarrow bcB \mid bc$

$C \rightarrow cC \mid c$

G já satisfaz as condições iniciais que impusemos: não tem símbolo de partida recursivo, não tem regras λ, não tem regras unitárias e não tem símbolos inúteis. Se alguma dessas condições não fosse satisfeita, bastaria aplicarmos os métodos estudados em seções anteriores.

A transformação seguinte (item 1 do método explicado nesta seção), consiste em fazer com que toda regra cujo lado direito tenha comprimento maior ou igual a 2 seja composta apenas por variáveis. Aplicando o método sugerido, teremos então:

$S \rightarrow X_1ABC \mid a$

$A \rightarrow X_1A \mid a$

$B \rightarrow X_2X_3B \mid X_2X_3$

$C \rightarrow X_3C \mid c$

$X_1 \rightarrow a$

$X_2 \rightarrow b$

$X_3 \rightarrow c$

Veja que foram criados 3 novos símbolos, X_1 , X_2 e X_3 que produzem unicamente os símbolos terminais a, b e c. Então as ocorrências de a, b e c no lado direito das regras com comprimento maior ou igual a 2 são substituídas por X_1 , X_2 e X_3 , respectivamente.

Importante: não pode ser usado o símbolo S, que produz a, nem o símbolo C, que produz c, porque esses símbolos terminais têm também outras regras associadas a eles. Caso houvesse algum símbolo não terminal que só produzisse a, por exemplo, ele poderia ser aproveitado, sem necessidade de criar um novo símbolo X_1 para isso.

A gramática obtida está mais próxima da FNC. As únicas regras que violam as restrições dessa forma normal são $S \rightarrow X_1ABC$ e $B \rightarrow X_2X_3B$, pois têm lado direito com comprimento maior que 2. Para sanar esse problema, basta aplicar o item 2 do método apresentado nesta seção, chegando ao seguinte resultado:

$S \rightarrow X_1X_4 \mid a$

$X_4 \rightarrow AX_5$

$X_5 \rightarrow BC$

$A \rightarrow X_1A \mid a$

$B \rightarrow X_2X_6 \mid X_2X_3$

$X_6 \rightarrow X_3B$

$C \rightarrow X_3C \mid c$

$X_1 \rightarrow a$

$X_2 \rightarrow b$

$X_3 \rightarrow c$

Observe que o procedimento é simples. Basta trocar o lado direito que tinha comprimento maior que 2 por uma palavra com o primeiro símbolo, seguido de um novo não terminal que deverá produzir o resto da sequência. Esse processo deve ser repetido até restem exatamente 2 não terminais.

A gramática acima está na Formal Normal de Chomsky. ❑

8.2. Forma Normal de Greibach

Definição 8.1.1

Uma GLC G = (V, Σ, R, P) é dita estar na forma normal de Greibach (FNG) se todas as suas regras estão na forma:

- $P \rightarrow \lambda \ se \ \lambda \in L(G);$
- $X \rightarrow ay \ para \ a \in \Sigma \ e \ y \in V^*$

Na Forma Normal de Greibach, os lados direitos das regras, se não forem λ, são sempre formados por um único terminal, seguido de 0 ou mais não terminais.

Dada uma GLC qualquer, para gerar outra GLC equivalente à primeira e que esteja na Forma Normal de Greibach, pode-se seguir os passos descritos a seguir. Vamos apresentar o método passo a passo aplicando a um exemplo.

Primeiro, assume-se que a GLC está na Forma Normal de Chomsky. Se esse não for o caso, aplicam-se as transformações apresentadas na Seção 8.1 e uma GLC na Forma Normal de Chomsky pode ser obtida, equivalente à primeira.

Em seguida, atribui-se uma numeração sequencial para cada não terminal da GLC. O símbolo de partida deve estar associado com o número 1, e os demais não terminais são associados a 2, 3, ..., em sequência. Pode-se escolher os demais não terminais em qualquer ordem.

Por exemplo, a gramática G a seguir já está na Forma Normal de Chomsky:

$$\begin{aligned} \text{G: } & S \rightarrow AB \mid \lambda \\ & A \rightarrow AB \mid CB \mid a \\ & B \rightarrow AB \mid b \\ & C \rightarrow AC \mid c \end{aligned}$$

Podemos atribuir uma numeração como: S = 1; A = 2; B= 3; C = 4.

O passo seguinte é construir uma gramática em que cada regra tem uma das seguintes formas:

i) $S \rightarrow \lambda$

ii) $A \rightarrow aw$

iii) $A \rightarrow Bw$

onde w ∈ V* e o número associado a B na ordenação escolhida é sempre **maio**r que o número associado a A.

Uma vez que a gramática já está na Forma Normal de Chomsky, então o símbolo de partida não pode ser recursivo. Neste caso, certamente as regras do símbolo de partida vão satisfazer o critério da ordenação, pois qualquer variável que estiver no início do lado direito de suas regras terá numeração maior que 1. De fato, podemos ver que na GLC G acima, o símbolo de partida é S, e a regra S → AB satisfaz a ordenação, com S=1 e A = 2.

Vamos analisar então as regras dos símbolos não terminais seguindo exatamente a numeração escolhida, na ordem, assim começamos pelo símbolo A. Para regra A → CB, a ordenação está OK, pois A =2 e C = 4. Mas a regra A → AB não satisfaz o critério, pois o primeiro símbolo no lado direito é também A e logo tem o mesmo número. Note que não seria possível ter símbolo no lado direito com número menor que o número associado a A, pois é o primeiro da nossa sequência e não pode ser S. Uma violação do critério de numeração só pode mesmo acontecer com um símbolo com o mesmo número, e esse caso é obviamente uma situação de regra com recursividade à esquerda. Podemos então aplicar a técnica apresentada na Seção 7, para eliminação de recursividade à esquerda. Teremos então a seguinte gramática:

$$S \rightarrow AB \mid \lambda$$
$$A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$$
$$B \rightarrow AB \mid b$$
$$C \rightarrow AC \mid c$$
$$R_1 \rightarrow BR_1 \mid B.$$

Observe que uma nova variável R_1 foi criada, para simular a funcionalidade de A usando recursividade à direita. Obviamente, as novas regras de A sempre irão satisfazer o critério de ordenação, pois o primeiro não terminal do lado direito não poderá ser S (símbolo de partida não é recursivo) e nem o próprio A, já que eliminamos a recursividade à esquerda. No nosso exemplo, a nova regra introduzida começa com C, que tem número C = 4.

Vamos analisar agora as regras de B, que é o próximo na sequência escolhida. A regra B → AB não satisfaz o critério de numeração, pois B = 3 e A = 2. Quando isso acontecer, iremos aplicar a técnica apresentada na Seção 6, que permite eliminar um não terminal no lado direito de uma regra substituindo-o pelo lado direito de cada a regra do terminal substituído. No caso do exemplo, vamos substituir em B → AB a variável A pelo lado direito de cada uma de suas regras. Neste momento, A tem 4 regras associadas, assim B → AB será trocada por 4 novas regras, obtendo o seguinte resultado:

$$S \rightarrow AB \mid \lambda$$
$$A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$$
$$B \rightarrow CBR_1B \mid aR_1B \mid CBB \mid aB \mid b$$
$$C \rightarrow AC \mid c$$
$$R_1 \rightarrow BR_1 \mid B.$$

As novas regras inseridas para B irão ter as primeiras variáveis do lado direito com número maior que o de A (pois são originadas de B). No caso do exemplo, elas começavam com C e assim satisfazem o critério de ordenação quando substituídas na regra de B. Poderia ocorrer um caso em que recursividade à esquerda seria gerada, mas nunca um símbolo com numeração inferior, por causa do método de construção.

Vamos analisar agora as regras de C. A regra C → AC não satisfaz o critério de numeração, pois C = 4 e A = 2. Vamos novamente aplicar a técnica da Seção 6, substituindo A por cada um dos lados direitos de suas regras. As novas regras poderão então começar com B (neste caso fazemos nova aplicação da técnica da Seção 6) ou C (recursividade à esquerda, demandando a técnica da Seção 7). Podemos ver na gramática abaixo, resultante das substituições em C → AC, que será gerado o caso em que ocorre recursividade à esquerda em C:

$$S \rightarrow AB \mid \lambda$$
$$A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$$
$$B \rightarrow CBR_1B \mid aR_1B \mid CBB \mid aB \mid b$$
$$C \rightarrow CBR_1C \mid aR_1C \mid CBC \mid aC \mid c$$
$$R_1 \rightarrow BR_1 \mid B$$

Para tratar o caso da recursividade à esquerda em C, uma nova variável R_2 é inserida:

$$S \rightarrow AB \mid \lambda$$
$$A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$$
$$B \rightarrow CBR_1B \mid aR_1B \mid CBB \mid aB \mid b$$
$$C \rightarrow aR_1C \mid aC \mid c \mid aR_1CR_2 \mid aCR_2 \mid cR_2$$
$$R_1 \rightarrow BR_1 \mid B$$
$$R_2 \rightarrow BR_1CR_2 \mid BCR_2 \mid BR_1C \mid BC$$

Neste ponto, todas as variáveis originais S, A, B, C satisfazem o critério de ordenação, ou seja, o primeiro símbolo do lado direito de suas regras, se for um não terminal, terá número associado maior que o número associado ao símbolo do lado esquerdo da regra. Obviamente, a variável com maior número associado (C = 4) não poderia ter nenhuma regra cujo lado direito comece com não terminal, pois assim não seria possível satisfazer o critério de ordenação. de fato, todas as regras de C, na versão atual da gramática, começam com terminal.

Ao chegar a este ponto, deve-se então aplicar a técnica da Seção 6 abordando as variáveis B, A e S, nesta ordem, e depois as novas variáveis que foram criadas no processo (R_1 e R_2). Primeiro, substituir C por cada lado direito de suas regras, nas regras de B que começarem com C:

$$
\begin{aligned}
S &\to AB \mid \lambda \\
A &\to CBR_1 \mid aR_1 \mid CB \mid a \\
B &\to aR_1B \mid aB \mid b \\
&\to aR_1CBR_1B \mid aCBR_1B \mid cBR_1B \mid aR_1CR_2BR_1B \mid aCR_2BR_1B \mid cR_2BR_1B \\
&\to aR_1CBB \mid aCBB \mid cBB \mid aR_1CR_2BB \mid aCR_2BB \mid cR_2BB \\
C &\to aR_1C \mid aC \mid c \mid aR_1CR_2 \mid aCR_2 \mid cR_2 \\
R_1 &\to BR_1 \mid B \\
R_2 &\to BR_1CR_2 \mid BCR_2 \mid BR_1C \mid BC.
\end{aligned}
$$

Deve-se aplicar a mesma técnica para substituir C no lado direito das regras de A que começam com C. Em seguida, substituir A no lado direito da regra de S que começa com A. O resultado será:

$$
\begin{aligned}
S &\to \lambda \\
&\to aR_1B \mid aB \\
&\to aR_1CBR_1B \mid aCBR_1B \mid cBR_1B \mid aR_1CR_2BR_1B \mid aCR_2BR_1B \mid cR_2BR_1B \\
&\to aR_1CBB \mid aCBB \mid cBB \mid aR_1CR_2BB \mid aCR_2BB \mid cR_2BB \\
A &\to aR_1 \mid a \\
&\to aR_1CBR_1 \mid aCBR_1 \mid cBR_1 \mid aR_1CR_2BR_1 \mid aCR_2BR_1 \mid cR_2BR_1 \\
&\to aR_1CB \mid aCB \mid cB \mid aR_1CR_2B \mid aCR_2B \mid cR_2B \\
B &\to aR_1B \mid aB \mid b \\
&\to aR_1CBR_1B \mid aCBR_1B \mid cBR_1B \mid aR_1CR_2BR_1B \mid aCR_2BR_1B \mid cR_2BR_1B \\
&\to aR_1CBB \mid aCBB \mid cBB \mid aR_1CR_2BB \mid aCR_2BB \mid cR_2BB \\
C &\to aR_1C \mid aC \mid c \mid aR_1CR_2 \mid aCR_2 \mid cR_2 \\
R_1 &\to BR_1 \mid B \\
R_2 &\to BR_1CR_2 \mid BCR_2 \mid BR_1C \mid BC.
\end{aligned}
$$

Finalmente, as únicas regras que não satisfazem a Forma Normal de Greibach são as dos novos símbolos criados no processo (R_1 e R_2). Par se obter a gramática final na FNG, a técnica da Seção 6 é utilizada pela última vez. As regras de R_1 e R_2 deverão ser trocadas pelas exibidas a seguir, chegando assim a uma gramática na FNG.

A gramática original perdeu completamente sua simplicidade. Obter uma gramática na FNG deve ser um procedimento realizado de forma automática, evitando assim o trabalho excessivo e os erros que tal processo pode trazer.

$$
\begin{aligned}
R_1 &\to aR_1BR_1 \mid aBR_1 \mid bR_1 \\
&\to aR_1CBR_1BR_1 \mid aCBR_1BR_1 \mid cBR_1BR_1 \mid aR_1CR_2BR_1BR_1 \mid aCR_2BR_1BR_1 \mid cR_2BR_1 \\
&\to aR_1CBBR_1 \mid aCBBR_1 \mid cBBR_1 \mid aR_1CR_2BBR_1 \mid aCR_2BBR_1 \mid cR_2BBR_1 \\
R_1 &\to aR_1B \mid aB \mid b \\
&\to aR_1CBR_1B \mid aCBR_1B \mid cBR_1B \mid aR_1CR_2BR_1B \mid aCR_2BR_1B \mid cR_2BR_1B \\
&\to aR_1CBB \mid aCBB \mid cBB \mid aR_1CR_2BB \mid aCR_2BB \mid cR_2BB \\
R_2 &\to aR_1BR_1CR_2 \mid aBR_1CR_2 \mid bR_1CR_2 \\
&\to aR_1CBR_1BR_1CR_2 \mid aCBR_1BR_1CR_2 \mid cBR_1BR_1CR_2 \mid aR_1CR_2BR_1BR_1CR_2 \mid \\
&\quad aCR_2BR_1BR_1CR_2 \mid cR_2BR_1BR_1CR_2 \\
&\to aR_1CBBR_1CR_2 \mid aCBBR_1CR_2 \mid cBBR_1CR_2 \mid aR_1CR_2BBR_1CR_2 \mid aCR_2BBR_1CR_2 \\
&\quad cR_2BBR_1CR_2 \\
R_2 &\to aR_1BCR_2 \mid aBCR_2 \mid bCR_2 \\
&\to aR_1CBR_1BCR_2 \mid aCBR_1BCR_2 \mid cBR_1BCR_2 \mid aR_1CR_2BR_1BCR_2 \mid aCR_2BR_1BCR_2 \\
&\quad cR_2BR_1BCR_2 \\
&\to aR_1CBBCR_2 \mid aCBBCR_2 \mid cBBCR_2 \mid aR_1CR_2BBCR_2 \mid aCR_2BBCR_2 \mid cR_2BBCR_2 \\
R_2 &\to aR_1BR_1C \mid aBR_1C \mid bR_1C \\
&\to aR_1CBR_1BR_1C \mid aCBR_1BR_1C \mid cBR_1BR_1C \mid aR_1CR_2BR_1BR_1C \mid aCR_2BR_1BR_1C \mid \\
&\quad cR_2BR_1BR_1C \\
&\to aR_1CBBR_1C \mid aCBBR_1C \mid cBBR_1C \mid aR_1CR_2BBR_1C \mid aCR_2BBR_1C \mid cR_2BBR_1C \\
R_2 &\to aR_1BC \mid aBC \mid bC \\
&\to aR_1CBR_1BC \mid aCBR_1BC \mid cBR_1BC \mid aR_1CR_2BR_1BC \mid aCR_2BR_1BC \mid cR_2BR_1BC \\
&\to aR_1CBBC \mid aCBBC \mid cBBC \mid aR_1CR_2BBC \mid aCR_2BBC \mid cR_2BBC.
\end{aligned}
$$