Material de Apoio #1

Transformações sobre Gramáticas Livres de Contexto

Transformações sobre gramáticas livres de contexto (GLC) podem ser necessárias por várias razões como, por exemplo, possibilitar o funcionamento de uma determinada classe de algoritmos de reconhecimento sintático para uma gramática. Este material de apoio detalha o funcionamento de três transformações: eliminação de produções vazias; eliminação da recursividade à esquerda e, por fim, a mais simples dentre elas, fatoração.

1 Eliminação de produções vazias

Produções vazias em uma gramática livre de contexto são aquelas produções da forma $A \rightarrow \epsilon$, ou ainda situações onde $B \rightarrow AAA$ e onde o não-terminal A pode derivar para vazio. Nestas duas situações, tanto A quanto B podem derivar para vazio: A de forma direta; e B porque os três não-terminais A podem, conjuntamente, derivar para vazio. Vemos a seguir dois métodos para a eliminação de produções vazias. O primeiro, baseado na intuição e no conhecimento da gramática, serve mais como uma motivação para o segundo método, mais genérico e automático.

1.1 Método manual: seguindo a intuição

A remoção das produções vazias pode eventualmente ser feita utilizando a intuição e o conhecimento sobre a gramática. Veja, por exemplo, a gramática seguinte que reconhece chamada de funções em uma linguagem imperativa:

Chamada_de_função	→	**id** (Argumentos_opcionais)
Argumentos_opcionais	→	Lista_argumentos
Argumentos_opcionais	→	ϵ
Lista_argumentos	→	Arg
Lista_argumentos	→	Arg , Lista_argumentos

No exemplo acima, a produção vazia Argumentos_opcionais →ϵ poderia ser facilmente removida alterando ligeiramente a gramática. Isto pode ser feito neste caso porque os argumentos da função são opcionais. Podemos portanto adicionar uma regra de produção para o não-terminal Chamada_de_função sem o não-terminal entre os dois parênteses, removendo a produção vazia da gramática. O resultado final fica assim:

Chamada_de_função	→	**id** (Argumentos_opcionais)
Chamada_de_função	→	**id** ( )
Argumentos_opcionais	→	Lista_argumentos
Lista_argumentos	→	Arg
Lista_argumentos	→	Arg , Lista_argumentos

1.2 Método automático

Embora possamos seguir a nossa intuição para remover produções vazias de uma gramática, podemos observar várias situações mais complexas onde a aplicação do nosso conhecimento da gramática é insuficiente para remover as produções vazias. Nestas situações, convém aplicar um algoritmo automático para removê-las. Para ilustrar o funcionamento do algoritmo, vejamos o exemplo abaixo baseado na gramática seguinte:

A	→	B **z** B
B	→	**b**
B	→	ϵ

Ao remover a produção $B \rightarrow \epsilon$, somos obrigados a replicar as produções que contém B no lado direito. No exemplo, a produção que contém B do lado direito é a produção $A \rightarrow B\mathbf{z}B$. Uma vez que B pode se tornar vazio, a única forma de remover com segurança a produção vazia é considerar todas as combinações possíveis para a forma sentencial $B\mathbf{z}B$, ou seja, adicionar as regras $A \rightarrow \mathbf{z}|B\mathbf{z}|\mathbf{z}B|B\mathbf{z}B$ que representam todas as formas sentenciais válidas a partir de A sabendo que a produção vazia a partir de B foi removida. Sendo assim a gramática resultante – capaz de gerar exatamente a mesma linguagem da gramática original com a produção vazia – termina sendo da seguinte forma:

$$
\begin{array}{lcl}
\text{A} & \rightarrow & \mathbf{z} \\
\text{A} & \rightarrow & \mathbf{z}\ \text{B} \\
\text{A} & \rightarrow & \text{B}\ \mathbf{z} \\
\text{A} & \rightarrow & \text{B}\ \mathbf{z}\ \text{B} \\
\text{B} & \rightarrow & \mathbf{b}
\end{array}
$$

Podemos generalizar o exemplo acima. Se B aparece n vezes em alguma produção P qualquer, esta produção P será replicada 2^n vezes, cada qual com uma combinação diferente. Esta descrição genérica pode ser posta em funcionamento através do seguinte algoritmo, dividido em três passos, que considera que a gramática inicial é $G = (N, T, P, S)$, onde N são os símbolos não-terminais, T os terminais, P as produções e S o símbolo não-terminal inicial:

1. **Reunir todos os não-terminais que geram ϵ**

   Para obter todos os não-terminais que geram ϵ e reuní-los em um conjunto identificado por N_ϵ, devemos inicializar N_ϵ com todos os não-terminais que derivam *diretamente* para vazio. Esta configuração inicial é representada por $N_\epsilon = \{A | A \rightarrow \epsilon\}$.

   Para identificar todos os não-terminais que derivam *indiretamente* para vazio, devemos repetir a regra $N_\epsilon = N_\epsilon \cup \{X \mid X \rightarrow X_1 ... X_n \in P \text{ tal que } X_1 ... X_n \in N_\epsilon\}$. Isso significa que devemos olhar para todas as produções da gramática tentando identicar aquelas cujo corpo deriva inteiramente para vazio. Caso isto ocorra, adicionamos em N_ϵ o não-terminal da cabeça da produção sob análise. Este processo é repetido até que N_ϵ não aumente mais de tamanho.

2. **Construir um conjunto de produções sem produções vazias**

   Neste passo, criamos uma outra gramática sem produções vazias. Esta gramática é $G_1 = (N, T, P_1, S)$, onde N são os símbolos não-terminais, T os terminais, P_1 as produções sem derivações para vazio e S o símbolo não-terminal inicial. N, T e S são idênticos aos da gramática original. Para obter P_1, devemos inicializá-lo com todas as produções de P (da gramática original) sem as produções que derivam *diretamente* para vazio. Portanto, P_1 é configurado inicialmente como $P_1 = \{A \rightarrow \alpha | \alpha \neq \epsilon\}$.

   Para criar da forma correta o conjunto de produções P_1, devemos repetir a seguinte regra: fazer $P_1 = P_1 \in \{A \rightarrow \alpha_1 \alpha_2\}$ para todo $A \rightarrow \alpha \in P_1$ e $X \in N_\epsilon$ tal que $\alpha = \alpha_1 X \alpha_2$ e $\alpha_1 \alpha_2 \neq \epsilon$. Isso significa que devemos olhar para cada uma das produções previamente inseridas (na etapa de inicialização) em P_1 e que contém um não-terminal que está em N_ϵ, e inserir uma outra produção cuja forma sentencial do corpo é idêntica a produção sendo considerada mas sem o não-terminal escolhido de N_ϵ. Esta regra iterativa realiza, dentro desse algoritmo, a geração de todas as combinações possíveis do corpo da produção que contém pelo menos um não-terminal que deriva para vazio. Como o processo é iterativo, a cada vez que uma nova produção é adicionada em P_1, devemos analisá-la posteriormente através desta mesma regra.

3. **Incluir a produção vazia se necessário**

   Este passo somente deve ser realizado se a palavra vazia fizer parte da linguagem gerada da gramática original. Neste caso, devemos adicionar a regra $S \rightarrow \epsilon$ ao conjunto P_1 do passo anterior, considerando que S é o símbolo inicial. Teremos então finalmente a gramática $G_2 = (N, T, P_2, S)$ onde $P_2 = P_1 \cup \{S \rightarrow \epsilon\}$ e N, T, e S são idênticos àqueles da gramática G.

Ilustramos o funcionamento deste algoritmo com a seguinte gramática:

$$
\begin{array}{lcl}
\text{S} & \rightarrow & \text{B}\ \mathbf{z}\ \text{B} \\
\text{B} & \rightarrow & \text{A A A} \\
\text{A} & \rightarrow & \mathbf{a} \\
\text{A} & \rightarrow & \epsilon
\end{array}
$$

No primeiro passo, inicializamos N_ϵ com o único não-terminal que deriva *diretamente* para vazio pela produção $A \rightarrow \epsilon$. Portanto, $N_\epsilon = \{A\}$. Depois, analisamos as outras produções onde o corpo pode derivar para vazio de forma *indireta*. Observando a produção $B \rightarrow AAA$, vemos que como A pode derivar para vazio, toda a forma sentencial AAA também pode se tornar vazia. Caso isto ocorra, vemos que B pode se tornar vazio. Por causa disso, adicionamos B também em N_ϵ, obtendo $N_\epsilon = \{A, B\}$. Olhando para a única produção com S, vemos que S nunca pode derivar para vazio, visto que na forma sentencial do corpo da sua produção temos o terminal **z** que nunca se torna vazio. A

produção $A \to \mathbf{a}$ não deriva para vazio de forma indireta pela mesma razão: **a** é um símbolo terminal. Terminamos portanto este passo com $N_\epsilon = \{A, B\}$ finalizando a análise de todas as produções.

No segundo passo deste exemplo, inicializamos P_1 com todas as produções de P menos aquelas que derivam de forma *direta* para vazio. Portanto, P_1 tem $S \to B\mathbf{z}B$, $B \to AAA$ e $A \to \mathbf{a}$ (todas menos a produção $A \to \epsilon$). Para terminarmos o processo, analisamos separadamente cada uma dessas três produções que fazem parte agora de P_1 tendo em vista os não-terminais que foram adicionados em N_ϵ no passo anterior. Analisando a produção $S \to B\mathbf{z}B$, sabemos que B deriva para vazio (pois se encontra em N_ϵ). Sendo assim, devemos incluir a combinação $S \to \mathbf{z}B$ em P_1 pois $\alpha_1 = \epsilon$ e $\alpha_2 = \mathbf{z}B$. Vejam que a forma sentencial $\mathbf{z}B$ é a concatenção de $\alpha_1\alpha_2$, como descrito na regra acima. Repetimos o processo considerando a mesma produção, mas com $\alpha_1 = B\mathbf{z}$ e $\alpha_2 = \epsilon$, rendendo a adição da nova produção $S \to B\mathbf{z}$. Como o processo é iterativo, estas duas novas produções adicionais a P_1 devem ser analisadas pelo mesmo algoritmo, pois contém em suas produções um não-terminal que está em N_ϵ. Considerando portanto a produção $S \to B\mathbf{z}$, vemos que $\alpha_1 = \epsilon$ e $\alpha_2 = \mathbf{z}$. Como devemos adicionar a concatenação $\alpha_1\alpha_2$ como corpo de uma produção cuja cabeça é S, adicionamos a nova produção $S \to \mathbf{z}$. Realizando o mesmo para a produção $S \to \mathbf{z}B$, terminamos com produções adicionais em P_1 a partir da produção $S \to B\mathbf{z}B$: $S \to \mathbf{z}B$, $S \to B\mathbf{z}$ e $S \to \mathbf{z}$. Estas quatro produções representam todas as combinações possíveis de $B\mathbf{z}B$, como descrito na parte introdutória desta seção. Para terminar, repetimos todo esse processo considerando a produção $B \to AAA$, pois A também está em N_ϵ. Uma vez isto terminado, teremos um P_1 com as seguintes produções:

S → B **z** B
S → B **z**
S → **z** B
S → **z**
B → A A A
B → A A
B → A
A → **a**

Como a palavra vazia não faz parte da linguagem gerada pela gramática original do exemplo acima, P_1 é final.

2 Remoção da recursão à esquerda

Ainda por ser escrito.

3 Fatoração gramatical

Ainda por ser escrito.