Linguagens formais e autômatos

Manipulação de Gramáticas Livres de Contexto

Gabriel V C Candido
gabriel.candido@ifpr.edu.br

Instituto Federal do Paraná - Pinhais

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Sumário

Sumário

Análise sintática

- Saber se uma palavra $w \in L(G)$: definir um caminho no grafo da gramática;
- Algoritmos de busca podem sofrer problemas: memória, recursão à esquerda;

Nosso objetivo é restringir as regras para facilitar o processo de análise sintática.

Sumário

Eliminação da recursividade no símbolo de partida

Podemos acrescentar um símbolo S', que será o novo símbolo de partida, e uma única regra $S' \rightarrow S$.

Sumário

Identificação de regras λ

Regras que geram λ permitem reduzir o tamanho das formas sentenciais (produções vazias).

É interessante que isso *não* aconteça, pois a palavra w que estamos procurando no grafo possui um tamanho fixo; e isso pode ser útil para limitar nossa busca.

Identificação de regras λ

1. Começar com o conjunto NULL de todas as regras que geram λ
2. Adicionar toda regra que produza formas sentenciais que estão em NULL*

Identificação de regras λ

Exemplo 1

G:

$$\begin{cases} S \rightarrow ACA \\ A \rightarrow aAa|B|C \\ B \rightarrow bB|b \\ C \rightarrow cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Identificação de regras λ

Exemplo 2

G:

$$\begin{cases} S \rightarrow aS|AB|AC \\ A \rightarrow aA|\lambda \\ B \rightarrow bB|bS \\ C \rightarrow cC|\lambda \end{cases}$$

$\text{NULL} = \{S, A, C\}$

G_1:

$$\begin{cases} S' \rightarrow S \\ S \rightarrow aS|AB|AC \\ A \rightarrow aA|\lambda \\ B \rightarrow bB|bS \\ C \rightarrow cC|\lambda \end{cases}$$

$\text{NULL} = \{S', S, A, C\}$

INSTITUTO FEDERAL
Paraná
Campus Pinhais

Sumário

Acrescentando regras na gramática

Dada uma GLC G:

Se $A \Rightarrow^* w$, então podemos adicionar a regra $A \rightarrow w$ à G.

Sumário

Eliminando regras λ

Nosso objetivo é manipular a gramática G de forma que ela:

- continue gerando a mesma linguagem;
- o símbolo de partida não é recursivo;
- não seja contratível; ou seja, só existe a produção de λ no símbolo inicial, caso $\lambda \in L(G)$

A ideia é adicionar regras que geram todas as possibilidades de formas sentenciais, removendo as produções λ.

Eliminando regras λ

Exemplo 1, de novo

G:

$$\begin{cases} S \to ACA \\ A \to aAa|B|C \\ B \to bB|b \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Agora:

1. $\lambda \in L(G)$?

Eliminando regras λ

Exemplo 1, de novo

G:
$$\begin{cases} S \to ACA \\ A \to aAa|B|C \\ B \to bB|b \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Agora:

1. $\lambda \in L(G)$? Sim, então teremos a regra $S' \to \lambda$
2. Analisar cada regra que gera λ:

Eliminando regras λ

Exemplo 1, de novo

G:

$$\begin{cases} S \to ACA \\ A \to aAa|B|C \\ B \to bB|b \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Agora:

1. $\lambda \in L(G)$? Sim, então teremos a regra $S' \to \lambda$
2. Analisar cada regra que gera λ:
   - $C \in$NULL: adiciona a regra $C \to c$

Eliminando regras λ

Exemplo 1, de novo

G:

$$\begin{cases} S \to ACA \\ A \to aAa|B|C \\ B \to bB|b \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Agora:

1. $\lambda \in L(G)$? Sim, então teremos a regra $S' \to \lambda$
2. Analisar cada regra que gera λ:
   - $C \in$NULL: adiciona a regra $C \to c$
   - $B \notin$NULL

Eliminando regras λ

Exemplo 1, de novo

G:

$$\begin{cases} S \to ACA \\ A \to aAa|B|C \\ B \to bB|b \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Agora:

1. $\lambda \in L(G)$? Sim, então teremos a regra $S' \to \lambda$
2. Analisar cada regra que gera λ:
   - $C \in$NULL: adiciona a regra $C \to c$
   - $B \notin$NULL
   - $A \in$NULL: adiciona a regra $A \to aa$

Eliminando regras λ

Exemplo 1, de novo

G:

$$\begin{cases} S \to ACA \\ A \to aAa|B|C \\ B \to bB|b \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Agora:

1. $\lambda \in L(G)$? Sim, então teremos a regra $S' \to \lambda$
2. Analisar cada regra que gera λ:
   - $C \in$NULL: adiciona a regra $C \to c$
   - $B \notin$NULL
   - $A \in$NULL: adiciona a regra $A \to aa$
   - $S \in$NULL: adiciona *todas* as combinações possíveis: $S \to AC|CA|AA|A|C$

Eliminando regras λ

Exemplo 1, de novo, continuação

G:

$$\begin{cases} S \to ACA \\ A \to aAa|B|C \\ B \to bB|b \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Agora:

3. Remover todas as regras que levam a λ que não são o novo símbolo inicial

Eliminando regras λ

Exemplo 1, nova gramática

$$
G': \begin{cases} S' \to S|\lambda \\ S \to ACA|AC|CA|AA|C|A \\ A \to aAa|aa|B|C \\ B \to bB|b \\ C \to cC \end{cases}
$$

Note que a gramática é maior, mas tente derivar *aba* e veja que você consegue fazer isso em menos passos!

Eliminando regras λ

Exemplo 2: $a^*b^*c^*$

G:
$$\begin{cases} S \to ABC \\ A \to aA|\lambda \\ B \to bB|\lambda \\ C \to cC|\lambda \end{cases}$$

Iteração	NULL	PREV
0	$\{A, B, C\}$	
1	$\{S, A, B, C\}$	$\{A, B, C\}$
3	$\{S, A, B, C\}$	$\{S, A, B, C\}$

Agora:

1. $\lambda \in L(G)$?
2. Analisar cada regra que gera λ
3. Remover regras que produzem λ e não são o símbolo partida

Eliminando regras λ

Exemplo 2: $a^*b^*c^*$

G:

$$\begin{cases} S \to ABC \\ A \to aA|\lambda \\ B \to bB|\lambda \\ C \to cC|\lambda \end{cases}$$

G':

$$\begin{cases} S' \to S|\lambda \\ S \to ABC|AB|AC|BC|A|B|C \\ A \to aA|a \\ B \to bB|b \\ C \to cC|c \end{cases}$$

Sumário

Eliminação de regras de cadeia

Note que nossa gramática ainda tem regras que mantém o mesmo tamanho da forma sentencial produzida: $S \rightarrow A$

G:
$$\begin{cases} A \rightarrow aA|a|B \\ B \rightarrow bB|b|c \end{cases}$$

G':
$$\begin{cases} A \rightarrow aA|a|bB|b|c \\ B \rightarrow bB|b|c \end{cases}$$

Eliminação de regras de cadeia

O algoritmo segue a **mesma** ideia da identificação de regras λ!

Eliminação de regras de cadeia

Exemplo 3

G:

$$\begin{cases} S \rightarrow ACA|CA|AA|AC|A|C|\lambda \\ A \rightarrow aAa|aa|B|C \\ B \rightarrow bB|b \\ C \rightarrow cC|c \end{cases}$$

Eliminação de regras de cadeia

Exemplo 3

G:

$$\begin{cases} S \to ACA|CA|AA|AC|A|C|\lambda \\ A \to aAa|aa|B|C \\ B \to bB|b \\ C \to cC|c \end{cases}$$

G':

$$\begin{cases} S \to ACA|CA|AA|AC|aAa|aa|bB|b|cC|c|\lambda \\ A \to aAa|aa|bB|b|cC|c \\ B \to bB|b \\ C \to cC|c \end{cases}$$

Sumário

Símbolos úteis

Seja G uma GLC. Um símbolo x é útil se:

- Derivar x gera, em algum momento, terminais: $uxv \Rightarrow^* w, w \in \Sigma^*$
- Ele é atingível a partir do símbolo de partida: $S \Rightarrow^* uxv$

Um símbolo que não é útil é inútil.

Exemplo 4

$L(G) = b^*$

G:

$$\begin{cases} S \rightarrow AC|BS|B \\ A \rightarrow aA|aF \\ B \rightarrow CF|b \\ C \rightarrow cC|D \\ D \rightarrow aD|BD|C \\ E \rightarrow aA|BSA \\ F \rightarrow bB|b \end{cases}$$

Identificando variáveis que geram terminais

Mesma ideia dos algoritmos anteriores!

- Começa com o óbvio: regras diretas
- Vai acrescentado as regras que chegam nas diretas

Identificando variáveis que geram terminais

Mesma ideia dos algoritmos anteriores!

- Começa com o óbvio: regras diretas
- Vai acrescentado as regras que chegam nas diretas

Podemos remover as regras que não geram terminais!

Exemplo 4

$L(G) = b^*$

G:

$$\begin{cases} S \to AC|BS|B \\ A \to aA|aF \\ B \to CF|b \\ C \to cC|D \\ D \to aD|BD|C \\ E \to aA|BSA \\ F \to bB|b \end{cases}$$

Iteração	TERM	
0	$\{B, F\}$	
1	$\{B, F, A, S\}$	$\{$
2	$\{B, F, A, S, E\}$	$\{B,$
3	$\{B, F, A, S, E\}$	$\{B, F$

Identificando variáveis atingíveis

Mesma ideia dos algoritmos anteriores!

- Começa com o óbvio: símbolo de partida
- Vai acrescentado as regras que são atingíveis

Identificando variáveis atingíveis

Mesma ideia dos algoritmos anteriores!

- Começa com o óbvio: símbolo de partida
- Vai acrescentado as regras que são atingíveis

Podemos remover as regras que não são atingíveis.

Exemplo 4

$L(G) = b^*$

G:

$$\begin{cases} S \to BS|B \\ A \to aA|aF \\ B \to b \\ E \to aA|BSA \\ F \to bB|b \end{cases}$$

Iteração	REACH	PREV
0	$\{S\}$	
1	$\{S, B\}$	$\{B\}$
2	$\{S, B\}$	$\{S, B\}$

Exemplo 4

$L(G) = b^*$

G:

$$\begin{cases} S \rightarrow BS|B \\ B \rightarrow b \end{cases}$$

Eliminação de símbolos inúteis

A ordem dos algoritmos importa!

1. Eliminar variáveis que não geram terminais
2. Eliminar símbolos que não são atingíveis

Sumário

Conclusão

Nossa gramática agora tem regras da forma:

- $S \rightarrow \lambda$
- $A \rightarrow a$
- $A \rightarrow w$, w pode ter variáveis, mas $|w \geq 2$!!

Isso já é suficiente para analisadores *bottom-up*.