Autômatos e computabilidade

Simplificação de CFGs

Pedro A D Rezende
UnB – IE – CIC

A & C: Simplificação de CFGs

Motivação

Evitar gramáticas ambíguas

- $\{(ab)^n a \mid n \in \mathbb{N}\}$
- $\{\alpha \in \{id, +, *, (,)\}^* \mid \alpha$ é expressão aritmética$\}$

podem ser geradas tanto por CFG ambígua como por CFG inambígua

A & C: Simplificação de CFGs

Motivação

Evitar gramáticas ambíguas

- $\{(ab)^n a \mid n \in \mathbb{N}\}$
- $\{\alpha \in \{id, +, *, (,)\}^* \mid \alpha$ é expressão aritmética$\}$

  podem ser geradas tanto por CFG ambígua como por CFG inambígua

Permitir modelos computacionais reconhecedores (e tradutores) eficientes.

A & C: Simplificação de CFGs

Motivação

Evitar gramáticas ambíguas

- $\{(ab)^n a \mid n \in \mathbb{N}\}$
- $\{\alpha \in \{id, +, *, (,)\}^* \mid \alpha$ é expressão aritmética$\}$

  podem ser geradas tanto por CFG ambígua como por CFG inambígua

Permitir modelos computacionais reconhecedores (e tradutores) eficientes.

Notação: $S \overset{L}{\Rightarrow}{}^* x$ denota derivação à esquerda (pode ser representada pela lista de produções)

A & C: Simplificação de CFGs

Definições: Linearidade

Dado $G = (V, T, P, S)$ CFG dizemos que G é

linear se toda produção de P for da forma

$$A \to uBv \quad \text{ou} \quad A \to u$$

onde $u, v, \in T^*$, $B \in V$

(toda produção tem no máximo uma variável).

A & C: Simplificação de CFGs

Definições: Linearidade

Dado $G = (V, T, P, S)$ CFG dizemos que G é

linear se toda produção de P for da forma

$$A \to uBv \quad \text{ou} \quad A \to u$$

onde $u, v, \in T^*$, $B \in V$

(toda produção tem no máximo uma variável).

G é dita linear à direita se toda produção de P for da forma

$$A \to uB \quad \text{ou} \quad A \to u$$

A & C: Simplificação de CFGs

Definições: Ambiguidade inerente

Dado L CFL dizemos que L é inerentemente ambígua se qualquer G que gere L é ambígua.

A & C: Simplificação de CFGs

Definições: Ambiguidade inerente

Dado L CFL dizemos que L é inerentemente ambígua se qualquer G que gere L é ambígua.

Exemplo de linguagem CFL inerentemente ambígua:

$$L = \{a^i b^j c^k \mid i, j, k \in \mathbb{N} \wedge i=j \vee j=k\}$$

($a^n b^n c^n$, $n>n_0$ tem duas derivações à esqerda)

A & C: Simplificação de CFGs

Definições: Ambiguidade inerente

Dado L CFL dizemos que L é inerentemente ambígua se qualquer G que gere L é ambígua.

Exemplo de linguagem CFL inerentemente ambígua:

$$L = \{a^i b^j c^k \mid i, j, k \in \mathbb{N} \wedge i=j \vee j=k\}$$

($a^n b^n c^n$, $n>n_0$ tem duas derivações à esqerda)

Exemplo de linguagem que não é CFL:

$$L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

A & C: Simplificação de CFGs

Definições: Símbolos úteis e inúteis

Dado $G=(V, T, P, S)$ CFG, $X \in (V \cup T)$ dizemos que

X é útil se $\exists$ derivação final em G da forma.

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w$$

Caso contrário, dizemos que X é inútil

A & C: Simplificação de CFGs

Definições: Símbolos úteis e inúteis

Dado $G=(V, T, P, S)$ CFG, $X \in (V \cup T)$ dizemos que

X é útil se $\exists$ derivação final em G da forma.

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w$$

Caso contrário, dizemos que X é inútil

OBS: Existe algoritmo para eliminar simbolos inúteis de uma CFG (HU L4.1-4.2, T4.2)

A & C: Simplificação de CFGs

Definições: Símbolos úteis e inúteis

Dado G=(V, T, P, S) CFG, $X \in (V \cup T)$ dizemos que

X é útil se $\exists$ derivação final em G da forma.

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w$$

Caso contrário, dizemos que X é inútil

OBS: Existe algoritmo para eliminar simbolos inúteis de uma CFG (HU L4.1-4.2, T4.2)

(idéia: a partir de produções terminais, seleciona-se iterativamente as variáveis que derivam formas sentenciais com símbolos úteis)

A & C: Simplificação de CFGs

Definições: Produções vazias

Dado G=(V, T, P, S) CFG, produções da forma

$A \rightarrow \lambda$ são chamadas produções vazias.

A & C: Simplificação de CFGs

Definições: Produções vazias

Dado G=(V, T, P, S) CFG, produções da forma

$A \rightarrow \lambda$ são chamadas produções vazias.

Fato: $\lambda \notin L \Rightarrow [\exists$ G' sem produções vazias $|$ L=L(G)]

A & C: Simplificação de CFGs

Definições: Produções vazias

Dado G=(V, T, P, S) CFG, produções da forma

$A \rightarrow \lambda$ são chamadas produções vazias.

Fato: $\lambda \notin L \Rightarrow$ [$\exists$ G' sem produções vazias | L=L(G)]

OBS: $\exists$ algoritmo para obter G' de G (HU T4.3)

(idéia: identifica-se iterativamente as variáveis anuláveis, i.e., B tal que $B \Rightarrow^* \lambda$. Depois subsitui-se cada produção com símbolos anuláveis à direita, por um conjunto de produções, cada uma sem algum(s) desses símbolos anuláveis)

A & C: Simplificação de CFGs

Definições: Forma normal de Chomski

Dado G=(V, T, P, S), $\lambda \notin L(G)$, existe G' tal que L(G)=L(G') onde as produções de G' tem a forma:

$$A \rightarrow BC \quad \text{ou} \quad A \rightarrow a$$

onde A, B, C $\in$ V; a $\in$ T

A & C: Simplificação de CFGs

Definições: Forma normal de Chomski

Dado G=(V, T, P, S), $\lambda \notin L(G)$, existe G' tal que L(G)=L(G') onde as produções de G' tem a forma:

$$A \rightarrow BC \quad \text{ou} \quad A \rightarrow a$$

onde A, B, C $\in$ V; a $\in$ T

OBS: $\exists$ algoritmo para obter G' de G (HU T4.5) (idéia: primeiro, introduz-se novas variáveis e produções da forma $C_a \rightarrow a$. Depois, subsitui-se a por C_a nas produções originais. Depois, da mesma forma para sufixos de variáveis originais.)

A & C: Simplificação de CFGs

Definições: Forma normal de Greibach

Dado G=(V, T, P, S), $\lambda \notin L(G)$, existe G' tal que L(G)=L(G') onde as produções de G' tem a forma:

$$A \rightarrow aX_1 \ldots X_m$$

onde $A, X_i \in V = \{A_1, \ldots, A_s\}$; $a \in T$; $m \geq 0$

OBS: $\exists$ algoritmo para obter G' de G (HU T4.6)

Idéia do algoritmo:

Passo 1- converte-se G à forma normal de Chomski.

Passo 2- aplica-se produções em lados direitos de produções de forma que as novas produções satisfaçam $A_i \rightarrow A_j\alpha \Rightarrow j > i$.

A & C: Simplificação de CFGs

Forma normal de Greibach: conversão

Dado G=(V, T, P, S), $\lambda \notin L(G)$, existe G' tal que L(G)=L(G') onde as produções de G' tem a forma:

$$A \rightarrow aX_1 \ldots X_m \; ; \quad X_i \in V = \{A_1, \ldots, A_s\}$$

Continuação da idéia do algoritmo:

Idéia do passo 2: Por indução, supondo $A_i \rightarrow A_j\alpha \Rightarrow j > i$ se i < k, para j=1, 2,... em $A_k \rightarrow A_j\alpha$ $(j < k)$ subsitui-se A_j pelas produções que a expandem, iterativamente. Em $A_k \rightarrow A_k\alpha$, usa-se HU L4.4

Passo 3- Para j=s, s-1,... em $A_i \rightarrow A_j\alpha$ $(j > i)$ subsitui-se A_j pelas produções que a expandem.

A & C: Simplificação de CFGs

Algoritmo conversor p/ forma normal de Greibach:

para k = 1 até m faça { /* A_1 , ... , A_m são as variáveis gramaticais originais de G */

para j = 1 até k-1 faça {

para cada produção da forma $A_k \rightarrow A_j\alpha$ faça {

para cada produção da forma $A_j \rightarrow \beta$ faça {

adicione a produção $A_k \rightarrow \beta\alpha$

}

remova a produção $A_k \rightarrow A_j\alpha$

}

}

para cada produção da forma $A_k \rightarrow A_k\alpha$ faça {

adicione as produções $B_k \rightarrow \alpha$ e $B_k \rightarrow \alpha B_k$

remova a produção $A_k \rightarrow A_k\alpha$

}

para cada produção da forma $A_k \rightarrow \beta$ onde β não omeça com A_k faça {

adicione a produção $A_k \rightarrow \beta B_k$

}

}

A & C: Simplificação de CFGs

Definições: Forma normal de Bakus-Naur (BNF)

Dado G=(V, T, P, S), denota-se G com a seguinte "convenção gráfica":

$$<A> ::= \alpha_1 | ... | \alpha_m$$

Representa o subconjunto de produções

$<A> \rightarrow \alpha_1$, $<A> \rightarrow \alpha_m$ onde $<\_> \in V$; $\alpha \in (V \cup T)^*$