Grammar Cheatsheet

TLP

2019

Contents

List of Tables

List of Figures

Build CFG for a given language

Reduce a CFG

Dada una gramática $G = (N, T, S, P)$:

- Un símbolo útil $\in N \cup T$ es aquel:
  - $X \in N \cup T$ accesible si: $S \Rightarrow^* \alpha X \beta$
  - $X \in N$ co-accesible si: $X \Rightarrow^* \omega, \omega \in T^*$
- El orden importa, primero calcular co-accesibles y luego accesibles.

Algoritmo para calcular símbolos co-accesibles

Símbolos co-accesibles: $S_{co} = \{A \in N \mid A \rightarrow \alpha, \alpha \in T^*\}$

$S_{co_i+1} = S_[co_i]\{A \in N \mid A \rightarrow \alpha \in P, \alpha \in (S_[co_i] \cup T)^*\}$

STOP WHEN: $S_{co_i} = S_{co_i+1}$

Algoritmo para calcular símbolos accesibles

Se construye un grafo:

- Los nodos son símbolos(dependencias)
- $X \rightarrow Y$ si $X \rightarrow \alpha Y \beta \in P$

X es accesible si $\exists$ un camino de S hasta X.

CFG is finite

1. Reduce the grammar.
2. Transform into CNF.
3. Look for loops in the dependency graph.

CFG is empty

1. Calculate co-accesible symbols.
2. If $S \in S_c \rightarrow L(G) \neq \emptyset$ else $L(G) = \emptyset$

A word belongs to L(G)

CYK

Brute force

Chomsky

La CNF es una gramatica del tipo:

1. $A \rightarrow BC$, donde $A, B,$ y C, son no-terminales o
2. $A \rightarrow a$ donde A es un no-terminal y a es una terminal
3. Cabe notar que un CNF no tiene simbolos inutiles (se debe reducir antes) ni tampoco tiene producciones ϵ

Los pasos para transformar una CFG a una CNF son:

a. Conseguir que todos los cuerpos de tamano 2 o mas consistan solo de no-terminales.
b. Romper los cuerpos de tamano 3 o superior en cuerpos pequenos para cumplir la condicion anterior.

ejemplo:

$$
\begin{array}{lcl}
E & \rightarrow & EPT \mid TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
T & \rightarrow & TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
F & \rightarrow & LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
I & \rightarrow & a \mid b \mid IA \mid IB \mid IZ \mid IO \\
A & \rightarrow & a \\
B & \rightarrow & b \\
Z & \rightarrow & 0 \\
O & \rightarrow & 1 \\
P & \rightarrow & + \\
M & \rightarrow & * \\
L & \rightarrow & (\\
R & \rightarrow &)
\end{array}
$$

$$
\begin{array}{lcl}
E & \rightarrow & EC_1 \mid TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
T & \rightarrow & TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
F & \rightarrow & LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
I & \rightarrow & a \mid b \mid IA \mid IB \mid IZ \mid IO \\
A & \rightarrow & a \\
B & \rightarrow & b \\
Z & \rightarrow & 0 \\
O & \rightarrow & 1 \\
P & \rightarrow & + \\
M & \rightarrow & * \\
L & \rightarrow & (\\
R & \rightarrow &) \\
C_1 & \rightarrow & PT \\
C_2 & \rightarrow & MF \\
C_3 & \rightarrow & ER
\end{array}
$$

Greibach

PDA

Deterministic PDA

$PDA = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if:

1. $\mid \delta(q, a, A) \mid \leq 1, \forall q \in Q, a \in \Sigma, A \in \Gamma$
2. $\delta(q, \lambda, A) \neq \emptyset, \delta(q, a, A) = \emptyset \forall A \in \Sigma$

LL(k) Grammars

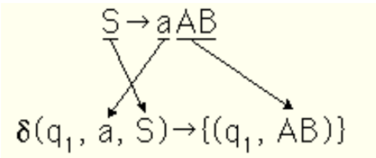
CFG to NPDA

For any context-free grammar in Greibach Normal Form we can build an equivalent nondeterministic pushdown automaton. This establishes that an npda is at least as powerful as a cfg. It will always produce a PDA with **three states**

1. Start state q_0 will serve as initialization.

$$(q_0, \lambda, z) \rightarrow \{(q_1, S_z)\}$$

2. State q_1 will contain the actual grammar computation.

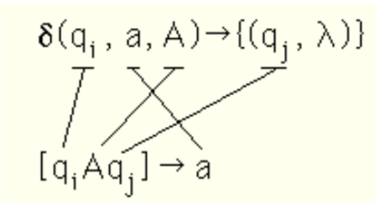


3. Transition q_1 to q_f to accept the string

$$delta(q_1, \lambda, z) \rightarrow \{(q_f, z)\}$$

NPDA to CFG

1. Las transiciones del tipo $\delta(q_i, a, A) = (q_j, \lambda)$ se transforman en reglas gramaticas del tipo:



2. Las transiciones del tipo $\delta(q_i, a, A) = (q_j, BC)$ resultan en una multitud de reglas. Una para cada par de estados q_x, q_y en el NPDA, muchas *unreachable* pero las utiles definen la gramatica: