
Grammar Cheatsheet

TLP

2019

Contents

| | |
|--|----------|
| Build CFG for a given language | 3 |
| Reduce a CFG | 3 |
| Algoritmo para calcular símbolos co-accesibles | 3 |
| Algoritmo para calcular símbolos accesibles | 3 |
| Algorithm for: | 3 |
| CFG is finite | 3 |
| CFG is finite | 4 |
| CFG is empty | 4 |
| A word belongs to $L(G)$ | 4 |
| CYK | 4 |
| Brute force | 4 |
| Normal Forms | 4 |
| Chomsky | 4 |
| Greibach | 4 |
| PDA | 4 |
| Deterministic PDA | 4 |
| LL(k) Grammars | 5 |
| CFG to NPDA | 5 |
| NPDA to CFG | 5 |
| Misc | 6 |
| Eliminate common prefixes | 6 |
| Ambiguity | 6 |

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.

Algorithm for:

CFG is finite

Given a CFG ...

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-accessible 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****Normal Forms****Chomsky****Greibach****PDA****Deterministic PDA**

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

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

LL(k) Grammars

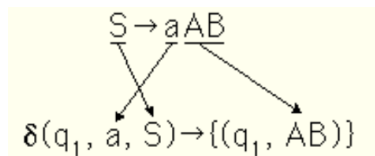
CFG to NPDA

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.

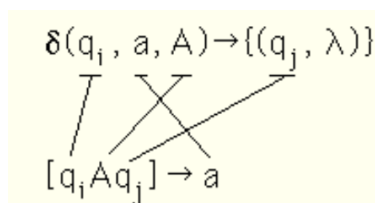


3. Transition q_1 to q_f to accept the string

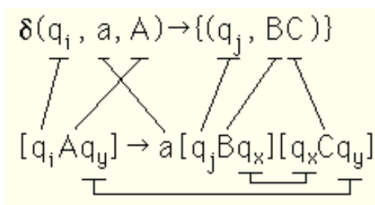
$$\delta(q_1, \lambda, z) \rightarrow \{(q_f, z)\}$$

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:



Misc

Eliminate common prefixes

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n$$

$$A \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_m$$

Transform into:

$$A \rightarrow A'$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

Ambiguity

A grammar $G = (N, T, S, P)$ is ambiguous if \exists a word that: - w can be derived with 2 different derivations to the right or left. - w have 2 different derivation trees.