# Grammar Cheatsheet

TLP

2019

# Contents

## List of Tables

## List of Figures

## Build CFG for a given language

## Reduce a CFG

## Algorithm for:

### CFG is finite

### CFG is empty

### A word belongs to L(G)

#### CYK

#### Brute force

## Normal Forms

### Chomsky

### Greibach

## LL(k) Grammars

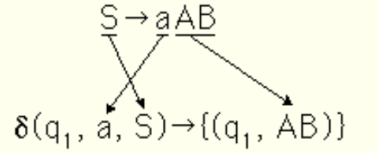
## CFG to NPDA

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.

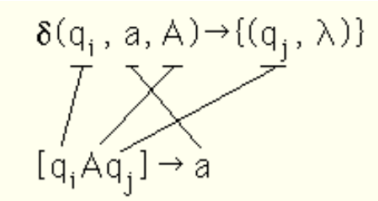



3. Transition $q_1$ to $q_f$ to accept the string

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

## 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: