Automata and Languages

Pranav Wadhwa

February 16, 2022

Contents

Chapter 1

Regular Languages

1.1 Terms

1.1.1 Alphabet

Alphabet is a non-empty finite set. Example: $\Sigma = \{0, 1\}, \Gamma = \{0, 1, x, y, z\}$

1.1.2 String

String is a finite sequence of symbols from an alphabet. There are **countably infinite** strings.

1.2 Decision Problems

Decision Problems is a function which takes an input string and decides whether to accept it or not.

input: string (finite length sequence from some finite alphabet Σ)

output: accept or reject, true or false, 1 or 0

A decision problem is specified by a function: strings $\rightarrow$ boolean

A decision problem is equivalently specified by a subset of strings $w \in S \iff f(w) = \text{accept}$

Cardinality Each string is finite

There are countably infinite strings

There are uncountably infinite decision problems (subset of strings).

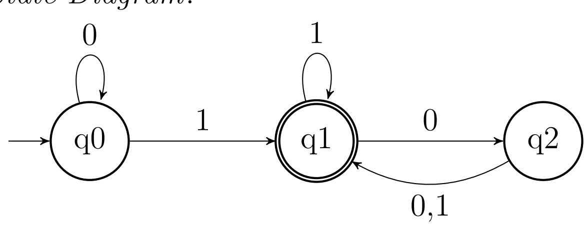
There are countably infinite finite automata, regular expressions, and regular language.

1.3 Finite Automaton

Definition: A finite automaton M is defined by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where
Q is finite set of State
Σ is the finite alphabet
$\delta : Q \times \Sigma \to Q$ is transition function
$q_0 \in Q$ is the start state
$F \subseteq Q$ is the set of accept state (this may be empty)
Example:
Consider M definied by: $Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
$q_0 is the start state$
$F = \{q_1\}$
Since Q and Σ finite, δ can be defined by a finite state transition table:

	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

State Diagram:



1.3.1 Definition of Computation

Given a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ and input string $w = \sigma_q \sigma_2 \dots \sigma_n$.

Let a configuration be a $c \in Q$.

Define the execution of M on w to eb the nuique sequence of configs $c_0 c_1, \dots, c_n$ s.t $c_0 = q_0, c_{i+1} = \delta(c_i, \sigma_{i+1})$ for $i = 0 \dots n - 1$.

If $c_n \in F$, then we say that M accepts w, else it rejects w.

Reconsider Example: Consider the execution of the previous finite automaton M on a few examples

1111: $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1$ accept

1110: $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_2$ reject This above program accepts even number of 0s (or no 0) after last 1.

1.4 Languages

A *language* is a subset of strings

Examples: $\emptyset$

all strings over $\Sigma = \{0, 1\}$

strings with more 1s than 0s etc

A machine is said to *accept* a string and *recognize* a language.

Definition M *recognizes* the language A iff $A = \{w : M \text{ accepts } w\}$

Definition A langauge is called *regular* iff some finite automaton M recognizes it.

Note

- For any regular langauge there are infinitely many finite automate that recogize it.
- For any ffinite automaton M, there is only one language it.

1.5 Regular operations on Languages

Let A and B be languages. The regular operations on languages are:

complement $\bar{A} = \{w : w \notin A\}$

intersection $A \cap B = \{w : w \in A \text{ and } w \in B\}$

union $A \cup B = \{w : w \in A \text{ or } w \in B\}$

concatenation $A \circ B = \{wx : w \in A \text{ and } x \in B\}$

star $A^* = \{w_1 w_2 \dots w_k : k \geq 0 \text{ and each } w_i \in A\}$

Theorem 1 *The class of regular languages is closed under complement*

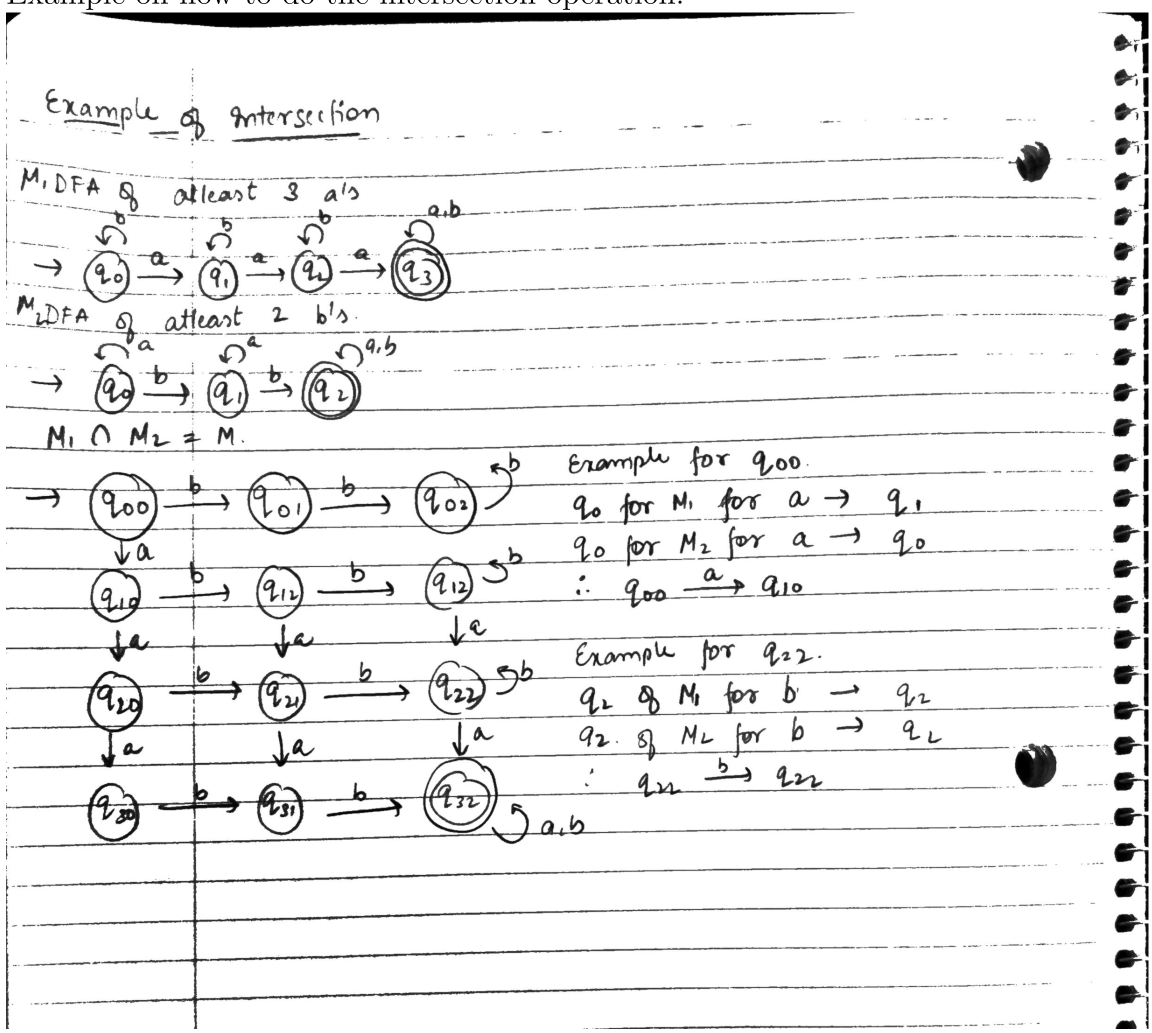
Intuition: When you have a finite automata M which recognize a langauge L then $\bar{L}$ is also regular and we can get a finite automata $\bar{M}$ which same as M but has the accept and reject states flipped

Theorem 2 *The class of regular langauge is closed uner intersection and union.*

Intuition/Hint: Let 2 regular language be L_1 and L_2 which have state machines M_1 and M_2. Let $M = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = Q_1 \times Q_2$
- $q_0 = (q_{01}, q_{02})$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

Example on how to do the intersection operation.



1.6 Nondeterministic finite automata

Definition A *nondeterministic finite automaton* (NFA) is defined by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:

Q is the finite set of states

Σ is the finite alphabet

$\delta : Q \times \Sigma_\lambda \to \mathcal{P}(Q)$ is the transition relation (i.e $\delta(q, \sigma) \subseteq Q$)

$q_0 \in Q$ is start state

$F \subseteq Q$ is set of accept states

where we let λ denote the null character, and let $\Sigma_\lambda = \Sigma \cup \{\lambda\}$, which will allow spontaneous state transitions that do not process the next input symbol.

Note Difference from DFA is the NFA allows multiple next possible state for one input.

Example Consider M defined by

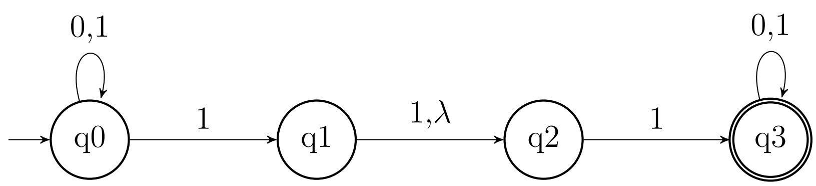
$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$q_0 = q_0$

$F = \{q_3\}$

	0	1	λ
q_0	$\{q_0\}$	$\{q_0, q_1\}$	$\emptyset$
q_1	$\{q_2\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\{q_3\}$	$\emptyset$
q_3	$\{q_3\}$	$\{q_3\}$	$\emptyset$



In this example, M accepts strings containing 11 or 101 as a substring.

1.6.1 Definition of nondeterministic computation

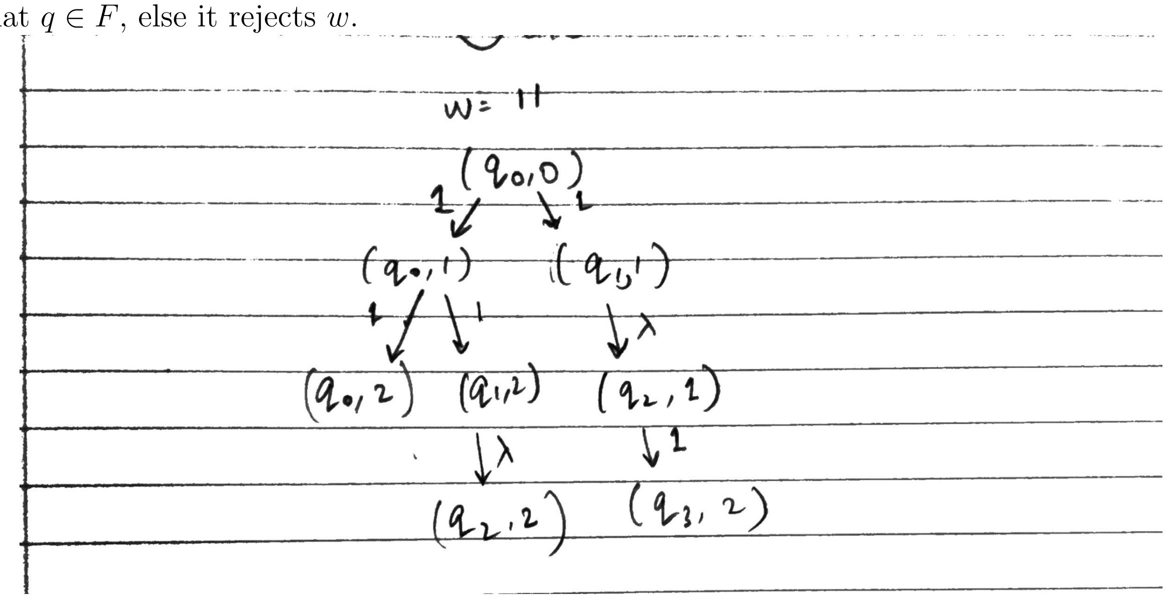
Instead of a sequence of configurations, we consider a tree of possible configurations.

Given an NFA $M = (Q, \Sigma, \delta, q_0, F)$ and input string $w = \delta_1\delta_2 \dots \delta_n$ for $\delta_i \in \Sigma$.

Define a *configuration* be $c = (q, i)$ s.t $q \in Q$ and i is position of last input symbol read.

Define the *execution tree* of M on w to be the directed tree rooted at $c = (q_0, 0)$ s.t there is an edge $c = (q, i)$ to $\bar{c} = (\bar{q}, \bar{i})$

We sau tht M accepts w if the execution tree contains a configuration (q, n) such that $q \in F$, else it rejects w.



Example of execution tree on input $w = 11$ for machine M defined above

1.7 NFA and DFA

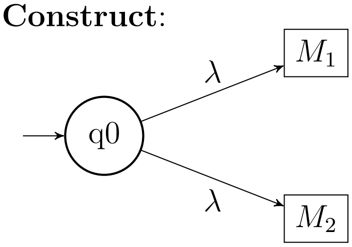
Theorem 3 *Every NFA M has an equivalent DFA $\bar{M}$.*

Corollary A language is regular $\leftrightarrow$ some NFA recogizes it ($\leftrightarrow$ some DFA recognizes it)

1.8 Regular Operations on Language

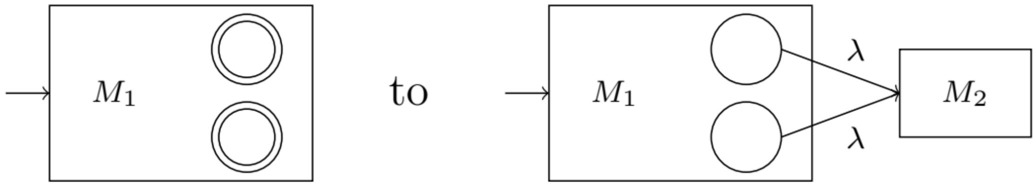
Theorem 4 *The class of regular langauges is closed under union.*

Construct:



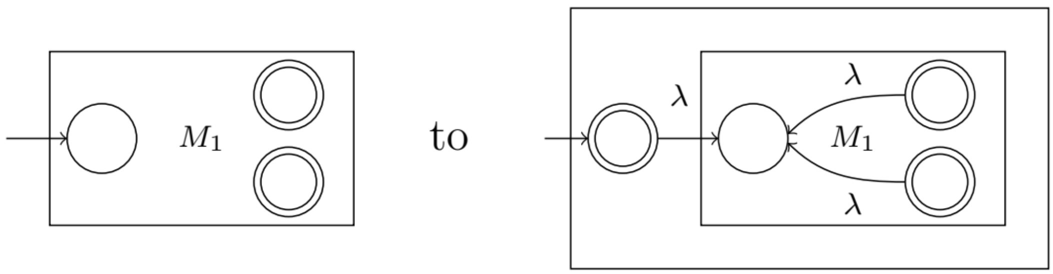
Theorem 5 *The class of regular languages is closed under concatenation*

Context



Theorem 6 *This class of regular languages is closed under star.*

Context



1.9 Regular Expression

A meta language for specifying decision problems over strings i.e a meta language for specifing a language

Definition R is a regular expression if
$R = a$ for some a in Σ
$R = \epsilon$
$R = \emptyset$
$R = R_1 \cup R_2$
$R = R_1 R_2$
$R = R_1^*$
$R = (R_1)$
Additionaly

- Precedence: $*, \circ, \cup$

- For a regular expression R, $L(R)$ denotes it language

Important Examples
$\Sigma = \{0, 1\}$

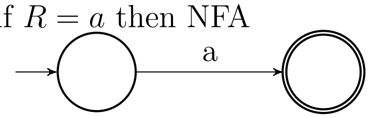
- $L(0 \cup 1) = \{0, 1\}$
- $L((0 \cup 1)^*) =$ all strings over $\{0, 1\}$
- $L((\Sigma\Sigma)^*) =$ strings of even length

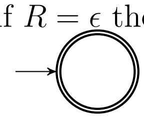
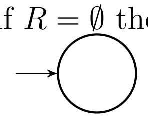
Identities

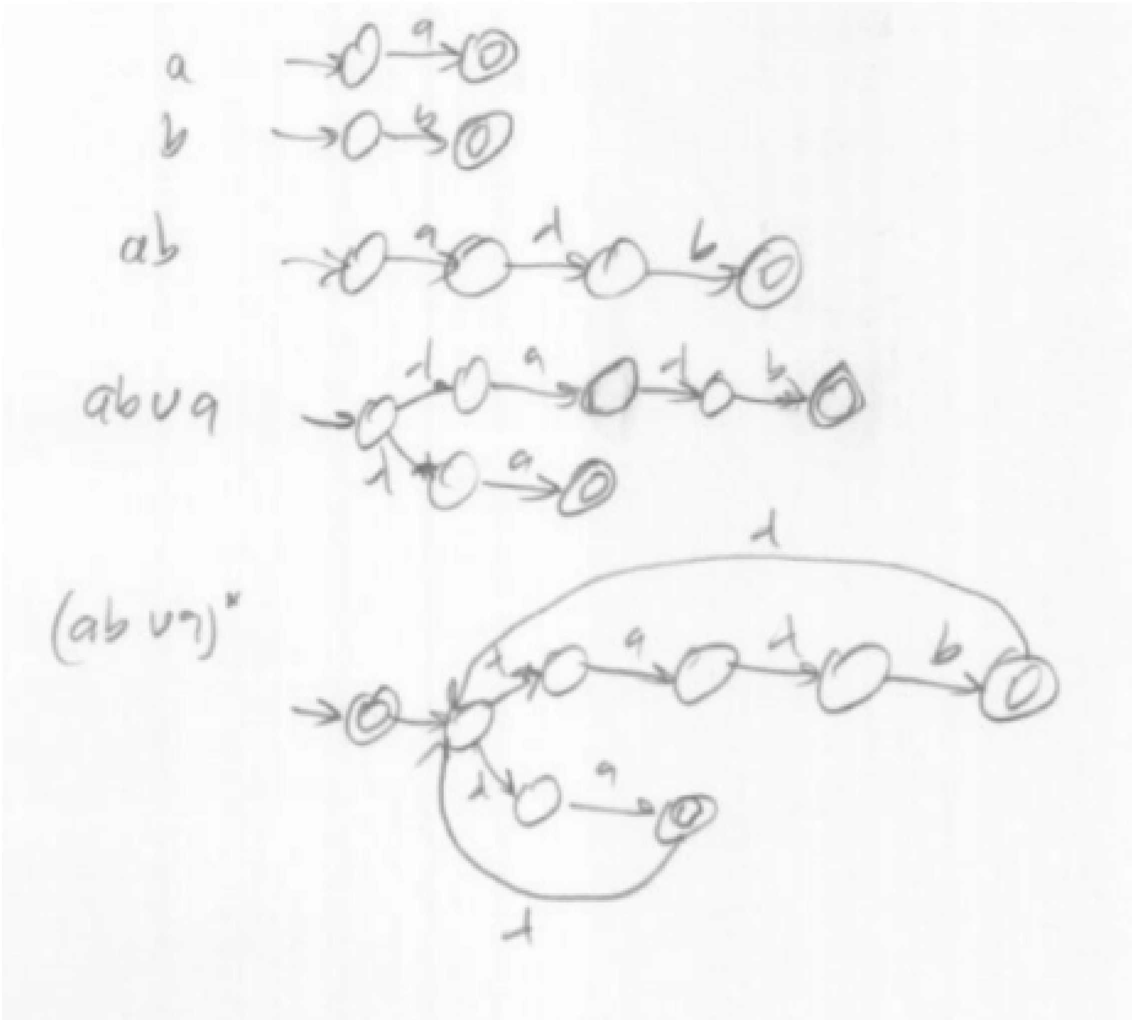
- $R \cup \emptyset = R$
- $R\epsilon = R$
- $R \cup \epsilon \neq R$
- $R\emptyset \neq R$

Theorem 7 *A language L is regular $\leftrightarrow$ exists a regular expression R such that* $L(R) = R$

- *Any decision problem that can be specified by an RE can be solved by an FA*
- *Any decision problem that can be implemented by an FA can be specified by an RE*

Regular Expression to NFA
if $R = a$ then NFA


if $R = \epsilon$ then NFA

if $R = \emptyset$ then NFA

Note: Rest operation already defined above
Example:

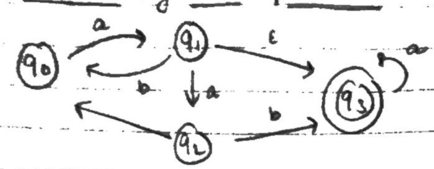


FA to Regular Expression
Steps:

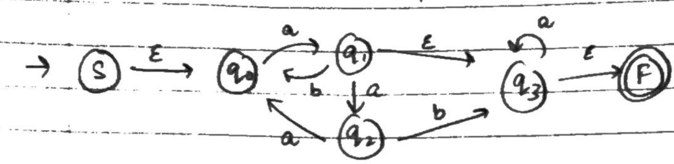
1. Start new start and accept states
2. Rip nodes out untill only the start and accept states are left.
3. To rip a node out, write down the "in" and "out" for the rip node

Example

NFA to Regular Expression



① Add new start & accept states



rip q_3

in → out

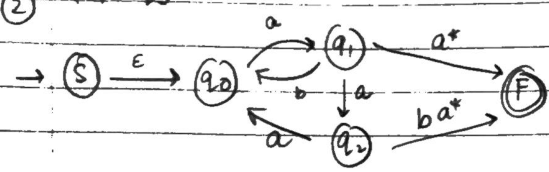
$q_1 \longrightarrow F$

q_2

$q_1 \xrightarrow{a^*} F$

$q_2 \xrightarrow{ba^*} F$

② Rip q_3



rip q_1

in → out

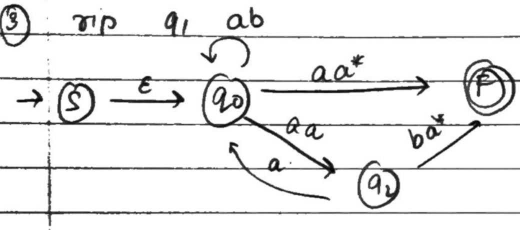
q_0 → F, q_2, q_0

$q_0 \xrightarrow{aa^*} F$

$q_0 \xrightarrow{aa} q_2$

$q_0 \xrightarrow{ab} q_0$

③ rip q_1



rip q_2

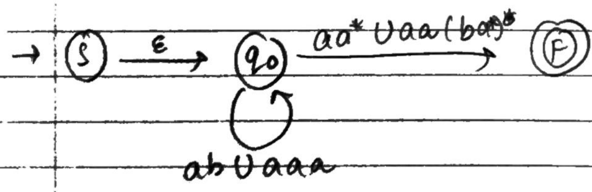
in → out

q_0 → F, q_0

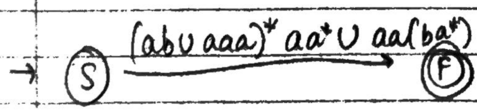
$q_0 \xrightarrow{aa^* \cup aa(ba^*)} F$

$q_0 \xrightarrow{ab \cup aaa} q_0$

④ rip q_2



⑤ rip q_0



1.10 Limits of Finite State Computation

Pumping Lemma for Regular Languages

if L is regular, then there must be some length p s.t any sufficiently long string $w \in L$, $|w| \geq p$ can be split as $w = xyz$ where:

$|y| > 0$ (y can't be ϵ)

$|xy| \leq p$

$xy^iz \in L$ for all $i \geq 0$ (i.e $L(xy^*z) \subseteq L$)

For sufficiently long strings, there is always a substring y that is arbitrarily repeatable.

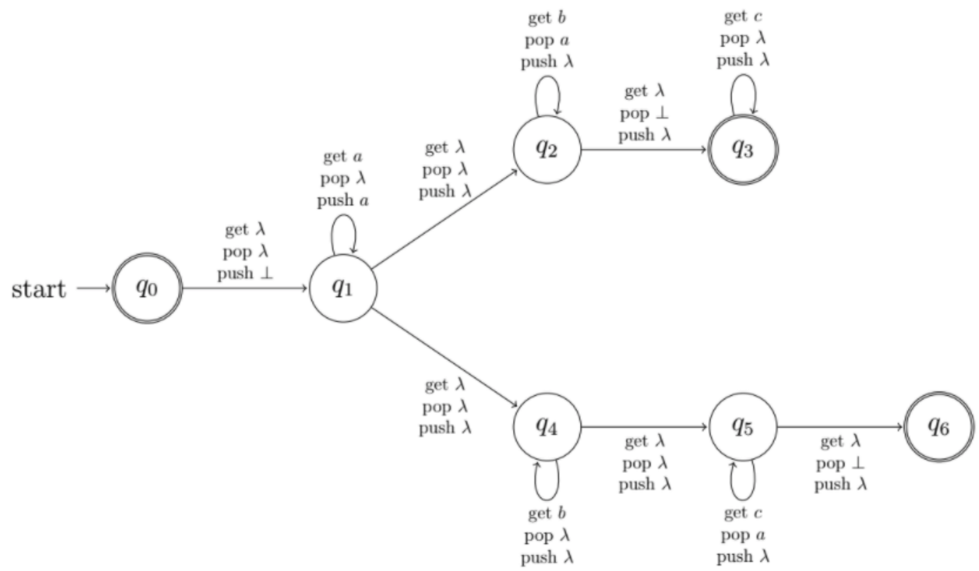
Chapter 2

Context-Free Languages

2.1 Pushdown Automata

Definition $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where
Q finite set of states
Σ finite input alphabet
Γ finite stack alphabet
$\delta : Q \times \Sigma_\lambda \times \Gamma_\lambda \to \{Q, \Gamma_\lambda\} \cup \emptyset$
q_0 start state
$F \subseteq Q$ set of accepting states
Example

Example A PDA that recognizes $\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k : i, j, k \geq 0,\ i = j \text{ or } i = k\}$



2.2 Context-Free Grammar

Definition A *context free grammar* is defined as $G = (V, \Sigma, R, S)$ where
V is the finite set of Variables ("non terminals")
Σ is the finite alphabet ("terminals")
R is the set of finite rules
S is the start variable