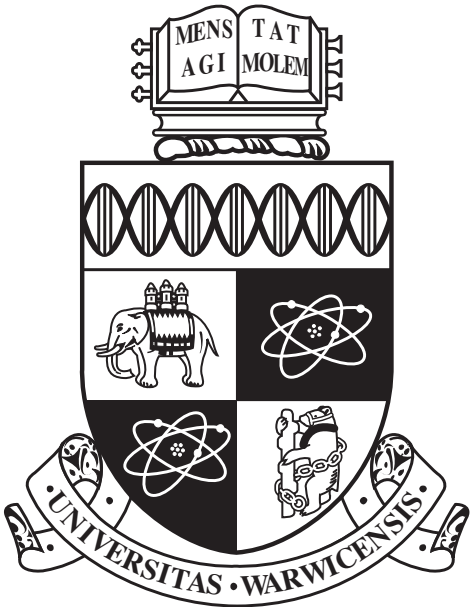
University of Warwick
Department of Computer Science

CS255

Formal Languages



Cem Yilmaz
August 13, 2023

Contents

1 Languages **2**
- 1.1 Alphabet . . . 2
- 1.2 Deterministic Final State Automata . . . 2
- 1.3 Languages . . . 4
- 1.4 Operations on Regular Languages . . . 4
- 1.5 Non-Deterministic Finite Automata (NFA) . . . 6

2 Regular Expressions **8**
- 2.1 Definition . . . 8
- 2.2 Conversion to NFA . . . 9

3 Proving Regularity **13**
- 3.1 Myhill-Nerode . . . 13
- 3.2 Pumping Lemma . . . 14

4 Grammars **16**
- 4.1 Definition . . . 16
- 4.2 Grammars and DFAs . . . 18

5 Push Down Automaton **18**
- 5.1 CFG to PDA . . . 19
- 5.2 PDA to CFG . . . 20
- 5.3 String in Grammar . . . 21
- 5.4 Pumping Lemma for CFL . . . 22

6 Turing Machines **23**
- 6.1 Definition . . . 23
- 6.2 Decidability . . . 23

7 Revision **25**

1 Languages

1.1 Alphabet

Definition 1.1. Alphabet

Alphabet is a non-empty finite set of symbols. For example,

$$\Sigma_1 = \{a, b, c\} \qquad \Sigma_2 = \{5, 8, 10\} \tag{1}$$

Are alphabets which contain those symbols.

Definition 1.2. Language

Language is a potentially infinite set of finite strings over an alphabet. Using our previous alphabets, for example, we could obtain

$$L_1 = \{ab, abc, aaab, ccc, ba\} \qquad L_2 = \{a, aa, aaa, aaaa, \ldots\} \tag{2}$$

We could further define

$$\Sigma^* = \{\text{ALL finite strings (also called words) over the alphabet } \Sigma\} \tag{3}$$

1.2 Deterministic Final State Automata

Definition 1.3. Deterministic Finate State Automaton

A machine M defined by the tuple

$$M = (Q, \Sigma, q_0, F, \delta) \tag{4}$$

is called the deterministic finite state automaton or a deterministic finite state machine. The Q refers to states, Σ to the alphabet, q_0 to the initial/starting state, F to the final state and δ as the state transition.

To represent a deterministic finite state automaton, consider the following diagram:

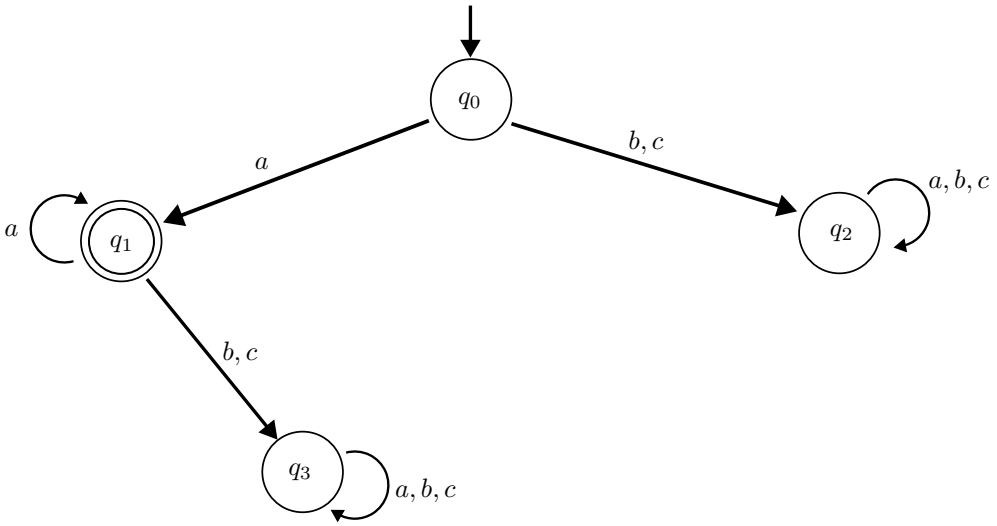


Figure 1: Exemplar deterministic finite state automata state diagram

Which represents the following table of values:

Table 1: State Transition Table

δ	a	b	c
q_0	q_1	q_2	q_2
q_1	q_1	q_3	q_3
q_2	q_2	q_2	q_2
q_3	q_3	q_3	q_3

In other words, if the input string finishes at q_1, we accept the input. If it finishes in any other node otherwise, reject.

Definition 1.4. The Empty Word

The length of $|\varepsilon| = 0$. The Language L_1 is the empty language, $L_2 = \{\varepsilon\}$ is a non-empty language. Note that Σ^* always contains ε. The role that of the empty string is to be a monoid in our system.

Definition 1.5. Monoid

Comprise of a set, an associative binary operation on the set with an identity element.

$$(\mathbb{N}_0, +, 0) \text{ is a monoid. Here, } + \text{ denotes addition.} \tag{5}$$
$$(\mathbb{N}, \times, 1) \text{ is a monoid. Here, } \times \text{ is multiplication.} \tag{6}$$
$$(\Sigma^*, \circ, \varepsilon) \text{ is a monoid. Here, } \circ \text{ denotes string concatenation.} \tag{7}$$

Definition 1.6. Transition Function

The transition function is denoted as

$$\delta(q_i, \text{string}) = q_j \tag{8}$$

k In other words, we take a state q_i, a string input, and after running the string, we get output state q_j. Note that the string can be a single letter or a bigger string. In case of a non-letter string, sometimes the δ is denoted as $\hat{\delta}$ instead. Formally,

$$\hat{\delta} : Q \times \Sigma^* \to Q \tag{9}$$

such that

$$\forall q \in Q, \hat{\delta}(q, \varepsilon) = q \tag{10}$$
$$\forall q \in Q \wedge s \in \Sigma^* \text{ s.t. } s = wa \text{ for some } w \in \Sigma^* \wedge a \in \Sigma, \hat{\delta}(q, s) = \delta(\hat{\delta}(q, w), a) \tag{11}$$

Definition 1.7. Language accepted by DFA

Consider a DFA $M = (Q, \Sigma, q_0, F, \delta)$. The language accepted or recognised by M is denoted by $L(M)$ and is defined as

$$L(M) = \{s \in \Sigma^* | \hat{\delta}(q_0, s) \in F\} \tag{12}$$

Definition 1.8. Run of a DFA

Consider a DFA $m = (Q, \Sigma, q_0, F, \delta)$. Consider a string $s = s_1 s_2 \dots s_n$, where $s_i \in \Sigma$ for each $i \in [n]$. The run of M on the empty word ε is just the state q_0. The run of M on the word s is a sequence of states $r_0, r_1, \dots, r_n$, where

$$r_0 = q_0 \tag{13}$$
$$\forall i \in [n], r_i = \delta(r_{i-1}, s_i) \tag{14}$$

1.3 Languages

Definition 1.9. Regular Language

A language L is called regular if it is accepted by some deterministic finite state automata (DFA)

Definition 1.10. NFA

Formally, the extended transition $\hat{\delta}$ for an NFA $(Q, \Sigma, q_0, F, \delta)$ is a function $\hat{\delta} : Q \times \Sigma^* \to \mathbb{P}(Q)$ and is defined as follows:

$$\forall q \in Q, \hat{\delta}(q, \varepsilon) = ECLOSE(q) \tag{15}$$
$$\forall q \in Q \wedge s \in \Sigma^* : s = wa \text{ for some } w \in \Sigma^* \wedge a \in \Sigma, \hat{\delta}(q, s) = ECLOSE(\cup_{q' \in \hat{\delta}(q,w)} \delta(q', a)) \tag{16}$$

It is useful to first compute the ε closure of an input and then consider the input string to see where it possible leads, repeating the process.

Corollary. *Language of NFA*

Consider an NFA $M = (Q, \Sigma, q_0, F, \delta)$. The run of M on the word s is a sequence of states $r_0, r_1, \dots, r_n$ such that

$$r_0 = q_0 \tag{17}$$
$$\exists s_1, s_2, \dots, s_n \in \Sigma \cup \{\varepsilon\} \text{ such that } s = s_1 s_2 \dots s_n \wedge \forall i \in [n], r_i \in \delta(r_{i-1}) \in \delta(r_{i-1}, s_i) \tag{18}$$

1.4 Operations on Regular Languages

Definition 1.11. Complement Language

Suppose L is regular. Is $\overline{L} = \Sigma^* - L$ also regular? Yes. If

$$M = (Q, \Sigma, q_0, F, \delta) \tag{19}$$

is a DFA for L then

$$M' = (Q, \Sigma, q_0, Q - F, \delta) \tag{20}$$

is a DFA for $\overline{L}$

Definition 1.12. Intersection

If L_1 and L_2 are regular, then

$$L_1 \cap L_2 \tag{21}$$

Is also a regular language. Consider

$$M_1 = (Q_1, \Sigma, q_1, F_1, \delta_1) \tag{22}$$
$$M_2 = (Q_2, \Sigma, q_2, F_2, \delta_2) \tag{23}$$

We define M_3 where

$$M_3 = (Q, \Sigma, q, F, \delta) \tag{24}$$
$$Q = Q_1 \times Q_2 \tag{25}$$
$$q = (q_1, q_2) \tag{26}$$
$$F = F_1 \times F_2 \tag{27}$$

For every $a \in \Sigma$, $x \in Q_1$, $y \in Q_2$ we have

$$\delta((x, y), a) = (\delta_1(x, a), \delta_2(y, a)) \tag{28}$$

Definition 1.13. Union

If L_1 and L_2 are regular, then

$$L_1 \cup L_2 \tag{29}$$

Is also a regular language. Consider

$$M_1 = (Q_1, \Sigma, q_1, F_1, \delta_1) \tag{30}$$
$$M_2 = (Q_2, \Sigma, q_2, F_2, \delta_2) \tag{31}$$

We define M_3 where

$$M_3 = (Q, \Sigma, q, F, \delta) \tag{32}$$
$$Q = Q_1 \times Q_2 \tag{33}$$
$$q = (q_1, q_2) \tag{34}$$
$$F = (F_1 \times Q_2) \times (Q_1 \times F_2) \tag{35}$$

For every $a \in \Sigma$, $x \in Q_1$, $y \in Q_2$ we have

$$\delta((x, y), a) = (\delta_1(x, a), \delta_2(y, a)) \tag{36}$$

Definition 1.14. Subtraction

We know that

$$L_1 - L_2 = L_1 \cap \overline{L}_2 \tag{37}$$

Therefore it is closed from our previous proofs.

Definition 1.15. Reverse

Suppose L is regular. Is L^{Rev} also regular?

$$L^{Rev} = \{w | w \text{ is the reverse of some string in } L\} \tag{38}$$

1.5 Non-Deterministic Finite Automata (NFA)

Definition 1.16. NFA

An NFA can have NONE or MULTIPLE transitions out of a state on reading the same symbol. The definition of an NFA is similar to that of a DFA, except the transition function is re-defined as follows:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q \tag{39}$$

Definition 1.17. Extended Transition Function

The extended transition function is similar to that of a DFA, except it also considers ε transitions. Formally,

$$\forall q \in Q, \hat{\delta}(q, \varepsilon) = ECLOSE(q) \tag{40}$$

$$\forall q \in Q \wedge s \in \Sigma^* : s = wa, w \in \Sigma^* \wedge a \in \Sigma, \hat{\delta}(q, s) = ECLOSE\left(\bigcup_{q' \in \hat{\delta}(q,w)} \delta(q', a)\right) \tag{41}$$

Definition 1.18. E-closure

The function $ECLOSE(q)$ denotes all states that can be reached from q by following ε-transitions alone.

Definition 1.19. Language

Consider an NFA $M = (Q, \Sigma, q_0, F, \delta)$. The language accepted or recognised by M denoted by $L(M)$ and is defined as

$$L(M) = \{s \in \Sigma^* | \hat{\delta}(q_0, s) \cap F \neq \varnothing\} \tag{42}$$

Definition 1.20. Run of M

A run of M on the word s is a sequence of states $r_0, r_1, \ldots, r_n$ such that

$$r_0 = q_0 \tag{43}$$

$$\exists s_1, s_2, \ldots s_n \in \Sigma \cup \{\varepsilon\} \text{ such that } s = s_1 s_2 \ldots s_n \wedge \forall i \in [n], r_i \in \delta(r_{i-1}, s_i) \tag{44}$$

Definition 1.21. Subset Construction

Let $N = (Q, \Sigma, q_0, F, \delta)$ be the NFA we want determinise. Denote using $M = (Q_1, \Sigma, q_1, F_1, \delta_1)$ the target DFA. We have

$$Q_1 = 2^Q \tag{45}$$

$$q_1 = ECLOSE(q_0) \tag{46}$$

$$F_1 = \{X \subseteq Q | X \cap F \neq \varnothing\} \tag{47}$$

$$\delta_1(X, a) = \bigcup_{x \in X} ECLOSE(\delta(x, a)) = \{z | \text{ for some } x \in X, z \in ECLOSE(\delta(x, a))\} \tag{48}$$

Example 1.1. Subset Construction

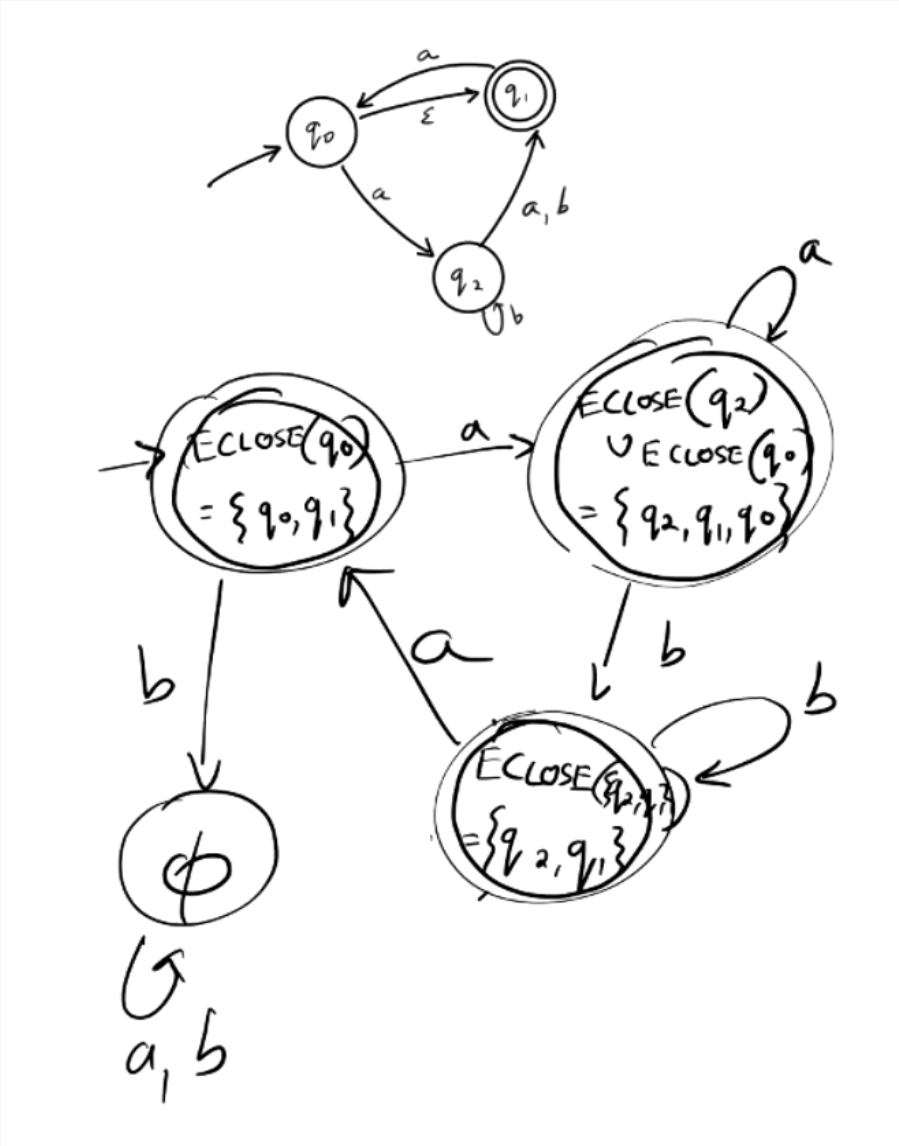


Figure 2: Subset Construction Example

2 Regular Expressions

2.1 Definition

Definition 2.1. Regular Expression

The language generated by a regular expression R is $L(R)$ and is defined as

$$L(R) = \{a\} \quad \Leftarrow R = a \text{ for some } a \in \Sigma \tag{49}$$
$$L(R) = \{\varepsilon\} \quad \Leftarrow R = \varepsilon \tag{50}$$
$$L(R) = \varnothing \quad \Leftarrow R = \varnothing \tag{51}$$
$$L(R) = L(R_1) \cup L(R_2) \quad \Leftarrow R = R_1 + R_2 \tag{52}$$
$$L(R) = L(R_1).L(R_2) \quad \Leftarrow R = R_1.R_2 \tag{53}$$
$$L(R) = (L(R_1))^* \quad \Leftarrow R = R_1^* \tag{54}$$

Definition 2.2. Star

For any $S \subseteq \Sigma^*$, we have

$$S^* = \{\varepsilon\} \cup S \cup S^2 \cup \dots \tag{55}$$

Example 2.1. Regular Expressions

Consider $\Sigma = \{a, b\}$ and let $R = (a + b)^*$. What is $L(R)$?

$$L(R) = \Sigma^* \tag{56}$$

Let $R = (a + b)^*(a + bb)$. What is $L(R)$?

$$L(R) = \Sigma^*.\{a, bb\} \tag{57}$$

Theorem 2.1. *Regularity*

A language is accepted by an NFA/DFA if and only if it is generated by a regular expression. Note that this also implies that NFA are regular.

2.2 Conversion to NFA

Definition 2.3. Union / Plus

For NFA N_1 expressing R_1 and NFA N_2 expressing R_2, their plus is

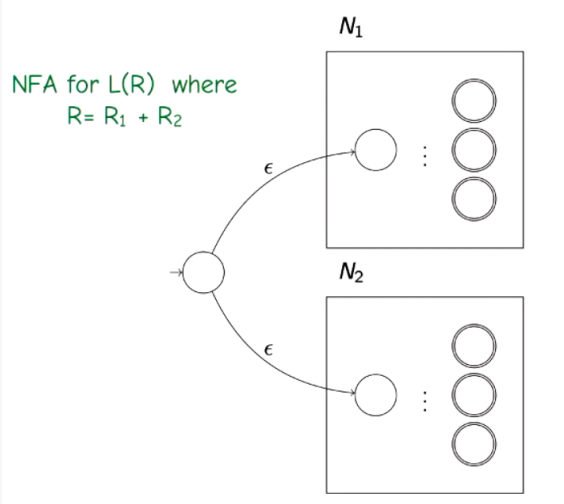


Figure 3: Plus as an NFA

Definition 2.4. Concat

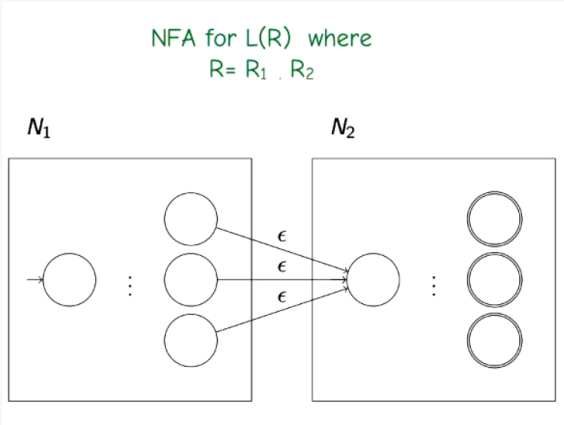


Figure 4: Concat

Definition 2.5. Star

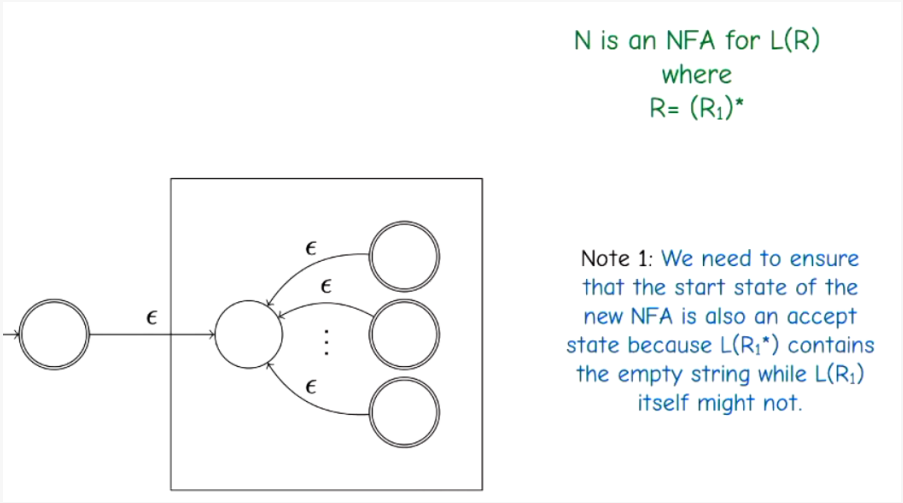


Figure 5: Star as an NFA

Example 2.2. Regular Expression to NFA

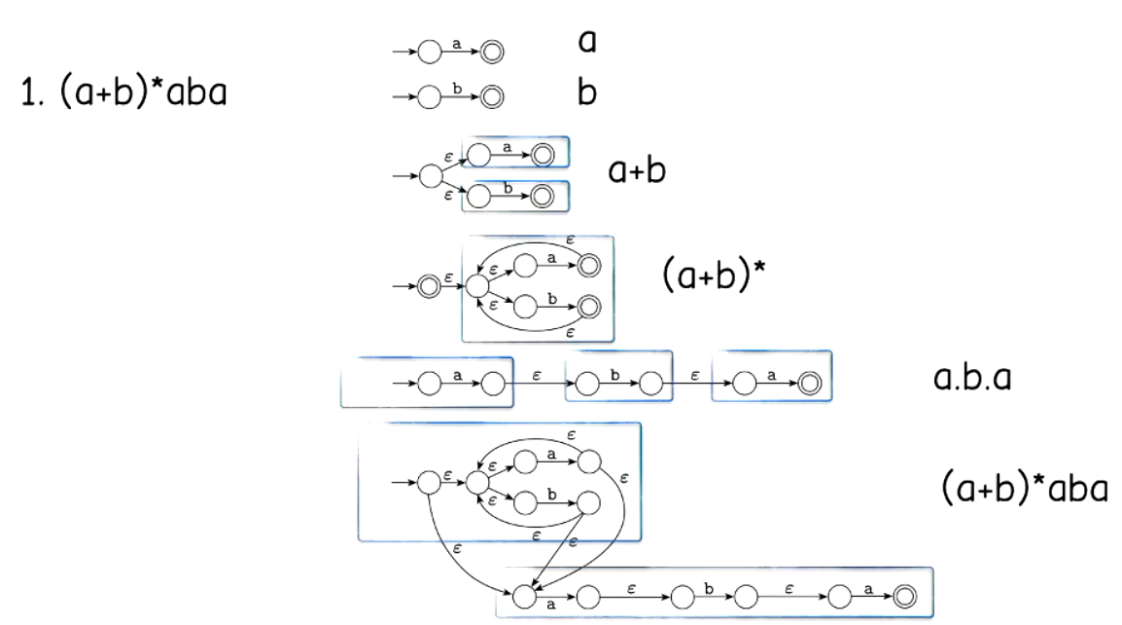


Figure 6: Regular Expression to NFA

Example 2.3. NFA To Regular Expression

Consider the NFA

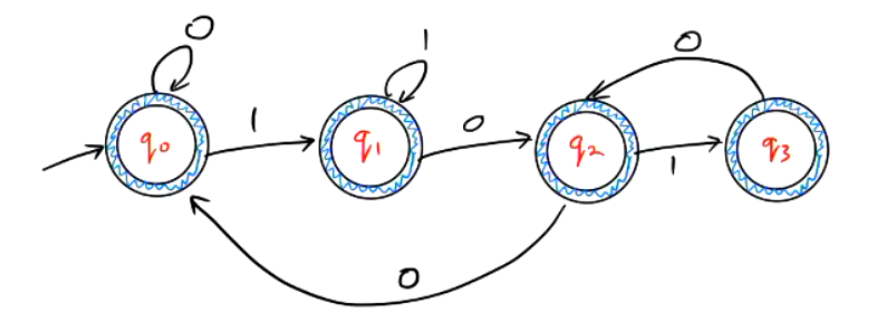


Figure 7: NFA

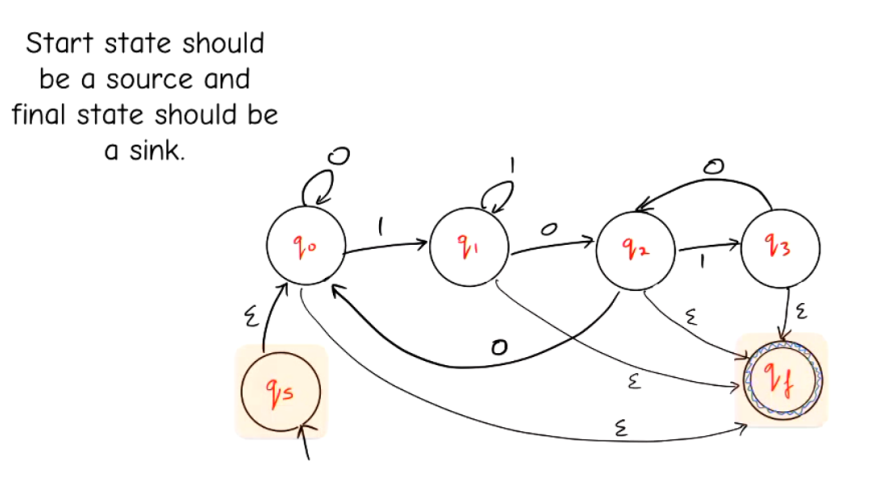


Figure 8: Sink addition

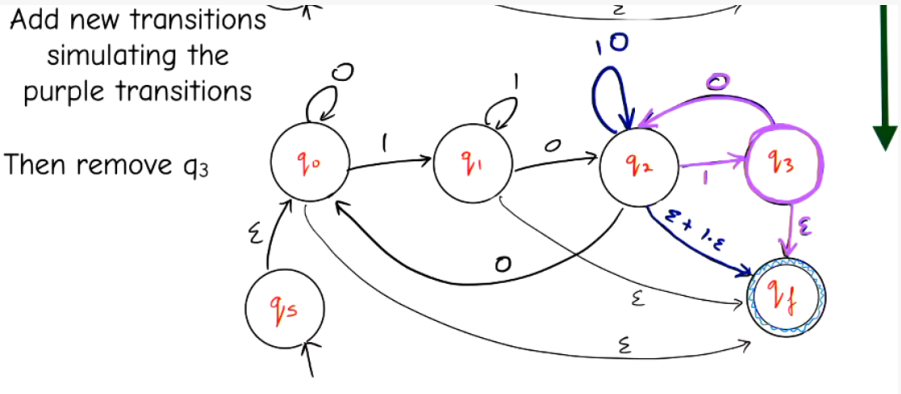


Figure 9: Re-making the transition

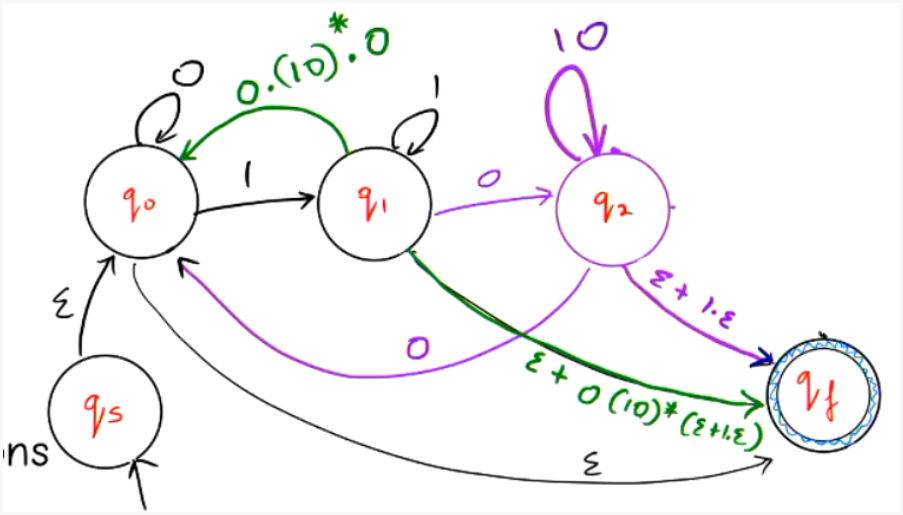


Figure 10: Removal of next node

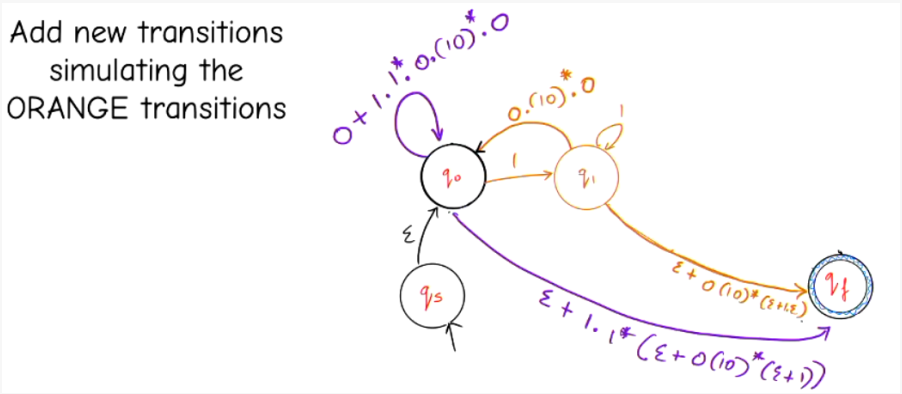


Figure 11: Another node

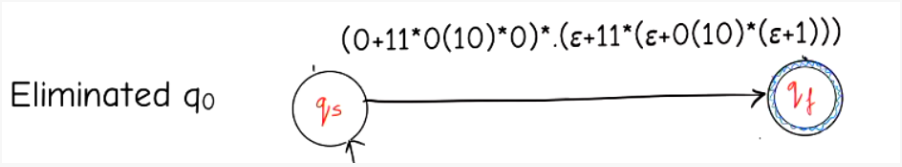


Figure 12: Last

Definition 2.6. GNFA

A run on GNFA on the word s is a sequence of states $r_0, r_1, \ldots, r_n$ such that

$$r_0 = q_0 \tag{58}$$

$$\exists s_1, s_2, \ldots, s_n \in \Sigma^* \text{ such that } s = s_1 s_2 \ldots s_n \wedge \forall i \in [n], s_i \in L(\delta(r_{i-1}, r_i)) \tag{59}$$

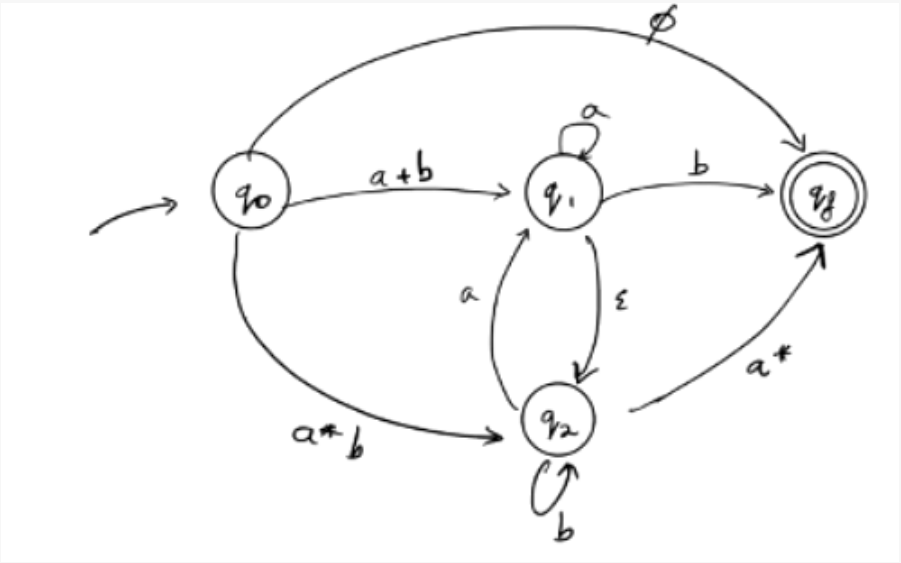


Figure 13: Example GNFA

3 Proving Regularity

3.1 Myhill-Nerode

Example 3.1. Language L

Let

$$L = \{0^n 1^n | n \geq 0\} \tag{60}$$

Is L regular?

Definition 3.1. Distinguishability

Strings x and y distinguishable by L if there is a string z such that $x.z \in L$ but $y.z \notin L$ or vice-versa. If they are indistinguishable by L otherwise, we write

$$X \equiv_L y \tag{61}$$

Note, $\equiv_L$ is an equivalence relation. In other words, two words are in the same equivalent class if they end in the same node.

Theorem 3.1. *Myhill-Nerode*

L is a regular language if and only iff $\equiv_L$ has a finite index. This will prove that they must all lie in distinct equivalence classes of $\equiv_L$ and so $\equiv_L$ must have an infinite number of equivalence classes.

Example 3.2. Myhill-Nerode Example

We will solve our initial example using our new theorem. Let $L = \{0^n 1^n | n \geq 0\}$

1. 0
2. 00
3. 000
4. $\vdots$
5. 0^r

We show that each 0 belongs to a different equivalence class. Consider concatenating 1 into every end of 0. This means that the first string is distinguishable. Consider concatenating 11 now. Second is distinguishable, but rest is not. Following this logic for every 0, from our theorem, we know that each 0 must belong to an unique state, therefore, it is an infinite DFA with infinite number of nodes, meaning it is not regular.

Example 3.3. Myhill-Nerode Example Two

Prove that the language

$$\{a^n b^n c^n | n \geq 0\} \tag{62}$$

over $\{a, b, c\}$ is not regular.

Proof. By Myhill-Nerode/index of the language. Consider the infintie set of strings generated by a^*. For any a^i, a^j where $i \neq j$, observe that $a^i \cdot a^{m-i} b^m c^m = a^m b^m c^m$, which is in L, while $a^j \cdot a^{m-i} b^m c^m \notin L$. This implies that the set of strings generated by a^* is pairwise distinguishable by L, and so L has an infinite index, implying that L must be non-regular. □

Example 3.4. Myhill-Nerode Example Three

Prove that the language

$$L = \{1^n | n \text{ is prime}\} \tag{63}$$

over $\{1\}$ is not regular.

Proof. We claim that 1^i is distinguishable from 1^j for every distinct i, j. Assume without loss of generality that $i < j$. Pick an arbitrary prime $p > i$, where $p > 2$. Since $p > 2$ and $j - i > 0$, we know that $p + 0(j-i)$ is prime and $p + p(j - i)$ is not prime. Walking down this list, at some point a prime will switch to a none prime. Let $k \leq p$ be the first such point. We now have that $p + (k - 1)(j - i)$ is prime. We show that

$$1^i . 1^{[p+(k-1)(j-i)]-i} \in L \tag{64}$$

$$1^j . 1^{[p+(k-1)(j-i)]-i} \notin L \tag{65}$$

□

3.2 Pumping Lemma

Definition 3.2. Intuition

If there is a cycle that is reachable from the start state and which can reach an accept state, in the state diagram of the DFA M, then $L(M)$ is infinite. In fact this is an iff statement. Furthermore, this is equivalent to saying that $L(M)$ accepts words more than the number of states it has.

Definition 3.3. Pumping Lemma

If L is a regular language, then $\exists m > 0$ such that $\forall w \in L : |w| > m$, $\exists$ a decomposition $w = xyz : |y| > 0, |xy| < m$ such that $\forall i \geq 0, xy^i z \in L$

Proof. Let $M = (Q, \Sigma, q_0, F, \delta)$ be a DFA accepting L. Let us set

$$m = |Q| \tag{66}$$

Consider any $w \in L : |w| \geq m$. Consider the run of w on M. (If w does not exist, this statement is vacuously true). Let the run of w on M be denoted by $(\alpha_0, \alpha_1, \ldots, \alpha_{|w|})$, where $\alpha_0 = q_0$, $\alpha_{|w|} \in F$ and $\forall j \in \{0, \ldots, |w|\}$, $\alpha_j \in Q$. By pigeon-hole principle, $\exists j_1, j_2 : j_1 < j_2$ and $\alpha_{j_1} = \alpha_{j_2}$. By our choice of j_1, either $j_1 = 0$ or the partial run $(\alpha_0, \ldots, \alpha_{j_1})$ has no repeating states. Therefore, $j_1 \leq |Q|$. Moreover, the partial run $(\alpha_{j_1}, \ldots, \alpha_{j_2})$ is a loop in the state-transition diagram of M. We define

- x to be the substring of w consumed by M in the partial run $(\alpha_0, \ldots, \alpha_j)$
- y to be the substring of w consumed by M in the partial run $(\alpha_{j_1}, \ldots, \alpha_{j_2})$
- z to be the substring of w such that $w = xyz$

We remain to prove that $\forall i \geq 0, xy^i z \in L$. But this is true since $\forall i \geq 0$, the run $(\alpha_0, \ldots, \alpha_{j_1}) \cdot (\alpha_{j_1}, \ldots, \alpha_{j_2})^i \cdot (\alpha_{j_2}, \ldots, \alpha_{|w|})$ is an accepting run in M. □

Definition 3.4. Steps for showing non-regularity

1. Suppose L is a regular and let m be the pumping length of L
2. Choose a string $w \in L$ such that $|w| \geq m$
3. Let $w = x.y.z$ be an arbitrary decomposition of w such that $|xy| \leq m$, $|y| > 0$
4. Carefully pick an integer i and argue that $x.y^i.z \notin L$

Example 3.5. Pumping Lemma Example

Show that the language

$$L = \{w \in \{a, b\}^* | n_a(w) <_b (w)\} \tag{67}$$

Is not regular.

Proof. Let m denote the pumping length of L. Let us choose $w = a^m b^{m+1}$. Let $w = x.y.z$ be an arbitrary decomposition of w such that $|xy| \leq m, |y| > 0$. Consider $x = a^\alpha, y = a^\beta, z = a^\gamma b^{m+1}$ where $\beta \geq 1$ and $\alpha + \beta + \gamma = m$. Let us choose $i = 2$. We have that $\alpha + 2\beta + \gamma \geq \alpha + \beta + \gamma + 1 = m + 1$ and this is not in L. □

Example 3.6. Pumping Lemma Example Two

Show that the language

$$L = \{w \in \{0, 1\}^* | n_0(w) = n_1(w)\} \tag{68}$$

Is not regular.

Proof. Let m denote the pumping length of L. Let us choose $w = 0^m 1^m$. Then, consider the breakdown $w = x.y.z$ where $|xy| \leq m$ and $y > 0$. In particular, let $x = 0^\alpha, y = 0^\beta, 0^\gamma 1^m$ where $\alpha + \beta + \gamma = m$ and $\beta \geq 1$. Let us choose $i = 2$. We have that $\alpha + 2\beta + \gamma \geq m + 1$. However, $m + 1 \neq m$ therefore this is not in L. □

Example 3.7. Pumping Lemma Example 3

Show that the language

$$\{a^n b^n c^n | n \geq 0\} \tag{69}$$

over $\{a, b, c\}$ is not regular

Proof. By the pumping lemma. Let L denote this language and for a contradiction suppose that L is regular and let m be its pumping length. Let $w = a^m b^m c^m$ and let $w = xyz$ where $|xy| \leq m$ and $y > 0$. Then, let $x = a^\alpha, y = a^\beta, z = a^\gamma b^m c^m$ for some α, β, γ where $\alpha + \beta + \gamma = m$ and $\beta > 0$. Observe that $a^{\alpha+\gamma} b^m c^m$, where $\alpha + \gamma < m$, implying that $xy^0 z \notin L$, a contradiction to the pumping lemma. This proves that L is not regular. □

4 Grammars

4.1 Definition

Definition 4.1. Grammar

The formal definition of a grammar is given by

$$G = (V, \Sigma, R, S) \tag{70}$$

Where V is the finite set of variables, Σ is the finite alphabet, R is the finite set of production rules or productions and $S \in V$ is the START variable.

Example 4.1. Grammar Example

Consider the grammar G

$$G = (\{S\}, \{0, 1\}, R, S) \tag{71}$$
$$R : S \to {}_0S_1 \tag{72}$$
$$: S \to \varepsilon \tag{73}$$

Therefore we have

$$S \implies {}_0S_1 \implies {}_{00}S_{11} \implies {}_{000}S_{111} \implies 000111 \tag{74}$$

Definition 4.2. Left-most Derivation

The left-most derivation of some grammar G means that you keep replacing the left-most variable.

Definition 4.3. Parse Tree

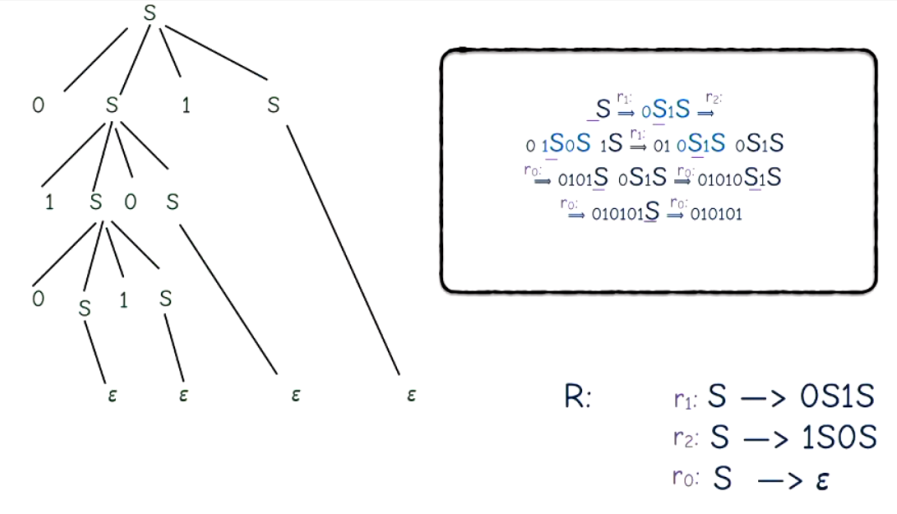


Figure 14: Parse Tree

Definition 4.4. Ambiguity

A grammar G is ambiguous if it can generate the same string with multiple parse trees if and only if the same string can be derived with two left-most derivations. Some but not all ambiguous grammars can be re-written with rules in order to get rid of the ambiguity.

Definition 4.5. Chomsky hierarchy of Grammars

1. Regular grammars - follow the rules of $A \to xB, A \to x$ or $A \to Bx, A \to x$
2. Context free grammars - $A \to w$ (letter) and B can be anything
3. Context sensitive - $\alpha A \gamma \to \alpha \beta \gamma$
4. Recursively enumerable - $\alpha \to \beta$

4.2 Grammars and DFAs

Example 4.2. DFA to right-linear Grammar

Consider the DFA

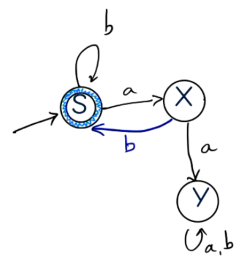


Figure 15: DFA

and the grammar

$$G = (\{S, X, Y\}, \{a, b\}, R, S) \tag{75}$$

We now write the rules according to the nodes of our DFA, that is

$$S \to b.S|a.X|\varepsilon \tag{76}$$
$$X \to b.S|a.Y \tag{77}$$
$$Y \to a.Y|b.Y \tag{78}$$

Definition 4.6. DFA to left-linera Grammar

For left linear grammars, instead of starting at the beginning state, we begin at the finish state. Instead of focusing on the nodes where it goes to, we care about the nodes that lead to our finish state from the start state and repeat the process.

5 Push Down Automaton

Definition 5.1. Push Down Automaton

A push down automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ and F are all finite sets and

1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet (Usually includes Σ and \$ the empty stack symbol)
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q\times$
5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is the set of accept states

Note that push down machines are non-deterministic.

Definition 5.2. Transitions in PDM

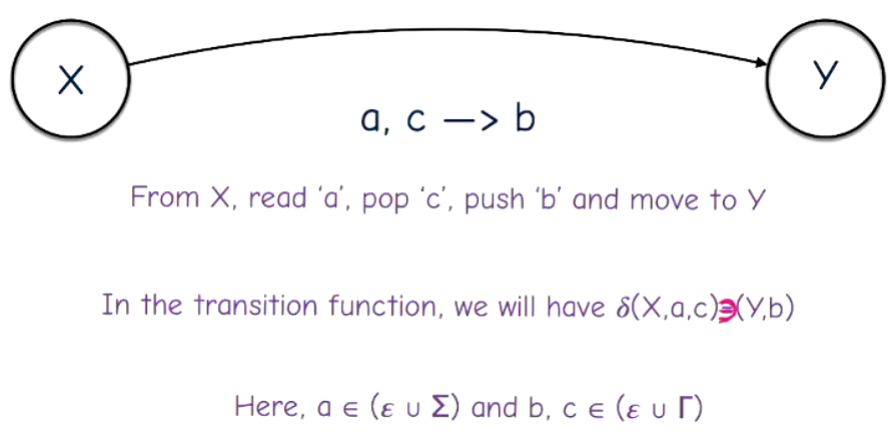


Figure 16: Push Down Transitions

When pushing in strings, we push the last letter first so the pop can work properly.

Theorem 5.1. *Acceptance*

Pushdown Automata accept precisely the set of Context Free Languages

5.1 CFG to PDA

Definition 5.3. CFG to PDA

The following construction of a PDA from a given CFG will rely on the following three states

- Start state q_s
- Loop state q_I
- Accept state q_a

In addition, we will implicitly use a number of auxiliary states whenever we take advantage of the shorthand notation for multiple pushes

Example 5.1. CFG to PDA example

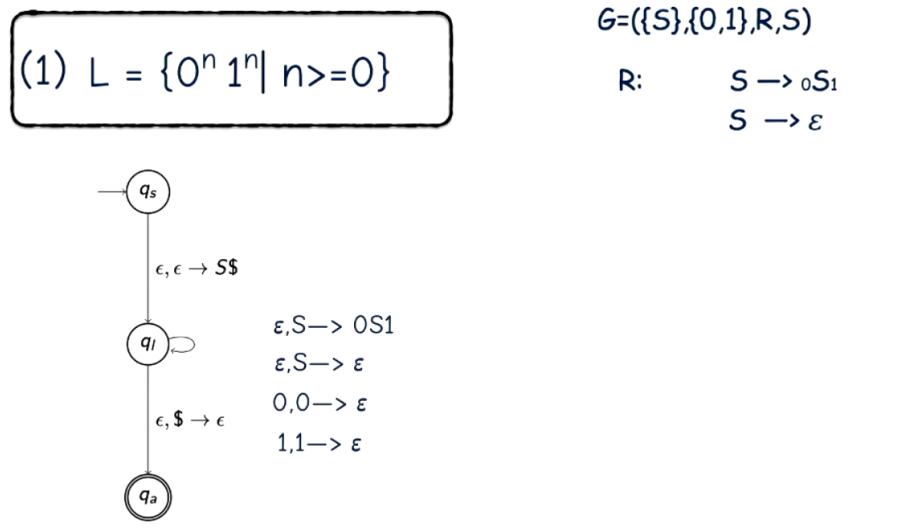


Figure 17: CFG to PDA

5.2 PDA to CFG

In order to convert a PDA to a CFG, we first need to normalise it.

Definition 5.4. Normalised PDA

A normalised PDA has

1. It has a single accept, q_{accept}
2. It empties its stack before accepting
3. Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.

Definition 5.5. A

There is a non-terminal A_{pq} for each pair of states p and q. A_{pq} generates all strings which take us from state p (with empty stack) to state q (with empty stack). Languages accepted by the PDA is the set of strings which can be derived from the non-terminal $A_{q-initial,q-final}$

Definition 5.6. Three Rules

1. For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$ if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \rightarrow aA_{rs}b$ in G.
2. For each $p, q, r \in Q$, put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G
3. Finally, for each $p \in Q$, put the rule $A_{pp} \rightarrow \varepsilon$ in G

5.3 String in Grammar

Definition 5.7. Cocke-Younger-Kasami

A grammar $G = (V, \Sigma, R, S)$ is in Chomsky Normal Form if every production has one of the following shapes:

1. $S \to \varepsilon$
2. $A \to x (x \in \Sigma$ is a terminal$)$
3. $A \to BC (B, C \in V$ where $B, C \neq S)$

Example 5.2. CYK Algorithm

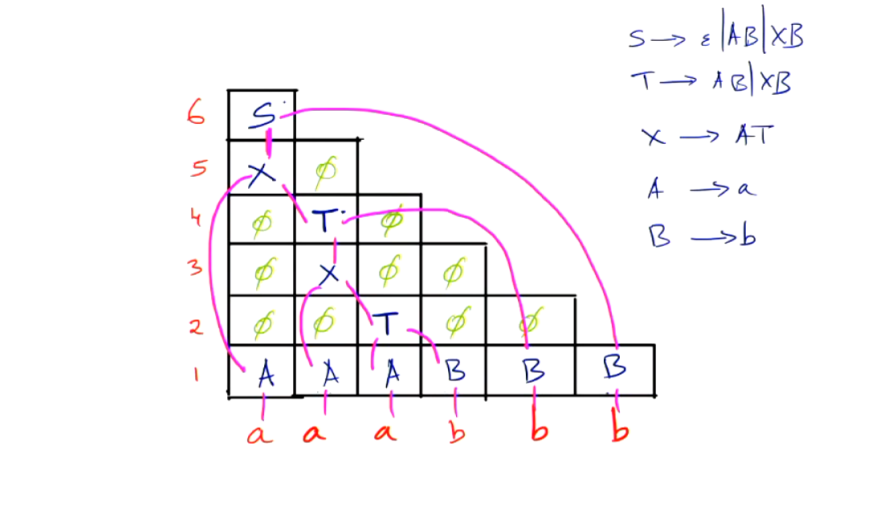


Figure 18: CYK Algorithm

Definition 5.8. To Chomsky Normal Form

1. Create a new start variable S_0 and add $S_0 \to S$
2. Eliminate $A \to \varepsilon$ productions where $A \neq S$
3. Eliminate unit productions $A \to B$
4. Add new variables and productions to eliminate remaining violations in the rules of the form $A \to u$ where $|u| > 2$ and in rules of the form $A \to xy$ where x and y are both terminals

The CYK algorithm can be used to perform a bottom up parsing of strings in polynomial time. Every string of length n can be derived in exactly $2n - 1$ steps

Definition 5.9. Useless

A grammar G can be called useless if it generates something that can fail to generate a string

1. Find and mark all variables with a terminal or ε on the right hand side
2. Repeat until on new variables get marked
3. If S is unmarked, return $L(G)$ is empty, else return $L(G)$ is none-empty

5.4 Pumping Lemma for CFL

Definition 5.10. Pumping Lemma for CFL

If L is a CFL, then $\exists m > 0$ such that $\forall w \in L : |w| \geq m, \exists$ a decomposition $w = uvxyz : |vy| > 0, |vxy| \leq m, \forall i \geq 0, uv^i xy^i z \in L$

Definition 5.11. Main Idea

If the length of the longest string in the right hand side of any production rule in R is b, then any node in any parse tree yielding a string in $L(G)$ has at most b children.
If every node of a rooted tree has at most b children and the height of the tree is h, then the number of leaves is at most b^h.

Definition 5.12. Using Lemma to show it is not CFL

1. Assume that L is a CFL.
2. Let m be the pumping length given by the pumping lemma
3. Choose a string $w \in L$ such that $|w| \geq m$
4. Take an arbitrary decomposition $w = uvxyz$ where $|vxy| \leq m$ and $|vy| > 0$
5. Carefully choose an i such that $uv^i xy^i z \notin L$, contradicting the pumping lemma

Example 5.3. CFL Pumping Example

Prove that the statement

$$L = \{a^n b^n c^n | n \geq 0\} \tag{79}$$

Is not a CFL.

Proof. Assume that L is a CFL. Let m be the pumping length given by the pumping lemma. We choose the string $n = m$ and clearly, $|w| \geq m$. There are multiple cases. Consider when vxy is equal to only the a, b or c, i.e., only $aaaaa\ldots$ or $bbbbb\ldots$ etc. In each of these three cases, $uv^0xy^0z \notin L$ so we just take $i = 0$. For the cases where cxy is between a and b or b and c, we can just take $i = 0$ again. In the case where vxy spans all letters, this cannot possibly be true because it is then bigger than m. □

Definition 5.13. Properties

CFLs are closed under

- Union
- Concatenation
- Kleene Closure

6 Turing Machines

6.1 Definition

Definition 6.1. Turing Machine

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\}$

1. Q is the set of states
2. Σ is the input alphabet not containing the blank symbol $\sqcup$
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. q_{accept} is the accept state and
7. q_{reject} is the reject state, where $q_{accept} \neq q_{reject}$

Definition 6.2. Configurations

The configuration is given by

$$X = (u, q, v) \tag{80}$$

which is a configuration if current state is q, the tape contains the string uv and the reading head is on the first symbol of v. The configuration $X = (u, q, v)$ yields configuration $Y = (u', p, v')$ if the turing machine can legally go from X to Y in one step

Definition 6.3. Decidability

A language L is Turing decidable if it is the language accepted by some dicider or total Turing machine. Note that Turing decidable = Recursive = Deciders

6.2 Decidability

Definition 6.4. Halting Problem

The halting problem is defined as

$$HP := \{< M, x > | M \text{ halts on } x\} \tag{81}$$

The halting problem is Turing recognisable but not decidable.

Definition 6.5. Membership Problem

The membership problem is defined as

$$MP := \{< M, x > | x \in L(M)\} \tag{82}$$

It is also Turing recognisable but not decidable. We construct a new machine M' obtained from M as follows

- Add a new accept state to M
- Make all incoming transitions to the old accept and reject states go to the new accept state

Simulate the assumed decider N on input $< M', x >$ and accept if and only if N accepts.

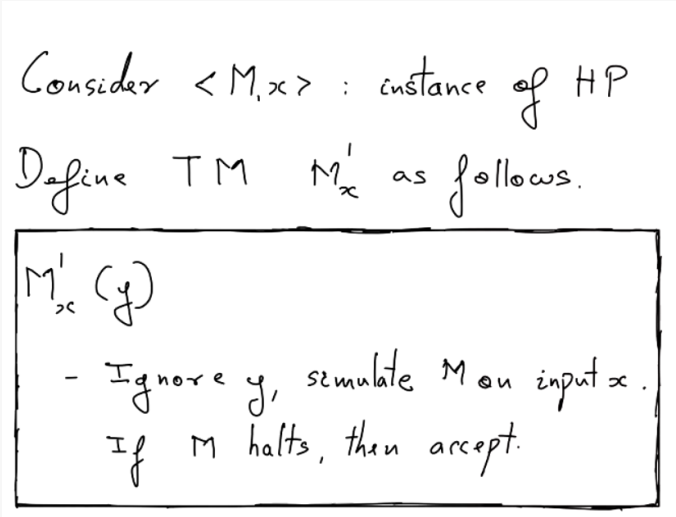


Figure 19: HP to MP

Definition 6.6. Mapping

For $A \subseteq \Sigma^*$, $B \subseteq \Delta$ where $\sigma : \Sigma^* \to \Delta^*$ is called a mapping (reduction) from A to B if

1. for all $x \in \Sigma^*, x \in A \iff \sigma(x) \in B$
2. σ is a computable function

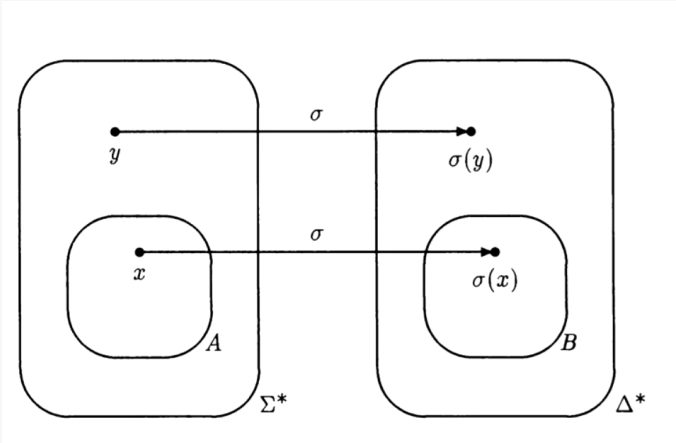


Figure 20: Mapping

Definition 6.7. Computability

The expression

$$\sigma : \Sigma^* \to \Delta^* \tag{83}$$

Is a computable function if there is a decider that, on input x, halts with $\sigma(x)$ written on the tape.

Definition 6.8. Reduction

For $A \subseteq \Sigma^*$, $B \subseteq \Delta^*$ we say that $A \leq_m B$ if there is a function $\sigma : \Sigma^* \to \Delta^*$

1. for all $x \in \Sigma^*, x \in A \iff \sigma(x) \in B$
2. σ is a computable function

Theorem 6.1. *Decidable*

A language L is decidable if and only if L and $\overline{L}$ are both Turing-recognisable.

7 Revision

	Reg.	CFL	Decidable	Tur. recog.
Complement	Y	N	Y	N
Union	Y	Y	Y	Y
Intersection	Y	N	Y	Y
Kleene Closure	Y	Y	Y	Y
Concatenation	Y	Y	Y	Y

Figure 21: Properties table: Closure

	Reg.	CFL	Decidable	Tur. recog.
Reg.	Reg			
CFL	CFL	Dec		
Decidable	Dec	Dec	Dec	
Tur.recog.	Tur. rec.	Tur. rec.	Tur. rec.	Tur. rec.

Figure 22: Pairwise closure: Intersection