CENG 280 - Review

Burak Metehan Tunçel - April 2022

1 Chapter 1

1.1 Alphabets and Languages

Alphabet: A finite set of symbols, e.g., $\{a, b, \ldots, z\}$, $\{0, 1\}$.

String: A string over an alphabet is a finite sequence of symbols from the alphabet, e.g., "0011", "a", "e" (empty string). The set of all strings (including the empty string) over Σ is denoted by Σ^*. The length of a string is the length of the sequence, e.g. $|abc| = 3$.

Language: Any subset L of Σ^* for an alphabet Σ is called a language over Σ.

1.1.1 String Operations

Concatenation: Two strings x, y over the same alphabet, e.g. $x, y \in \Sigma^*$, can be combined. $w = x \circ y$, or simply $w = xy$.

Substring: A string v is a substring of w if w can be written as $w = xvy$. If $w = vx$ then v is a *prefix* of w, and if $w = xv$, then v is a *suffix* of w.

Reversal: The reverse of a string w, denoted by w^R, is the string spelled backwards. For example, $w = abc \Rightarrow w^R = cba$.

Theorem 1.1

For any two strings x, w, $(wx)^R = x^R w^R$.

1.1.2 Languages

Given an alphabet Σ, any subset of Σ^* is called a language.

$$L = \{w \in \Sigma^* \mid w \text{ has the property } P\}$$

Language Operations Languages are sets, so set operations (union, intersection, difference) can be used on languages.

- **Complement:** $\overline{L} = \Sigma^* \setminus L$
- **Concatenation:** L_1, L_2 are languages over Σ. $L = L_1 \circ L_2$ (or simply $L = L_1 L_2$) is defined as
$$L = \{w_1 w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$$
- **Kleene star:** The Kleene star of a language L, denoted by L^*, is the set of strings obtained by concatenating 0 or more strings from L.
- $L^+ = LL^*$

1.2 Finite Representations of Languages

Definition 1.1: Regular Expression

The regular expressions over an alphabet Σ are all strings over the alphabet $\Sigma \cup \{(,), \emptyset, \cup, *\}$ that can be obtained as follows:

1. $\emptyset$ and each member of Σ is a regular expression.
2. If α and β are regular expressions than so is $(\alpha\beta)$.
3. If α and β are regular expressions than so is $(\alpha \cup \beta)$.
4. If α is a regular expression than so is α^*.
5. Nothing is a regular expression unless it follows from 1-4.

Every regular expression defines a language.

Definition 1.2: Languages defined by regular expressions

For a regular expression α, $L(\alpha)$ is the language represented by α and it is defined as

1. $L(\emptyset) = \emptyset$ and $L(a) = \{a\}$ for each $a \in \Sigma$.
2. If α and β are regular expressions than $L(\alpha\beta) = L(\alpha)L(\beta)$.
3. If α and β are regular expressions than $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$.
4. If α is a regular expression than so is $L(\alpha^*) = L(\alpha)^*$

The class of **regular languages** consists of all languages L such that $L = L(\alpha)$ for some regular expression α.

Language recognition device: For some language L, an algorithm that answers the question is $w \in L$

Language generators: Descriptions of how a string from a language can be produced.

2 Chapter 2

2.1 DFA: Deterministic Finite Automata

Definition 2.1

A **deterministic finite automaton** is a quintuple $M = (K, \Sigma, \delta, s, F)$ where

- K is a finite set of **states**
- Σ is an alphabet,
- $s \in K$ is the **initial state**
- $F \subseteq K$ is the set of **final states**
- δ, the **transition function**, is a function from $K \times \Sigma$ to K.

The **configuration** of the machine is the *current state and the unread part of the input string*, i.e., a configuration is an element of $K \times \Sigma^*$.

Let (q, w) and (q', w') be two configurations of M. Then $(q, w) \vdash_M (q', w')$ if and only if $w = aw'$ for some $a \in \Sigma$ and $q' = \delta(q, a)$. For example, $(q, aabb) \vdash_M (q', abb)$ where $q' = \delta(q, a)$.

$(q, w) \vdash_M (q', w')$ reads (q, w) **yields** (q', w') in **one step**. $\vdash_M^*$ is the **reflexive transitive closure** of $\vdash_M$ (It can be though like the multiple steps). A string $w \in \Sigma^*$ is **accepted** by M if and only if $(s, w) \vdash_M^* (f, e)$ for some $f \in F$. The **language** of M, $L(M)$, is the set of strings *accepted* by M.

2.2 NFA: Nondeterministic Finite Automata

Definition 2.2

A **nondeterministic finite automaton** is a quintuple $M = (K, \Sigma, \Delta, s, F)$, where

- K is a finite set of **states**
- Σ is an alphabet
- $s \in K$ is the **initial state**
- $F \subseteq K$ is the set of **final states**, and
- Δ, the **transition relation**, is a subset of $K \times (\Sigma \cup \{e\}) \times K$

$(q, a, p) \in \Delta$ is called a **transition** of M. (q, e, p) indicates that the machine can pass to state p from state q without reading an input symbol.

The **configuration** of the machine is the current state and the unread part of the input string, i.e., a configuration is an element of $K \times \Sigma^*$.

Let (q, w) and (q', w') be two configurations of M. Then $(q, w) \vdash_M (q', w')$ if and only if $w = aw'$ for some $a \in \Sigma \cup \{e\}$ and $(q, a, q') \in \Delta$.

$(q, w) \vdash_M (q', w')$ reads (q, w) **yields** (q', w') **in one step**. $\vdash_M^*$ is the **reflexive transitive closure** of $\vdash_M$. $(q, w) \vdash_M^* (q', w')$ reads (q, w) yields (q', w'). A string $w \in \Sigma^*$ is **accepted** by M if and only if there is a state $f \in F$ such that $(s, w) \vdash_M^* (f, e)$. The **language** of M, $L(M)$, is the set of strings accepted by M.

A deterministic finite state automaton is just a special type of nondeterministic finite state automaton. We obtain a DFA when Δ defines a function from $K \times \Sigma$ to K. In other words, an NFA $M = (K, \Sigma, \Delta, s, F)$ is deterministic if there are no transitions of the form (q, e, p) and for each $q \in K$ and $a \in \Sigma$, there exists *exactly one* $p \in K$ such that $(q, a, p) \in \Delta$.

A nondeterministic finite automaton can always be converted to an *equivalent* deterministic finite state automaton.

Theorem 2.1

For each nondeterministic finite automaton, there exists an equivalent deterministic finite automaton.

Proof of the theorem is constructive. In proof, one can use subset construction algorithm to construct a DFA from anNFA and then show they are equivalent.

Two automaton (DFA or NFA, one can be DFA and the other can be NFA) M_1 and M_2 are said to be **equivalent** when $L(M_1) = L(M_2)$.

2.2.1 Subset Construction

In here, the following is main and formal definition.

Given an NFA $M = (K, \Sigma, \Delta, s, F)$, the algorithm constructs an equivalent DFA $M' = (K', \Sigma, \delta, s', F')$ as follows. For each state $q \in K$, the set of states that can be reached without reading an input symbol is defined as

$$E(q) = \{p \in K \mid (q, e) \vdash_M^* (p, e)\}$$

Essentially, $E(q)$ is the reflexive transitive closure of the set $\{q\}$ under the relation $\{(p, r) \mid (p, e, r) \in \Delta\}$. The DFA is defined as:

$$\begin{aligned} K' &= 2^K \\ s' &= E(s) \\ F' &= \{Q \subseteq K \mid Q \cap F \neq \emptyset\} \\ \delta'(Q, a) &= \{E(p) : p \in K, (q, a, p) \in \Delta \text{ for some } q \in Q\} \text{ for each } Q \in K' \text{ and } a \in \Sigma \\ &= \bigcup \{E(p) : p \in K, (q, a, p) \in \Delta \text{ for some } q \in Q\} \end{aligned}$$

$$E(q) = \{q\} \cup \{p \in K \mid (q, e) \vdash_M^* (p, e)\}$$

If there is question about transforming an NFA to a DFA, the following steps can be used in question solving:

1. The NFA $M = (K, \Sigma, \Delta, s, F)$ is given and we want to construct DFA. In other words we want to acquire:

   $$M' = (K', \Sigma, \delta', s', F')$$

2. Start from the initial state of NFA, $s' = E(s)$. For each $q_j \in E(s)$ find the transition for each $k \in \Sigma$ (q_j, k, q_l).

   For example, if $E(s) = \{q_0, q_1\}$. Look the transition from q_0 and q_1. If $\Sigma = \{a, b\}$, then look for a and b.

3. Then calculate the union of $E(q_i)$ of the reachable states from the state we consider.

   For example, if transitions for a are $(q_0, a, q_1), (q_1, a, q_2)$, then write

   > $(q_0, a, q_1), (q_1, a, q_2)$ are all the transitions (q, a, p) for some $q \in E(s)$

   and calculate the $\delta'(s', a) = E(q_1) \cup E(q_2)$. If it is new state, write this new state on DFA, if not connect it the old one.

4. Follow this steps for each $k \in \Sigma$ and newly introduced steps.

Look at the example below, it is taken from 2.2.4 in the textbook.

Example 1: NFA to DFA

For this example:

- $E(q_0) = \{q_0, q_1, q_2, q_3\}$
- $E(q_1) = \{q_1, q_2, q_3\}$
- $E(q_2) = \{q_2\}$
- $E(q_3) = \{q_3\}$
- $E(q_4) = \{q_3, q_4\}$

$s' = E(q_0) = \{q_0, q_1, q_2, q_3\}$,

$$(q_1, a, q_0),\ (q_1, a, q_4),\ \text{and}\ (q_3, a, q_4)$$

are all the transitions (q, a, p) for some $q \in s'$. It follows that (Instead of s' any state symbol can be used such as q_{100})

$$\delta'(s', a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}$$

Similarly,

$$(q_0, b, q_2) \text{ and } (q_2, b, q_4)$$

are all the transitions of the form (q, b, p) for some $q \in E(q_0)$, so

$$\delta'(s', b) = E(q_2) \cup E(q_4) = \{q_2, q_3, q_4\}$$

In here $\{q_0, q_1, q_2, q_3, q_4\}$ and $\{q_2, q_3, q_4\}$ are new states.
Repeat this calculation for the newly introduced states. In the end, there will be no new states or you will be in wrong way.

2.3 Finite Automata and Regular Expressions

Theorem 2.2

The class of languages accepted by finite automata is closed under

a) union

b) concatenation

c) Kleene star

d) complementation

e) intersection

Let $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$ be nondeterministic finite automata.

(a) *Union.* $L(M) = L(M_1) \cup L(M_2)$.

$M = (K, \Sigma, \Delta, s, F)$, where s is a new state not in K_1 or K_2,

- $K = K_1 \cup K_2 \cup \{s\}$
- $F = F_1 \cup F_2$
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, e, s_1), (s, e, s_2)\}$

(b) *Concatenation.* $L(M) = L(M_1)L(M_2)$.

Then the finite automaton M that accepts $L(M_1)L(M_2)$ is defined as follows (*May include mistakes*). $M = (K, \Sigma, \Delta, s, F)$

- $s = s_1$
- $K = K_1 \cup K_2$
- $F = F_2$
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{(f, e, s_2) \mid f \in F_1\}$

(c) *Kleene star.* $L(M) = L(M_1)^*$.

Then the finite automaton M that accepts $L(M_1)^*$ is defined as follows (*May include mistakes*). $M = (K, \Sigma, \Delta, s, F)$, where s is not in M_1

- $K = K_1 \cup \{s\}$
- $F = F_1 \cup \{s\}$
- $\Delta = \Delta_1 \cup \{(s, e, s_1)\}$

(d) *Complementation.* $\overline{L} = \Sigma^* - L(M)$. $\overline{M}$ is identical to M except that final and non final states are interchanged.

$M = (K, \Sigma, \Delta, s, F)$.

- $s = s_1$
- $K = K_1$
- $F = K \setminus F_1$
- $\Delta = \Delta_1$

(e) *Intersection.*

$$L(M) = L(M_1) \cap L(M_2) = \overline{\overline{L(M_1)} \cup \overline{L(M_2)}}$$

Theorem 2.3

A language is regular if and only if it is accepted by a finite automaton.

2.3.1 Converting FA to RE

There are two popular methods for converting a DFA to its regular expression:

- Arden's Method
- State elimination method

Arden's Method is a little more complicated. Consider the state elimination method to convert FA to RE.

Rules The rules for state elimination method are as follows

1. *The initial state of FA must not have any incoming edge.*

   If there is any incoming edge to the initial edge, then create a new initial state having no incoming edge to it.

2. *There must exist only one final state in FA.*

   If there exist multiple final states, then convert all the final states into non-final states and create a new single final state.

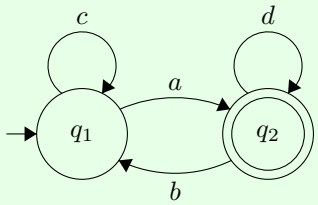
3. *The final state of DFA must not have any outgoing edge.*

   If this exists, then create a new final state having no outgoing edge from it.

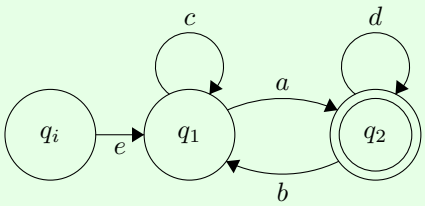
4. *Eliminate all intermediate states one by one.*

Example 2

Get the regular expression from the following FA:

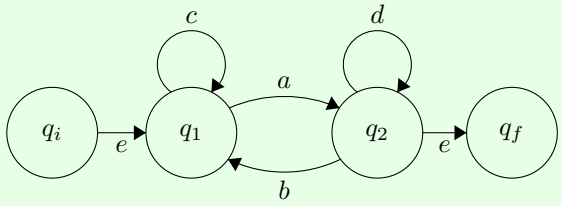


Step 1:
Initial state q_1 has an incoming edge so create a new initial state q_i.



Step 2:

Final state q_2 has an outgoing edge. So, create a new final state q_f.

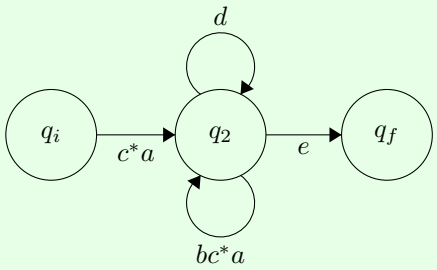


Step 3:
Start eliminating intermediate states
First eliminate q_1:

There is a path going from q_i to q_2 via q_1. So, after eliminating q_1 we can connect a direct path from q_i to q_2 having cost: $ec^*a = c^*a$.

There is a loop on q_2 using state q_i. So, after eliminating q_1 we put a direct loop to q_2 having cost: bc^*a.

After eliminating q_1, the FA looks like following



Second eliminate q_2: There is a direct path from q_i to q_f so, we can directly eliminate q_2 having cost.

$$c^*a(d + bc^*a)^*e = c^*a(d + bc^*a)^*$$

which is our final regular expression for given finite automata.

2.4 Languages that are and are not Regular

To show that a language L is regular, one of the following methods can be used:

- write a regular expression α such that $L = L(\alpha)$
- construct an NFA M such that $L = L(M)$
- use closure properties, e.g., for regular languages L_1 and L_2, show $L = L_1 \cap L_2$, $L = L_1 \cup L_2$, $L = L_1L_2$, or $L = \Sigma^* \setminus L_1$ (more examples can be added)

There are two properties shared by all regular languages, but not by certain nonregular languages, may be phrased intuitively as follows:

1. As a string is scanned left to right, the amount of memory that is required in order to determine at the end whether or not the *string is in the language must be bounded*, fixed in advance and dependent on the language, not the particular input string. For example, we would expect that $\{a^nb^n \mid n \geq 0\}$ is not regular, since it is difficult to imagine how a finite-state device could be constructed that would correctly remember, upon reaching the border between the a's and the b's, how many a's it had seen, so that the number could be compared against the number of b's.
2. Regular languages with an infinite number of strings are represented by automata with cycles and regular expressions involving the Kleene star. Such languages must have infinite subsets with a certain simple repetitive structure that arises from the Kleene star in a corresponding regular expression or a cycle in the state diagram of a finite automaton. This would lead us to expect, for example, that $\{a^n \mid n \geq 1 \text{ is a prime}\}$ is not regular, since there is no simple periodicity in the set of prime numbers.

In brief,

To show that a language L is not regular, we use the following property of regular languages:

- as a string is scanned from left to right, the amount of memory required to determine if $w \in L$ or $w \notin L$ must be bounded.
- In RL, infinite languages can be represented with Kleene star (cycle in automata), which induce a periodicity/pattern.

2.4.1 Pumping Lemma

These intuitive ideas are formalized in the following theorem known as *Pumping Lemma*.

Theorem 2.4: Pumping Lemma

Let L be a regular language. There is an integer $n \geq 1$ such that any string $w \in L$ with $|w| \geq n$ can be rewritten as $w = xyz$ such that $y \neq e$, $|xy| \leq n$ and $xy^iz \in L$ for each $i \geq 0$.

Assume L is a regular language and there is a string $w \in L$. Now, imagine, not find or determine, that there is a integer n that is $n \geq 1$, $n > 0$. Also, this integer is smaller than the length of the string w, $n \leq |w|$. So,

$$|w| \geq n \geq 1 \qquad \text{or} \qquad |w| \geq n > 0$$

Also, notice that the string can be written as three parts: $w = xyz$.
If the followings are satisfied, it shows that $w \in L$ and L is regular:

- $|y| > 0$: y is not empty string
- $|xy| \leq n$: the length of the part of the string xy is less than n
- $xy^iz \in L$, for each $i \geq 0$: When part y is repeated, or pumped, as many times we want, the expression is still in language

In here, x or z can be empty string, e, but y cannot be the empty string, e. Notice that the chosen part of string that will be compared with n should be located in first n.

Pumping lemma is also used to show that a language is not regular: start applying the pumping lemma, then find a contradiction.

For each regular language L,
there exists $n \geq 1$ such that (this is a general term do not pick a number!!!)
for each $w \in L$ with $|w| \geq n$ (write/pick a string with respect to n, that can be used to reach a contradiction)
there exists x, y, z with $w = xyz$, $y \neq e$, $|xy| \leq n$ (consider each possible split satisfying these constraints)
for each $i \geq 0$, $xy^iz \in L$ (show that there exists an i such that $xy^iz \neq L$, contradiction)

2.5 State Minimization

The process of reducing a given DFA to its minimal form is called as minimization of DFA.

- It contains the minimum number of states.
- The DFA in its minimal form is called as a Minimal DFA.

The two popular methods for minimizing a DFA are

- Equivalence Theorem
- Myhill Nerode Theorem

Since Myhill Nerode Theorem is a little bit cumbersome and prone to error compared to Equivalence Theorem, it is better to focus on the Equivalence Theorem.

2.5.1 Equivalence Theorem

The steps as follows:

1. Eliminate all the dead states and inaccessible states from the given DFA (if any).
   - **Dead State:** All those non-final states which transit to itself for all input symbols in Σ are called as dead states.
   - **Inaccessible State:** All those states which can never be reached from the initial state are called as inaccessible states.
2. Draw a state transition table for the given DFA. Transition table shows the transition of all states on all input symbols in Σ.
3. Now, start applying equivalence theorem.
   - Take a counter variable k and initialize it with value 0.
   - Divide Q (*set of states*) into two sets such that one set contains all the non-final states and other set contains all the final states.
   - This partition is called P_0.
4.
   - Increment k by 1.
   - Find P_k by partitioning the different sets of P_{k-1}.
   - In each set of P_{k-1}, consider all the possible pair of states within each set and if the two states are distinguishable, partition the set into different sets in P_k.

   Two states q_1 and q_2 are distinguishable in partition P_k for any input symbol 'a', if $\delta(q_1, a)$ and $\delta(q_2, a)$ are in different sets in partition P_{k-1}.
5. Repeat *step-4* until no change in partition occurs. In other words, when you find $P_k = P_{k-1}$, stop.
6. All those states which belong to the same set are equivalent. The equivalent states are merged to form a single state in the minimal DFA.

$$\#\ of\ states\ in\ Minimal\ DFA = \#\ of\ sets\ in\ P_k$$

3 Chapter 3

3.1 Context Free Grammars

Definition 3.1

A **context-free grammar** G is a quadruple (V, Σ, R, S) where

- V is an alphabet
- Σ (the set of **terminals**) is a subset of V
- R (the set of **rules**) is a finite subset of $(V - \Sigma) \times V^*$
- S (the **start symbol**) is an element of $V - \Sigma$

The members of $V - \Sigma$ are called **nonterminals**. For any $A \in V - \Sigma$ and $u \in V^*$, we write $A \to_G u$ whenever $(A, u) \in R$.

For any strings $u, v \in V^*$, we write $u \Rightarrow_G v$ if and only if there are strings $x, y \in V^*$ and $A \in V - \Sigma$ such that $u = xAy$, $v = xv'y$, and $A \to_G v'$.

The relation $\Rightarrow_G^*$ is the *reflexive*, *transitive closure* of $\Rightarrow_G$. Finally, $L(G)$, the **language generated** by G, is $\{w \in \Sigma^* : S \Rightarrow_G^* w\}$; we also say that G **generates** each string in $L(G)$. A language L is said to be a **context-free language** if $L = L(G)$ for some context-free grammar G.

We call any sequence of the form

$$w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n$$

a **derivation** in G of w_n from w_0. Here $w_o, \cdots, w_n$ may be any strings in V^*, and n, the **length** of the derivation, may be any natural number, including zero. We also say that the derivation has n **steps**.

- $u \Rightarrow v$: u ***directly yields*** v; $A \to w$: ***(production) rule***.
- V, alphabet, can include symbols such as start symbol, S, or A.
- (From example 3.1.4) The same string may have several derivations in a context-free grammar. Two derivations in this grammar are

  $$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S((S)) \Rightarrow S(()) \Rightarrow ()(()) \quad \text{and} \quad S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()(())$$

- Some context-free languages are not regular. However, all regular languages are context-free.
- Context-free languages are precisely the languages accepted by certain language acceptors called ***push-down automata***.