Unit-2

Regular Expressions:

Regular Expressions: Formal Definition

We construct REs from primitive constituents (basic elements) by repeatedly applying certain recursive rules as given below. (In the definition)

Definition : Let S be an alphabet. The regular expressions are defined recursively as follows.

Basis :

i) ϕ is a RE

ii) $\in$ is a RE

iii) $\forall\ a \in S$, a is RE.

These are called primitive regular expression i.e. Primitive Constituents

Recursive Step :

If r_1 and r_2 are REs over, then so are

i) $r_1 + r_2$

ii) $r_1 r_2$

iii) r_1^*

iv) (r_1)

Language described by REs : Each describes a language (or a language is associated with every RE). We will see later that REs are used to attribute regular languages.

Notation : If r is a RE over some alphabet then $L(r)$ is the language associate with r. We can define the language $L(r)$ associated with (or described by) a REs as follows.

1. ϕ is the RE describing the empty language i.e. $L(\phi) = \phi$.

2. $\in$ is a RE describing the language $\{\in\}$ i.e. $L(\in) = \{\in\}$.

3. $\forall\ a\ \in S$, a is a RE denoting the language $\{a\}$ i.e . $L(a) = \{a\}$.

4. If r_1 and r_2 are REs denoting language $L(r_1)$ and $L(r_2)$ respectively, then

i) $r_1 + r_2$ is a regular expression denoting the language $L(r_1 + r_2) = L(r_1) \cup L(r_2)$

ii) $r_1 r_2$ is a regular expression denoting the language $L(r_1 r_2) = L(r_1)\, L(r_2)$

iii) r_1^* is a regular expression denoting the language $L(r_1^*) = (L(r1))^*$

iv) (r_1) is a regular expression denoting the language $L((r_1)) = L(r_1)$

Example : Consider the RE (0*(0+1)). Thus the language denoted by the RE is

$L(0^*(0+1)) = L(0^*)\, L(0+1)$ .....by 4(ii)

$= L(0)^* L(0) \cup L(1)$

$= \{\in, 0,00,000,.....\}\ \{0\} \cup \{1\}$

$= \{\in, 0,00,000,.....\}\ \{0,1\}$

= {0, 00, 000, 0000,.....,1, 01, 001, 0001,.....}

Example : The RE ab^*+b is grouped as $((a(b^*))+b)$ which describes the language $L(a)(L(b))^* \cup L(b)$

Example : The RE $(ab)^*+b$ represents the language $(L(a)L(b))^* \cup L(b)$.

Example : It is easy to see that the RE (0+1)*(0+11) represents the language of all strings over {0,1} which are either ended with 0 or 11.

Example : The regular expression r =(00)*(11)*1 denotes the set of all strings with an even number of 0's followed by an odd number of 1's i.e. $L(r) = \{0^{2n}1^{2m+1} \mid n \geq 0,\ m \geq 0\}$

Note : The notation r^+ is used to represent the RE rr^*. Similarly, r^2 represents the RE rr, r^3 denotes $r^2 r$, and so on.

An arbitrary string over Σ = {0,1} is denoted as (0+1)*.

Regular Expression:

FA to regular expressions:

FA to RE (REs for Regular Languages) :

Lemma : If a language is regular, then there is a RE to describe it. i.e. if $L = L(M)$ for some DFA M, then there is a RE r such that $L = L(r)$.

Proof : We need to construct a RE r such that $L(r) = \{w \mid w \in L(M)\}$. Since M is a DFA, it has a finite no of states. Let the set of states of M is $Q = \{1, 2, 3, \ldots, n\}$ for some integer n. [Note : if the n states of M were denoted by some other symbols, we can always rename those to indicate as 1, 2, 3,..., n]. The required RE is constructed inductively.

Regular Grammar:

A *grammar* $G = (N, \Sigma, P, S)$ is right-linear if each production has one of the following three forms:

- $A \rightarrow cB$,
- $A \rightarrow c$,
- $A \rightarrow \in$

Where $A, B \in N'$ (with $A = B$ allowed) and $c \in \Sigma$. A *grammar* G is left-linear if each production has once of the following three forms.

$A \rightarrow Bc$, $A \rightarrow c$, $A \rightarrow \in$

A right or left-linear grammar is called a regular grammar.

Regular grammar and Finite Automata are equivalent as stated in the following theorem.

Lemma 1 : If L is a regular language, then L is generated by some right-linear grammar.

Lemma 2 : Let $G = (N, \Sigma, P, S)$ be a right-linear grammar. Then $L(G)$ is a regular language.

Example : Consider the grammar

$G: S \rightarrow 0A \mid 0$

$A \rightarrow 1S$

It is easy to see that G generates the language denoted by the regular expression (01)*0.
The construction of lemma 2 for this grammar produces the follwoing FA.
This FA accepts exactly (01)*1.

Decisions Algorithms for CFL

In this section, we examine some questions about CFLs we can answer. A CFL may be represented using a CFG or PDA. But an algorithm that uses one representation can be made to work for the others, since we can construct one from the other.

Equivalence of Finite Automata and Regular Expressions

Theorem 1. *L is a regular language iff there is a regular expression R such that $\mathbf{L}(R) = L$ iff there is a DFA M such that $\mathbf{L}(M) = L$ iff there is a NFA N such that $\mathbf{L}(N) = L$.*

i.e., regular expressions, DFAs and NFAs have the same computational power.

Proof.
- Given regular expression R, will construct *NFA* N such that $\mathbf{L}(N) = \mathbf{L}(R)$
- Given *DFA* M, will construct regular expression R such that $\mathbf{L}(M) = \mathbf{L}(R)$ □

Regular Expressions to NFA

Regular Expressions to Finite Automata
... to Non-deterministic Finite Automata

Lemma 2. *For any regex R, there is an NFA N_R s.t. $\mathbf{L}(N_R) = \mathbf{L}(R)$.*

Proof Idea
We will build the NFA N_R for R, inductively, based on the number of operators in R, $\#(R)$.

- *Base Case:* $\#(R) = 0$ means that R is $\emptyset, \epsilon$, or a (from some $a \in \Sigma$). We will build NFAs for these cases.
- *Induction Hypothesis:* Assume that for regular expressions R, with $\#(R) < n$, there is an NFA N_R s.t. $\mathbf{L}(N_R) = \mathbf{L}(R)$.
- *Induction Step:* Consider R with $\#(R) = n$. Based on the form of R, the NFA N_R will be built using the induction hypothesis.

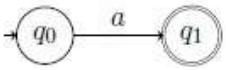
Regular Expression to NFA

Base Cases
If R is an elementary regular expression, NFA N_R is constructed as follows.

$R = \emptyset$

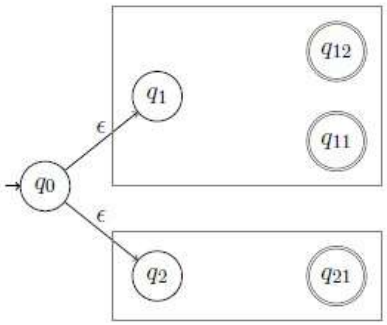
$R = \epsilon$

$R = a$



Induction Step: Union

Case $R = R_1 \cup R_2$
By induction hypothesis, there are N_1, N_2 s.t. $\mathbf{L}(N_1) = \mathbf{L}(R_1)$ and $\mathbf{L}(N_2) = \mathbf{L}(R_2)$. Build NFA N s.t. $\mathbf{L}(N) = \mathbf{L}(N_1) \cup \mathbf{L}(N_2)$



Concatenation

Case $R = R_1 \circ R_2$

- By induction hypothesis, there are N_1, N_2 s.t. $\mathbf{L}(N_1) = \mathbf{L}(R_1)$ and $\mathbf{L}(N_2) = \mathbf{L}(R_2)$
- Build NFA N s.t. $\mathbf{L}(N) = \mathbf{L}(N_1) \circ \mathbf{L}(N_2)$

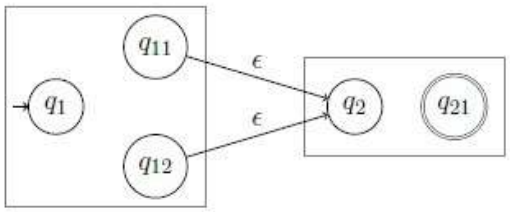
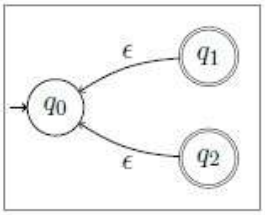


Figure 2: NFA for $\mathbf{L}(N_1) \circ \mathbf{L}(N_2)$

Kleene Closure

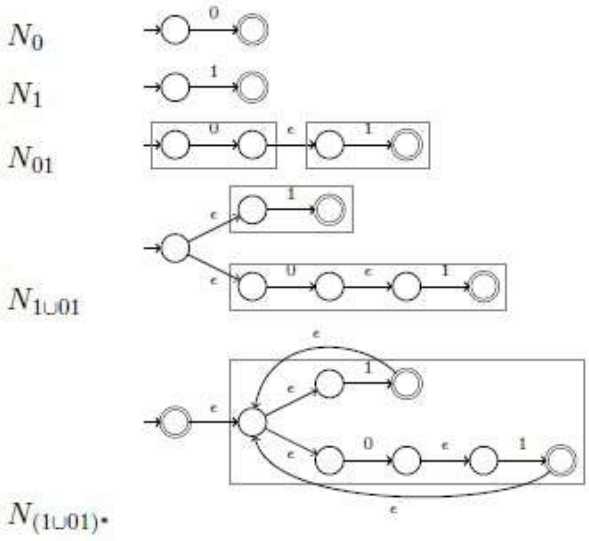
Case $R = R_1^*$

- By induction hypothesis, there is N_1 s.t. $\mathbf{L}(N_1) = \mathbf{L}(R_1)$
- Build NFA N s.t. $\mathbf{L}(N) = (\mathbf{L}(N_1))^*$



Regular Expressions to NFA

Build NFA for $(1 \cup 01)^*$



Closure Properties

- Recall that we can carry out operations on one or more languages to obtain a new language
- Very useful in studying the properties of one language by relating it to other (better understood) languages
- Most useful when the operations are sophisticated, yet are guaranteed to preserve interesting properties of the language.
- Today: A variety of operations which preserve regularity
  - i.e., the universe of regular languages is *closed* under these operations

Pumping Lemma:

Limitations of Finite Automata and Non regular Languages :

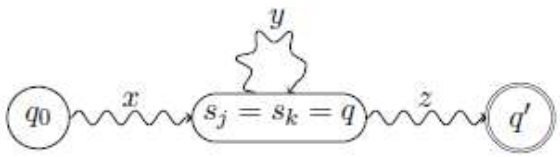
The class of languages recognized by FA s is strictly the regular set. There are certain languages which are non regular i.e. cannot be recognized by any FA

Consider the language $L = \{a^n b^n \mid n \geq 0\}$

In order to accept is language, we find that, an automaton seems to need to remember when passing the center point between a's and b's how many a's it has seen so far. Because it would have to compare that with the number of b's to either accept (when the two numbers are same) or reject (when they are not same) the input string.

But the number of a's is not limited and may be much larger than the number of states since the string may be arbitrarily long. So, the amount of information the automaton need to remember is unbounded.

A finite automaton cannot remember this with only finite memory (i.e. finite number of states). The fact that FA s have finite memory imposes some limitations on the structure of the languages recognized. Inductively, we can say that a language is regular only if in processing any string in this language, the information that has to be remembered at any point is strictly limited. The argument given above to show that $a^n b^n$ is non regular is informal. We now present a formal method for showing that certain languages such as $a^n b^n$ are non regular



- We have $\hat{\delta}_M(q_0, xy^i) = \hat{\delta}_M(q_0, x)$ for all $i \geq 1$
- Since $w \in L$, we have $\hat{\delta}_M(q_0, w) = \hat{\delta}_M(q_0, xyz) \subseteq F$
- Observe, $\hat{\delta}_M(q_0, xz) = \hat{\delta}_M(q, z) = \hat{\delta}_M(q_0, w)$, where $\{q\} = \hat{\delta}_M(q_0, x) = \hat{\delta}_M(q_0, xy)$. So $xz \in L$
- Similarly, $\hat{\delta}_M(q_0, xy^i z) = \hat{\delta}_M(q_0, xyz) \subseteq F$ and so $xy^i z \in L$

Finite Languages and Pumping Lemma

Question
Do finite languages really satisfy the condition in the pumping lemma?

Recall Pumping Lemma: If L is regular then *there is a number* p (the pumping length) such that $\forall w \in L$ with $|w| \geq p$, $\exists x, y, z \in \Sigma^*$ such that $w = xyz$ and

1. $|y| > 0$
2. $|xy| \leq p$
3. $\forall i \geq 0$. $xy^iz \in L$

Answer
Yes, they do. Let p be larger than the longest string in the language. Then the condition "$\forall w \in L$ with $|w| \geq p$, ..." is *vaccuously* satisfied as there are no strings in the language longer than p!

Proposition 4. *$L_p = \{0^i \mid i$ prime$\}$ is not regular*

Proof. Suppose L_p is regular. *Let p be the pumping length for L_p.*

- Consider $w = 0^m$, where $m \geq p + 2$ and m is prime.
- *Since $|w| > p$, there are x, y, z such that $w = xyz$, $|xy| \leq p$, $|y| > 0$, and $xy^iz \in L_p$, for all i.*
- Thus, $x = 0^r$, $y = 0^s$ and $z = 0^t$. Further, as $|y| > 0$, we have $s > 0$. $xy^{r+t}z = 0^r(0^s)^{(r+t)}0^t = 0^{r+s(r+t)+t}$. Now $r + s(r + t) + t = (r + t)(s + 1)$. Further $m = r + s + t \geq p + 2$ and $s > 0$

mean that $t \geq 2$ and $s + 1 \geq 2$. Thus, $xy^{r+t}z \notin L_p$. Contradiction!

Is $L_{\text{eq}} = \{xx \mid x \in \{0, 1\}^*\}$ is regular?

Suppose L_{eq} is regular, and let p be the pumping length of L_{eq}.

- Consider $w = 0^p0^p \in L$.
- Can we find substrings x, y, z satisfying the conditions in the pumping lemma? Yes! Consider $x = \epsilon, y = 00, z = 0^{2p-2}$.
- Does this mean L_{eq} satisfies the pumping lemma? Does it mean it is regular?
  - No! We have chosen a bad w. To prove that the pumping lemma is violated, we only need to exhibit *some* w that cannot be pumped.
- Another bad choice $(01)^p(01)^p$.