CS340: Theory of Computation — Sem I 2017-18

Lecture Notes 4: Regular Expressions

Raghunath Tewari — IIT Kanpur

1 Regular Expression

- An algebraic way to represent regular languages.
- Some practical applications: pattern matching in text editors, used in compiler design.

Some examples

Expression	**Language**
0	$\{0\}$
1	$\{1\}$
$0 \cup 1$	$\{0, 1\}$
0^*	$\{\epsilon, 0, 00, 000, \ldots\}$
$(0 \cup 1)^*$	$\{\epsilon, 0, 1, 00, 01, 10, \ldots\}$
$(0 \cup 1) \cdot 1^*$	$\{0, 1, 01, 11, 011, 111, \ldots\}$
ϵ	$\{\epsilon\}$
$\emptyset$	$\{\}$

Each expression corresponds to a language. Regular expressions are defined inductively as shown below.

Definition 1.1. R is said to be a *regular expression* (or RE in short) if R has one of the following forms:

Regular Expression	**Language of the regular expression or $L(R)$**	**Comment**
$\emptyset$	$\{\}$	the empty set
ϵ	$\{\epsilon\}$	the set containing ϵ only
a	$\{a\}$	$a \in \Sigma$
$R_1 \cup R_2$	$L(R_1) \cup L(R_2)$	for two regular expressions R_1 and R_2
$R_1 \cdot R_2$	$L(R_1) \cdot L(R_2)$	for two regular expressions R_1 and R_2
R_1^*	$(L(R_1))^*$	for a regular expression R_1
(R_1)	$L(R_1)$	for a regular expression R_1

Remark. Note the following

- Regular expressions are well defined. In other words, each regular expression corresponds to a unique language. Is the converse true?
- $\cup$ is often replaced by $+$. Hence $R_1 \cup R_2$ is the same as $R_1 + R_2$.
- The dot symbol is often discarded.
- () gives precedence to an expression (similar to standard arithmetic).

- Order of precedence (higher to lower): () $\quad * \quad \cdot \quad \cup$
- The language corresponding to the RE $\emptyset^*$ is $\{\epsilon\}$. (since ϵ is the concatenation of zero symbols from the set $\emptyset$)

Some more examples of REs and their corresponding languages.

R	**L(R)**
01	$\{01\}$
$01+1$	$\{01, 1\}$
$(01+\epsilon)1$	$\{011, 1\}$
$(0+10)^*(\epsilon+1)$	$\{\epsilon, 0, 10, 00, 001, 010, 0101, \ldots\}$

Informally, $L(R)$ consists of all those strings that "matches" the regular expression R. Let us see some examples of the other type. That is given a regular language, what is the corresponding regular expression.

Language	**RE**
$\{w \mid w$ has a single $1\}$	0^*10^*
$\{w \mid w$ has at most a single $1\}$	$0^* + 0^*10^*$
$\{w \mid \lvert w\rvert$ is a multiple of $3\}$	$((0+1)(0+1)(0+1))^*$
$\{w \mid w$ has a 1 at every odd position and $\lvert w\rvert$ is odd$\}$	$1((0+1)1)^*$
$\{w \mid w$ has a 1 at every even position$\}$	$((0+1)1)^* + (0+1)(1(0+1))^*$

We say that two regular expressions R_1 and R_2 are equivalent (denoted as $R_1 = R_2$) if $L(R_1) = L(R_2)$.

Note 1. Some basic algebraic properties of REs.

1. $R_1 + (R_2 + R_3) = (R_1 + R_2) + R_3$
2. $R_1(R_2R_3) = (R_1R_2)R_3$
3. $R_1(R_2 + R_3) = R_1R_2 + R_1R_3$
4. $(R_1 + R_2)R_3 = R_1R_3 + R_2R_3$
5. $R_1 + R_2 = R_2 + R_1$ (only addition is commutative))
6. $(R^*)^* = R^*$
7. $R\epsilon = \epsilon R = R$
8. $R\emptyset = \emptyset R = \emptyset$
9. $R + \emptyset = R$

2 Regular Expressions and Regular Languages

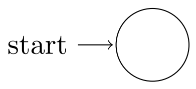
Theorem 1. *A language L is regular if and only if $L = L(R)$ for some regular expression R.*

In other words, REs are equivalent in power to NFAs/DFAs.

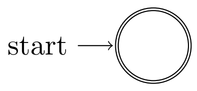
2.1 Converting an RE to an NFA

Given a regular expression, we will convert it into an NFA N such that $L(R) = L(N)$. We will give a case based analysis based on the inductive definition of REs.

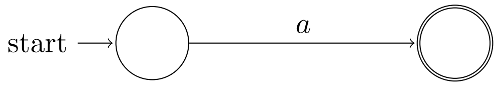
Case 1: $R = \emptyset$. NFA is



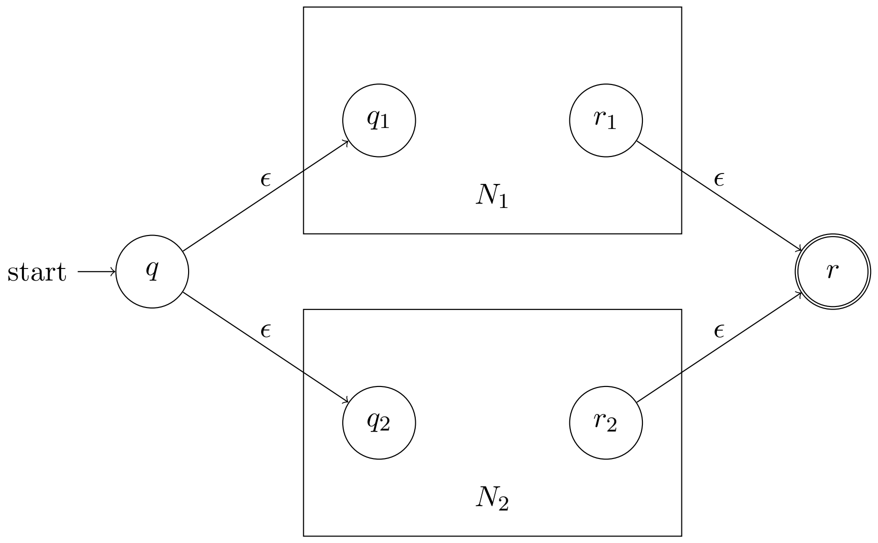
Case 2: $R = \epsilon$. NFA is



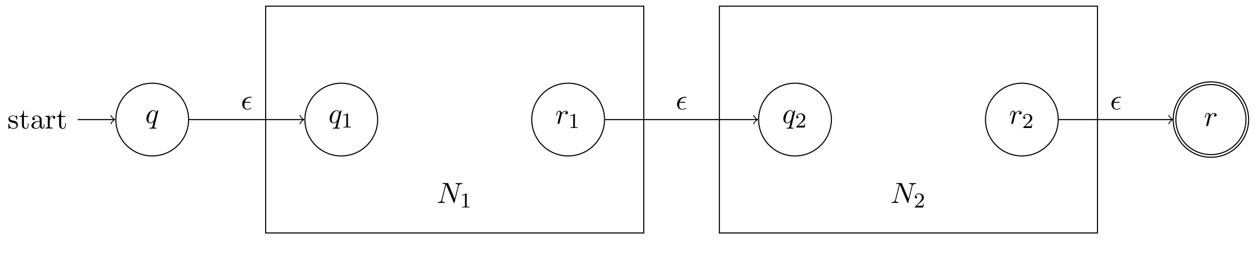
Case 3: $R = a$ for some $a \in \Sigma$. NFA is



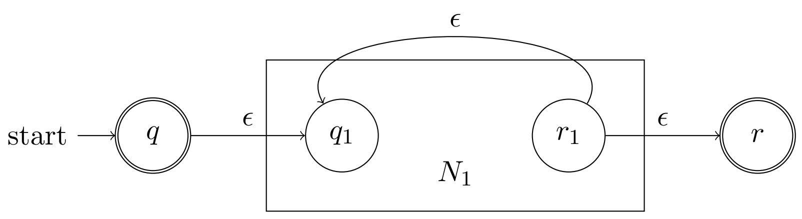
Case 4: $R = R_1 + R_2$, where R_1 and R_2 are two REs. Let N_1 and N_2 be the NFAs for R_1 and R_2 respectively. Then the NFA for R is



Case 5: $R = R_1R_2$, where R_1 and R_2 are two REs. Let N_1 and N_2 be the NFAs for R_1 and R_2 respectively. Then the NFA for R is



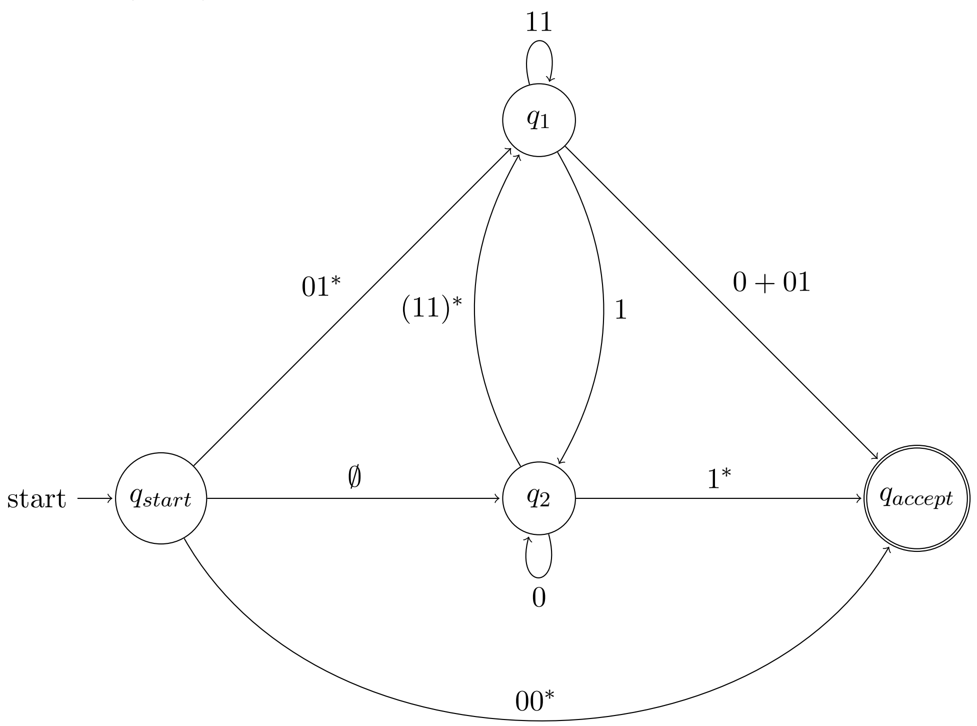
Case 6: $R = R_1^*$, where R_1 is an RE. Let N_1 be the NFA for R_1. Then the NFA for R is



The above construction constructs an NFA from an RE in an inductive manner. Therefore the class of languages accepted by regular expressions are a subset of regular languages.

2.2 Generalized Nondeterministic Finite Automaton

We will now prove that for every regular language there exists a regular expression. For this we will introduce another type of finite automaton known as *generalized non-deterministic finite automaton* (or GNFA).
A GNFA is a non-deterministic automaton with transitions being labeled with regular expressions instead of just symbols from the alphabet or ϵ. Here is an example of a GNFA.



Strings accepted by the above GNFA:

- 01101 : in multiple ways.

- 00 : at least 2 ways.

- 0100

Strings not accepted by the above GNFA:

- 10 : no way to partition so that it matches a sequence from start to accept state

- ϵ

A string $w \in \Sigma^*$ is accepted by a GNFA if $w = w_1 w_2 \dots w_k$, where each $w_i \in \Sigma^*$ and there exists a sequence of states $q_0, q_1, \dots q_k$, such that

- q_0 is the start state,

- q_k is the accept state, and

- for each i, if the transition from q_{i-1} to q_i is labeled with the regular expression R_i, then $w_i \in L(R_i)$.

We assume the following conditions on a GNFA without loss of generality.

1. Has a unique start state and a unique accept state.

2. The start state has a transition going out to every other state (excluding itself).

3. No transition coming into the start state from any other state.

4. The accept state has a transition coming in from every other state (excluding itself).

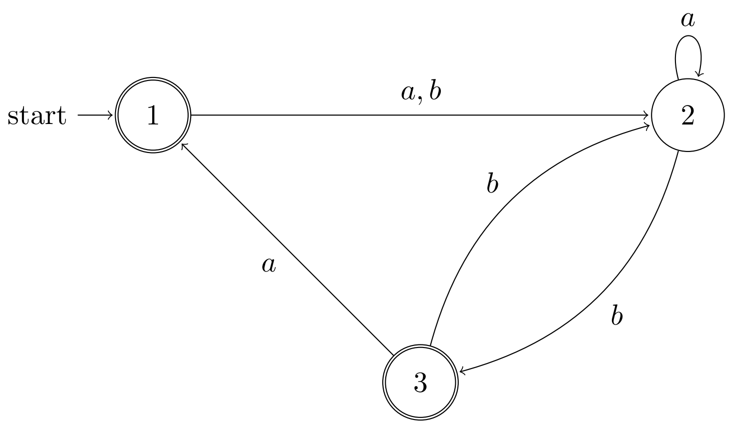
5. No transition going out of the accept state to any other state.

6. Except for the start and accept states, there are transitions between every pair of states (in both directions), and also from a state to itself.

2.3 Converting a DFA to an RE

We will illustrate the algorithm with an example.

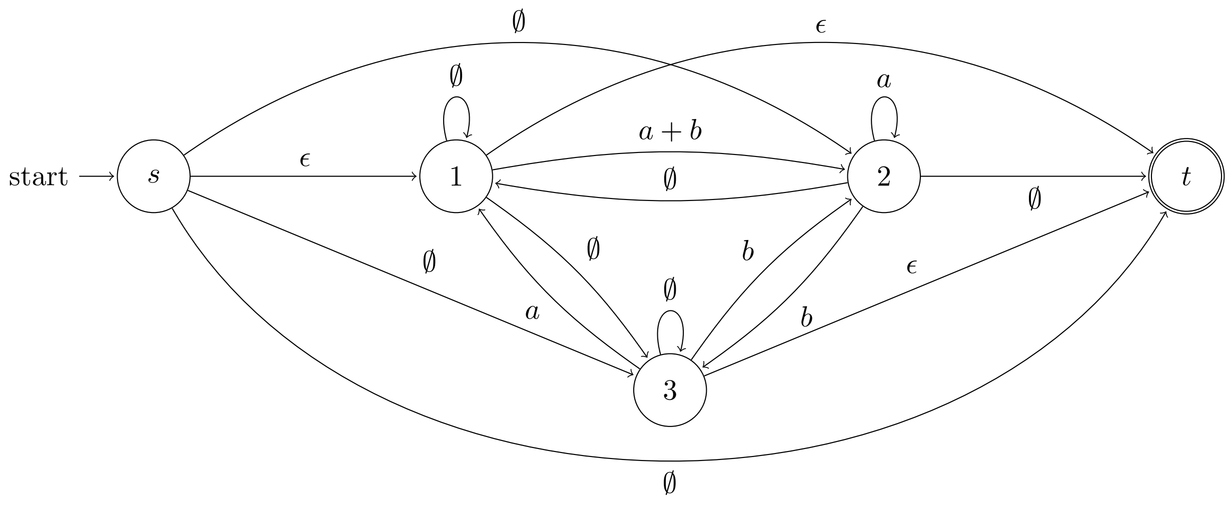
1. Consider the following DFA.



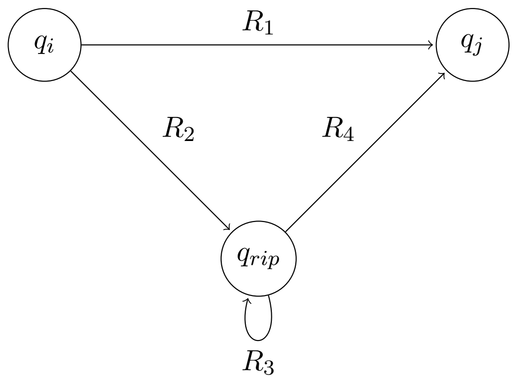
2. We convert the DFA into a GNFA satisfying the above assumptions.

   - Create new start state s and new start accepting state t. Let the new set of states be Q
   - Add ϵ transition from s to old start state.
   - Add ϵ transitions from old accept states to t.

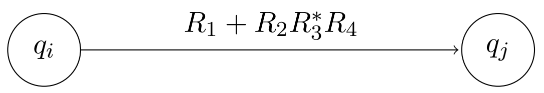
- Make sure there are transitions from s to every state in the GNFA (except s itself), and from every state (except t) to t.
- Add transitions from every state in $Q \setminus \{s,t\}$ to every other state in $Q \setminus \{s,t\}$, putting the label $\emptyset$, if a transition did not exist there earlier.



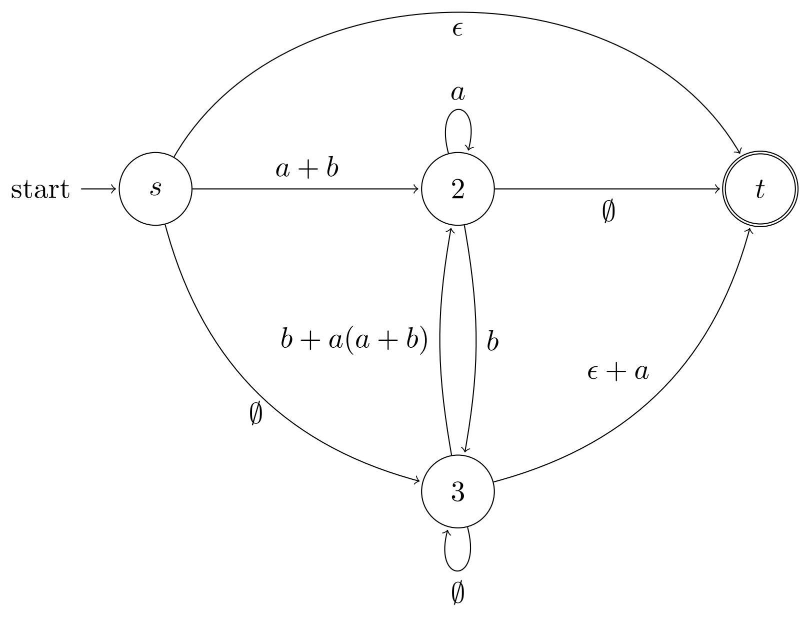
3. We now remove states in $Q \setminus \{s,t\}$, one at a time. replace the resulting transitions with suitable labels as described below. Consider the following set of 3 states with regular expressions labeled on the transitions, and q_{rip} is the state that we want to remove.



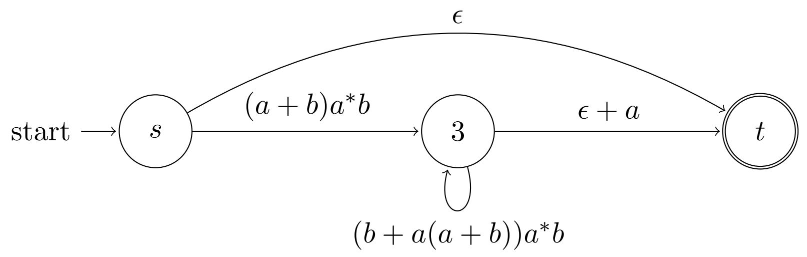
Then on removing q_{rip}, the resulting GNFA will be



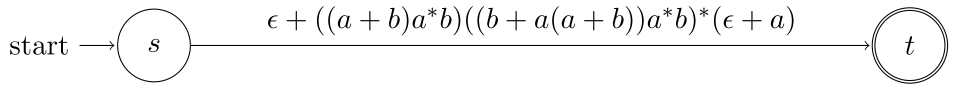
- GNFA after removing state **1**.



- GNFA after removing state **2**.



- GNFA after removing state **3**.



Therefore regular expression corresponding to the given DFA is

$$\epsilon+((a+b)a^*b)((b+a(a+b))a^*b)^*(\epsilon+a)$$