Regular Languages and Regular Expressions

According to our definition, a language is regular if there exists a finite state automaton that accepts it. Therefore every regular language can be described by some nfa or dfa.

Regular expressions:
One way of describing regular languages is via the notation of regular expressions.
This notation involves a combination of strings of symbols from some alphabet, parentheses and the operators +, ., and *.
If $\sum$ is an alphabet, the regular expression $\sum$ describes the language consisting of all strings of length 1 over this alphabet and $\sum^*$ describes the language consisting of all strings over that alphabet.

Formal Definition of a Regular Expression:
Let $\sum$ be a given alphabet, then:

- ε, $\emptyset$, and $a \in \sum$ are all regular expressions.
  These are called primitive regular expressions.
- if R1 and R2 are regular expressions , then
  - (R1 $\cup$ R2) or R1 + R2,
  - R1.R2,
  - R1* and
  - (R1) are also regular expressions
- A string is a regular expression iff it can be derived from primitive regular expressions by a finite number of applications of the rules in 2

In item 1 the regular expressions a and ε represent the languages $\{a\}$, and $\{\varepsilon\}$, respectively; the regular expression $\emptyset$ represents the empty language.

In items 2, the expressions represent the languages obtained by taking the union, the concatenation of the languages R1 and R2 or the star of the language R1, respectively.
Regular expressions are useful tools in the design of compilers for programming languages.

Languages associated with Regular Expressions:
Regular expressions can be used to describe some simple languages. If R is a regular expression, we will let L(R) denote the language associated with R.

1. is a regular expression denoting the empty set
2. ε is a regular expression denoting $\{\varepsilon\}$
3. For every $a \in \sum$, a is a regular expression denoting {a},

if r1 and r2 are regular expressions, then:

a. $L(r1 + r2) = L(r1) \cup L(r2)$
b. $L(r1.r2) = L(r1)L(r2)$
c. $L((r1))=L(r1)$
d. $L(r1^*)=(L(r1))^*$

Precedence of Regular Expressions Operators:
Like any algebra, the regular-expression operators have an assumed order of “precedence”, which means that operators are associated with the operands in a particular order. The following is the order of precedence:

1. The star operator is of the highest precedence.
2. Next comes the dot or concatenation operator
3. finally all the unions (+) are grouped with their operands.

The expression 01* +1 is grouped (0(1)*) +1

Write the language L(a*.(a+b)) in set notation

$L(a^*.(a+b)) = L(a^*)L(a+b)$
$\quad = (L(a))^*(L(a) \cup L(b))$
$\quad = \{\varepsilon, a,aa,aaa,\ldots.\}\{a,b\}$

= {a, aa,aaa, …, b, ab, aab,aaab,..}

Write a regular expression for the set of strings that consists of alternating 0's and 1's.
Let us first develop a regular expression for the language consisting of a single string 01. We can then use the star operator to get an expression of all strings of the form 0101…01
The basis rule for RE tells us that 0 and 1 are expressions denoting the languages {0} and {1}, respectively. If we concatenate the two expressions, we get a regular expression for the language {01}; this is the expression 01. Now, to get all strings consisting of zero or more occurrences of 01, we use the regular expression (01)* and get L(01)*. However this is not exactly what we want, since it only includes strings beginning with 0. To account for strings that start with 1 and strings that end with 0 or with 1: (01)* +(10)*+ 0 (10)*+ 1(01)*

Regular expressions and Regular Languages

The class R of regular languages over an alphabet ∑, has the following properties:
i) The emptyset ∅ ∈ R, ε ∈ R and if a ∈ ∑, then {a} ∈ R
(ii) If s2 and s2 ∈ R, then s1 ∪ s2, s1.s2, s1* ∈ R
(iii) Only sets formed using (i) and (ii) ∈ R

Let S and T be subsets of ∑*, if S= T* then S is generated by T.
If S is generated by T, then every element of S is the finite concatenation of elements of T.
The elements in the concatenation forming a given word are not necessarily unique.
Example:

$\Sigma = \{a, b\}$, Σ^* consists of the empty word and all possible finite strings of the symbols a and b.
$T' = \{a, ab, b\}$, then $\Sigma^* = T'^*$. However, although every string in Σ^* can be expressed uniquely as the concatenation of elements of Σ, this is not true of T', since the expression abab can be expressed as:
(a).(b)(ab), (a).(b).(a).(b) and (ab).(ab)

Let S and C be subsets of Σ^*.
if $S=C^*$, then every string in S can be expressed as the concatenation of elements of in C and we say that C is a code.
A code C is uniquely decipherable if every string in S can be uniquely expressed as the concatenation of elements of C.
Is $C = \{a,b,c,de\}$ uniquely decipherable?
same for $C = \{ba, ab, c\}$, $C = \{a, ab, bc, c\}$

Let Σ be an alphabet. A code C subset of Σ^* is called a prefix code if for all words u, v, in C, if $u=vw$ for w in Σ^*, then $u=v$ or $u=\varepsilon$.
This means that one word in a code cannot be the beginning of another word in the same code.

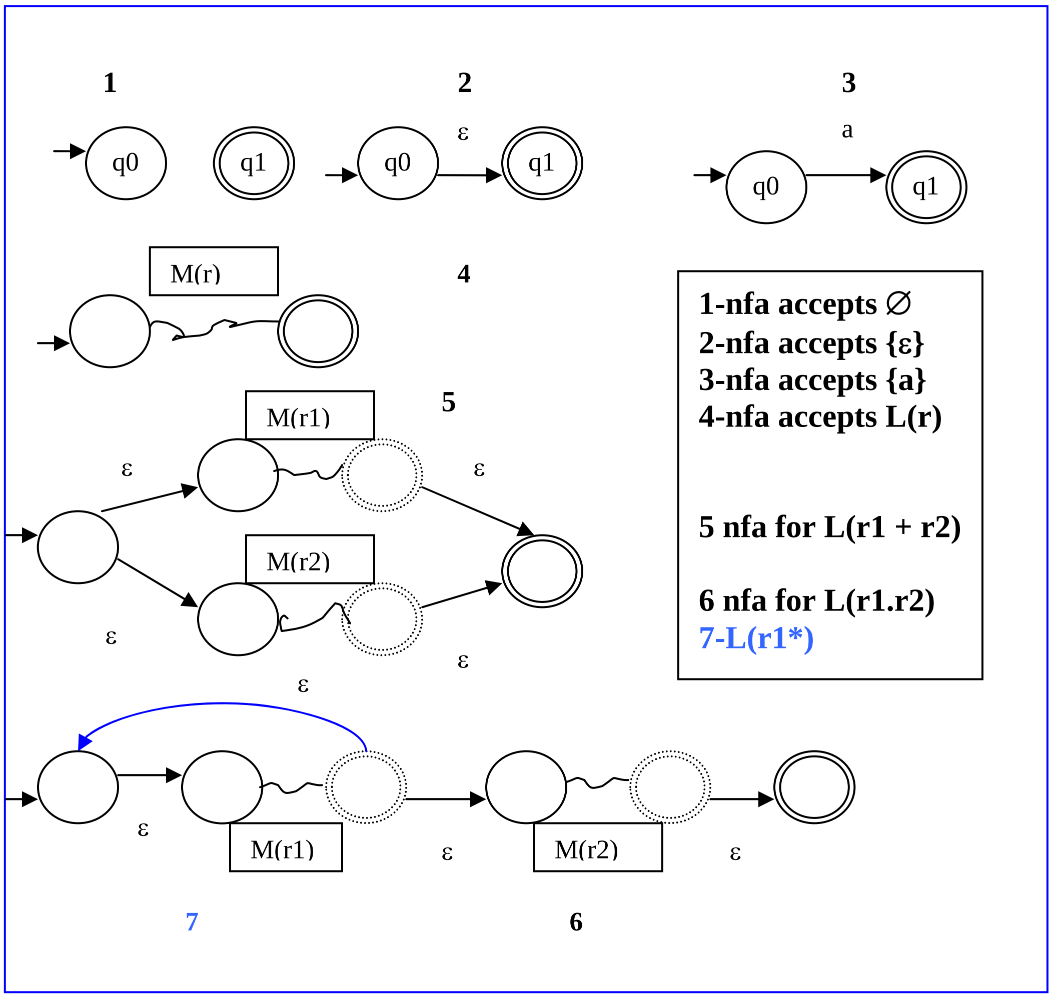
Equivalence with finite automata:

Regular expressions and finite automata are equivalent in their descriptive power. That is any regular expression can be converted into a finite automaton that recognizes the language and vice versa.

Theorem:

A language is regular if and only if some regular expression describes it.

Lemma:
If a language is described by a regular expression, then it is regular.
Proof:
We consider the six cases in the formal definition of regular expressions and build an nfa that accepts them based on the rules below:



Lemma:

If a language is regular, then it is described by a regular expression

Algorithm for converting a DFA into an equivalent regular expression:

1) Add two states to the DFA: a start state called s and an accept state called a. This defines a new automaton called G
CONVERT (G):
Let k be the number of states of G
If k=2, then G must consist of a start state, an accept state and a single arrow connecting them and labeled with a regular expression R. Return R.
If k >2, we select any state $q_{rip} \in$ Q different from s and a and let G'= (Q', $\sum$, δ',s, a), where: Q'= Q-{q_{rip}}
and for any $q_i \in$ Q'-{a} and any $q_j \in$ Q'-{s} let:

$\delta'(q_i, q_j) = (R1)(R2)^*(R3) \cup R4$ for

R1= $\delta(q_i, q_{rip})$, R2= $\delta(q_{rip}, q_{rip})$, R3= $\delta(q_{rip}, q_j)$ and
R4= $\delta(q_i, q_j)$

Compute CONVERT(G') and return this value

Simplification Rules:
Let r be the transition symbol from q_i to q_j, the following simplification rules apply:

1. $r + \emptyset = r$
2. $r . \emptyset = \emptyset$
3. $\emptyset^* = \varepsilon$