Regular Expressions

1.3

REGULAR EXPRESSIONS

In arithmetic, we can use the operations $+$ and $\times$ to build up expressions such as

$$(5 + 3) \times 4 .$$

Similarly, we can use the regular operations to build up expressions describing languages, which are called ***regular expressions***. An example is:

$$(0 \cup 1)0^* .$$

- What are values of these expressions?

Where used

- Very useful to describe a set of strings having certain patterns.
  - In UNIX, rm *.c → removes all files ending with .c
  - Lex, a tool used in compiler generators
  - grep, awk available utilities in UNIX use regular expressions.

Meaning …

0 means the language {0}

1 means {1}

$(0 \cup 1)$ means $\{0\} \cup \{1\}$

0^* means $\{0\}^*$

$(0 \cup 1)0^*$ actually is shorthand for $(0 \cup 1) \circ 0^*$

$(0 \cup 1)0^* = \{0, 1\}\{0\}^*$

Inductive Definition

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

In items 1 and 2, the regular expressions a and ε represent the languages $\{a\}$ and $\{\varepsilon\}$, respectively. In item 3, the regular expression $\emptyset$ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2, or the star of the language R_1, respectively.

Don’t confuse between null string and null set

Don’t confuse the regular expressions ε and $\emptyset$. The expression ε represents the language containing a single string—namely, the empty string—whereas $\emptyset$ represents the language that doesn’t contain any strings.

Parentheses in an expression may be omitted. If they are, evaluation is done in the precedence order: star, then concatenation, then union.

- Precedence is *, ◦, U
- So, aUb* is different from (aUb)*

  aUb* = (a U (b)*)
- aUb*c is same as aU(b*c)

- Many authors use + for U

  So, aUb = a+b But + is overloaded.

  Sipser reserved the + to mean only one thing.

+ (positive closure)

$$R^* = R^0 \cup R^1 \cup R^2 \cup \ldots \cup R^i \cup \ldots$$

$$R^+ = R^1 \cup R^2 \cup \ldots \cup R^i \cup \ldots$$

$$R^+ = RR^*$$

$$R^* = R^+ \cup \epsilon$$

- The value of a regular expression R is nothing but the language represented by R
- When we want to distinguish between r.e. and the language represented by it. We use *L(R)* to mean language represented by *R*.

We can write Σ as shorthand for regular expression $(0 \cup 1)$

From the context it should be clear for us by saying Σ do we mean alphabet or the language consisting of all possible strings of length 1.

Σ^* is a regular expression which is the language of all strings (including ϵ) over the alphabet Σ.

1. $0^*10^* = \{w | \ w \text{ contains a single } 1\}$.

2. $\Sigma^*1\Sigma^* = \{w | \ w \text{ has at least one } 1\}$.

3. $\Sigma^*001\Sigma^* = \{w | \ w \text{ contains the string } 001 \text{ as a substring}\}$.

Can you describe the language?

$$1^*(01^+)^* =$$

$$(\Sigma\Sigma)^* =$$

$$(\Sigma\Sigma\Sigma)^* =$$

$$01 \cup 10 =$$

$$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 =$$

$$(0 \cup \varepsilon)(1 \cup \varepsilon) =$$

$$1^*\emptyset =$$

$$\emptyset^* =$$

$$1^*(01^+)^* = \{w | \text{ every } 0 \text{ in } w \text{ is followed by at least one } 1\}.$$

$$(\Sigma\Sigma)^* = \{w | \ w \text{ is a string of even length}\}.$$

$$(\Sigma\Sigma\Sigma)^* = \{w | \text{ the length of } w \text{ is a multiple of } 3\}.$$

$$01 \cup 10 = \{01, 10\}.$$

$$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w | \ w \text{ starts and ends with the same symbol}\}.$$

$$(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}.$$

$$1^*\emptyset = \emptyset.$$

$$\emptyset^* = \{\varepsilon\}.$$

Understood?

If we let R be any regular expression, we have the following identities. They are good tests of whether you understand the definition.

$R \cup \emptyset = R$.
Adding the empty language to any other language will not change it.

$R \circ \varepsilon = R$.
Joining the empty string to any string will not change it.

Understood?

However, exchanging $\emptyset$ and ε in the preceding identities may cause the equalities to fail.

$R \cup \varepsilon$ may not equal R.
For example, if $R = \mathtt{0}$, then $L(R) = \{\mathtt{0}\}$ but $L(R \cup \varepsilon) = \{\mathtt{0}, \varepsilon\}$.

$R \circ \emptyset$ may not equal R.
For example, if $R = \mathtt{0}$, then $L(R) = \{\mathtt{0}\}$ but $L(R \circ \emptyset) = \emptyset$.

Compilers -- Tokens

- Lexical Analysis
  - Automatic tools can be used (like Lex)
    - But , you need to describe what you want.
- For Decimal Numbers:

$$(+ \cup - \cup \varepsilon)\,(D^+ \cup D^+ . D^* \cup D^* . D^+)$$

where $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the alphabet of decimal digits. Examples of generated strings are: 72, 3.14159, +7., and -.01.

Equivalence of RE with DFA/NFA

- It is somewhat surprising to note that, RE can be used to describe any regular language.
  - This is not true for other higher level languages, like CFL {We can describe a CFL by a CFG, not by an expression}.
- Now, how is that we prove this?

Proof has two directions

- Given RE, show that we can build a DFA/NFA recognizing the language given by the RE.
- Given DFA/NFA, show that we can convert this in to a RE.

To Show

- Given RE, show that we can build a NFA recognizing the language given by the RE.

- We use inductive definition of RE.

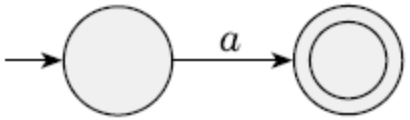
Inductive Definition of RE

DEFINITION 1.52

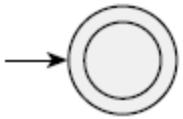
Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

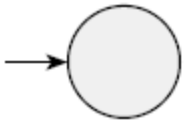
1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.



2. $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$, and the following NFA recognizes $L(R)$.



3. $R = \emptyset$. Then $L(R) = \emptyset$, and the following NFA recognizes $L(R)$.



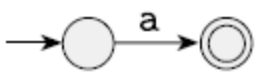
- One should be able to do these formally !!

4. $R = R_1 \cup R_2$.
5. $R = R_1 \circ R_2$.
6. $R = R_1^*$.

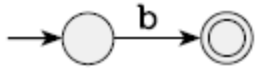
- **Use the construction proofs.**

We convert the regular expression (ab ∪ a)* to an NFA in a sequence of stages.

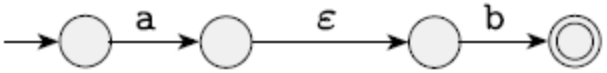
a



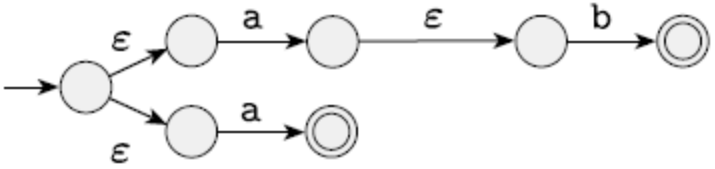
b



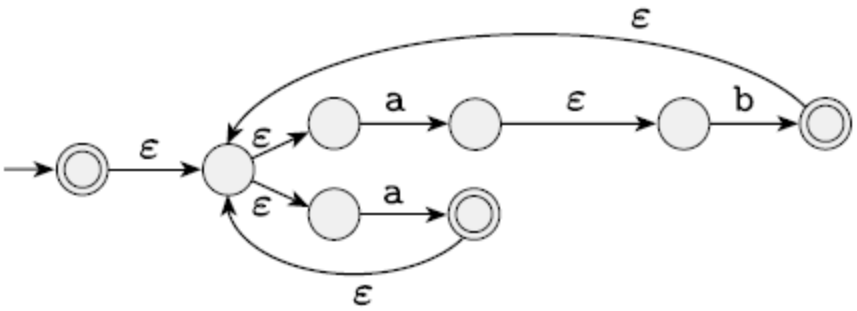
ab



ab ∪ a



(ab ∪ a)*



Equivalence between RE and DFA/NFA

- The second part is,
- Given DFA/NFA convert this to equivalent RE.

- There are various ways for this.
  - The Ullman's book gives a rigorous algorithm
  - Same thing in essence is achieved by the Sipser's book in a different way.
    - We follow Sipser's book.

DFA/NFA ➔RE

- Go for GNFA (Generalized Nondeterministic Finite Automata)
- RE can be on arrows.

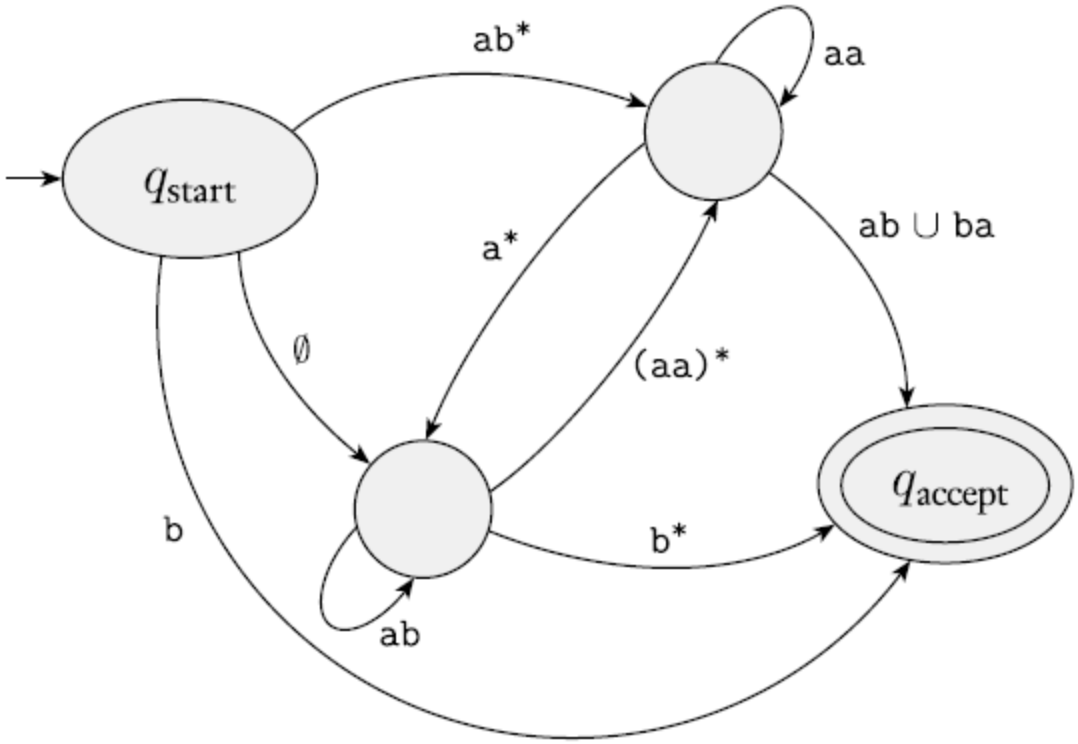


FIGURE 1.61
A generalized nondeterministic finite automaton

GNFA should -

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

What we do?

- For example we are given a 3 state DFA

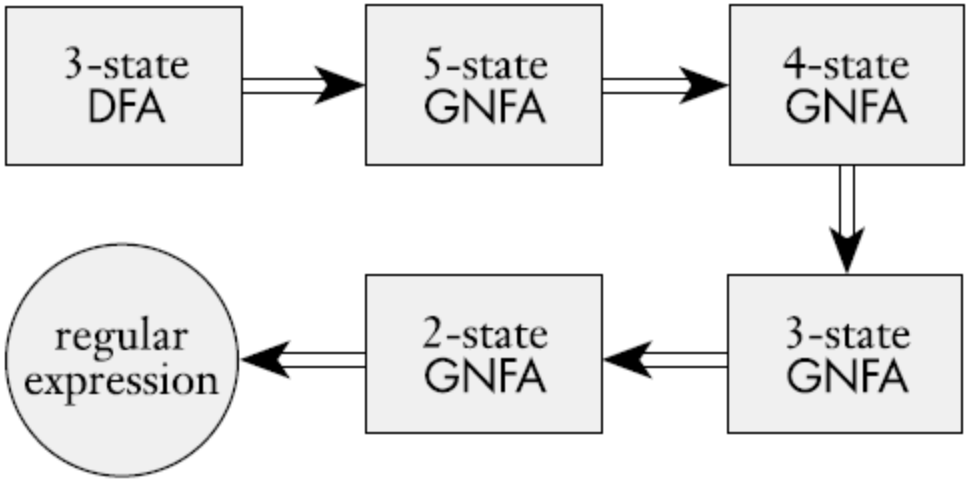


FIGURE **1.62**
Typical stages in converting a DFA to a regular expression

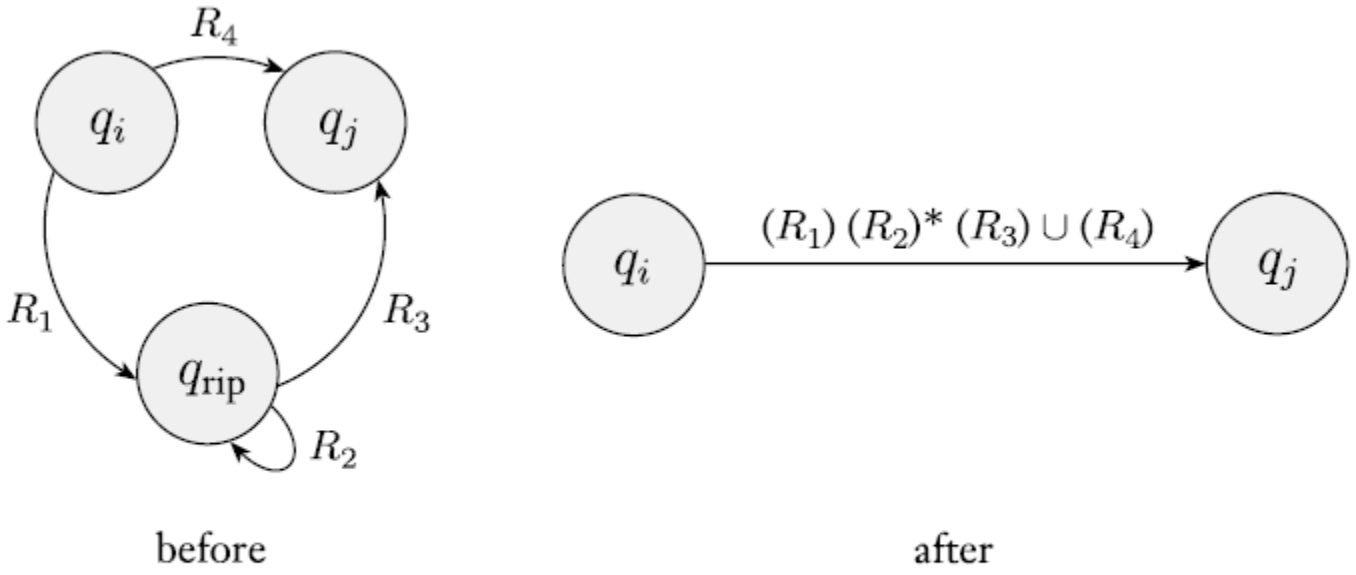
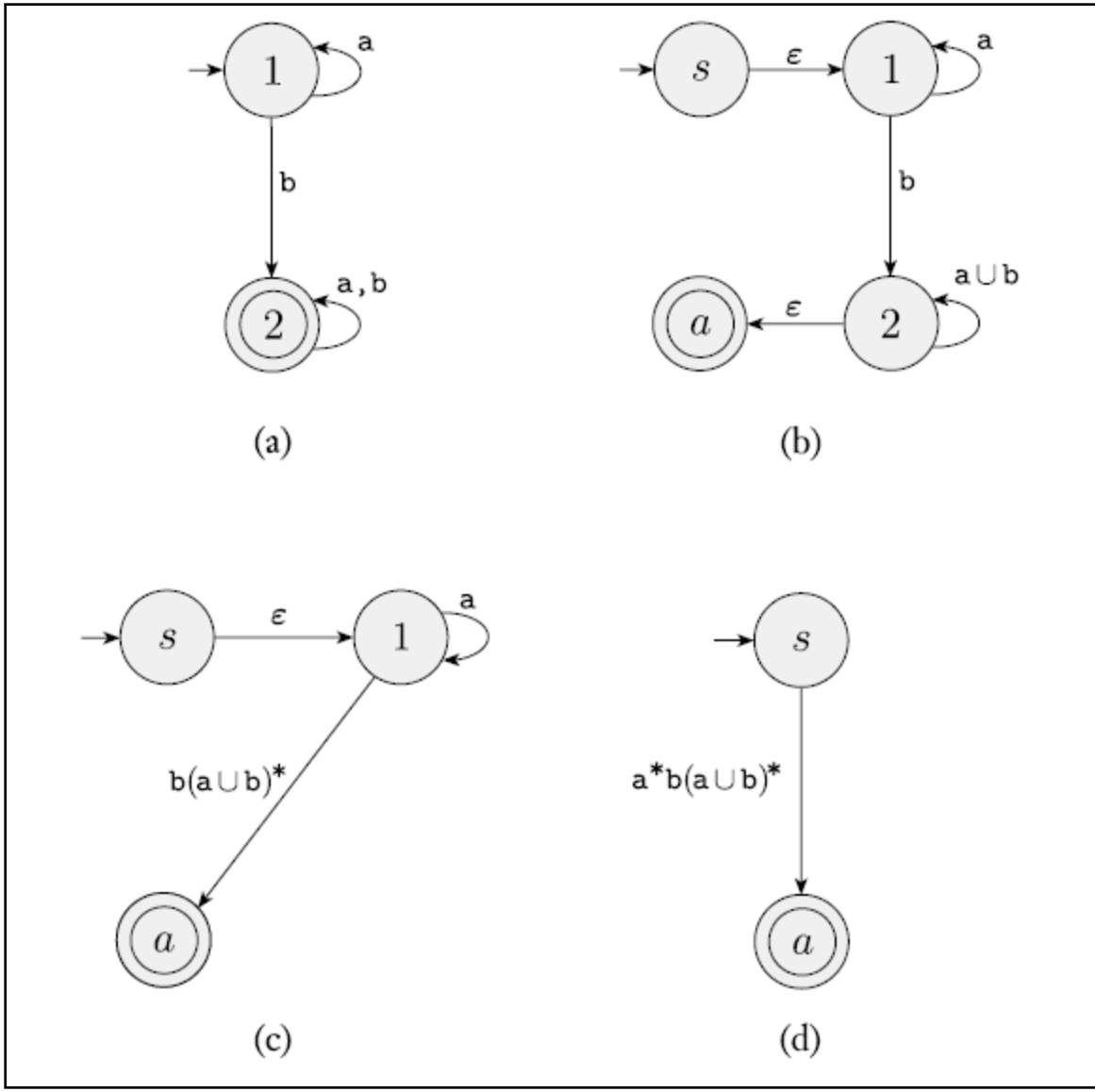


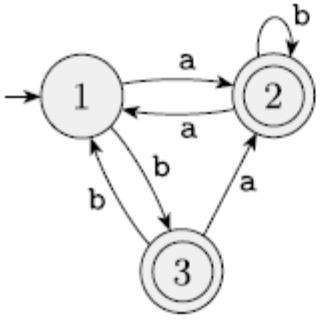
FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

Example 1

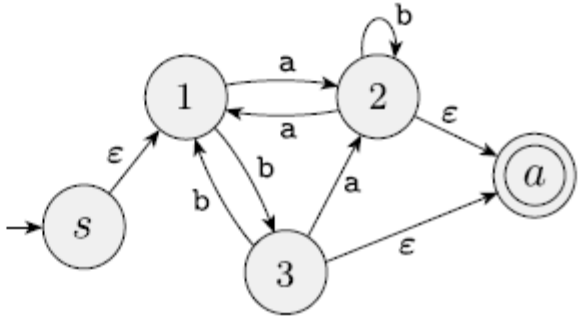


To avoid cluttering up the figure, we do not draw the arrows labeled ∅, even though they are present.

Example 2

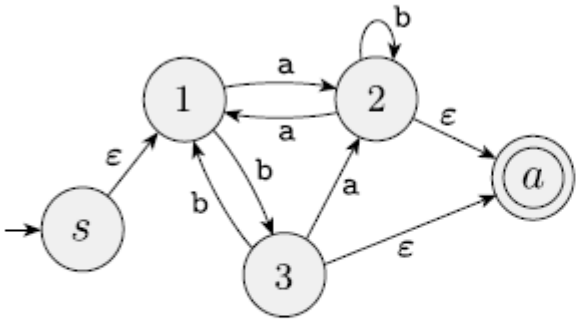


(a)

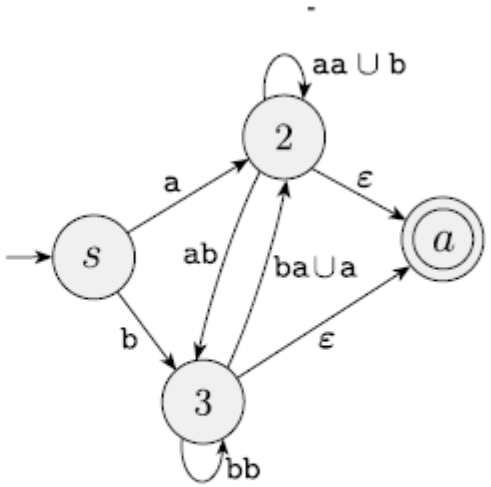


(b)

Example 2…

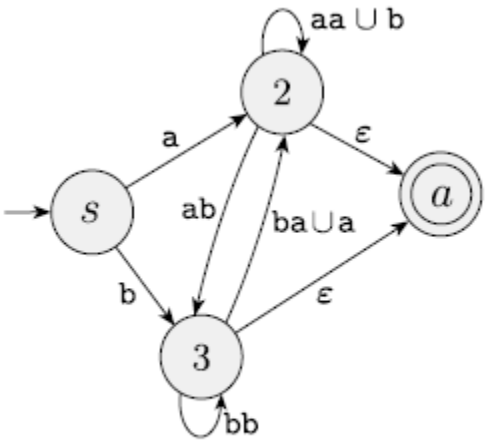


(b)

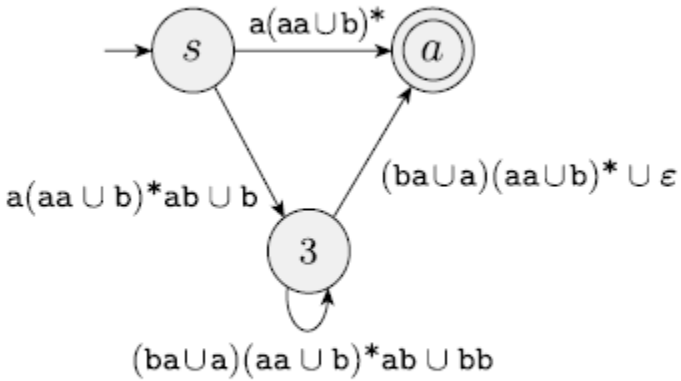


(c)

Example 2…

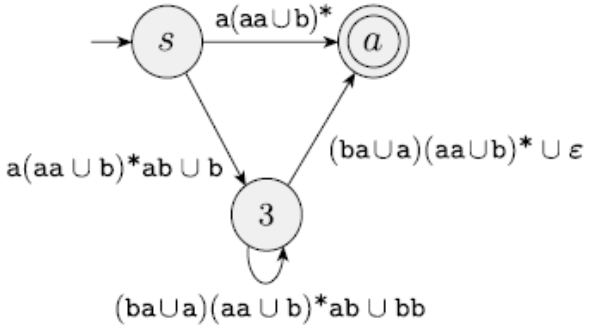


(c)



(d)

Example 2…



(d)

(a(aa∪b)*ab∪b)((ba∪a)(aa∪b)*ab∪bb)*((ba∪a)(aa∪b)* ∪ε)∪a(aa∪b)*

(e)

Laws concerning R.E.

- We overload + to mean U also.
- a+b = a U b

3.4 Algebraic Laws for Regular Expressions

- Commutativity and Associativity
  - $L + M = M + L$ (Remember, $L \cup M = M \cup L$)
  - $(L + M) + R = L + (M + R) = L + M + R$
  - $(LM)R = L(MR) = LMR$
- Left distribution and right distribution
  - $L(M + N) = LM + LN$
  - $(M + N)L = ML + NL$

Identities and Annihalors

- $\emptyset + L = L + \emptyset = L$. This law asserts that $\emptyset$ is the identity for union.
- $\epsilon L = L\epsilon = L$. This law asserts that ϵ is the identity for concatenation.
- $\emptyset L = L\emptyset = \emptyset$. This law asserts that $\emptyset$ is the annihilator for concatenation.

Idempotent Laws

- L+L = L
- (L*)* = L

How to prove?

- Inequality can be easily proved by a counter example.
- But, equality, to be proved is cumbersome.
  - You can follow your set theory knowledge to deduct that from LHS, RHS is deductible.
  - These is an established simple way of doing this.

Assuming only three regular operators, viz., +, . * are only used.

- To test whether E = F is true or false.

1. Convert E and F to concrete regular expressions C and D, respectively, by replacing each variable by a concrete symbol.
2. Test whether $L(C) = L(D)$. If so, then $E = F$ is a true law, and if not, then the "law" is false.

- What do you mean by concrete r.e. ?

Concretizing a r.e.

Let $E = P + Q(RS^*)$ be a regular expression where P, Q, R, S are some regular expressions.

Here, we say E has variables P, Q, R, S.

Concretizing E means replacing each variable in E by a distinct symbol.

In this example, we can concretize $P + Q(RS^*)$ to $a + b(cd^*)$

To prove, $P(M+N) = PM + PN$.

Concretizing L.H.S will give us $a(b+c)$

Concretizing R.H.S will give us $ab + ac$

Now, one has to show $L\big(a(b+c)\big) = L(ab+ac)$, which can be done easily.

Now, verify whether $PR = RP$ is true or false.

Concretize L.H.S in to ab

Concretize R.H.S in to ba

Now, it is clear that $L(ab) = \{ab\}$ is not equal to $L(ba) = \{ba\}$.

! **Exercise 3.4.2:** Prove or disprove each of the following statements about regular expressions.

* a) $(R+S)^* = R^* + S^*$.

b) $(RS+R)^*R = R(SR+R)^*$.

* c) $(RS+R)^*RS = (RR^*S)^*$.

d) $(R+S)^*S = (R^*S)^*$.

e) $S(RS+S)^*R = RR^*S(RR^*S)^*$.

Don’t use operators beyond $+ \cdot *$

- That is, stick to, regular operators only.

Extensions of the Test Beyond Regular Expressions May Fail

Consider the “law” $L \cap M \cap N = L \cap M$;

Concretizing, we get, $\{a\} \cap \{b\} \cap \{c\} = \{a\} \cap \{b\}$. This is true.

But clearly the above "law" is false.

Counter example to disprove the "law".

For example, let $L = M = \{a\}$ and $N = \emptyset$.

Clearly L.H.S and R.H.S are distinct languages.