REGULAR EXPRESSION

Dr. Nadeem Akhtar
Assistant Professor
Department of Computer Science & IT
The Islamia University of Bahawalpur

PhD – Formal methods in Software engineering
IRISA – University of South Brittany – FRANCE.

STRINGS AND LANGUAGES (1)

An **alphabet** is any finite set of symbols.

Examples of symbols: letters, digits, and punctuation.

The set {0, 1} is the binary alphabet

ASCII (example of alphabet)

Unicode (100,000 characters from alphabets of different languages around the world).

STRINGS AND LANGUAGES (2)

String:

- A string over an alphabet is a finite sequence of symbols drawn from that alphabet.
- “sentence” and “word” synonyms for “string”.

STRINGS AND LANGUAGES (3)

Language:

- Any countable set of strings over some fixed alphabet
- Examples:
  - Set of well-defined C programs
  - Set of all grammatically correct English sentences
- Also **abstract languages** like Ø , the empty set or {ϵ}, the set containing only the empty string, are languages under this definition.

STRINGS AND LANGUAGES (4)

Concatenation:

x and y are strings, concatenation of x and y denoted xy, is the string formed by appending y to x.

Example: x=dog, y=house then xy=doghouse

Empty String is the identity under concatenation that is for any string s

$\epsilon s = s\epsilon = s$

STRINGS AND LANGUAGES (5)

- **Exponentiation**

  S^0 to be ϵ

  For all $i>0$, S^i is $S^{i-1}S$

  since $\epsilon s = s$, it follows that $S^1=S$

  then $S^2=SS$, and $S^3=SSS$, and so on

STRINGS AND LANGUAGES (6)

prefix: of string S is any string obtained by removing zero or more symbols from the end of S.

Example: ban, banana, and ϵ are prefixes of banana.

suffix: of String S is any String obtained by removing zero or more symbols from the beginning of S.

Example: nana, banana, and ϵ are suffixes of banana.

Substring: of string S is obtained by deleting any prefix and any suffix from S.

Example: banana, nan, and ϵare substrings of banana.

STRINGS AND LANGUAGES (7)

- **Proper prefixes, suffixes and substrings** of a string S are those, prefixes, suffixes, and substrings of S that are not ϵ or not equal to S itself.
- **Subsequence** of S is any string formed by deleting zero or more not necessarily consecutive positions of S. Example: baan is a substring of banana

Operations on Languages (1)

Union, Concatenation, Closure

Union of L and M

LUM = {s | s is in L or s is in M}

Concatenation of L and M

LM = {st | s is in L and t is in M}

Kleene closure of L

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Set of strings you get by concatenating L zero or more times

L^0 "Concatenation of L zero times" and L^i is $L^{i-1}L$

Operations on Languages (2)

- **Positive closure of L**

$$L^{+} = \bigcup_{i=1}^{\infty} L^{i}$$

(same as Kleene closure without the term L^0)

Є will not be in L^+ unless it is L itself

OPERATIONS ON LANGUAGES (3)

- **Example**

  L = {A, B, … ,Z, a, b, …, z} (52)

  D = {0, 1,…,9} (10)

(1) LUD is the set of letters and digits – Language of 62 strings of length one, each of which strings is either one letter or one digit.

(2) LD is the set of 520 strings of length two, each consists of one letter followed by one digit.

(3) L^4 is the set of of all 4-letter strings.

(4) L* is the set of all strings of letters, including **Є**, the empty string.

(5) L(LUD)* is the set of all strings of letters and digits beginning with a letter.

(6) D^+ is the set of all strings of one or more digits.

REGULAR EXPRESSION (1)

- An algebraic notation that is compact and easy for humans to use and understand.

-Regular expressions that describe simple sets of strings can be combined to form regular expressions that describe more complex sets of Strings.

e.g. (*r*, *s*, and *t*) in italics used for regular expression.

Regular expressions built recursively out of smaller regular expressions. Each regular expression r denotes a language L(r), which is also defined recursively from the languages denoted by r's sub-expressions.

REGULAR EXPRESSION (2)

Regular operations are used to build regular expressions. These regular expressions describe regular languages.

Example:

(0 U 1)0*

The value of the above Regular Expression is a language. In this case the value is the language consisting of all strings starting with a 0 or a 1 followed by any number of 0s.

REGULAR EXPRESSION (3)

Example:

Another example of a regular expression is

(0 U 1)*

It starts with the language (0 U 1) and applies the * operation. The value of this expression is the language consisting of all possible strings of Os and 1s.

If ∑ = {0, 1}, we can write as shorthand for the regular expression (0 U 1).

REGULAR EXPRESSION (4)

Example:

More generally, if ∑ is any alphabet, the regular expression ∑ describes the language consisting of all strings of length 1 over this alphabet, and ∑* describes the language consisting of all strings over that alphabet.

Similarly ∑*1 is the language that contains all strings that end in a 1.

The language (O ∑ *) U (∑ *1) consists of all strings that either start with a 0 or end with a 1.

REGULAR EXPRESSION (5)

- In arithmetic, we say that x has precedence over + to mean that, when there is a choice, we do the x operation first. Thus in 2 + 3 x 4 the 3 x 4 is done before the addition. To have the addition done first we must add parentheses to obtain (2 + 3) x 4.

- In regular expressions, the star operation is done first, followed by concatenation, and finally union, unless parentheses are used to change the usual order.

FORMAL DEFINITION OF A REGULAR EXPRESSION (1)

Say that R is a regular expression if R is

1) a for some a in the alphabet,

2) ϵ ,

3) Ø,

4) (R_1 U R_2), where R_1 and R_2 are regular expressions,

5) (R_1 o 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 e represent the languages {a} and {ϵ }, respectively.

In item 3, the regular expression 0 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.

FORMAL DEFINITION OF A REGULAR EXPRESSION (2)

The expression ϵ represents the language containing a single string-namely, the empty string-

whereas Ø represents the language that doesn't contain any strings.

FORMAL DEFINITION OF A REGULAR EXPRESSION (3)

- Regular expressions do not have a **Circular definition** i.e. A regular expression are not defined in terms of itself.

- Regular expressions have **Inductive definition**. R_1 and R_2 always are smaller than R. Regular expressions are defined in terms of smaller regular expressions. A definition of this type is called an **inductive definition**.

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

Formal definition of a Regular Expression (4)

- $R^+ = RR^*$ (R^+ be shorthand for RR^*)

- In other words, whereas R* has all strings that are 0 or more concatenations of strings from R,

  The language R^+ has all strings that that are 1 or more concatenations of strings from R.

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

FORMAL DEFINITION OF A REGULAR EXPRESSION (5)

R^k be shorthand for the concatenation of k R's with each other.

When we want to distinguish between a regular expression R and the language that it describes, we write L(R) to be the language of R.

FORMAL DEFINITION OF A REGULAR EXPRESSION (6)

If we let R be any regular expression, we have the following identities.

$R \cup \emptyset = R$.
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.

FORMAL DEFINITION OF A REGULAR EXPRESSION (7)

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 $R = 0$, then $L(R) = \{0\}$ but $L(R \cup \varepsilon) = \{0, \varepsilon\}$.

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

Regular Expression and Design of Compilers

Regular expressions are useful tools in the design of compilers for programming languages. Elemental objects in a programming language, called ***tokens***, such as the variable names and constants, may be described with regular expressions. For example, a numerical constant that may include a fractional part and/or a sign may be described as a member of the language

$$\left(+ \cup - \cup \varepsilon\right)\left(D^{+} \cup D^{+} . D^{*} \cup D^{*} . D^{+}\right)$$

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 .

Once the syntax of the tokens of the programming language have been described with regular expressions, automatic systems can generate the ***lexical analyzer***, the part of a compiler that initially processes the input program.

REGULAR EXPRESSION (1)

- **BASIS**

(1) Є is a regex, and L(Є) is {Є}, that is, the languages whose sole member is the empty string.

(2) If a is the symbol in ∑, then a is a regex and L(a) = {a}, that is, the language with one string, of length one, with a in its one position.

REGULAR EXPRESSION (2)

- **INDUCTION:** Suppose r and s are regular expressions denoting languages L(r) and L(s) respectively

(1) (r) | (s) is a regex denoting the languages L(r) U L(s).

(2) (r)(s) is a regex denoting the language L(r) L(s).

(3) (r)* is a regex denoting (L(r))*.

(4) (r) is a regex deboting L(r).

We can add additional pairs of parenthesis around the expressions without changing the language they denote.

REGULAR EXPRESSION (3)

- **PRECEDENCE:**

(1) The unary operator * and + has the highest precedence and is left associative

(2) Concatenation has second highest precedence and is left associative

(3) + (Union) has lowest precedence and is left associative

REGULAR EXPRESSION (4)

Example:

Let ∑ = {a, b}

(1) The regex a |b denotes the language {a, b}

(2) (a|b)(a|b) denotes {aa, ab, ba, bb}

aa | ab | ba | bb regex of the same language.

(3) a* denotes the language consisting of all strings of zero or more a's, that is {Є, a, aa, aaa, aaaa, … }

(4) (a|b)* denotes the set of all strings consisting of zero or more instances of a or b, that is, all strings of a's and b's: {Є, a, b, aa, ab, ba, bb, aaa, … }

(5) a|a*b denotes the language {a, b, ab, aab, aaab, …} that is, the string a and all strings consisting of zero or more a's and ending in b.

Regular Expression (5)

Regular expression	Language (set of strings)	Informal description
a	{“a”}	The set consisting of the one-letter string “a”.
ε	{“”}	The set containing the empty string.
$s\|t$	$L(s) \cup L(t)$	Strings from both languages
st	$\{vw \mid v \in L(s), w \in L(t)\}$	Strings constructed by concatenating a string from the first language with a string from the second language. Note: In set-formulas, “|” reads as “where”.
$s*$	$\{\text{“”}\} \cup \{vw \mid v \in L(s), w \in L(s*)\}$	Each string in the language is a concatenation of any number of strings in the language of *s*.

REGULAR EXPRESSION (6)

- *st (pronounced "s t") describes the concatenation of the languages L(s) and L(t),* i.e., the sets of strings obtained by taking a string from L(*s) and putting this in* front of a string from L(*t). For example, if L(s) is {"a", "b"} and L(t) is {"c",* "d"}, then L(*st) is the set {"ac", "ad", "bc", "bd"}.*

REGULAR EXPRESSION (7)

- The language for *s∗ (pronounced "s star") is described recursively: It consists of* the empty string plus whatever can be obtained by concatenating a string from L(*s) to a string from L(s∗). This is equivalent to saying that L(s∗) consists of* strings that can be obtained by concatenating zero or more (possibly different) strings from L(*s). If, for example, L(s) is {"a", "b"} then L(s∗) is {"", "a", "b",* "aa", "ab", "ba", "bb", "aaa", . . . }, i.e., any string (including the empty) that consists entirely of as and bs.

REGULAR EXPRESSION (8)

- **Note** that while we use the same notation for concrete strings and regular expressions denoting one-string languages, the context will make it clear which is meant.
- We will often show strings and sets of strings without using quotation marks, e.g., write {a, bb} instead of {“a”, “bb”}. When doing so, we will use *ε to denote the* empty string, so the example from L(*s∗) above is written as {ε, a, b, aa, ab, ba,* bb, aaa, . . . }. The letters *u, v and w in italics will be used to denote unspecified* single strings, i.e., members of some language.

As an example, ab*w denotes any* string starting with ab.

REGULAR EXPRESSION (9)

- Shorthands

non-negative integer constants

(0|1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)∗

- Sequences of letters within square brackets represent the set of these letters. For example, we use [ab01] as a shorthand for a|b|0|1.
- Interval notation to abbreviate [0123456789] to [0–9].
- Write [a–zA–Z] to denote all alphabetic letters in both lower and upper case.

- integer constants above, we can write this much shorter as [0–9][0–9]∗

- *s∗ denotes zero or more occurrences of s*
- *s+* denotes *one or more occurrences of s*
- With this notation, we can abbreviate our description of integers to [0–9]+

REGULAR EXPRESSION (10)

- it is common that we can have zero or one occurrence of something (e.g., an optional sign to a number)

s? for zero or more occurrence of s (i.e. s+ε)

+ and ? bind with the same precedence as ∗.

Algebraic properties of regular expressions

- Algebraic properties of regular expressions

$(r\|s)\|t = r\|s\|t = r\|(s\|t)$	| is associative
$s\|t = t\|s$	| is commutative
$s\|s = s$	| is idempotent
$s? = s\|\varepsilon$	by definition
$(rs)t = rst = r(st)$	concatenation is associative
$s\varepsilon = s = \varepsilon s$	*ε is a neutral element for concatenation*
$r(s\|t) = rs\|rt$	concatenation distributes over |
$(r\|s)t = rt\|st$	concatenation distributes over |
$(s*)* = s*$	$*$ is idempotent.
$s*s* = s*$	0 or more twice is still 0 or more
$ss* = s^+ = s*s$	by definition

EXAMPLES (1)

In the following instances we assume that the alphabet ∑ is {O, 1 }.

1) $0^*10^* = \{w \mid w \text{ contains a single 1}\}$.
2) $\sum^*1\sum^* = \{w \mid w \text{ has at least one 1}\}$.
3) $\sum^*001\sum^* =$

   $\{w \mid w \text{ contains the string 001 as substring}\}$.

EXAMPLES (2)

4) $1 * (01^{+})^{*}$ =

{w| every 0 in w is followed by at least one 1}.

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

6)$(\sum\sum\sum)^{*}$ = {w | the length of w is a multiple of three}.

EXAMPLES (3)

7) 01 U 10 = {01, 10}.

8) 0∑*0 U 1∑*1 U 0 U 1 = {w|w starts and ends with the same symbol}.

EXAMPLES (4)

9) (0U ϵ)1* = 01* U 1*.

The expression 0 U ϵ describes the language {O, ϵ}, so the concatenation operation adds either 0 or ϵ before every string in 1 * .

10) (0 U ϵ) (1 U ϵ) = {ϵ, 0, 1, 01 }.

11) 1* Ø = Ø.

Concatenating the empty set to any set yields the empty set.

EXAMPLES (5)

- $\emptyset^* = \{\epsilon\}$.

The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

EXAMPLES (6)

Let R be any regular expression, we have the following identities.

$R \cup \emptyset = R$.

Adding the empty language to any other language will not change it.

$R . \in = R$.

Joining the empty string to any string will not change it.

EXAMPLES (7)

- RU Є may not equal R.

For example, if R = 0, then L(R) = {0} but
L(R U Є) = {0, Є }.

- R. Ø may not equal R.

For example, if R = 0, then L(R) = {0} but

L(R. Ø) = Ø.

Question - Exercise

- Describe the languages denoted by the following Regular expression: (Note: the alphabet ∑ is {a, b})

(1) a(a | b)*a

(2) ((Є | a)b*)*

(3) (a | b)*a(a | b)(a | b)

(4) a*ba*ba*ba*

Regular Expression (1)

Example

Variable names. In the programming language C, a variable name consists of letters, digits and the underscore symbol and it must begin with a letter or underscore. This can be described by the regular expression [a–zA–Z_][a–zA–Z_0–9]*

Example

An integer constant is an optional sign followed by a non-empty sequence of digits: [+-]?[0–9]+

REGULAR EXPRESSION (2)

Example

- A floating-point constant can have an optional sign. After this, the mantissa part is described as a sequence of digits followed by a decimal point and then another sequence of digits. Either one (but not both) of the digit sequences can be empty. Finally, there is an optional exponent part, which is the letter e (in upper or lower case) followed by an (optionally signed) integer constant. If there is an exponent part to the constant, the mantissa part can be written as an integer constant i.e., without the decimal point). Some examples:

  3.14 -3. .23 3e+4 11.22e-3.

REGULAR EXPRESSION (3)

3.14 -3. .23 3e+4 11.22e-3.

Format can be described by the following regular expression:

[+-]? *((([0–9]+. [0–9]∗|. [0–9]+)([eE][+-]?[0–9]+)?)* |[0–9]+[eE][+-]?[0–9]+*)*

- This regular expression is complicated by the fact that the exponent is optional if the mantissa contains a decimal point, but not if it does not (as that would make the number an integer constant). We can make the description simpler if we make the regular expression for floats also include integers, and instead use other means of distinguishing integers from floats. If we do this, the regular expression can be simplified to

[+-]?*(([0–9]+(. [0–9]∗)?|. [0–9]+)([eE][+-]?[0–9]+)?)*

REGULAR EXPRESSION (4)

- **Example:**

  The set of valid C identifiers.

  letter_ stand for any letter or the underscore

  digit_ stand for any digit

  letter_(letter_ + digit)*

Plus + used means Union

Parenthesis are used to group sub-expressions

Star means “zero or more occurrences of”

REGULAR EXPRESSION (5)

Regular Definitions

If ∑ is an alphabet of basic symbols, then a regular definition is a sequence of definitions of the form:

d1 → r1

d2 → r2

d3 → r3

…..

dn → rn

where

(1) Each d*i* is a new symbol, not in ∑ and not the same as any other of the d's and

(2) Each r*i* is a regex, over the alphabet ∑ U {d1, d2, d3, …, di-1} .

By restricting ri to ∑ and previously defined d's, we avoid recursive definitions, we construct a regex over ∑ alone for each r*i*.

REGULAR EXPRESSION (6)

- **Example:** C identifiers are strings of letters, digits, and underscores

  letter_ → A|B|,…,|Z|a|b|,…,|z|_

  digit → 0|1|,…,|9

  id → letter_ (letter_ | digit)*

REGULAR EXPRESSION (7)

- **Example:** Unsigned numbers (integers or floating point) e.g. 5280, 0.01234, 6.336E4, or 1.89E-4

digit → 0|1|,…,|9

digits → digit digit*

optionalFraction → .digits | Є

optionalExponent → (Є(+|-|Є) digits) | Є

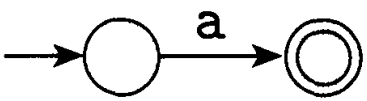
number → digits optionalFraction optionalExponent

QUESTION-1 (1/3)

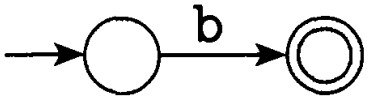
Convert the regular expression (ab U a) * to an NFA?

The sequence of stages are:

a

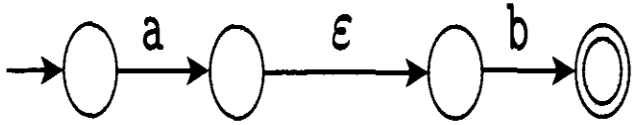


b

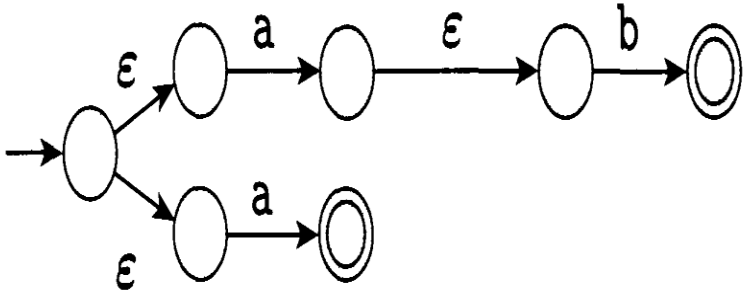


QUESTION-1 (2/3)

ab

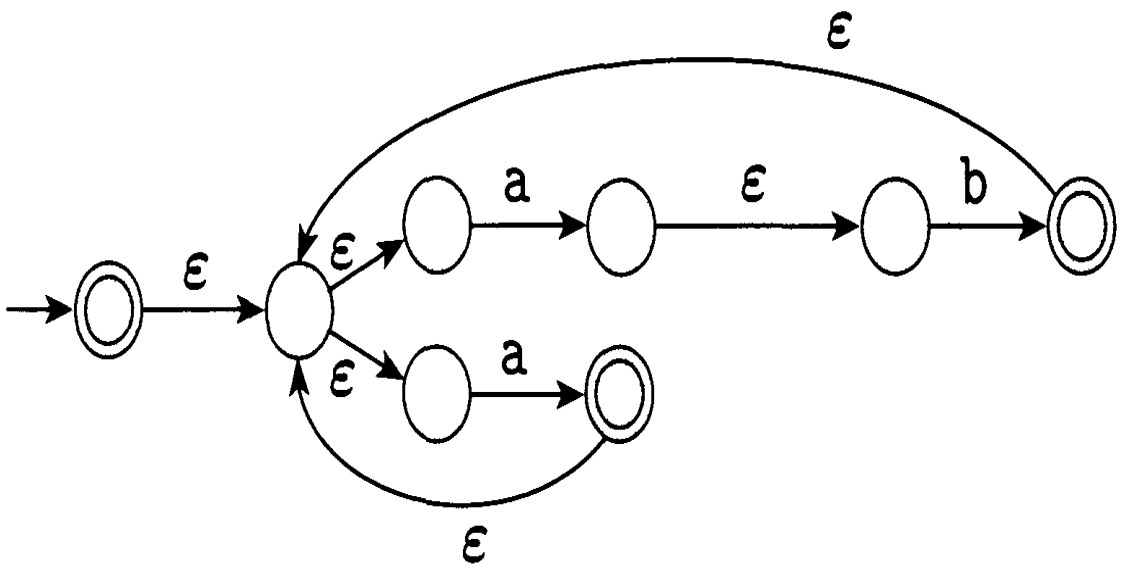


ab ∪ a



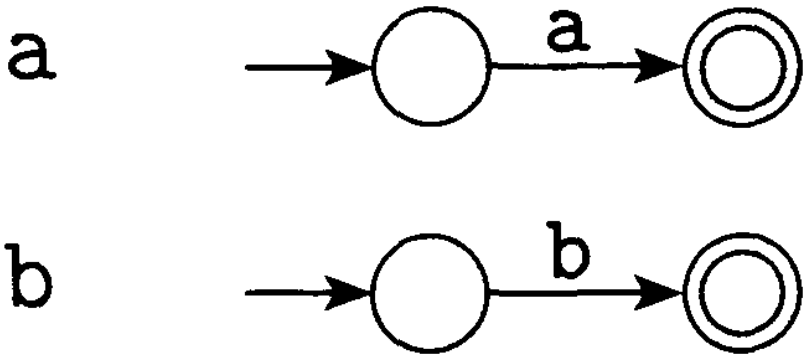
QUESTION-1 (3/3)

$(ab \cup a)^*$



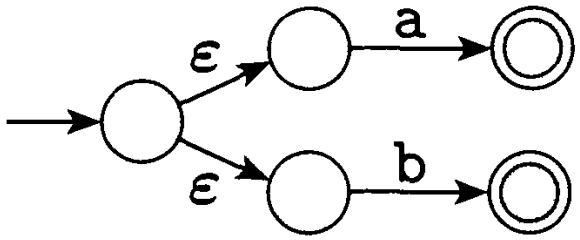
QUESTION-2 (1/3)

- Convert the regular expression (aUb)*aba to an NFA?

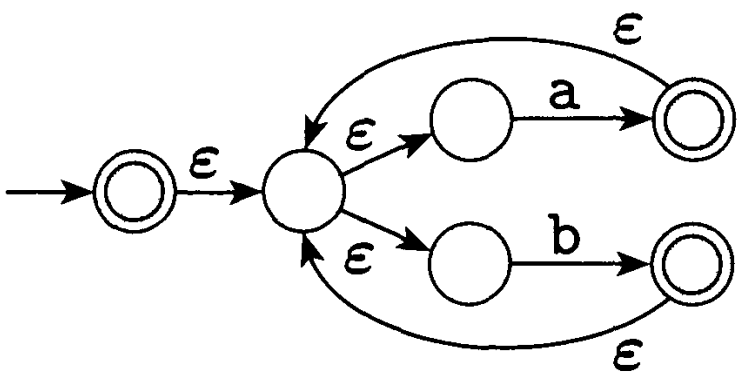


QUESTION-2 (2/3)

a∪b

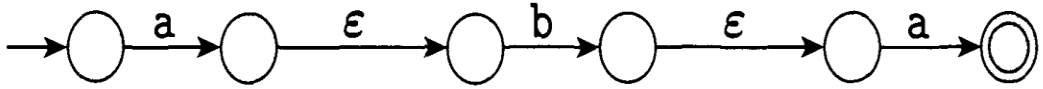


(a∪b)*



QUESTION-2 (3/3)

aba



(a∪b)*aba