University of sulaimani
College of science
Department of Computer Science

Computation

Regular Expression

Lecture four

Mzhda Hiwa Hama
2024-2025

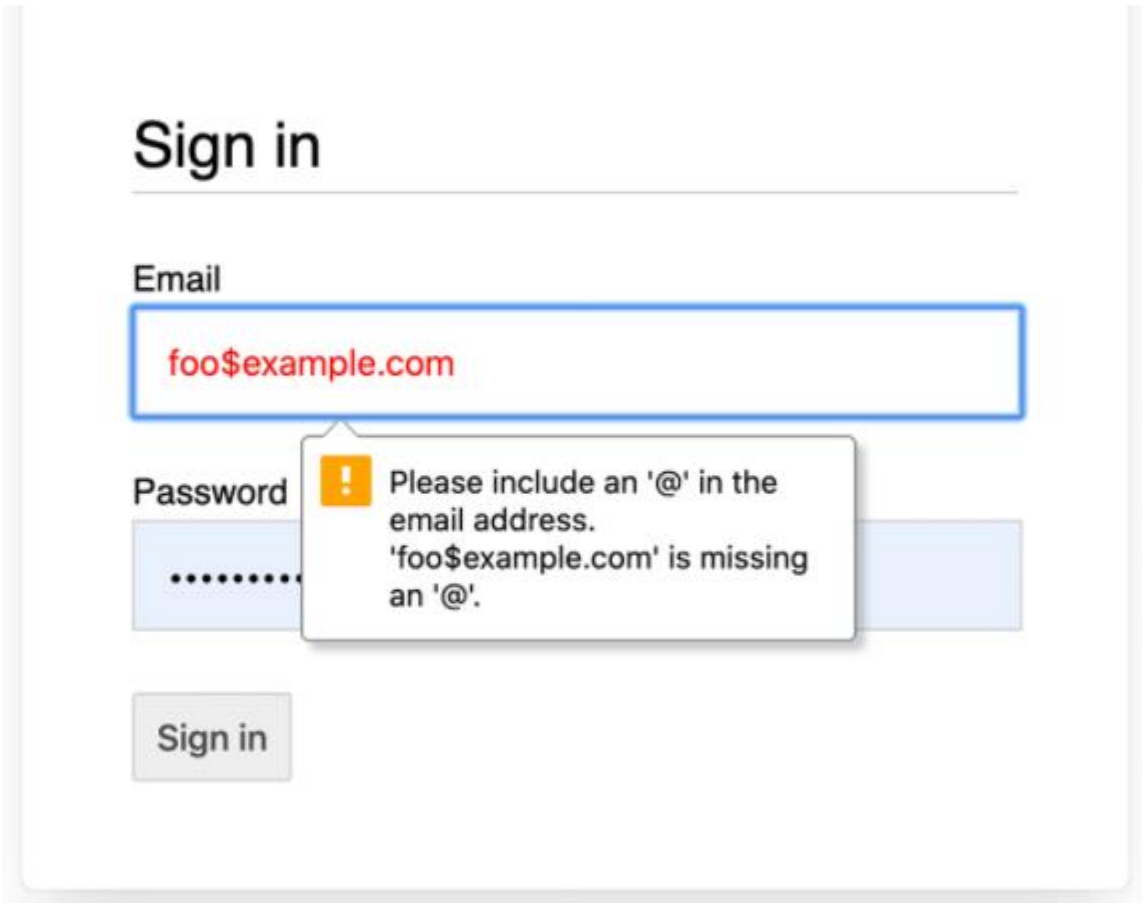
Regular Expression

- Regular Expressions (shortened as "regex") are used in many programming languages and tools. They can be used in finding and extracting patterns in texts and programs.

- Regular expressions are a way to search for substrings ("matches") in strings. This is done by searching with "patterns" through the string.

- 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.

- Using regular expressions, we can also specify and validate forms of data such as passwords, e-mail addresses, user IDs, etc.



Regular Expression's metacharacters

[ ]	A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches "a", "b", or "c". [a-z] specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: [abcx-z] matches "a", "b", "c", "x", "y", or "z“.
.	Matches any single character. Within bracket expressions, the dot character matches a literal dot. For example, a.c matches "abc", etc., but [a.c] matches only "a", ".", or "c".
[^]	Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than "a", "b", or "c". [^a-z] matches any single character that is not a lowercase letter from "a" to "z".

Regular Expression's metacharacters

( )	Defines a marked sub expression. The string matched within the parentheses can be recalled later . A marked subexpression is also called a block or capturing group. (abc)
*	Matches the preceding element zero or more times. For example, ab*c matches "ac", "abc", "abbbc", etc. [xyz]* matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. (ab)* matches "", "ab", "abab", "ababab", and so on.
?	Matches the preceding element zero or one time. For example, ab?c matches only "ac" or "abc".
+ على الأقل يكون موجود مرة	Matches the preceding element one or more times. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac".
|	The choice (also known as alternation or set union) operator matches either the expression before or the expression after the operator. For example, abc|def matches "abc" or "def".

Regular Expression’s metacharacters

^	Matches the starting position within the string.
$	Matches the ending position of the string or the position just before a string-ending newline.
{n}	Matches Exactly the specified number of occurrences, a{3} contains {aaa} , exactly three a

{m,n}	Matches the preceding element at least m and not more than n times. For example, a{3,5} matches only "aaa", "aaaa", and "aaaaa".

Formal Definition of Regular Expression

Say that R is a regular expression if R is

1. a for some a in the alphabet Σ, so a represented as {a}
2. ε, represent {ε} language.
3. ∅, represent { } empty language.

Note: Don't confuse the regular expressions ε and ∅. The expression ε represents the language containing a single string—namely, the empty string—whereas ∅ represents the language that doesn't contain any strings.

Regular Expression Operation

1. **Union(OR) :** where R1 and R2 are regular expressions, then (R1 U R2), also written as(R1 | R2 or R1 + R2) is also a regular expression. L(R1|R2) = L(R1) U L(R2).

2. **Concatenation:** (R1 ∘ R2), where R1 and R2 are regular expressions then R1R2 (also written as R1.R2) is also a regular expression. L(R1R2) = L(R1) concatenated with L(R2).

3. **Kleene closure(star):** (R1*), where R1 is a regular expression then R1* (the Kleene closure of R1) is also a regular expression. L(R1*) = epsilon U L(R1) U L(R1R1) U L(R1R1R1) U…

Regular Expression and languages

- The origins of regular expressions lie in Automata Theory and Formal Language Theory.
- We can use RE to identify Regular Languages.
- So, The value of regular expression is a language.
- Regular language is one accepted by some FA or described by an RE.

Note

- In arithmetic, we can use the operations + and × 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: $(0 \cup 1)0*$. The value of the arithmetic expression is the number 32. The value of a regular expression is a language.

In arithmetic, we say that × has precedence over + to mean that when there is a choice, we do the × operation first. Thus in 2+3×4, the 3×4 is done before the addition. To have the addition done first, we must add parentheses to obtain (2 + 3)×4. In regular expressions, the star operation is done first, followed by concatenation, and finally union, unless parentheses change the usual order.

اذا كان فيه

$(a+b) = (a|b) = (a \cup b)$

Examples

- In the following instances, we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w$ contains a single 1$\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w$ has at least one 1$\}$. $\Sigma^* = (0+1)^*$

   لو كان 5 فهذا راح يعني 5 مرات ناخذ في كل مرة لو صفر لو واحد

3. $\Sigma^*001\Sigma^* = \{w \mid w$ contains the string 001 as a substring$\}$.
4. $(\Sigma\Sigma)^* = \{w \mid w$ is a string of even length$\}$.
5. $(\Sigma\Sigma\Sigma)^* = \{w \mid$ the length of w is a multiple of 3$\}$.
6. $01 \cup 10 = \{01, 10\}$.
7. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
8. $1^*\emptyset = \emptyset$. Concatenating the empty set to any set yields the empty set.

$(0|1)^*$
$(0+1)^*$

9. (0 u 1)* Consists of all possible strings of 0s and 1s

| = or

10. (0∑*) u (∑*1) Consists of all strings that start with 0 or end with 1.

11. The set of strings over {0,1} that end in 3 consecutive 1's.

    (0 | 1)* 111

12. The set of strings over {0,1} that have at most one 1

    0* | 0* 1 0*

Homework

- Write a regular expressions for each of the following languages:

1. {w| w starts with a 0 or a 1 and followed by any number of 0s} $0(0)^* \mid 1(0)^*$ $(0+1)0^*$
2. {w| w contains the string 101 as a substring} $(0|1)^*101(0|1)^*$
3. {w| w starts with the string 11 and ends with 10} $11(0+1)^*10$
4. Start and end with same symbol. $0(0|1)^*0 \cup 1(0|1)^*1$
5. {w| w contains at least three 1s}

$(0+1)^*1(0+1)^*1(0+1)^*1(0+1)^*$

Equivalence with Finite Automata

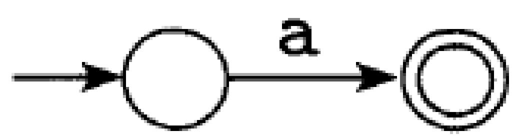
- Every regular language is FA recognizable, ie. Any RE can be converted into Finite Automata that recognizes the language it describes, and vice versa. Recall that a regular language is one that is recognized by some finite automaton.

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

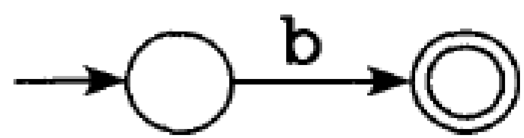
Example1

- We convert the regular expression **(abUa)*** to an NFA in a sequence of stages. We build up from the smallest subexpressions to larger subexpressions until we have an NFA for the original expression, as shown in the following diagram.

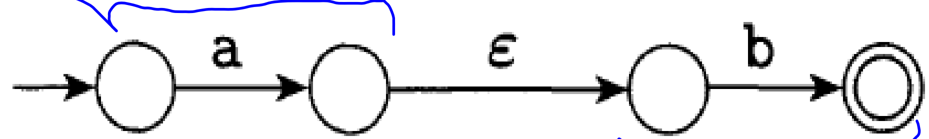
a



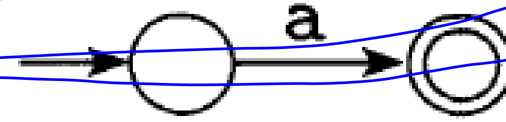
b



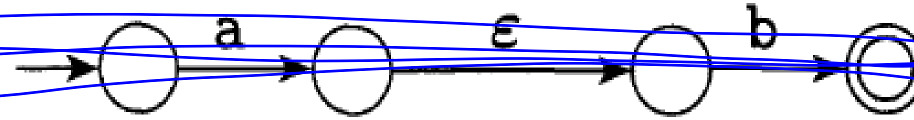
ab



a



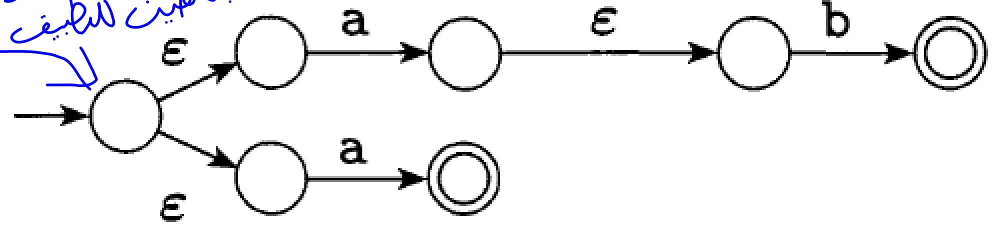
ab



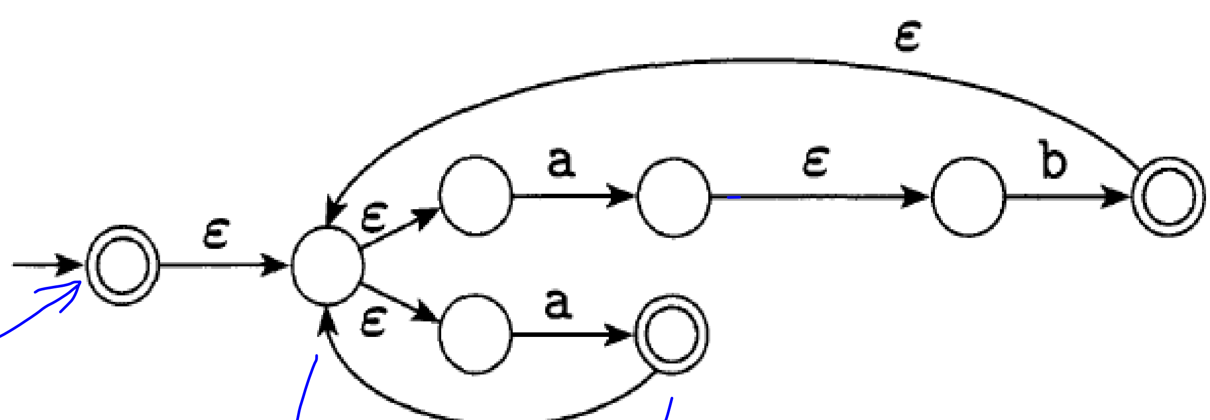
OR

لذلك راح يكون فيه
اتجاهين للتطبيق

ab ∪ a



$(ab \cup a)^*$



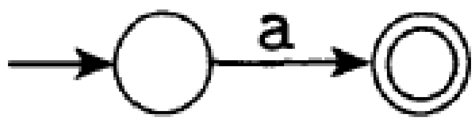
لازم يرجع لهذا وليس الى

Example 2

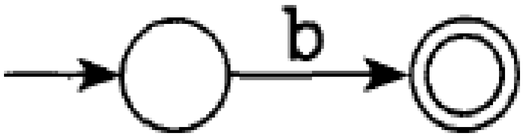
- (a u b)* aba

concatination بيضم

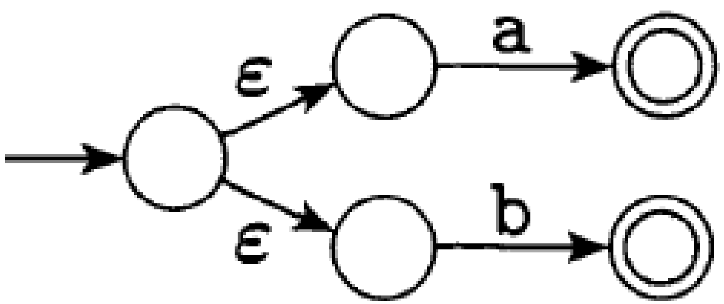
a



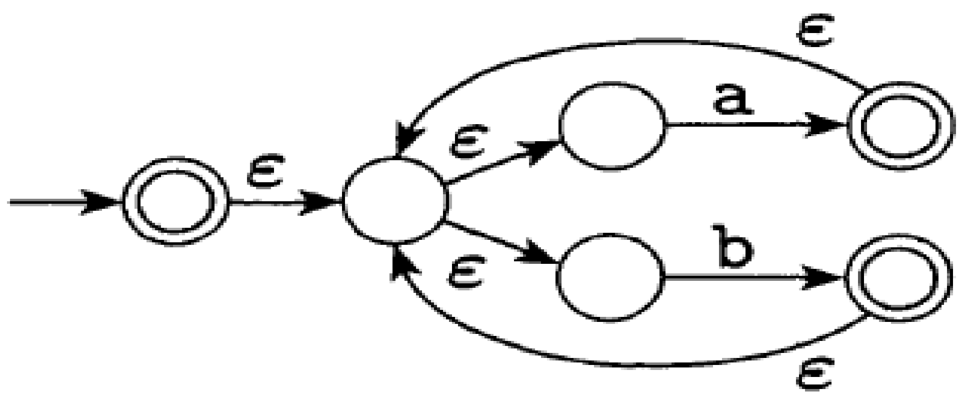
b



a∪b

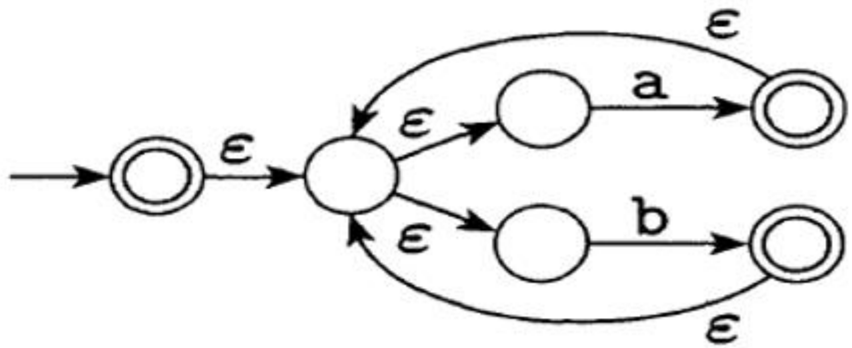


$(a \cup b)^*$

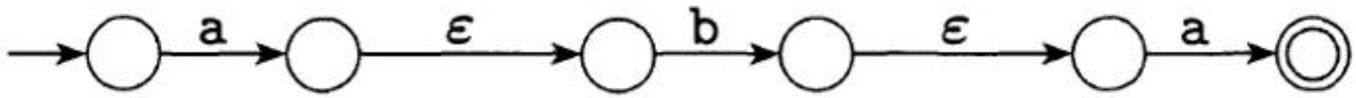


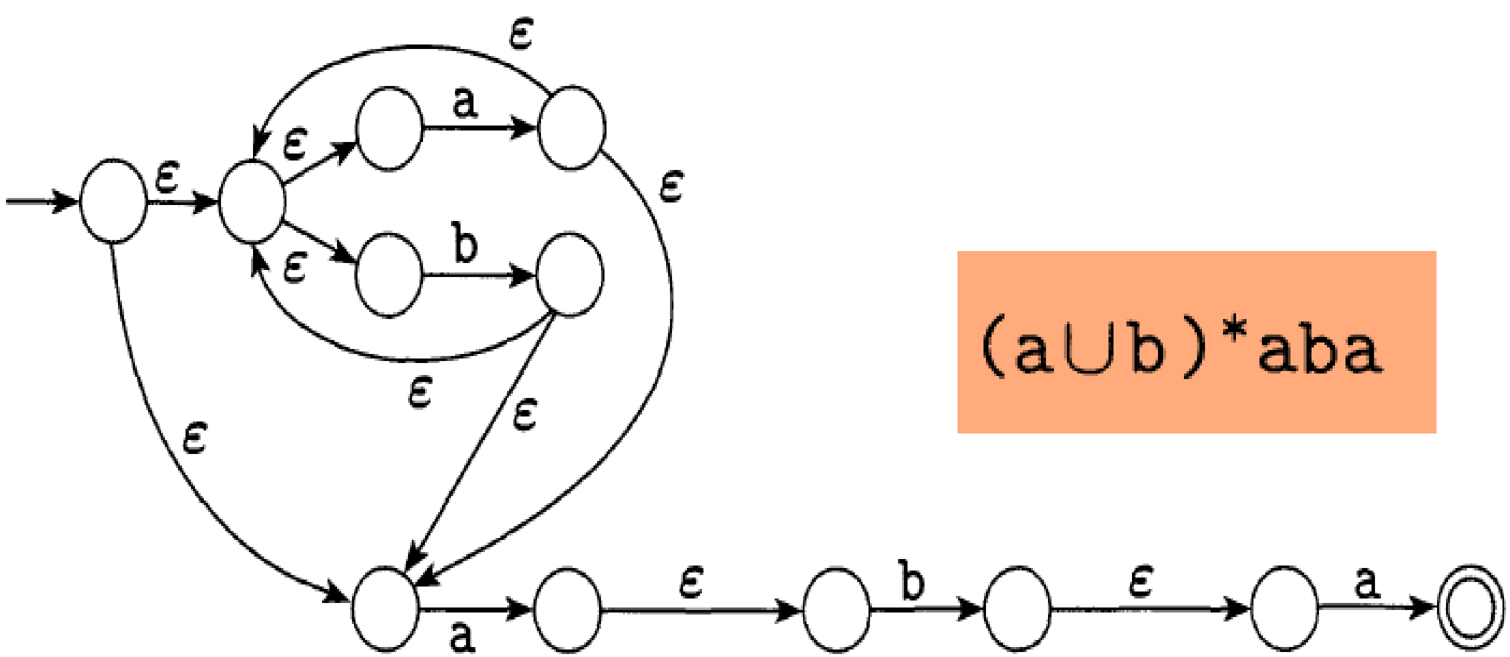
aba

(a∪b)*



aba





(a∪b)*aba

Look Ahead and Look Behind collectively called "lookaround"

You can have assertions in your pattern like look*ahead* or *behind* to ensure that a substring does or does not occur. These “look around” assertions are specified by putting the substring checked for in a string, whose leading characters are:

مهم

- ?= (for positive lookahead),
- ?! (negative lookahead),

البداية

- ?<= (positive lookbehind),
- ?<! (negative lookbehind).

النهاية

Look Ahead and Look Behind…cont'd

- Use ?! (for negative lookahead), if the query was to avoid appearing a specific substring in a string. At the beginning of the string

- Ex: ^(?!101)[01]* // Doesn't have 101 at beginning of the string.

Look Ahead and Look Behind...cont'd

- Use ?= (for positive lookahead), if the query required appearing a specific substring in a string. At the beginning of the string

Ex: ^(?=101)[01]* // String must contain 101 at beginning of the string.

Look Ahead and Look Behind…cont’d

- Use **?<!** (for negative lookbehind), if the query was to avoid appearing a specific substring only at the end of the string

➢ Ex: **^[01]*(?<!101)$** // Doesn’t end with 101

- Use **?<=** (for positive lookbehind), if the query required appearing a specific substring only at the end of the string

➢ Ex: **^[01]*(?<=101)$** // must end with 101

- Note: always specify the end position with $ when using lookbehind.