Theory of Computation

النظرية الاحتسابية
المحاضرة الاولى

كلية التربية للعلوم الصرفة / جامعة ديالى

اعداد
م.د. محمد سامي محمد

قسم علوم الحاسوب – المرحلة الثانية

BACKGROUND

We are concerned with the **Theory of Computers**, which means that we form several abstract mathematical models that will describe with varying degrees of accuracy parts of computers and types of computers and similar machines.

Our models will not be used to discuss the practical engineering details of the hardware of computers, but the more abstract questions of the **frontiers of capability of these mechanical devices**.

There are separate courses that deal with circuits and switching theory (computer logic) and with instruction sets and register arrangements (computer architecture) and with data structures and algorithms and operating systems and compiler design and artificial intelligence and so forth.

BACKGROUND

All of these courses have a theoretical component, but they differ from our study in two basic ways.

First, they deal only with computers that already exist; our models, on the other hand, will encompass all computers that do exist, will exist, and that can ever be dreamed of.

Second, they are interested in how best to do things; we shall not be interested in optimality at all, but rather we shall be concerned with the question of possibility- what can and what cannot be done.

what can and what cannot be done.

LANGUAGES

In English we distinguish the three different entities: **letters**, **words**, and **sentences**.

Not all collections of letters form a valid word, and not all collections of words form a valid sentence.

The analogy can be continued. Certain groups of sentences make up coherent paragraphs, certain groups of paragraphs make up coherent stories, and so on.

This situation also exists with computer languages. Certain character strings are recognizable words (GOTO, END . . .). Certain strings of words are recognizable commands. Certain sets of commands become a program (with or without data).

LANGUAGES

To construct a general theory that unifies all these examples, it is necessary for us to adopt a definition of a "**most universal language structure**," that is, a structure in which the decision of whether a given string of units constitutes a valid larger unit is not a matter of guesswork but is based on explicitly stated rules.

When we call our study the **Theory of Formal Languages**, the word "**formal**" refers to the fact that all the rules for the language are explicitly stated in terms of what strings of symbols can occur.

We begin with only one finite set of fundamental units out of which we build structures. We shall call this the alphabet.

A certain specified set of strings of characters from the alphabet will be called the language.

Those strings that are permissible in the language we call words.

LANGUAGES AND SETS

The symbols in the alphabet do not have to be Latin letters, and the sole universal requirement for a possible string is that it have only finitely many symbols in it.

The question of what it means to "specify" a set of strings is one we discuss presently.

We shall wish to allow a string to have no letters. This we call the empty string or null string, and we shall denote it by the symbol Λ.

No matter what language we are considering, the null string is always Λ.

Two words are considered the same if all their letters are the same and in the same order so there is only one possible word of no letters.

For clarity, we do not allow the symbol Λ to be part of the alphabet for any language.

LANGUAGES AND SETS

The most familiar example of a language for us is English. The alphabet is the usual set of letters plus the apostrophe and hyphen. Let us denote the whole alphabet by the Greek letter capital sigma.

$$\Sigma = \{a\ b\ c\ d\ e\ \ldots\ z, -\}$$

Sometimes we shall list a set of elements separated by spaces and sometimes by commas.

We can now specify which strings of these letters are valid words in our language by listing them all, as is done in a dictionary. It is a long list, but a finite list, and it makes a perfectly good definition of the language. If we call this language ENGLISH-WORDS we may write ENGLISH-WORDS = {all the words (main entries) in a standard dictionary}

In the line above, we have intentionally mixed mathematical notation (the equal sign, the braces denoting sets) and a prose phrase. This results in Perfectly understandable communication; we take this liberty throughout. All of our investigations will be agglomerates of informal discussion and precise symbolism.

LANGUAGES AND SETS

Of course, the language ENGLISH-WORDS, as we have specified it, does not have any grammar. If we wish to make a formal definition of the language of the sentences in English, we must begin by saying that this time our basic *alphabet* is the entries in the dictionary. Let us call this alphabet Γ, the capital gamma.

Γ= **{** the entries in a standard dictionary, plus a blank space, plus the usual punctuation marks **}**

In order to specify which strings of elements from Γ produce valid words in the language ENGLISH-SENTENCES, we must rely on the grammatical rules of English. This is because we could never produce a complete list of all possible words in this language; that would have to be a list of all valid English sentences. Theoretically, there are infinitely many different words in the language ENGLISH-SENTENCES. For example:

ate one apple.
I ate two apples.
I ate three apples.

LANGUAGES AND SETS

The trick of defining the language ENGLISH-SENTENCES by listing all the rules of English grammar allows us to give a finite description of an infinite language.

If we go by the rules of grammar only, many strings of alphabet letters seem to be valid words, for example, "I ate three Tuesdays." In a *formal* language we must allow this string. It is *grammatically* correct; only its *mean*ing reveals that it is ridiculous. Meaning is something we do not refer to in formal languages.

In general, the abstract languages we treat will be defined in one of two ways. Either they will be presented as an alphabet and the exhaustive list of all valid words, or else they will be presented as an alphabet and a set of rules defining the acceptable words.

LANGUAGES AND SETS

Earlier we mentioned that we could define a language by presenting the alphabet and then *specifying* which strings are words. The word "specify" is trickier than we may at first suppose.

Consider this example of the language called MY-PET. The alphabet for this language is

{a c d g o t}

There is only one word in this language, and for our own perverse reasons we wish to *specify* it by this sentence:

If the Earth and the Moon ever collide, then
MY-PET = { *cat* }
but, if the Earth and the Moon never collide, then
MY-PET = { *dog* }

LANGUAGES AND SETS

One or the other of these two events will occur, but at this point in the history of the universe it is impossible to be certain whether the word *dog* is or is not in the language MY-PET.

This sentence is not an adequate specification of the language MY-PET because it is not useful. To be an acceptable specification of a language, a set of rules must enable us to decide, in a finite amount of time, whether a given string of alphabet letters is or is not a word in the language.

The set of rules can be of two kinds. They can either tell us how to test a string of alphabet letters that we might be presented with, to see if it is a valid word; or they can tell us how to construct all the words in the language by some clear procedures.

LANGUAGES AND SETS

Let us consider some simple examples of languages. If we start with an alphabet having only one letter, the letter x,

$$\Sigma = \{ x \}$$

we can define a language by saying that any nonempty string of alphabet characters is a word.

$$L1 = \{x\ xx\ xxx\ xxxx \ldots\}$$

or to write this in an alternate form:

$$L1 = \{\ x^n \text{ for } n = 1\ 2\ 3\ \ldots \}$$

LANGUAGES AND SETS

In this example, when we concatenate the word xxx with the word xx, we obtain the word *xxxxx*. The words in this language are clearly analogous to the positive integers, and the operation of concatenation is analogous to addition:

$$x^n \textbf{ concatenated with } x^m \text{ is the word } x^{n+m}$$

It will often be convenient for us to designate the words in a given language by new symbols, that is, other than the ones in the alphabet.

For example, we could say that the word xxx is called a and that the word xx is b. Then to denote the word formed by concatenating a and b we write the letters side by side:

$$ab = xxxxx$$

LANGUAGES AND SETS

It is not always true that when two words are concatenated they produce another word in the language. For example if the language is

$$L2 = \{ x\ xxx\ xxxxx\ xxxxXX... = \{ x^{odd} \}$$

$$= \{ x^{n+1} \text{ forn=0 1 2 3 ... } \}$$

then a = xxx and b xxxxx are both words in L2, but their concatenation *ab* = xxxxxxxx is not in L2

Notice that the alphabet for L2 is the same as the alphabet for L1.

In these simple examples, when we concatenate a with b we get the same word as when we concatenate b with a. We can depict this by writing:

$$ab = ba$$

LANGUAGES AND SETS

But this relationship does not hold for all languages. In English when we concatenate "house" and "boat" we get "houseboat," which is indeed a word but distinct from "boathouse," which is a different thing-not because they have different meanings but because they are different words.

EXAMPLE Consider another language. Let us begin with the alphabet:

$$\Sigma=\{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\}$$

and define the set of words:
L3={any finite string of alphabet letters that does not start with the letter zero }

This language L 3 then looks like the set of all positive integers written in base 10.

$$L3 =\{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ .\ \}$$

LANGUAGES AND SETS

We say "looks like" instead of "is" because L 3 is only a formal collection of strings of symbols. The integers have other mathematical properties.

DEFINITION
We define the function "**length of a string**" to be the number of letters in the string. We write this function using the word "**length**."

For example, if a = xxxx in the language *L1* above, then length(a) = 4

If *c* = 428 in the language L3, then length(c) = 3

Or we could write directly that in L, length(xxxx) = 4 and in L3

length(428) = 3 In any language that includes the empty string Λ we have: length(Λ) = 0

For any word w in any language, if length(w) = 0 then w = Λ.

LANGUAGES AND SETS

We can now present yet another definition of *L3*.
L3 = {any finite string of alphabet letters that, if it has length more than one, does not start with a zero}

This is not necessarily a better definition of *L3* , but it does illustrate that there are often different ways of specifying the same language.

On the other hand, we may wish to define a language like *L1* but that does contain Λ.

$$L1 = \{\Lambda \;\; x \; xx \; xxx \; xxxx \ldots\}$$
$$= \{x^n \text{ forn } = \mathbf{0}\ 1\ 2\ 3 \ldots\}$$

LANGUAGES AND SETS

DEFINITION

Let us introduce the function reverse. If a is a word in some language L, then **reverse** (a) is the same string of letters spelled backward, called the reverse of a, even if this backward string is not a word in L.

EXAMPLE

reverse(xxx) = *xxx*
reverse(xxxxx) = *xxxxx*
reverse(145) = *541*
But let us also note that in L3
reverse(140) = *041*
which is not a word in L3.

LANGUAGES AND SETS

DEFINITION

Let us define a new language called **PALINDROME** over the alphabet

$$\Sigma = \{a, b\}$$

PALINDROME { Λ, and all strings x such that reverse(x) = x **}** If we begin listing the elements in PALINDROME we find

PALINDROME = {Λ, a, *b*, aa, *bb*, aaa, *aba*, *bab*, *bbb*, *aaaa*, *abba* **...** }

Sometimes when we concatenate two words in PALINDROME we obtain another word in PALINDROME such as when *abba* is concatenated with ab*baabba*. More often, the concatenation is not itself a word in PALINDROME, as when aa is concatenated with aba.

LANGUAGES AND SETS

DEFINITION

Given an alphabet Σ, we wish to define a language in which any string of letters from Σ is a word, even the null string. This language we shall call the **closure** of the alphabet. It is denoted by writing a star (an asterisk) after the name of the alphabet as a superscript Σ^* This notation is sometimes known as the **Kleene star** after the logician who was one of the founders of this subject.

EXAMPLE

If $\Sigma = \{x\}$, then
$\Sigma * = L4 = \{\Lambda$ x xx xxx... $\}$

EXAMPLE

If $\Sigma = \{0,1\}$, then
$\Sigma * = \{\Lambda$ 0 1 *00 01* 10 11 000 *001*...$\}$

LANGUAGES AND SETS

EXAMPLE

If $\Sigma = \{a,b,c\}$, then

$$\Sigma * = \{\Lambda\ a\ b\ c\ aa\ ab\ ac\ ba\ bb\ bc\ ca\ cb\ cc\ aaa...\}$$

Notice that when we wrote out the first several words in the language we put them in size order (**words of shortest length first**) and then listed all the words of the same length alphabetically.

LANGUAGES AND SETS

DEFINITION

If S is a set of words, then by S* we mean the set of all finite strings formed by concatenating words from S, where any word may be used as often as we like, and where the null string is also included.

EXAMPLE

If S = *{aa,b}*, then

*S**= {*Λ* plus any word composed of factors of aa and b }
{*Λ* plus all strings of a's and b's in which the a's occur in even clumps }
= {*Λ b* aa *bb aab baa bbb* aaaa *aabb baab bbaa bbbb*
aaaab aabaa aabbb baaaa baabb bbaab bbbaa bbbbb ... }

The string aabaaab is not in S* since it has a clump of a's of length 3. The phrase "clump of a's" has not been precisely defined, but we know what it means anyway.

LANGUAGES AND SETS

EXAMPLE

Let S = **{** a, ab **}.** Then

S* = {Λ plus any word composed of factors of a and *ab}*
= {Λ plus all strings of a's and b's except those that start with b and those that contain a double *b}*
={*Λ* a aa ab aaa aab aaaa *aaab aaba abaa abab* aaaaa aaaab aaaba aabaa aabab abaaa abaab *ababa*... }

By the phrase "double b" we mean the substring bb. For each word in ***S**** every b must have an a immediately to its left. The substring bb is impossible, as is starting with a b. Any string without the substring bb that begins with an a can be factored into terms of *(ab)* and (a).

LANGUAGES AND SETS

To prove that a certain word is in the closure language S*, we must show how it can be written as a **concatenate** of words from the base set S. In the last example, to show that *abaab* is in ***S**** we can factor it as follows:

(ab)(a)(ab)

These three factors are all in the set **S;** therefore their concatenation is in ***S***.* This is the only way to factor this string into factors of (a) and (ab). When this happens, we say that the factoring **is unique**. Sometimes the factoring is **not unique**. For example, consider S = *{xx,* xxx}. Then:

S^* = {Λ and all strings of more than one x }
=$\{x^n$ *for* n=0, 2, 3, 4, 5...}
={Λ xx xxx xxxx xxxxx xxxxxx.. . }

Notice that the word x is not in the language S*. The string xxxxxxx is in this closure for any of these three reasons. It is: (xx) (xx) (xxx) or (xx) (xxx) (xx) or (xxx) (xx) (xx) Also, x^6 is either x^2x^2 x^2 or else x^3 x^3. It is important to note here that the parentheses, **( ), are not letters in the alphabet but are used for the sole purpose** of demarcating the ends of factors. So we can write xxxxx = (xx)(xxx).

LANGUAGES AND SETS

In cases where **parentheses** are letters of the alphabet

If Σ= {x ( ) }, then

length(xxxxx) = 5

but length(*(xx)(xxx)*) = 9

Let us suppose that we wanted to prove mathematically that this set S* contains all x^n for $n \neq 1$. Suppose that somebody did not believe this and needed convincing. We could proceed as follows. First, we consider the possibility that there were some powers of x that we could not produce by concatenating factors of *(xx)* and *(xxx)*. Obviously, since we can produce x^4 , x^5, x^6, the examples of strings that we cannot produce must be large. Let us ask the question, "What is the smallest power of x (larger than 1) that we cannot form out of factors of xx and xxx?" Let us suppose that we start making a list of how to construct the various powers of x. On this list we write down how to form x^2, x^3, x^4, x^4, and so on. Let us say that we work our way successfully up to x^{373}, but then we cannot figure out how to form x^{374} . We become stuck, so a friend comes over to us and says, "Let me see your list. How did you form the word x^{372}?.

LANGUAGES AND SETS

Why don't you just concatenate another factor of xx in front of this and then you will have the word x^{374} that you wanted." Our friend is right, and this story shows that while writing this list out we can never really become stuck. This discussion can easily be generalized into a mathematical proof of the fact that S* contains all powers of x greater than 1.

We have just established a mathematical fact by a method of proof that we have rarely seen in other courses. It is a proof based on showing that something exists (the factoring) because we can describe how to create it (by adding xx to a previous case). What we have described **can be formalized into an algorithm for producing** all the powers of x from the factors xx and xxx. The method is to begin with xx and *xxx* and, when we want to produce x^n, we take the sequence of concatenations that we have already found will producx^{n-2}, and we concatenate *xx* on to that.

LANGUAGES AND SETS

The method of proving that something exists by showing how to create it is called **proof by constructive algorithm**. This is the most important tool in our whole study. We may have a difficult time selling powers of x broken into factors of xx and xxx. Let us observe that if the alphabet has no letters, then its closure is the language with the null string as its only word Symbolically, we write:

If $\Sigma = \{\emptyset\}$ (the empty set),
then $\Sigma * = \{\Lambda\}$ This is not the same as

If S = $\{\Lambda\}$,
then S* = $\{\Lambda\}$

An alphabet may look like a set of one-letter words. If for some reason we wish to modify the concept of closure to refer to only the concatenation of some (not zero) strings from a set S, we use the notation + instead of * For example,

LANGUAGES AND SETS

If Σ= {x}, then Σ + = { x *xx xxx*. . . } which is the language L, that we discussed before. If S = {xx, xxx} then S+ is the same as S* except for the word Λ, which is not in S+. This is not to say that S+ cannot in general contain the word Λ. It can, but only on condition that S contains the word Λ. In this case, Λ is in S+, since it is the concatenation of some (actually one) word from

S (Λ itself). Anyone who does not think that the null string is confusing has missed something. It is already a problem, and it gets worse later. If S is the set of three words *S* = { *w1 w2 w3* }

then,

S+ = { w 1 w 2 w3 w1w1 w1w2 w1w3 w2w1 w2w2 w2w3 w3w1 w3w2 w3w 3 w1w1w1 w1w1w2 . . . }

no matter what the words *w1, w2, and w3* are. If w1 = *aa,* w2 = *bbb,* w3 = Λ, then

S += { aa *bbb* Λ aaaa *aabbb...}*

LANGUAGES AND SETS

The words in the set S are listed above in the order corresponding to their w sequencing, not in the usual size-alphabetical order. What happens if we apply the closure operator twice? We start with a set of words S and look at its closure S*.

Now suppose we start with the set S* and try to form *its* closure, which we denote as (S*)* or *S*** If S is not the trivial empty set, then S* is infinite, so we are taking the closure of an infinite set.

THEOREM 1

For any set S of strings we have S* = S**. **CONVINCING** REMARKS
First let us illustrate what this theorem means. Say for example that S = *{a,b}*. Then **S*** is clearly all strings of the two letters a and *b* of any finite length whatsoever. Now what would it mean to take strings from S* and concatenate them?

LANGUAGES AND SETS

Let us say we concatenated *(aaba)* and *(baaa)* and *(aaba)*. The end result *(aababaaaaaba)* is no more than a concatenation of the letters a and b, just as with all elements of S*.

aababaaaaaba
= *(aaba)(baaa)(aaba)*
= *[(a)(a)(b)(a)]* [(b)(a)(a)(a)] [(a)(a)(b)(a)]
= *(a)(a)(b)(a)(b)(a)(a)(a)(a)(a)(b)(a)*

Let us consider one more illustration. If *S = {aa, bbb},* then S* is the set of all strings where the a's occur in even clumps and the b's in groups of 3, 6, 9. . . Some words in ***S**** are *aabbbaaaa bbb bbbaa* If we concatenate these three elements of *S**, we get one big word in ***S*****,** which is again in ***S*****.** *aabbbaaaabbbbbbaa = [(aa)(bbb)(aa)(aa)] [(bbb)] [(bbb)(aa)]* This theorem expresses a trivial but subtle point. It is analogous to saying that if people are made up of molecules and molecules are made up of atoms, then people are made up of atoms.

LANGUAGES AND SETS

PROOF

Every word in S^{**} is made up of factors from S*. Every factor from S* is made up of factors from S. Therefore, every word in S^{**} is made up of factors from S. Therefore, every word in S^{**} is also a word in S*. We can write this as

$S^{**} \subset S^{*}$

using the symbol " $\subset$ " from Set Theory, which means "is contained in or equal to.“

Now in general it is true that for any set A we know that $A \subset A^{*}$, since in A* we can chose as a word any one factor from A. So if we consider A to be our set S*, we have

$S^{*} \subset S^{**}$

Together, these two inclusions prove that

$S^{*} = S^{**}$

Next Lecture
RECURSIVE DEFINITIONS

One of the mathematical tools that we shall find extremely useful in our study, but which is largely unfamiliar in other branches of mathematics, is a method of defining sets called recursive definition. **A** recursive definition is characteristically a three-step process. First, we specify some basic objects in the set. Second, we give rules for constructing more objects in the set from the ones we already know. Third, we declare that no objects except those constructed in this way are allowed in the set.

قال رسول الله صلى الله عليه وسلم :

أعجز الناس من عجز عن الدعاء...

و أبخل الناس من بخل بالسلام .

صححه الألباني

واجب الدرس الاول

مادة الاحتسابية للعام الدراسي 2022/2021

Question: -

If you have this language ∑ (ab, bb, a, bab) find each of the followings: -

1- Find the closure (*) of this language (16 terms).
2- Find the palindrome of this language (16 terms).
3- Find the length of these words: -
   a- abbbaba.
   b- bbabbabab.
   c- bbΛ
4- check which one of these words are unique and which is not? (aab, babbaab, bbbbbab, baa, baab, babb, bababaaba).

ملاحظات الحلول (يجب ان تؤخذ بجدية):

1- كل ثلاثة واجبات يومية يعتبر امتحان شهري.
2- الحل يجب ان يكون مفصل كي ينفرد الطالب في الحلول.
3- اي تشابه يعتبر جميع من تتشابه حلولهم صفر بلا استثناء لذلك يرجى الابتعاد عن موضوع النقل ويجب ادراج التفاصيل والملاحظات عند الحل.
4- الحل يكون على برنامج الوورد حصرا ولا تعتبر الصورة مقبولة واي صورة او بي دي اف سيعتبر الواجب صفر.
5- ترك الواجبات بلا حلول يعتبر صفر ويتم محاسبة الطالب في نهاية كل فصل دراسي عن ترك الواجبات.
6- اخر موعد لحل الواجب هو يوم الاثنين المصادف 2021/11/15 الساعة التاسعة صباحا ومن الممكن الاستفسار عن الاسئلة يوم الاحد في الكلية.

م.د. محمد سامي محمد

مدرس المادة

Theory of Computation

النظرية الاحتسابية
المحاضرة الثانية والثالثة

كلية التربية للعلوم الصرفة / جامعة ديالى

اعداد
م.د. محمد سامي محمد

قسم علوم الحاسوب – المرحلة الثانية

RECURSIVE DEFINITIONS

One of the mathematical tools that we shall find extremely useful in our study,

A recursive definition is characteristically a three-step process:-

First, we specify some basic objects in the set.

Second, we give rules for constructing more objects in the set from the ones we already know.

Third, we declare that no objects except those constructed in this way are allowed in the set.

RECURSIVE DEFINITIONS

Example

Suppose that we are trying to define the set of positive even integers for someone who knows about arithmetic but has never heard of the even numbers.

Method 1:- EVEN is the set of all positive whole numbers divisible by 2.

Method 2:- EVEN is the set of all 2n where n = 1 2 3 4 ...

Method 3:-
The set EVEN is defined by these three rules:

Rule 1 2 is in EVEN.
Rule 2 If x is in EVEN then so is x + 2.
Rule 3 The *only* elements in the set EVEN are those that can be produced from the two rules above.

RECURSIVE DEFINITIONS

There is a reason that the third definition is less popular than the others: It is much harder to use in most practical applications.

suppose that we wanted to prove that 14 is in the set EVEN.

Using the first definition we divide 14 by 2 and find that there is no remainder. Therefore, it is in EVEN.

To prove that 14 is in EVEN by the second definition we have to somehow come up with the number 7 and then, since 14 = (2)(7), we know that it is in EVEN

By Rule 1, we know that 2 is in EVEN.
Then by Rule 2 we know that 2 + 2 = 4 is also in EVEN.
And, at last, by applying Rule 2 once more, to the number 12, we conclude that 12 + 2 = 14 is, indeed, in EVEN.

RECURSIVE DEFINITIONS

The set POLYNOMIAL is defined by these four rules:
Rule 1 Any number is in POLYNOMIAL.
Rule 2 The variable x is in POLYNOMIAL.
Rule 3 If *p* and q are in POLYNOMIAL, then so are p + q and (p) and *pq.*
Rule 4 POLYNOMIAL contains only those things which can be created by the three rules above.

RECURSIVE DEFINITIONS

Example

Some sequence of applications of these rules can show that $3x^2 + 7x - 9$ is in POLYNOMIAL.
By Rule 1 3 is in POLYNOMIAL
By Rule 2 x is in POLYNOMIAL
By Rule 3 (3)(x) is in POLYNOMIAL, call–it 3x
By Rule 3 *(3x)(x)* is in POLYNOMIAL, call it $3x^2$
By Rule 1 7 is in POLYNOMIAL
By Rule 3 (7)(x) is in POLYNOMIAL
By Rule 3 *3x′*+ 7x is in POLYNOMIAL
By Rule 1 –9 is in POLYNOMIAL
By Rule 3 3x 2 + 7x + (– 9) = 3x2 + 7x – 9 is in POLYNOMIAL.

REGULAR EXPRESSIONS

The language-defining symbols we are about to create are called Regular Expressions (RE).

We will define the term regular expression itself recursively. The languages that are associated with these regular expressions are called regular languages and are also said to be defined by finite representation.

ملاحظة : في هذا الفصل المجموعة محددة اي تحتوي على عناصر محددة وليست كالسابق

REGULAR EXPRESSIONS

Let us reconsider the language L4.

L4,={ Λ x xx *xxx xxxx . . .}*

In that chapter we presented one method for indicating this set as the closure of a smaller set.

Let S = {x}. Then L4 = *S**

As shorthand for this we could have written:
L4 = {x}*

We now introduce the use of the Kleene star applied not to a set but directly to the letter x and written as a superscript as if it were an exponent. x*

REGULAR EXPRESSIONS

This notation can be used to help us define languages by writing L4 = language (x*)

Suppose that we wished to describe the language L over the alphabet S = *{a,b}* where

L = {a ab abb abbb abbbb ... }

We could summarize this language by the English phrase "all words of the form one a followed by some number of b's (maybe no b's at all.)"

Using our star notation, we may write:
L = language (a b*) or without the space,
L = language (ab*)

REGULAR EXPRESSIONS

We can apply the Kleene star to the string *ab* if we want, as follows: (ab)* = Λ or *ab* or *abab* or *ababab* ...

EXAMPLE

The language defined by the expression ab*a

language (ab*a) = {aa *aba abba abbba abbbba* ... .}

REGULAR EXPRESSIONS

ملاحظة اذا احتوى القانون على نجمة هذا يعني ليس بالضرورة ان تكون موجودة

Remember x* can always be Λ.

EXAMPLE The language of the expression

a*b*

language (a*b*) = {Λ a *b* aa ab *bb* aaa aab *abb bbb* aaaa . . . }

Notice that ba and aba are not in this language. Notice also that there need not be the same number of a's and b's.

Here we should again be very careful to observe that

a*b* # (ab)*

REGULAR EXPRESSIONS

The language defined by the expression a*b*a* contains the word *baa* since it starts with zero a's followed by one b followed by two a's.

EXAMPLE

The following expressions both define the language

$$L2 = \{x^{Odd}\}$$

x(xx)* or (xx)*x but the expression x*xx* does not since it includes the word *(xx) x (x).*

REGULAR EXPRESSIONS

Question:-

Which one represent L = $\{x^{Even}\}$

- $x^* \; xx^*$
- $x^*(xx)^*$
- xx^*
- $(xx)^*$
- None
- All of above

REGULAR EXPRESSIONS

We now introduce another use for the plus sign. By the expression x + y, we mean "either x or y".

This means that x + y offers a choice, much the same way that x* does. Care should be taken so as not to confuse this with + as an exponent.

EXAMPLE

Consider the language T defined over the alphabet Σ = {a, *b, c*}
T = {a *c ab cb abb cbb abbb cbbb abbbb cbbbb* ... }
All the words in T begin with an a or a c and then are followed by some number of b's.
Symbolically, we may write this as

T = language ((a + c) b*)
= language (either a or *c* then some b's)

REGULAR EXPRESSIONS

EXAMPLE

Now let us consider a finite language L

L = {aaa *aab aba abb baa bab bba bbb}*

The first letter of each word in L is either an a or a b. The second letter of each word in L is either an a or a *b.* The third letter of each word in L is either an a or a *b.*

So we may write

L = language ((a + b)(a + b)(a + b))

or for short,

L = language ($(a + b)^3$)

REGULAR EXPRESSIONS

If we want to define the set of all seven letter strings of a's and b's, we could write $(a + b)^7$.
In general, if we want to refer to the set of all possible strings of a's and b's of any length whatsoever we could write, (a + b)*

This is the set of all possible strings of letters from the alphabet 2 = *{a, b}*

Again this expression represents a language. If we decide that * stands for 5, then (a + b)* gives

$(a + b)^5$= (a+b)(a+b)(a+b)(a+b)(a+b)

REGULAR EXPRESSIONS

We can describe all words that begin with the letter a simply as:

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

that is, first an a, then anything (as many choices as we want of either letter a or *b).*

All words that begin with an a and end with a b can be defined by the expression

$$a(a + b)^*b = a \text{ (arbitrary string) } b$$

REGULAR EXPRESSIONS

EXAMPLE

Let us consider the language defined by the expression

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

For example, the word *abbaab* can be considered to be of this form in three ways:

(Λ) a *(bbaab)* or *(abb)* a (ab) or (abba) a *(b)*

Question:- what about aaaaa and bbaa how many different expression can be described?

REGULAR EXPRESSIONS

EXAMPLE

The language of all words that have at least two a's can be described by the expression

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

= (some beginning)(the first important a)(some middle)(the second important a)(some end)

where the arbitrary parts can have as many a's (or b's) as they want.

In the last three examples we have used the notation $(a + b)^*$ as a factor to mean "any possible substring,"

REGULAR EXPRESSIONS

EXAMPLE

Another expression that denotes all the words with at least two a's is:

$$b^*ab^*a(a + b)^*$$

We scan through some jungle of b's (or no b's) until we find the first a, then more b's (or no b's), then the second a, then we finish up with anything.

In this set are *abbbabb* and aaaaa.
We can write:

$$(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$$

where by the equal sign we do not mean that these expressions are equal algebraically in the same way as

$$x+x=2x$$

REGULAR EXPRESSIONS

We could write
language ((a + b)*a(a + b)*a(a + b)*)
= language (b*ab*a (a + b)*)
= all words with at least two a's.

To be careful about this point, we say that two regular expressions are equivalent if they describe the same language.

$$(a + b)^* ab^* ab^*$$

↑ next to last a ↑ last a

and

$$b^*a(a + b)^*ab^*$$

↑ first a ↑ last a

REGULAR EXPRESSIONS

EXAMPLE

If we wanted all the words with *exactly* two a's, we could use the expression

b*ab*ab*

which describes such words as *aab, baba,* and *bbbabbbbab.*

To make the word *aab,* we let the first and second b* become A and the last becomes b.

REGULAR EXPRESSIONS

EXAMPLE

The language of all words that have at least one a and at least one b is somewhat trickier. If we write

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

= (arbitrary) a(arbitrary) b(arbitrary)

we could define this set by the expression:

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

Here we are still using the plus sign in the general sense of disjunction (or). We are taking the union of two sets,

REGULAR EXPRESSIONS

يمكن التعبير عن الصيغ بصيغاخرى مشابهة

EXAMPLE

All temptation to treat these language-defining expressions as if they were algebraic polynomials should be dispelled by these equivalences:

$(a+b)^* = (a+b)^* + (a+b)^*$

$(a+b)^* = (a+b)^* (a+b)^*$

$(a+b)^* = a(a+b)^* + b(a+b)^* + \Lambda$

$(a+b)^* = (a+b)^* ab(a+b)^* + b^*a^*$

REGULAR EXPRESSIONS

Usually when we employ the star operator, we are defining an infinite language.
We can represent a finite language by using the plus (union sign) alone. If the language L over the alphabet X = {a, *b*} contains only the finite list of words given below,

$$L = \{abba\ baaa\ bbbb\}$$

then we can represent L by the symbolic expression

$$L = \text{language (abba + baaa + bbbb)}$$

If L is a finite language that includes the null word Λ, then the expression that defines L must also employ the symbol Λ.
For example, if

$$L = \{\Lambda\ a\ aa\ bbb\}$$

then the symbolic expression for L must be

$$L = \text{language (}\Lambda + a + aa + bbb\text{)}$$

REGULAR EXPRESSIONS

EXAMPLE

Let V be the language of all strings of a's and b's in which the strings are either all b's or else there is an a followed by some b's. Let V also contain the word Λ.

$$V = \{\Lambda\ a\ bab\ bb\ abb\ bbb\ abbb\ bbbb\ \ldots.$$

We can define V by the expression

$$b^* + ab^*$$

where the word Λ is included in the term b^*. Alternatively, we could define V by the expression:

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

This would mean that in front of the string of some *b's* we have the option of either adding an a or nothing. Since we could always write $b^* = \Lambda\, b^*$,

$$\Lambda\, b^* + ab^* = (\Lambda + a)b^*$$

REGULAR EXPRESSIONS

Let us reconsider the language
T = {a c ab cb abb cbb … }.

T can be defined as above by
$(a + c)b^*$
but it can also be defined by
$ab^* + cb^*$

This is another example of the distributive law.

REGULAR EXPRESSIONS

If r1 = aa + b then the expression r1* technically refers to the expression

$$r1^* = aa + b^*$$

which is the formal concatenation of the symbols for r, with the symbol *, but what we generally mean when we write r1* is actually (r1)*

$$(r1)^* = (aa + b)^*$$

REGULAR EXPRESSIONS

DEFINITION

If S and T are sets of strings of letters (whether they are finite or infinite sets), we define the product set of strings of letters to be

ST = {all combinations of a string from S concatenated with a string from T }

EXAMPLE

If

$$S = \{a\ aa\ aaa\} \text{ and } T = \{bb\ bbb\}$$

then

$$ST = \{abb\ abbb\ aabb\ aabbb\ aaabb\ aaabbb\}$$

EXAMPLE

If $S = \{a\ bb\ bab\}$ $T = \{a\ ab\}$

Then $ST = \{aa\ aab\ bba\ bbab\ baba\ babab\}$

REGULAR EXPRESSIONS

EXAMPLE

If $P = \{a\ bb\ bab\}$ and $Q = \{\Lambda\ bbbb\}$

then

$PQ = \{a\ bb\ bab\ abbbb\ bbbbbb\ babbbbb\}$

ملاحظة :- فقط في حالة الصفر موجود في المجموعتين يكون في الناتج صفر

EXAMPLE

If $M = \{\Lambda\ x\ xx\}$ $N = \{\Lambda\ y\ yy\ yyy\ yyyy \ldots\}$

then

$MN = \{\Lambda\ y\ yy\ yyy\ yyyy$ **...**

x xy xyy xyyy xyyyy…

xx **xxy** xxyy xxyyy xxyyyy.. **.}.**

REGULAR EXPRESSIONS

Using regular expressions, these four examples can be written as:

$(a + aa + aaa)(bb + bbb) = abb + abbb + aabb + aabbb + aaabb + aaabbb$

$(a + bb + bab)(a + ab) = aa + aab + bba + bbab + baba + babab$

$(a + bb + bab)(A + bbbb) = a+bb+bab+abbbbb +bbbbbb + babbbbb$

$(\Lambda + x + xx)(y^*) = y^* + xy^* + xxy^*$

REGULAR EXPRESSIONS

THEOREM 5
If L is a finite language (a language with only finitely many words), then L can be defined by a regular expression.

PROOF

For example, the regular expression that defines the language

$$L = \{baa\ abbba\ bababa\}$$

is

$$baa + abbba + bababa$$

Another example If

$$L = \{aa\ ab\ ba\ bb\}$$

the algorithm described above gives the regular expression

$$aa + ab + ba + bb$$

Another regular expression that defines this language is

$$(a + b)(a + b)$$

REGULAR EXPRESSIONS

EXAMPLE

Let

$$L = \{\Lambda \text{ x xx } xxx\ xxxx\ xxxxx\}$$

The regular expression we get from the theorem is

$$\Lambda + x + xx + xxx + xxxx + xxxxx$$

A more elegant regular expression for this language is

$$(\Lambda + x)^5$$

Of course the 5 is, strictly speaking, not a legal symbol for a regular expression although we all understand it means

$$(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the expression:

$$(a + b)^*(aa + bb)(a + b)^*$$

This is the set of strings of a's and b's that at some point contain a double letter. We can think of it as

(arbitrary)(double letter)(arbitrary)

Example are: Λ a *b ab ba aba bab abab baba* …. The expression (ab)* covers all of these except those that begin with b or end in a. Adding these choices gives us the regular expression

(Λ+ b) (ab)* (Λ+ a)

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression below:

$$E = (a + b)^*a(a + b)^* (a + \Lambda) (a + b)^*a(a + b)^*$$

= (arbitrary) a (arbitrary) [a or nothing] (arbitrary) a (arbitrary).

$$= (a+b)^*a(a+b)^*a(a+b)^*a(a+b)^*+(a + b)^*a(a + b)^* \Lambda(a + b)^*a(a + b)^*$$

Before we analyze the second term let us make the observation that

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

REGULAR EXPRESSIONS

which occurs in the middle of the second term is only another way of saying "any string whatsoever" and could be replaced with the more direct expression

$$(a + b)^*$$

This would reduce the second term of the expression to

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

which we have already seen is a regular expression representing all words that have at least two a's in them. Therefore, the language associated with E is the union of all strings that have three or more a's with all strings that have two or more a's. But since all strings with three or more a's are themselves already strings with two or more a's, this whole language is just the second set alone.

REGULAR EXPRESSIONS

Example:- It is possible by repeated application of the rules for forming regular expressions to produce an expression in which the star operator is applied to a sub expression that already has a star in it. Some examples are:

$$(a + b^*)^* \; (aa + ab^*)^* \; ((a + bbba^*) + ba^*b)^*$$

In the first of these expressions, the internal * adds nothing to the language

$$(a + b^*)^* = (a + b)^*$$

$$(a^*)^* = a^*$$

However,

$$(aa + ab^*)^* \; \# \; (aa + ab)^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression:

$$(a^*b^*)^*$$

The language defined by this expression is all strings that can be made up of factors of the form a*b*, but since both the single letter a and the single letter b are words of the form a*b*, this language contains all strings of a's and b's. It cannot contain more than everything, so

$$(a^*b^*)^* = (a + b)^*$$

REGULAR EXPRESSIONS

EXAMPLE

One very interesting example, which we consider now in great detail is

$$E = [aa + bb + (ab+ba)(aa+bb)^*(ab+ba)]^*$$

This regular expression represents the collection of all words that are made up of "syllables" of three types:

type1 = aa
type2 = bb
type3 = (ab + ba)(aa + bb)*(ab + *ba)*

$$E = [\text{type}_1 + \text{type2} + \text{type 3}]^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the language defined by the regular expression:

$$b^*(abb^*)^*(\Lambda + a)$$

This is the language of all words without a double a.

"لا تُسيء اللّفظ وإن
ضَاق عليكَ الجَواب"

- الإمام علي بن ابي طالب

Theory of Computation

النظرية الاحتسابية
المحاضرة الثانية والثالثة

كلية التربية للعلوم الصرفة / جامعة ديالى

اعداد
م.د. محمد سامي محمد

قسم علوم الحاسوب – المرحلة الثانية

RECURSIVE DEFINITIONS

One of the mathematical tools that we shall find extremely useful in our study,

A recursive definition is characteristically a three-step process:-

First, we specify some basic objects in the set.

Second, we give rules for constructing more objects in the set from the ones we already know.

Third, we declare that no objects except those constructed in this way are allowed in the set.

RECURSIVE DEFINITIONS

Example

Suppose that we are trying to define the set of positive even integers for someone who knows about arithmetic but has never heard of the even numbers.

Method 1:- EVEN is the set of all positive whole numbers divisible by 2.

Method 2:- EVEN is the set of all 2n where n = 1 2 3 4 ...

Method 3:-
The set EVEN is defined by these three rules:

Rule 1 2 is in EVEN.
Rule 2 If x is in EVEN then so is x + 2.
Rule 3 The *only* elements in the set EVEN are those that can be produced from the two rules above.

RECURSIVE DEFINITIONS

There is a reason that the third definition is less popular than the others: It is much harder to use in most practical applications.

suppose that we wanted to prove that 14 is in the set EVEN.

Using the first definition we divide 14 by 2 and find that there is no remainder. Therefore, it is in EVEN.

To prove that 14 is in EVEN by the second definition we have to somehow come up with the number 7 and then, since 14 = (2)(7), we know that it is in EVEN

By Rule 1, we know that 2 is in EVEN.
Then by Rule 2 we know that 2 + 2 = 4 is also in EVEN.
And, at last, by applying Rule 2 once more, to the number 12, we conclude that 12 + 2 = 14 is, indeed, in EVEN.

RECURSIVE DEFINITIONS

The set POLYNOMIAL is defined by these four rules:
Rule 1 Any number is in POLYNOMIAL.
Rule 2 The variable x is in POLYNOMIAL.
Rule 3 If *p* and q are in POLYNOMIAL, then so are p + q and (p) and *pq.*
Rule 4 POLYNOMIAL contains only those things which can be created by the three rules above.

RECURSIVE DEFINITIONS

Example

Some sequence of applications of these rules can show that $3x^2 + 7x - 9$ is in POLYNOMIAL.
By Rule 1 3 is in POLYNOMIAL
By Rule 2 x is in POLYNOMIAL
By Rule 3 (3)(x) is in POLYNOMIAL, call–it 3x
By Rule 3 *(3x)(x)* is in POLYNOMIAL, call it $3x^2$
By Rule 1 7 is in POLYNOMIAL
By Rule 3 (7)(x) is in POLYNOMIAL
By Rule 3 *3x′*+ 7x is in POLYNOMIAL
By Rule 1 –9 is in POLYNOMIAL
By Rule 3 3x 2 + 7x + (– 9) = 3x2 + 7x – 9 is in POLYNOMIAL.

REGULAR EXPRESSIONS

The language-defining symbols we are about to create are called Regular Expressions (RE).

We will define the term regular expression itself recursively. The languages that are associated with these regular expressions are called regular languages and are also said to be defined by finite representation.

ملاحظة : في هذا الفصل المجموعة محددة اي تحتوي على عناصر محددة وليست كالسابق

REGULAR EXPRESSIONS

Let us reconsider the language L4.

L4,={ Λ x xx *xxx xxxx . . .}*

In that chapter we presented one method for indicating this set as the closure of a smaller set.

Let S = {x}. Then L4 = *S**

As shorthand for this we could have written:
L4 = {x}*

We now introduce the use of the Kleene star applied not to a set but directly to the letter x and written as a superscript as if it were an exponent. x*

REGULAR EXPRESSIONS

This notation can be used to help us define languages by writing L4 = language (x*)

Suppose that we wished to describe the language L over the alphabet S = *{a,b}* where

L = {a ab abb abbb abbbb ... }

We could summarize this language by the English phrase "all words of the form one a followed by some number of b's (maybe no b's at all.)"

Using our star notation, we may write:
L = language (a b*) or without the space,
L = language (ab*)

REGULAR EXPRESSIONS

We can apply the Kleene star to the string *ab* if we want, as follows: (ab)* = Λ or *ab* or *abab* or *ababab* ...

EXAMPLE

The language defined by the expression ab*a

language (ab*a) = {aa *aba abba abbba abbbba* ... .}

REGULAR EXPRESSIONS

ملاحظة اذا احتوى القانون على نجمة هذا يعني ليس بالضرورة ان تكون موجودة

Remember x* can always be Λ.

EXAMPLE The language of the expression

a*b*

language (a*b*) = {Λ a *b* aa ab *bb* aaa aab *abb bbb* aaaa . . . }

Notice that ba and aba are not in this language. Notice also that there need not be the same number of a's and b's.

Here we should again be very careful to observe that

a*b* # (ab)*

REGULAR EXPRESSIONS

The language defined by the expression a*b*a* contains the word *baa* since it starts with zero a's followed by one b followed by two a's.

EXAMPLE

The following expressions both define the language

$$L2 = \{x^{Odd}\}$$

x(xx)* or (xx)*x but the expression x*xx* does not since it includes the word *(xx) x (x).*

REGULAR EXPRESSIONS

Question:-

Which one represent L = $\{x^{Even}\}$

- $x^* \; xx^*$
- $x^*(xx)^*$
- xx^*
- $(xx)^*$
- None
- All of above

REGULAR EXPRESSIONS

We now introduce another use for the plus sign. By the expression x + y, we mean "either x or y".

This means that x + y offers a choice, much the same way that x* does. Care should be taken so as not to confuse this with + as an exponent.

EXAMPLE

Consider the language T defined over the alphabet Σ = {a, *b, c*}
T = {a *c ab cb abb cbb abbb cbbb abbbb cbbbb* ... }
All the words in T begin with an a or a c and then are followed by some number of b's.
Symbolically, we may write this as

T = language ((a + c) b*)
= language (either a or *c* then some b's)

REGULAR EXPRESSIONS

EXAMPLE

Now let us consider a finite language L

L = {aaa *aab aba abb baa bab bba bbb}*

The first letter of each word in L is either an a or a b. The second letter of each word in L is either an a or a *b.* The third letter of each word in L is either an a or a *b.*

So we may write

L = language ((a + b)(a + b)(a + b))

or for short,

L = language ($(a + b)^3$)

REGULAR EXPRESSIONS

If we want to define the set of all seven letter strings of a's and b's, we could write $(a + b)^7$.
In general, if we want to refer to the set of all possible strings of a's and b's of any length whatsoever we could write, (a + b)*

This is the set of all possible strings of letters from the alphabet 2 = *{a, b}*

Again this expression represents a language. If we decide that * stands for 5, then (a + b)* gives

$(a + b)^5$= (a+b)(a+b)(a+b)(a+b)(a+b)

REGULAR EXPRESSIONS

We can describe all words that begin with the letter a simply as:

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

that is, first an a, then anything (as many choices as we want of either letter a or *b).*

All words that begin with an a and end with a b can be defined by the expression

$$a(a + b)^*b = a \text{ (arbitrary string) } b$$

REGULAR EXPRESSIONS

EXAMPLE

Let us consider the language defined by the expression

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

For example, the word *abbaab* can be considered to be of this form in three ways:

(Λ) a *(bbaab)* or *(abb)* a (ab) or (abba) a *(b)*

Question:- what about aaaaa and bbaa how many different expression can be described?

REGULAR EXPRESSIONS

EXAMPLE

The language of all words that have at least two a's can be described by the expression

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

= (some beginning)(the first important a)(some middle)(the second important a)(some end)

where the arbitrary parts can have as many a's (or b's) as they want.

In the last three examples we have used the notation $(a + b)^*$ as a factor to mean "any possible substring,"

REGULAR EXPRESSIONS

EXAMPLE

Another expression that denotes all the words with at least two a's is:

$$b^*ab^*a(a + b)^*$$

We scan through some jungle of b's (or no b's) until we find the first a, then more b's (or no b's), then the second a, then we finish up with anything.

In this set are *abbbabb* and aaaaa.
We can write:

$$(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$$

where by the equal sign we do not mean that these expressions are equal algebraically in the same way as

$$x+x=2x$$

REGULAR EXPRESSIONS

We could write
language ((a + b)*a(a + b)*a(a + b)*)
= language (b*ab*a (a + b)*)
= all words with at least two a's.

To be careful about this point, we say that two regular expressions are equivalent if they describe the same language.

$$(a + b)^* ab^* ab^*$$

↑ next to last a ↑ last a

and

$$b^*a(a + b)^*ab^*$$

↑ first a ↑ last a

REGULAR EXPRESSIONS

EXAMPLE

If we wanted all the words with *exactly* two a's, we could use the expression

b*ab*ab*

which describes such words as *aab, baba,* and *bbbabbbbab.*

To make the word *aab,* we let the first and second b* become A and the last becomes b.

REGULAR EXPRESSIONS

EXAMPLE

The language of all words that have at least one a and at least one b is somewhat trickier. If we write

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

= (arbitrary) a(arbitrary) b(arbitrary)

we could define this set by the expression:

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

Here we are still using the plus sign in the general sense of disjunction (or). We are taking the union of two sets,

REGULAR EXPRESSIONS

يمكن التعبير عن الصيغ بصيغاخرى مشابهة

EXAMPLE

All temptation to treat these language-defining expressions as if they were algebraic polynomials should be dispelled by these equivalences:

$(a+b)^* = (a+b)^* + (a+b)^*$

$(a+b)^* = (a+b)^* (a+b)^*$

$(a+b)^* = a(a+b)^* + b(a+b)^* + \Lambda$

$(a+b)^* = (a+b)^* ab(a+b)^* + b^*a^*$

REGULAR EXPRESSIONS

Usually when we employ the star operator, we are defining an infinite language.
We can represent a finite language by using the plus (union sign) alone. If the language L over the alphabet X = {a, *b*} contains only the finite list of words given below,

$$L = \{abba\ baaa\ bbbb\}$$

then we can represent L by the symbolic expression

$$L = \text{language (abba + baaa + bbbb)}$$

If L is a finite language that includes the null word Λ, then the expression that defines L must also employ the symbol Λ.
For example, if

$$L = \{\Lambda\ a\ aa\ bbb\}$$

then the symbolic expression for L must be

$$L = \text{language (}\Lambda + a + aa + bbb\text{)}$$

REGULAR EXPRESSIONS

EXAMPLE

Let V be the language of all strings of a's and b's in which the strings are either all b's or else there is an a followed by some b's. Let V also contain the word Λ.

$$V = \{\Lambda\ a\ bab\ bb\ abb\ bbb\ abbb\ bbbb\ \ldots.$$

We can define V by the expression

$$b^* + ab^*$$

where the word Λ is included in the term b^*. Alternatively, we could define V by the expression:

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

This would mean that in front of the string of some *b's* we have the option of either adding an a or nothing. Since we could always write $b^* = \Lambda\, b^*$,

$$\Lambda\, b^* + ab^* = (\Lambda + a)b^*$$

REGULAR EXPRESSIONS

Let us reconsider the language
T = {a c ab cb abb cbb … }.

T can be defined as above by
$(a + c)b^*$
but it can also be defined by
$ab^* + cb^*$

This is another example of the distributive law.

REGULAR EXPRESSIONS

If r1 = aa + b then the expression r1* technically refers to the expression

$$r1^* = aa + b^*$$

which is the formal concatenation of the symbols for r, with the symbol *, but what we generally mean when we write r1* is actually (r1)*

$$(r1)^* = (aa + b)^*$$

REGULAR EXPRESSIONS

DEFINITION

If S and T are sets of strings of letters (whether they are finite or infinite sets), we define the product set of strings of letters to be

ST = {all combinations of a string from S concatenated with a string from T }

EXAMPLE

If

$$S = \{a\ aa\ aaa\} \text{ and } T = \{bb\ bbb\}$$

then

$$ST = \{abb\ abbb\ aabb\ aabbb\ aaabb\ aaabbb\}$$

EXAMPLE

If $S = \{a\ bb\ bab\}$ $T = \{a\ ab\}$

Then $ST = \{aa\ aab\ bba\ bbab\ baba\ babab\}$

REGULAR EXPRESSIONS

EXAMPLE

If $P = \{a\ bb\ bab\}$ and $Q = \{\Lambda\ bbbb\}$

then

$PQ = \{a\ bb\ bab\ abbbb\ bbbbbb\ babbbbb\}$

ملاحظة :- فقط في حالة الصفر موجود في المجموعتين يكون في الناتج صفر

EXAMPLE

If $M = \{\Lambda\ x\ xx\}$ $N = \{\Lambda\ y\ yy\ yyy\ yyyy \ldots\}$

then

$MN = \{\Lambda\ y\ yy\ yyy\ yyyy$ **...**

x xy xyy xyyy xyyyy…

xx **xxy** xxyy xxyyy xxyyyy.. **.}.**

REGULAR EXPRESSIONS

Using regular expressions, these four examples can be written as:

$(a + aa + aaa)(bb + bbb) = abb + abbb + aabb + aabbb + aaabb + aaabbb$

$(a + bb + bab)(a + ab) = aa + aab + bba + bbab + baba + babab$

$(a + bb + bab)(A + bbbb) = a+bb+bab+abbbbb +bbbbbb + babbbbb$

$(\Lambda + x + xx)(y^*) = y^* + xy^* + xxy^*$

REGULAR EXPRESSIONS

THEOREM 5
If L is a finite language (a language with only finitely many words), then L can be defined by a regular expression.

PROOF

For example, the regular expression that defines the language

$$L = \{baa\ abbba\ bababa\}$$

is

$$baa + abbba + bababa$$

Another example If

$$L = \{aa\ ab\ ba\ bb\}$$

the algorithm described above gives the regular expression

$$aa + ab + ba + bb$$

Another regular expression that defines this language is

$$(a + b)(a + b)$$

REGULAR EXPRESSIONS

EXAMPLE

Let

$$L = \{\Lambda \text{ x xx } xxx\ xxxx\ xxxxx\}$$

The regular expression we get from the theorem is

$$\Lambda + x + xx + xxx + xxxx + xxxxx$$

A more elegant regular expression for this language is

$$(\Lambda + x)^5$$

Of course the 5 is, strictly speaking, not a legal symbol for a regular expression although we all understand it means

$$(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the expression:

$$(a + b)^*(aa + bb)(a + b)^*$$

This is the set of strings of a's and b's that at some point contain a double letter. We can think of it as

(arbitrary)(double letter)(arbitrary)

Example are: Λ a *b ab ba aba bab abab baba* …. The expression (ab)* covers all of these except those that begin with b or end in a. Adding these choices gives us the regular expression

(Λ+ b) (ab)* (Λ+ a)

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression below:

$$E = (a + b)^*a(a + b)^* (a + \Lambda) (a + b)^*a(a + b)^*$$

= (arbitrary) a (arbitrary) [a or nothing] (arbitrary) a (arbitrary).

$$= (a+b)^*a(a+b)^*a(a+b)^*a(a+b)^*+(a + b)^*a(a + b)^* \Lambda(a + b)^*a(a + b)^*$$

Before we analyze the second term let us make the observation that

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

REGULAR EXPRESSIONS

which occurs in the middle of the second term is only another way of saying "any string whatsoever" and could be replaced with the more direct expression

$$(a + b)^*$$

This would reduce the second term of the expression to

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

which we have already seen is a regular expression representing all words that have at least two a's in them. Therefore, the language associated with E is the union of all strings that have three or more a's with all strings that have two or more a's. But since all strings with three or more a's are themselves already strings with two or more a's, this whole language is just the second set alone.

REGULAR EXPRESSIONS

Example:- It is possible by repeated application of the rules for forming regular expressions to produce an expression in which the star operator is applied to a sub expression that already has a star in it. Some examples are:

$$(a + b^*)^* \; (aa + ab^*)^* \; ((a + bbba^*) + ba^*b)^*$$

In the first of these expressions, the internal * adds nothing to the language

$$(a + b^*)^* = (a + b)^*$$

$$(a^*)^* = a^*$$

However,

$$(aa + ab^*)^* \; \# \; (aa + ab)^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression:

$$(a^*b^*)^*$$

The language defined by this expression is all strings that can be made up of factors of the form a*b*, but since both the single letter a and the single letter b are words of the form a*b*, this language contains all strings of a's and b's. It cannot contain more than everything, so

$$(a^*b^*)^* = (a + b)^*$$

REGULAR EXPRESSIONS

EXAMPLE

One very interesting example, which we consider now in great detail is

$$E = [aa + bb + (ab+ba)(aa+bb)^*(ab+ba)]^*$$

This regular expression represents the collection of all words that are made up of "syllables" of three types:

type1 = aa
type2 = bb
type3 = (ab + ba)(aa + bb)*(ab + *ba)*

$$E = [\text{type}_1 + \text{type2} + \text{type 3}]^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the language defined by the regular expression:

$$b^*(abb^*)^*(\Lambda + a)$$

This is the language of all words without a double a.

"لا تُسيء اللّفظ وإن
ضَاق عليكَ الجَواب"

- الإمام علي بن ابي طالب

واجب الدرس الثاني

مادة الاحتسابية للعام الدراسي 2022/2021

Question 1: -

If you have this regular expression (ab)*ba* do the followings: -

1- Open this term.
2- Are these words belonging to the term above: -?
   a- baa
   b- aabbaa
   c- aaa
3- which one of the above words are unique and which is not? Why?

Question 2: -

If you have this regular expression (112) *2*(1*1) * do the followings: -

1- Open this term.
2- Is these words belonging to the term above: -?
   d- 121212
   e- 2221
   f- 121
3- which one of the above words are unique and which is not? Why?

ملاحظات الحلول (يجب ان تؤخذ بجدية):

1- كل ثلاثة واجبات يومية يعتبر امتحان شهري.
2- الحل يجب ان يكون مفصل كي ينفرد الطالب في الحلول.
3- اي تشابه يعتبر جميع من تتشابه حلولهم صفر بلا استثناء لذلك يرجى الابتعاد عن موضوع النقل ويجب ادراج التفاصيل والملاحظات عند الحل.
4- الحل يكون على برنامج الوورد حصرا ولا تعتبر الصورة مقبولة واي صورة او بي دي اف سيعتبر الواجب صفر.
5- ترك الواجبات بلا حلول يعتبر صفر ويتم محاسبة الطالب في نهاية كل فصل دراسي عن ترك الواجبات.
6- اخر موعد لحل الواجب هو يوم الثلاثاء المصادف 2021/11/30 الساعة التاسعة صباحا ومن الممكن الاستفسار عن الاسئلة يوم الاحد في الكلية.

ملاحظات عن الواجب السابق :-

لا توجد تفاصيل في الحل وسيتم اعلان بعض اسماء الطلبة الذين تتشابه اجوبتهم علما انه اي جواب على شكل صورة او بي دي اف ستكون درجته صفرا

م.د. محمد سامي محمد

مدرس المادة

S = {a,b}

L = {a ab abb abbb abbbb ... }

write the R.E. of this language

L= ab*

S={b}

S*={ aA, ab, abb, abbb, abbbb, ……}

$$S = \{ab\}$$

$$S^* = (ab)^* = \{ A, ab, abab, ababab, abababab, \ldots\}$$

$$(ab)^* \qquad ab^*$$

EXAMPLE The language defined by the expression ab*a

L1= ab*a

b*={A, b, bb, bbb, bbbb, ……}

L1={ aAa, aba, abba, abbb a, abbbb a, abbbbb a, a a, a a, a a

L2=aab*

L2={ aaA, aab, aabb, aabbb, aa, aa, aa, aa, aa,

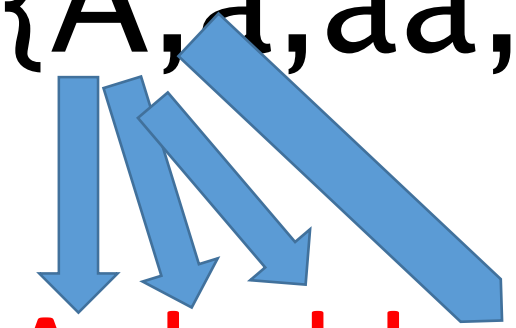
L3=b*aa

L3={ Aaa, baa, bbaa, bbbaa, aa, aa, aa, aa

L=a*b*

a*={A,a,aa,aaa, aaaa, aaaaa,

… }



b*={A, b, bb, bbb, bbbb, ……}

L={AA, Ab, Abb, Abbb, aA ,

ab, abb, ……}

L= a*b*a*

a*={A,a,aa,aaa, aaaa, aaaaa, … }

b*={A, b, bb, bbb, bbbb, ……}

a*={A,a,aa,aaa, aaaa, aaaaa, … }

L= {AAA, AAa, AAaa,AbA,,

b

E={x}

L = {$\boldsymbol{xOdd}$}

L1= x*={A, x, xx, xxx, xxxx, ……..}

L2= (xx)* ={A, xx, xxxx, xxxxxx, ……..}

L3=(xxx)*={A, xxx, xxxxxx, ……..}

L4= x(xx)*= {x,xxx, xxxxx,……..}

Question:- Which one represent L = $\{\boldsymbol{xEven}\}$

1-$\boldsymbol{x}* \ \boldsymbol{xx}*$

2-$\boldsymbol{x}*(\boldsymbol{xx})*$

3-$\boldsymbol{xx}*$

4-$(\boldsymbol{xx})*$

5-None

6-All of above

R.E.

L1 = (x+y) = {x, y}

L2= (a +b)= {a, b}

L3 = (12 + 17) = { 12, 17}

L4= (ab + bb) = {ab, bb}

L5= (cd+ ef +bb)= { cd, ef, bb}

R.E.

L1 = (x* + y) = {A, x, xx, xxx, xxxx, ...,y }

x*= { A, x, xx, xxx, xxxx, ... }

R.E.

L1 = (ab* + b*) = {aA, ab, abb, abbb, abbbb, abbbbb, abbbbbb,…. , A,b,bb,bbb,bbbb, …….}

ab*= {aA, ab, abb, abbb, abbbb, abbbbb,….

S = {a,b}

L = {a ab abb abbb abbbb ... }

write the R.E. of this language

L= ab*

S={b}

S*={ aA, ab, abb, abbb, abbbb, ……}

$$S = \{ab\}$$

$$S^* = (ab)^* = \{ A, ab, abab, ababab, abababab, \ldots\}$$

$$(ab)^* \qquad ab^*$$

EXAMPLE The language defined by the expression ab*a

L1= ab*a

b*={A, b, bb, bbb, bbbb, ……}

L1={ aAa, aba, abba, abbb a, abbbb a, abbbbb a, a a, a a, a a

L2=aab*

L2={ aaA, aab, aabb, aabbb, aa, aa, aa, aa, aa,

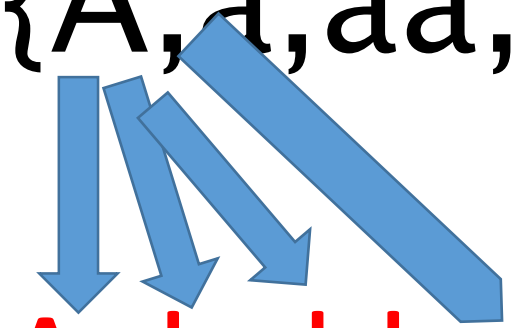
L3=b*aa

L3={ Aaa, baa, bbaa, bbbaa, aa, aa, aa, aa

L=a*b*

a*={A,a,aa,aaa, aaaa, aaaaa,

… }



b*={A, b, bb, bbb, bbbb, ……}

L={AA, Ab, Abb, Abbb, aA ,

ab, abb, ……}

L= a*b*a*

a*={A,a,aa,aaa, aaaa, aaaaa, … }

b*={A, b, bb, bbb, bbbb, ……}

a*={A,a,aa,aaa, aaaa, aaaaa, … }

L= {AAA, AAa, AAaa,AbA,,

b

E={x}

L = {$\boldsymbol{xOdd}$}

L1= x*={A, x, xx, xxx, xxxx, ……..}

L2= (xx)* ={A, xx, xxxx, xxxxxx, ……..}

L3=(xxx)*={A, xxx, xxxxxx, ……..}

L4= x(xx)*= {x,xxx, xxxxx,……..}

Question:- Which one represent L = $\{\boldsymbol{xEven}\}$

1-$\boldsymbol{x}* \ \boldsymbol{xx}*$

2-$\boldsymbol{x}*(\boldsymbol{xx})*$

3-$\boldsymbol{xx}*$

4-$(\boldsymbol{xx})*$

5-None

6-All of above

R.E.

L1 = (x+y) = {x, y}

L2= (a +b)= {a, b}

L3 = (12 + 17) = { 12, 17}

L4= (ab + bb) = {ab, bb}

L5= (cd+ ef +bb)= { cd, ef, bb}

R.E.

L1 = (x* + y) = {A, x, xx, xxx, xxxx, ...,y }

x*= { A, x, xx, xxx, xxxx, ... }

R.E.

L1 = (ab* + b*) = {aA, ab, abb, abbb, abbbb, abbbbb, abbbbbb,…. , A,b,bb,bbb,bbbb, …….}

ab*= {aA, ab, abb, abbb, abbbb, abbbbb,….

R.E.

a

(a* + b)*=

a*= {A,a,aa,aaa, aaaa, aaaaa, … }

(a**)= a* = a***

(ab* + (ba)*)*=

R.E.

a***=a*

(ab* + (ba)*)*=

(ab*) *= {

ab*= { a, ab, abb, abbb, abbbb, abbbbb, ... }*

= { a*, (ab)*, (abb)*, ...}

= { A, a, aa, aaa, ab, abab, ababab, abb, abbabb, abbabbabb,.....}

(ba)** = (ba)*= { A, ba, baba, bababa, ...}

(ab* + (ba)*)*= {A, a, aa, aaa, ab, abab, ababab, abb, abbabb, abbabbabb,....., ba, baba, bababa}

(12 + 21*)*=
(12)*= { A, 12, 1212, 121212,……}

(21*)* = {2, 21 , 211 , 2111 , 21111 , 211111,… } *
21* = { 2, 21 , 211 , 2111 , 21111 , 211111,……
(21*)* = {2*, (21)* , (211)* , (2111)* , (21111)* ,
211111,… }

L = {aaa aab aba abb baa bab bba bbb}

L= (a + b)(a + b) (a + b) =(a+b)3

q1/ L1= (a + b)(a + b)

= (aa + ab +ba +bb)

q2/ L2= (a + b)*(a + b)

L2= (a* + b*) (a + b) = (a*a + a*b+ b*a+ b*b)

{ }

L= (a + b)(a + b) (a + b)

L = (aa + ab+ ba + bb) (a+b)

L= (aaa + aab + aba+ abb + baa+ bab + bba + bbb)

L= { aaa aab aba abb baa bab bba bbb}

$$L = (a + b)\ 5$$
$$L = (a + b)\ (a + b)\ (a + b)\ (a + b)\ (a + b)$$
$$= (aa + ab + ba + bb)\ (aa + ab + ba + bb)\ (a + b)$$

$a(a + b)^*b = a(a^* + b^*)b = \cancel{ab(a^*+b^*)}$

$= (aa^*+ ab^*)b = (aa^*b + ab^*b)$

{ }

(a + b)* a (a + b)*

= (A, a, aa,aaa, aaaa, ……., b, bb, bbb,bbbb,)a

aaaaa

1- aaaaa True

2- aaaaa True

3- aaaaa True

all possible solutions

~~Unique~~ or not unique

Theory of Computation

النظرية الاحتسابية

المحاضرة الخامسة والسادسة

كلية التربية للعلوم الصرفة / جامعة ديالى

قسم علوم الحاسوب – المرحلة الثانية

اعداد

م.د. محمد سامي محمد

REGULAR EXPRESSIONS

EXAMPLE

Another expression that denotes all the words with at least two a's is:

$$b^*ab^*a(a + b)^*$$

We scan through some jungle of b's (or no b's) until we find the first a, then more b's (or no b's), then the second a, then we finish up with anything.

In this set are *abbbabb* and aaaaa.

We can write:

$$(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$$

where by the equal sign we do not mean that these expressions are equal algebraically in the same way as

x+x=2x

$$(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$$

Are they equal ?

REGULAR EXPRESSIONS

We could write
language ((a + b)*a(a + b)*a(a + b)*)
= language (b*ab*a (a + b)*)
= all words with at least two a's.

To be careful about this point, we say that two regular expressions are equivalent if they describe the same language.

(a + b) * ab * ab *
↑ ↑
next to last *a*
last *a*

and

b*a(a + b)*ab*
↑ ↑
first *a* last *a*

REGULAR EXPRESSIONS

EXAMPLE

If we wanted all the words with *exactly* two a's, we could use the expression

b*ab*ab*

which describes such words as *aab*, *baba*, and *bbbabbbab*.

To make the word *aab*, we let the first and second b* become Λ and the last becomes b.

b*ab*ab*

Prove is these one of the solutions and how?

aab

baba

and *bbbabbbab*

REGULAR EXPRESSIONS

EXAMPLE

The language of all words that have at least one a and at least one b is somewhat trickier. If we write

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

$$= (\text{arbitrary})\; a(\text{arbitrary})\; b(\text{arbitrary})$$

we could define this set by the expression:

(a+b)*a(a+b)*b(a+b)* + (a+b)*b(a+b)*a(a+b)*

Here we are still using the plus sign in the general sense of disjunction (or). We are taking the union of two sets,

(a + b)*a(a + b)* b(a + b)* One a and one b always

(a+b)*a(a+b)*b(a+b)* + (a+b)*b(a+b)*a(a+b)* Are they equal ?

REGULAR EXPRESSIONS

يمكن التعبير عن الصيغ بصيغ اخرى مشابهة

EXAMPLE

All temptation to treat these language-defining expressions as if they were algebraic polynomials should be dispelled by these equivalences:

(a+b)* = (a+b)* + (a+b)*
(a+b)* = (a+b)* (a+b)*

(a+b)* = a(a+b)* + b(a+b)* + Λ
(a+b)* = (a+b)* ab(a+b)* + b*a*

$(a+b)^* = (a+b)^* + (a+b)^*$

$(a+b)^* = (a+b)^* (a+b)^*$

Check these?

$(a+b)^* = a(a+b)^* + b(a+b)^* + \Lambda$

$(a+b)^* = (a+b)^* ab(a+b)^* + b^*a^*$

REGULAR EXPRESSIONS

Usually when we employ the star operator, we are defining an infinite language.
We can represent a finite language by using the plus (union sign) alone. If the language L over the alphabet X = {a, *b*} contains only the finite list of words given below,

$$L = \{abba\ baaa\ bbbb\}$$

then we can represent L by the symbolic expression

$$L = \text{language (abba + baaa + bbbb)}$$

If L is a finite language that includes the null word Λ, then the expression that defines L must also employ the symbol Λ.
For example, if

$$L = \{\Lambda\ a\ aa\ bbb\}$$

then the symbolic expression for L must be

$$L = \text{language (}\Lambda + a + aa + bbb\text{)}$$

L = {abba baaa bbbb}

What is Regular Expression of this language?

REGULAR EXPRESSIONS

Let us reconsider the language
T = {a c ab cb abb cbb ... }.

T can be defined as above by

$$(a + c)b^*$$

but it can also be defined by

$$ab^* + cb^*$$

This is another example of the distributive law.

T = {a c ab cb abb cbb ... }.

The distributive law

(a + c)b*

REGULAR EXPRESSIONS

If r1 = aa + b then the expression r1* technically refers to the expression

$$r1^* = aa + b^*$$

which is the formal concatenation of the symbols for r, with the symbol *, but what we generally mean when we write r1* is actually (r1)*

$$(r1)^* = (aa + b)^*$$

r1 = aa + b

r1*=

REGULAR EXPRESSIONS

DEFINITION

If S and T are sets of strings of letters (whether they are finite or infinite sets), we define the product set of strings of letters to be

***ST* = {all combinations of a string from S concatenated with a string from T }**

EXAMPLE

If

$$S = \{a\ aa\ aaa\} \text{ and } T = \{bb\ bbb\}$$

then

$$ST = \{abb\ abbb\ aabb\ aabbb\ aaabb\ aaabbb\}$$

EXAMPLE

If $S = \{a\ bb\ bab\}$ $T = \{a\ ab\}$

Then $ST = \{aa\ aab\ bba\ bbab\ baba\ babab\}$

S* = {a *aa aaa} and *T* = *{bb bbb}

Then ST = {abb abbb aabb aabbb aaabb aaabbb}

If*S* = *{a bb bab} T = {a ab}*

Then *ST* =

REGULAR EXPRESSIONS

EXAMPLE

If $P = \{a\ bb\ bab\}$ and $Q = \{\Lambda\ bbbb\}$

then

$PQ = \{ a\ bb\ bab\ abbbb\ bbbbbb\ babbbbb\}$

ملاحظة :- فقط في حالة الصفر موجود في المجموعتين يكون في الناتج صفر

EXAMPLE

If $M = \{\Lambda$ x $xx\}$ N-$=\{\Lambda$ y yy yyy yyyy...$\}$

then

$MN = \{\Lambda$ y yy yyy $yyyy$ **...**

x xy xyy xyyy xyyyy...

xx **xxy** xxyy xxyyy xxyyyy.. **.}.**

If $P = \{a\ bb\ bab\}$ *and* $Q = \{\mathbf{\Lambda}\ bbbb\}$

Then

$PQ =$

If $M = \{\mathbf{\Lambda}\ x\ xx\}$

$N = \{\mathbf{\Lambda}\ y\ yy\ yyy\ yyyy...\}$

Then MN =

REGULAR EXPRESSIONS

Using regular expressions, these four examples can be written as:

(a + aa + aaa)(bb + bbb) = abb + abbb + aabb + aabbb + aaabb + aaabbb

(a + bb + bab)(a + ab) = aa + aab + bba + bbab + baba + babab

(a + bb + bab)(A + bbbb) = a+bb+bab+abbbbb +bbbbbb + babbbbb

(Λ + x + xx)(y*) = y* + xy* + xxy*

REGULAR EXPRESSIONS

THEOREM 5
If L is a finite language (a language with only finitely many words), then L can be defined by a regular expression.

PROOF

For example, the regular expression that defines the language

$$L = \{baa\ abbba\ bababa\}$$

is

baa + abbba + bababa

Another example If

$$L = \{aa\ ab\ ba\ bb\}$$

the algorithm described above gives the regular expression

aa + ab + ba + bb

Another regular expression that defines this language is

(a + b)(a + b)

L = {baa abbba bababa}

R.E.= baa + abbba + bababa

L = {aa ab ba bb}

R.E.= aa + ab + ba + bb

REGULAR EXPRESSIONS

EXAMPLE

Let

$$L = \{\Lambda \text{ x xx } xxx \text{ } xxxx \text{ } xxxxx\}$$

The regular expression we get from the theorem is

$$\Lambda + x + xx + xxx + xxxx + xxxxx$$

A more elegant regular expression for this language is

$$(\Lambda + x)^5$$

Of course the 5 is, strictly speaking, not a legal symbol for a regular expression although we all understand it means

$$(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)$$

$L = \{\Lambda\ x\ xx\ xxx\ xxxx\ xxxxx\}$

$\Lambda + x + xx + xxx + xxxx + xxxxx$

$(\Lambda + x)^5$

REGULAR EXPRESSIONS

EXAMPLE

Consider the expression:

$$(a + b)^*(aa + bb)(a + b)^*$$

This is the set of strings of a's and b's that at some point contain a double letter. We can think of it as

(arbitrary)(double letter)(arbitrary)

Example are: Λ a *b ab ba aba bab abab baba* .... The expression (ab)* covers all of these except those that begin with b or end in a. Adding these choices gives us the regular expression

(Λ+ b) (ab)* (Λ+ a)

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression below:

$$E = (a + b)^*a(a + b)^* (a + \Lambda) (a + b)^*a(a + b)^*$$

= (arbitrary) a (arbitrary) [a or nothing] (arbitrary) a (arbitrary).

$$= (a+b)^*a(a+b)^*a(a+b)^*a(a+b)^*+(a + b)^*a(a + b)^* \Lambda(a + b)^*a(a + b)^*$$

Before we analyze the second term let us make the observation that

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

Simplify it?

$$(a + b^*)^* (aa + ab^*)^* ((a + bbba^*) + ba^*b)^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression:

$$(a^*b^*)^*$$

The language defined by this expression is all strings that can be made up of factors of the form a*b*, but since both the single letter a and the single letter b are words of the form a*b*, this language contains all strings of a's and b's. It cannot contain more than everything, so

$$(a^*b^*)^* = (a + b)^*$$

Prove it?

$$(a^*b^*)^* = (a + b)^*$$

REGULAR **EXPRESSIONS**

EXAMPLE

One very interesting example, which we consider now in great detail is

$$E = [aa + bb + (ab+ba)(aa+bb)^*(ab+ba)]^*$$

This regular expression represents the collection of all words that are made up of "syllables" of three types:

typeI = aa

type2 = bb

type3 = (ab + ba)(aa + bb)*(ab + *ba)*

$$E = [\text{type}, + \text{type2} + \text{type 3}]^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the language defined by the regular expression:

$$b^*(abb^*)^*(\Lambda + a)$$

This is the language of all words without a double a.

Theory of Computation

النظرية الاحتسابية

المحاضرة الخامسة والسادسة

كلية التربية للعلوم الصرفة / جامعة ديالى

قسم علوم الحاسوب – المرحلة الثانية

اعداد

م.د. محمد سامي محمد

REGULAR EXPRESSIONS

EXAMPLE

Another expression that denotes all the words with at least two a's is:

$$b^*ab^*a(a + b)^*$$

We scan through some jungle of b's (or no b's) until we find the first a, then more b's (or no b's), then the second a, then we finish up with anything.

In this set are *abbbabb* and aaaaa.

We can write:

$$(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$$

where by the equal sign we do not mean that these expressions are equal algebraically in the same way as

x+x=2x

$$(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$$

Are they equal ?

REGULAR EXPRESSIONS

We could write
language ((a + b)*a(a + b)*a(a + b)*)
= language (b*ab*a (a + b)*)
= all words with at least two a's.

To be careful about this point, we say that two regular expressions are equivalent if they describe the same language.

(a + b) * ab * ab *
↑ ↑
next to last *a*
last *a*

and

b*a(a + b)*ab*
↑ ↑
first *a* last *a*

REGULAR EXPRESSIONS

EXAMPLE

If we wanted all the words with *exactly* two a's, we could use the expression

b*ab*ab*

which describes such words as *aab*, *baba*, and *bbbabbbab*.

To make the word *aab*, we let the first and second b* become Λ and the last becomes b.

b*ab*ab*

Prove is these one of the solutions and how?

aab

baba

and *bbbabbbab*

REGULAR EXPRESSIONS

EXAMPLE

The language of all words that have at least one a and at least one b is somewhat trickier. If we write

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

$$= (\text{arbitrary})\; a(\text{arbitrary})\; b(\text{arbitrary})$$

we could define this set by the expression:

(a+b)*a(a+b)*b(a+b)* + (a+b)*b(a+b)*a(a+b)*

Here we are still using the plus sign in the general sense of disjunction (or). We are taking the union of two sets,

(a + b)*a(a + b)* b(a + b)* One a and one b always

(a+b)*a(a+b)*b(a+b)* + (a+b)*b(a+b)*a(a+b)* Are they equal ?

REGULAR EXPRESSIONS

يمكن التعبير عن الصيغ بصيغ اخرى مشابهة

EXAMPLE

All temptation to treat these language-defining expressions as if they were algebraic polynomials should be dispelled by these equivalences:

(a+b)* = (a+b)* + (a+b)*
(a+b)* = (a+b)* (a+b)*

(a+b)* = a(a+b)* + b(a+b)* + Λ
(a+b)* = (a+b)* ab(a+b)* + b*a*

$(a+b)^* = (a+b)^* + (a+b)^*$

$(a+b)^* = (a+b)^* (a+b)^*$

Check these?

$(a+b)^* = a(a+b)^* + b(a+b)^* + \Lambda$

$(a+b)^* = (a+b)^* ab(a+b)^* + b^*a^*$

REGULAR EXPRESSIONS

Usually when we employ the star operator, we are defining an infinite language.
We can represent a finite language by using the plus (union sign) alone. If the language L over the alphabet X = {a, *b*} contains only the finite list of words given below,

$$L = \{abba\ baaa\ bbbb\}$$

then we can represent L by the symbolic expression

$$L = \text{language (abba + baaa + bbbb)}$$

If L is a finite language that includes the null word Λ, then the expression that defines L must also employ the symbol Λ.
For example, if

$$L = \{\Lambda\ a\ aa\ bbb\}$$

then the symbolic expression for L must be

$$L = \text{language (}\Lambda + a + aa + bbb\text{)}$$

L = {abba baaa bbbb}

What is Regular Expression of this language?

REGULAR EXPRESSIONS

Let us reconsider the language
T = {a c ab cb abb cbb ... }.

T can be defined as above by

$$(a + c)b^*$$

but it can also be defined by

$$ab^* + cb^*$$

This is another example of the distributive law.

T = {a c ab cb abb cbb ... }.

The distributive law

(a + c)b*

REGULAR EXPRESSIONS

If r1 = aa + b then the expression r1* technically refers to the expression

$$r1^* = aa + b^*$$

which is the formal concatenation of the symbols for r, with the symbol *, but what we generally mean when we write r1* is actually (r1)*

$$(r1)^* = (aa + b)^*$$

r1 = aa + b

r1*=

REGULAR EXPRESSIONS

DEFINITION

If S and T are sets of strings of letters (whether they are finite or infinite sets), we define the product set of strings of letters to be

***ST* = {all combinations of a string from S concatenated with a string from T }**

EXAMPLE

If

$$S = \{a\ aa\ aaa\} \text{ and } T = \{bb\ bbb\}$$

then

$$ST = \{abb\ abbb\ aabb\ aabbb\ aaabb\ aaabbb\}$$

EXAMPLE

If $S = \{a\ bb\ bab\}$ $T = \{a\ ab\}$

Then $ST = \{aa\ aab\ bba\ bbab\ baba\ babab\}$

S* = {a *aa aaa} and *T* = *{bb bbb}

Then ST = {abb abbb aabb aabbb aaabb aaabbb}

If*S* = *{a bb bab} T = {a ab}*

Then *ST* =

REGULAR EXPRESSIONS

EXAMPLE

If $P = \{a\ bb\ bab\}$ and $Q = \{\Lambda\ bbbb\}$

then

$PQ = \{ a\ bb\ bab\ abbbb\ bbbbbb\ babbbbb\}$

ملاحظة :- فقط في حالة الصفر موجود في المجموعتين يكون في الناتج صفر

EXAMPLE

If $M = \{\Lambda$ x $xx\}$ N-$=\{\Lambda$ y yy yyy yyyy...$\}$

then

$MN = \{\Lambda$ y yy yyy $yyyy$ **...**

x xy xyy xyyy xyyyy...

xx **xxy** xxyy xxyyy xxyyyy.. **.}.**

If $P = \{a\ bb\ bab\}$ *and* $Q = \{\mathbf{\Lambda}\ bbbb\}$

Then

$PQ =$

If $M = \{\mathbf{\Lambda}\ x\ xx\}$

$N = \{\mathbf{\Lambda}\ y\ yy\ yyy\ yyyy...\}$

Then MN =

REGULAR EXPRESSIONS

Using regular expressions, these four examples can be written as:

(a + aa + aaa)(bb + bbb) = abb + abbb + aabb + aabbb + aaabb + aaabbb

(a + bb + bab)(a + ab) = aa + aab + bba + bbab + baba + babab

(a + bb + bab)(A + bbbb) = a+bb+bab+abbbbb +bbbbbb + babbbbb

(Λ + x + xx)(y*) = y* + xy* + xxy*

REGULAR EXPRESSIONS

THEOREM 5
If L is a finite language (a language with only finitely many words), then L can be defined by a regular expression.

PROOF

For example, the regular expression that defines the language

$$L = \{baa\ abbba\ bababa\}$$

is

baa + abbba + bababa

Another example If

$$L = \{aa\ ab\ ba\ bb\}$$

the algorithm described above gives the regular expression

aa + ab + ba + bb

Another regular expression that defines this language is

(a + b)(a + b)

L = {baa abbba bababa}

R.E.= baa + abbba + bababa

L = {aa ab ba bb}

R.E.= aa + ab + ba + bb

REGULAR EXPRESSIONS

EXAMPLE

Let

$$L = \{\Lambda \text{ x xx } xxx \text{ } xxxx \text{ } xxxxx\}$$

The regular expression we get from the theorem is

$$\Lambda + x + xx + xxx + xxxx + xxxxx$$

A more elegant regular expression for this language is

$$(\Lambda + x)^5$$

Of course the 5 is, strictly speaking, not a legal symbol for a regular expression although we all understand it means

$$(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)(\Lambda + x)$$

$L = \{\Lambda\ x\ xx\ xxx\ xxxx\ xxxxx\}$

$\Lambda + x + xx + xxx + xxxx + xxxxx$

$(\Lambda + x)^5$

REGULAR EXPRESSIONS

EXAMPLE

Consider the expression:

$$(a + b)^*(aa + bb)(a + b)^*$$

This is the set of strings of a's and b's that at some point contain a double letter. We can think of it as

(arbitrary)(double letter)(arbitrary)

Example are: Λ a *b ab ba aba bab abab baba* .... The expression (ab)* covers all of these except those that begin with b or end in a. Adding these choices gives us the regular expression

(Λ+ b) (ab)* (Λ+ a)

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression below:

$$E = (a + b)^*a(a + b)^* (a + \Lambda) (a + b)^*a(a + b)^*$$

= (arbitrary) a (arbitrary) [a or nothing] (arbitrary) a (arbitrary).

$$= (a+b)^*a(a+b)^*a(a+b)^*a(a+b)^*+(a + b)^*a(a + b)^* \Lambda(a + b)^*a(a + b)^*$$

Before we analyze the second term let us make the observation that

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

Simplify it?

$$(a + b^*)^* (aa + ab^*)^* ((a + bbba^*) + ba^*b)^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the regular expression:

$$(a^*b^*)^*$$

The language defined by this expression is all strings that can be made up of factors of the form a*b*, but since both the single letter a and the single letter b are words of the form a*b*, this language contains all strings of a's and b's. It cannot contain more than everything, so

$$(a^*b^*)^* = (a + b)^*$$

Prove it?

$$(a^*b^*)^* = (a + b)^*$$

REGULAR **EXPRESSIONS**

EXAMPLE

One very interesting example, which we consider now in great detail is

$$E = [aa + bb + (ab+ba)(aa+bb)^*(ab+ba)]^*$$

This regular expression represents the collection of all words that are made up of "syllables" of three types:

typeI = aa

type2 = bb

type3 = (ab + ba)(aa + bb)*(ab + *ba)*

$$E = [\text{type}, + \text{type2} + \text{type 3}]^*$$

REGULAR EXPRESSIONS

EXAMPLE

Consider the language defined by the regular expression:

$$b^*(abb^*)^*(\Lambda + a)$$

This is the language of all words without a double a.

Theory of Computation

النظرية الاحتسابية
المحاضرة السابعة

كلية التربية للعلوم الصرفة / جامعة ديالى

اعداد
م.د. محمد سامي محمد

قسم علوم الحاسوب
المرحلة الثانية

FINITE AUTOMATA

Finite Automaton (FA) –"finite" because the number of possible states and number of letters in the alphabet are both finite, and "automaton" because the change of states is totally governed by the input. It

A finite automaton is a collection of three things:

1. A finite set of states, one of which is designated as the initial state, called the start state, and some (maybe none) of which are designated as final states.

2. An alphabet **E** of possible input letters, from which are formed strings, that are to be read one letter at a time.

3. A finite set of transitions that tell for each state and for each letter of the input alphabet which state to go to next.

الدكتورالمهندس محمد سامي محمد

Example

Suppose that the input alphabet has only the two letters a and b.

Let us also assume that there are only three states, x, y, and z. Let the following be the rules of transition:

1. From state x and input a go to state y.
2. From state x and input b go to state z.
3. From state y and input a go to state x.
4. From state y and input b go to state z.
5. From state z and any input stay at state z.

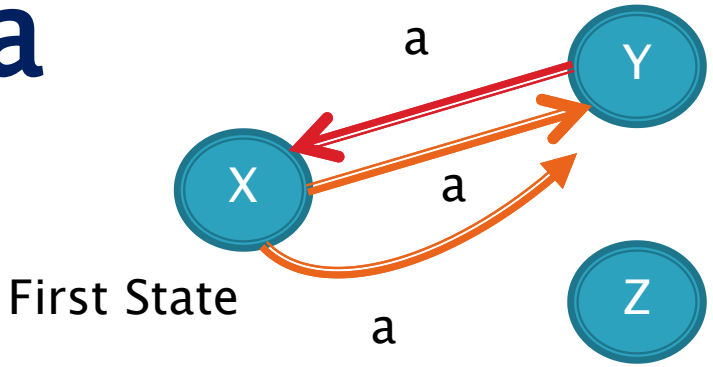
Example Continued

Let us also designate state x as the starting state and state z as the only final state.

We now have a perfectly defined finite automaton, since it fulfills all three requirements demanded above: states, alphabet, transitions.

Let us examine what happens to various input strings when presented to this FA. Let us start with the string aaa.

aaa



Q/ Check if these input are successful aaa

Example Continued

We did not finish up in the final state (state z), so we have an unsuccessful termination of our run.

The string aaa is not in the language of all strings that leave this FA in state z.

The set of all strings that do leave us in a final state is called the language defined **by** the finite automaton.

The input string aaa is not in the language defined by this FA.

Using other terminology, we may say that the string aaa is not accepted by this finite automaton because it does not lead to a final state.

We may also say "aaa is rejected by this FA."

Example Continued

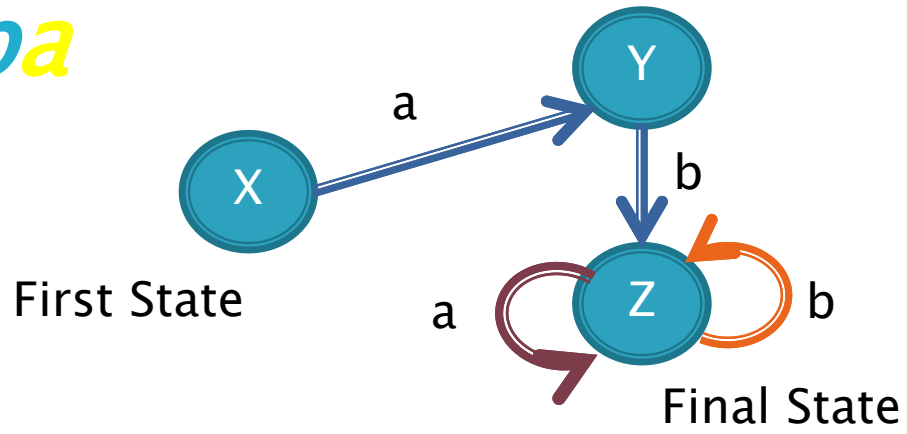
The set of all strings accepted is the language associated with the FA.
We say, this FA accepts the language L, or L is the language accepted by this FA.

we may also say that L is the language of the FA.

Example 2

Let us examine a different input string for this same FA. Let the input be *abba.*

abba



babaa

abbaa

after we have followed the instruction of each input letter we end up in state z.

The input string abba has taken us successfully to the final state.

The string *abba* is therefore a word in the language associated with this FA. The word *abba* is accepted by this FA.

Conclusion

It is not hard for us to predict which strings will be accepted by this FA.

If an input string is made up of only the letter a repeated some number of times, then the action of the FA will be to jump back and forth between state x and state y. No such word can ever be accepted.

To get into state z, it is necessary for the string to have the letter b in it.
As soon as a *b* is encountered in the input string, the FA jumps immediately to state z no matter what state it was in before.

Once in state z, it is impossible to leave. When the input string runs out, the FA will still be in state z, leading to acceptance of the string.

Conclusion

The FA above will accept all strings that have the letter b in them and no other strings.

Therefore, the language associated with (or accepted by) this FA is the one defined by the regular expression

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

It is much simpler to summarize them in a table format.

The **transition table** for the FA we have described is:

State	a	b
Start X	Y	Z
Y	X	Z
Final Z	Z	Z

Conclusion

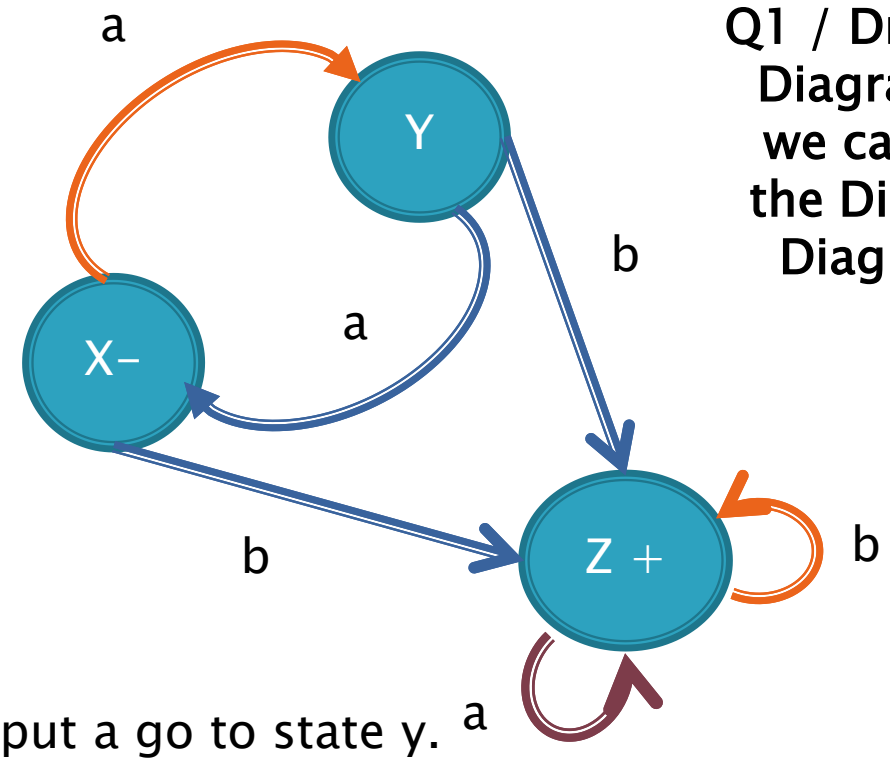
If a certain letter makes a state go back to itself, we indicate this by an arrow that returns to the same circle-this arrow is called a loop.

We can indicate the start state by labeling it with the word "start" or by a minus sign

And the final states by labeling them with the word "final" or plus signs.

The machine we have already defined by the transition list and the transition table can be depicted by the **transition diagram**:

Conclusion



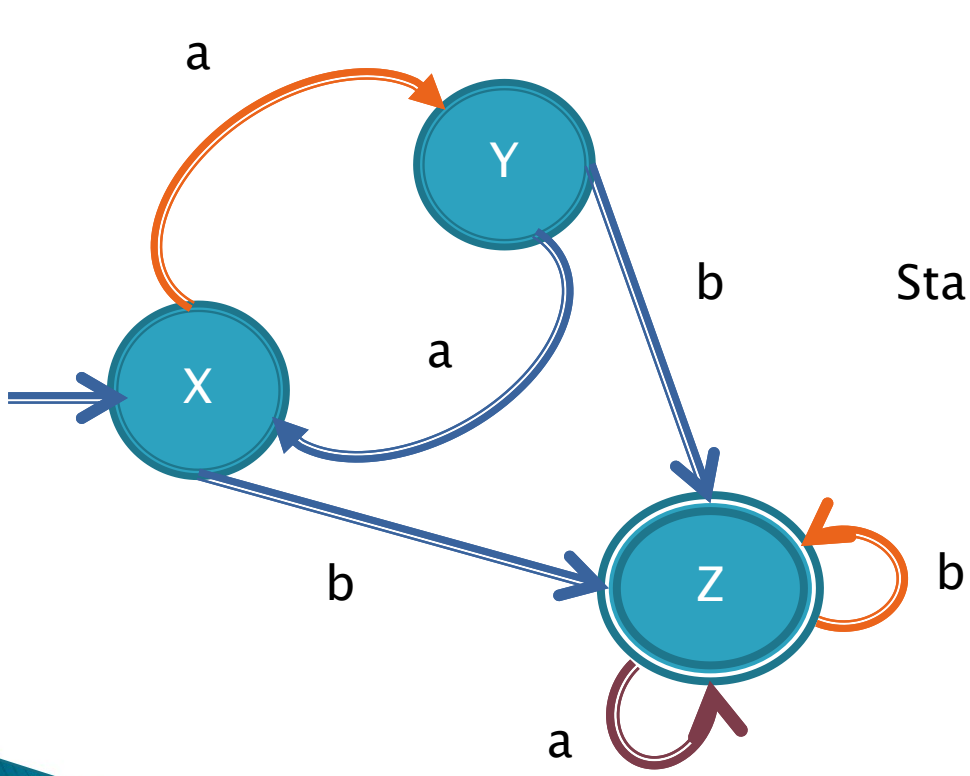
Q1 / Draw the State Diagram of FA or we can say Draw the Directed Edge Diagram of FA?

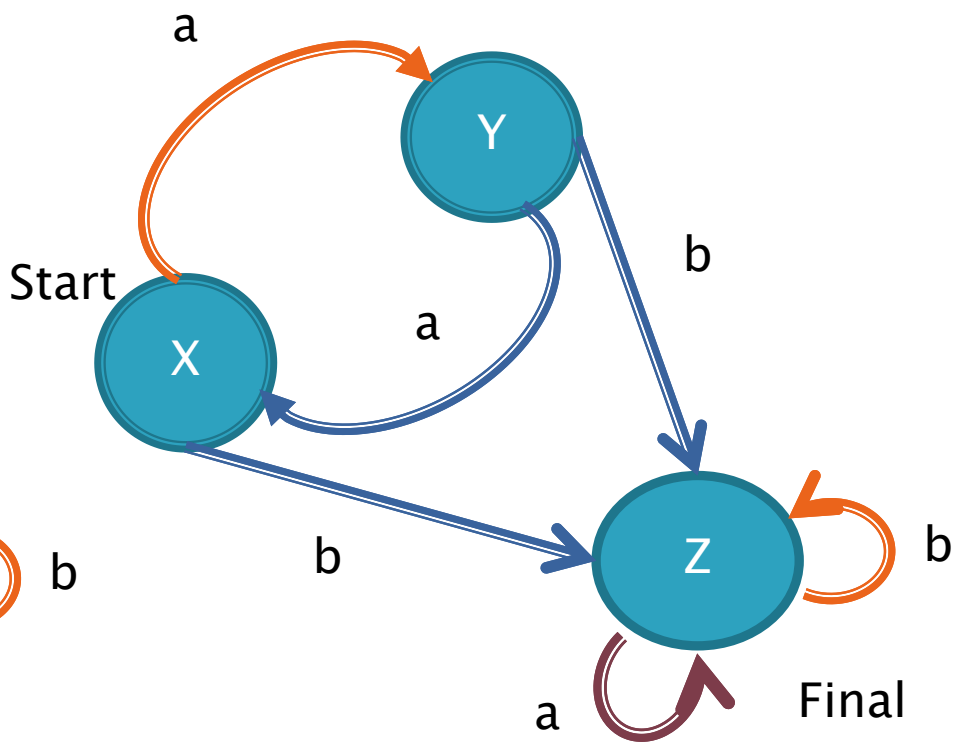
1. From state x and input a go to state y.
2. From state x and input b go to state z.
3. From state y and input a go to state x.
4. From state y and input b go to state z.
5. From state z and any input stay at state z.

Conclusion

Sometimes a start state is indicated by an arrow and a final state by drawing a box or another circle around its circle.

The minus and plus signs, when employed, are drawn inside or outside the state circles. This machine can also be depicted as:





الدكتور المهندس محمد سامي محمد

Conclusion

Every input string can be interpreted as traversing a path beginning at the start state and moving among the states (perhaps visiting the same state many times) and finally settling in some particular rest state.

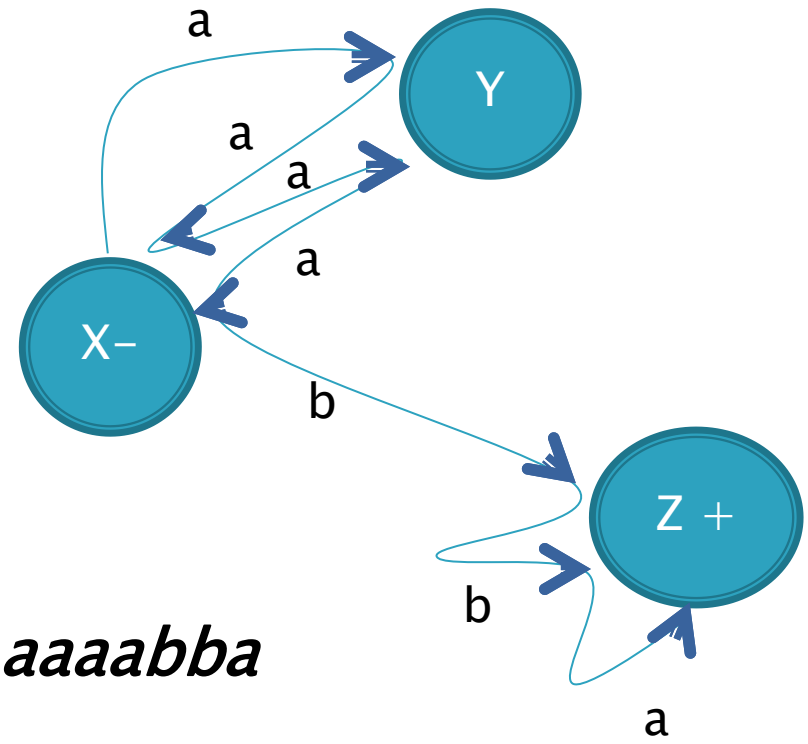
If it is a final state, then the path has ended in success.

The letters of the input string dictate the directions of travel. They are the map and the fuel needed for motion. When we are out of letters we must stop.

Example

Let us look at this machine again and at the paths generated by the input strings *aaaabba* and *bbaabbbb.*

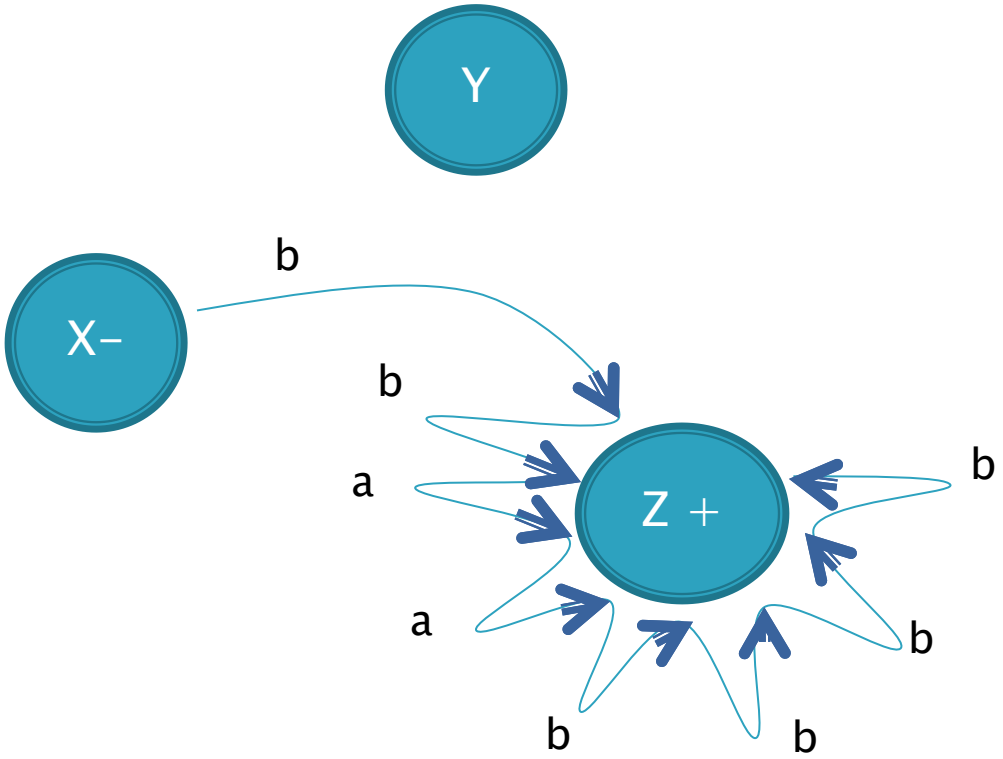
aaaabba



aaaabba

Example

bbaabbbb



Definition

When we depict an FA as circles and arrows, we say that we have drawn a **directed graph**.

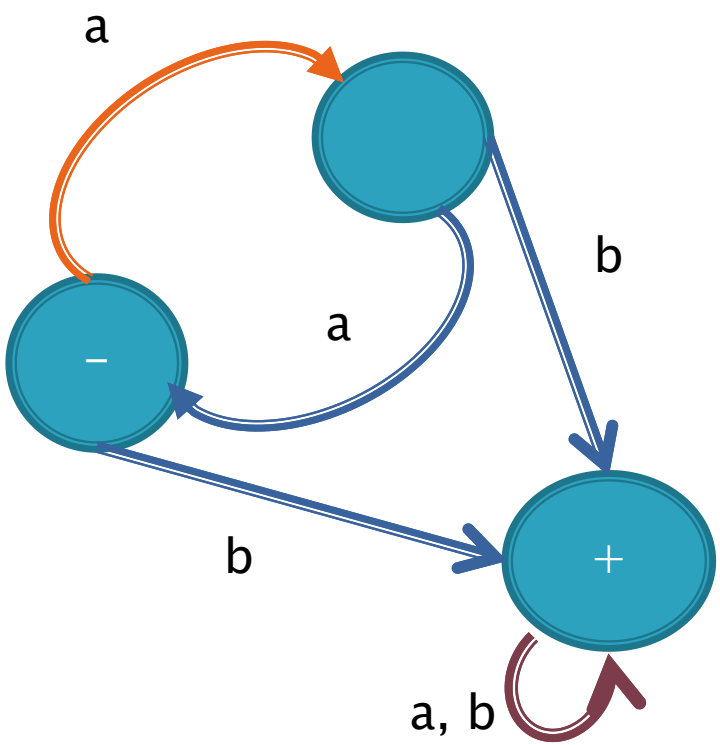
We borrow from Graph Theory the name **directed edge** or simply **edge** for the arrow between states.

Every state has as many **outgoing edges** as there are letters in the alphabet.

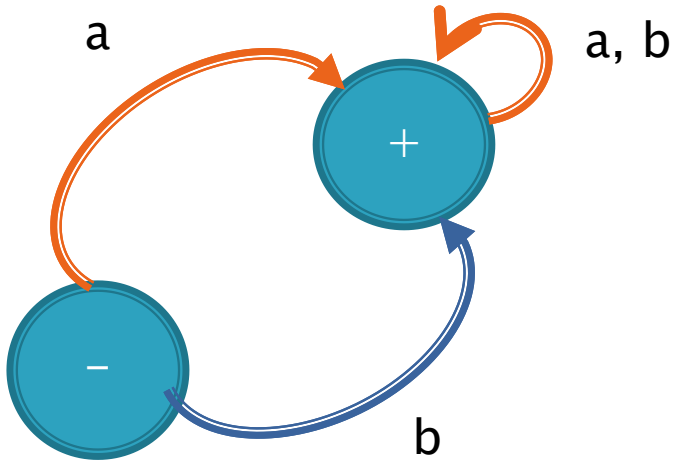
It is possible for a state to have no **incoming edges**.

Unnecessary Names

There are machines for which it is not necessary to give the states specific names. For example, the FA we have been dealing with so far can be represented simply as:

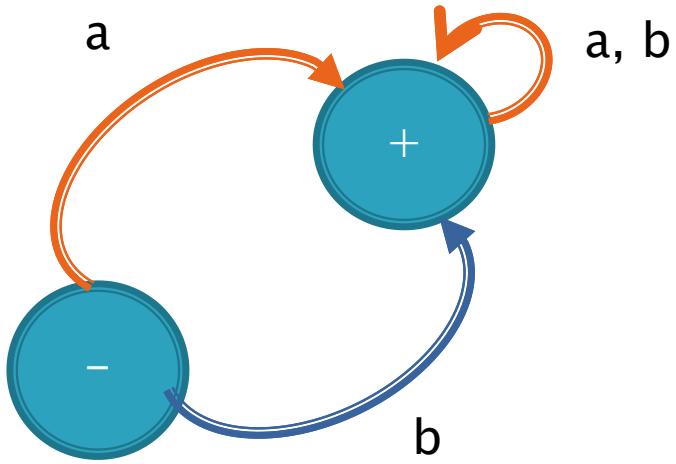


EXAMPLE



In the picture above we have drawn one edge from the state on the right back into itself and given this loop the two labels a and b, separated by a comma meaning that this is the path traveled if either letter is read.

EXAMPLE

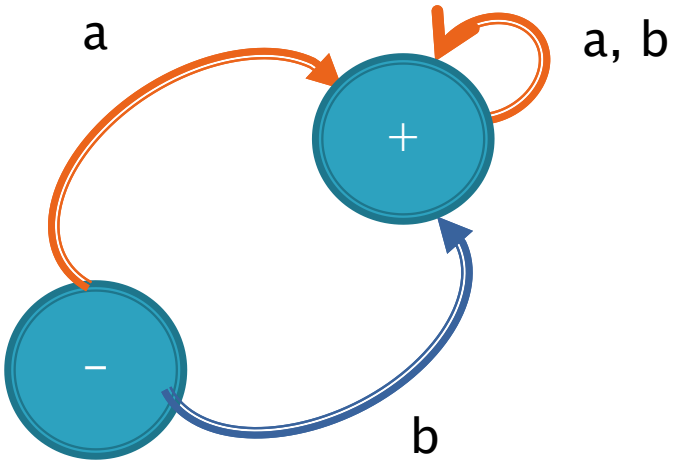


The first letter of the input takes us to the right-hand state and, once there, we are trapped forever.

When the input string runs out, there we are in the correct final state.

This description, however, omits the possibility that the input is the null string A. If the input string is the null string, we are left back at the left-hand state, and we never get to the final state.

EXAMPLE



The language accepted by this machine is the set of all strings except A. This has the regular expression definitions

$$(a + b)(a + b)^* = (a + b)+$$

EXAMPLE

One of the many FA's that accepts all words is:

Here the sign ± means that the same state is both a start and a final state.

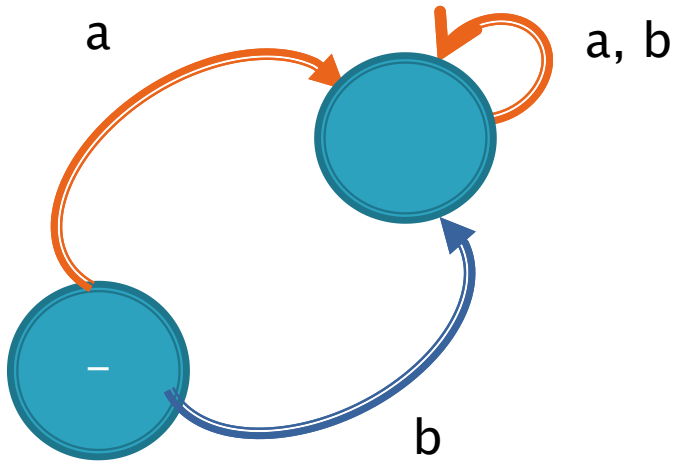
Since there is only one state and no matter what happens we must stay there, the language for this machine is:

$$(a + b)^*$$

EXAMPLE

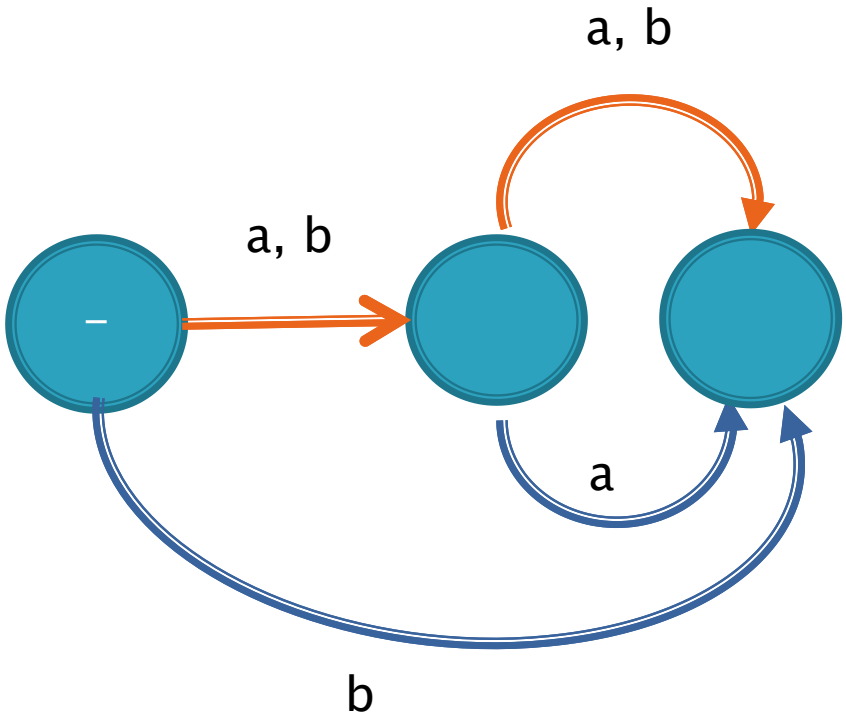
Similarly, there are FA's that accept no language.

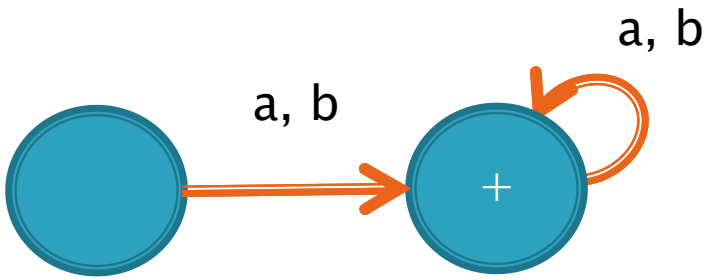
These are of two types:
FA's that have no final states, such as



EXAMPLE

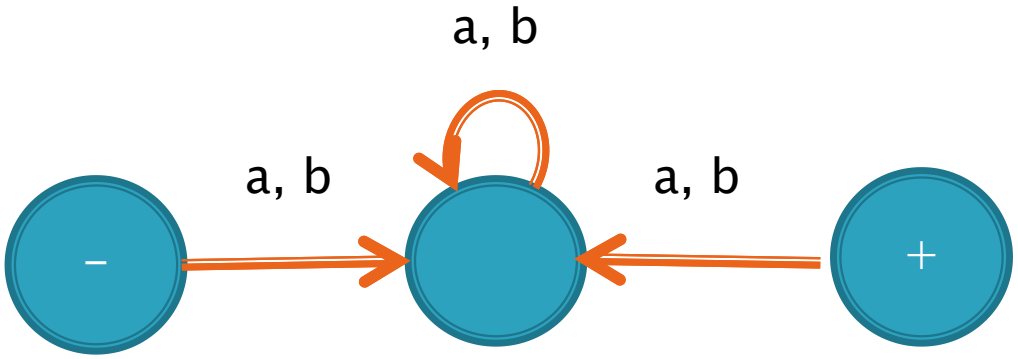
And FA's in which the circles that represent the final states cannot be reached from the start state.





EXAMPLE

(in this case we say that the. graph is disconnected) or for a reason such as that shown below.



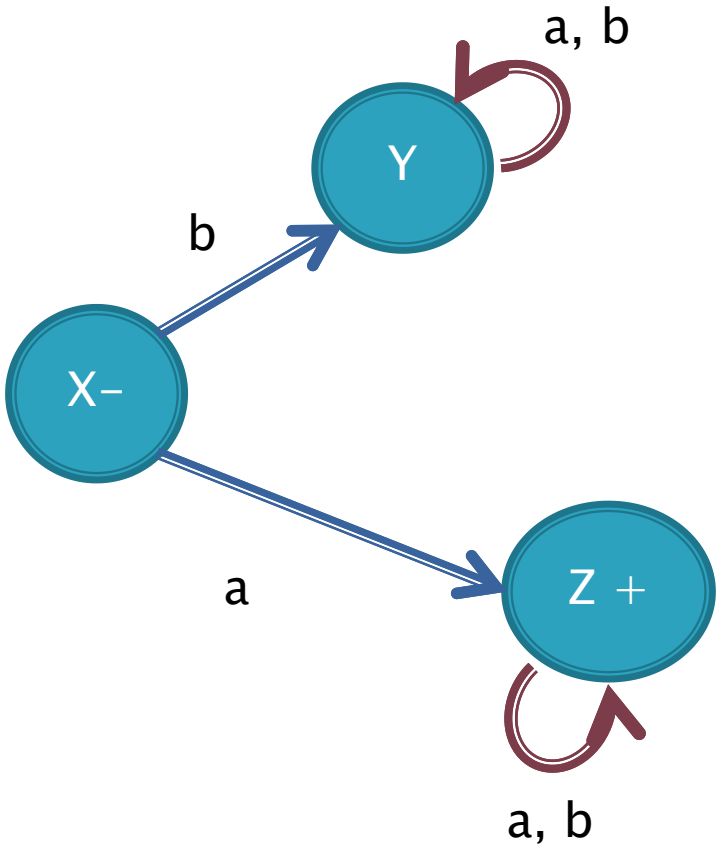
Example of Dead End

Suppose we want to build a finite automaton that accepts all the words in the language

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

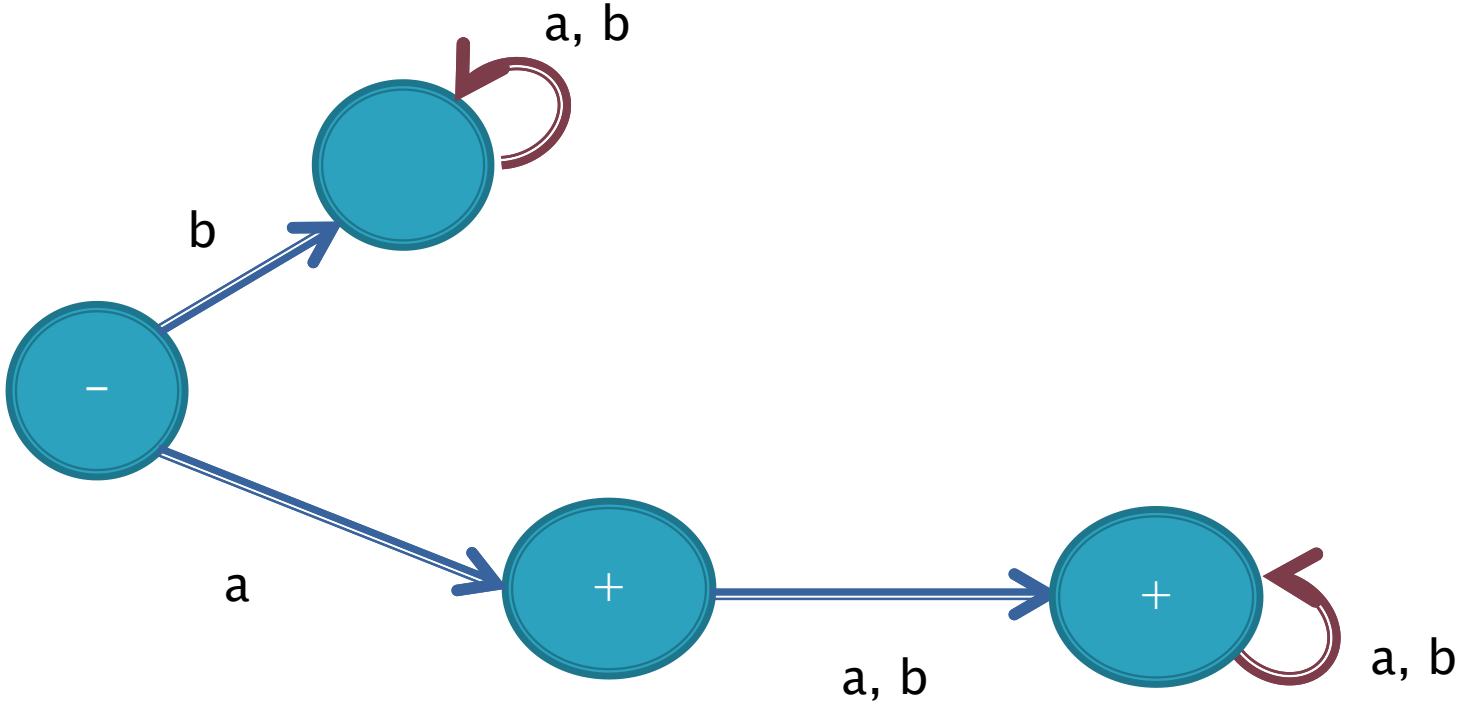
that is, all the strings that begin with the letter a. We start at state x and, if the first letter read is a ***b*** we go to a dead-end state y. (A "dead-end state" is an informal way of describing a state that no string can leave once it has entered.) If the first letter is an a we go to the dead-end state z, where z is a final state. The machine looks like this:

Example of Dead End



Example of Dead End

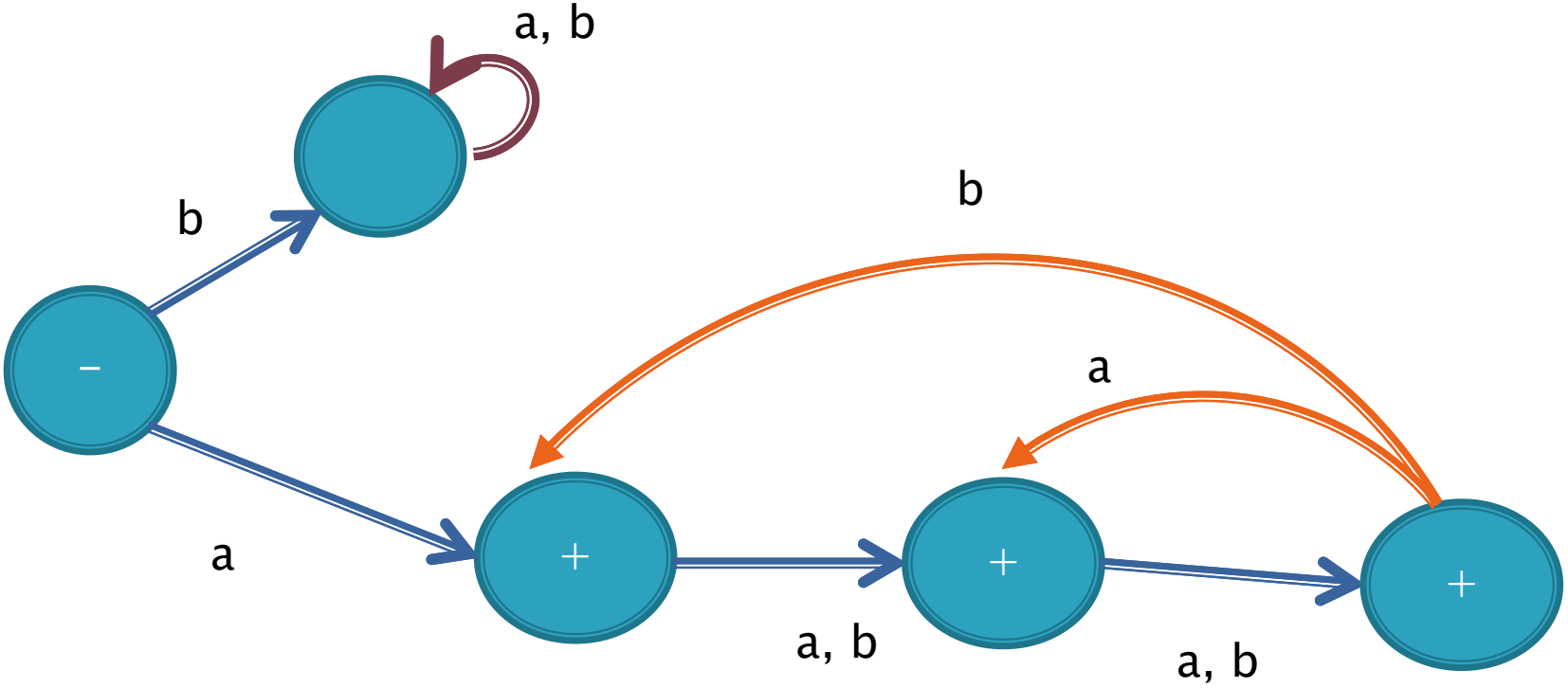
The same language may be accepted by a four-state machine, as below:



Only the word a ends in the first + state. All other words starting with an a reach and finish in the second + state where they are accepted.

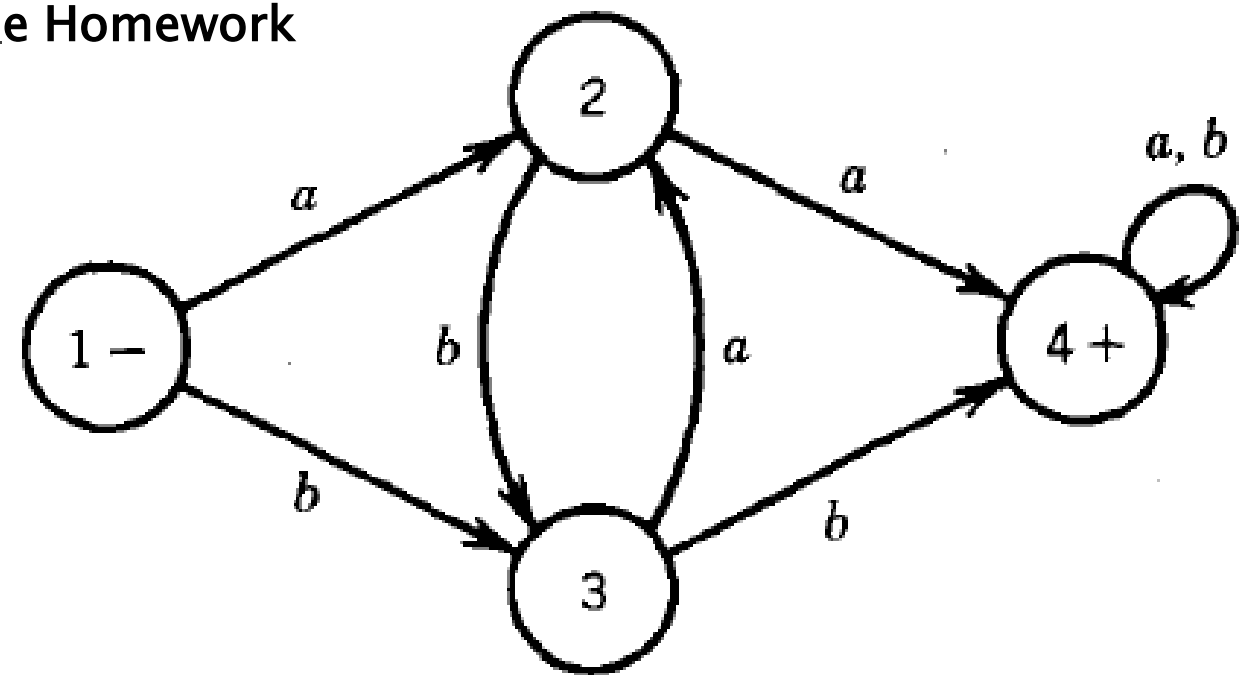
Example of Dead End

This idea can be carried further to a five-state FA as below:



The examples above are FA's that have more than one final state. From them we can see that there **is not a unique machine** for a given language.

Example Homework



ababa is It in language? Is it successful?

babbb is It in language? Is it successful?

abababb is It in language? Is it successful?

aabbabbba is It in language? Is it successful?

الدكتورالمهندس محمد سامي محمد

Example Homework notes

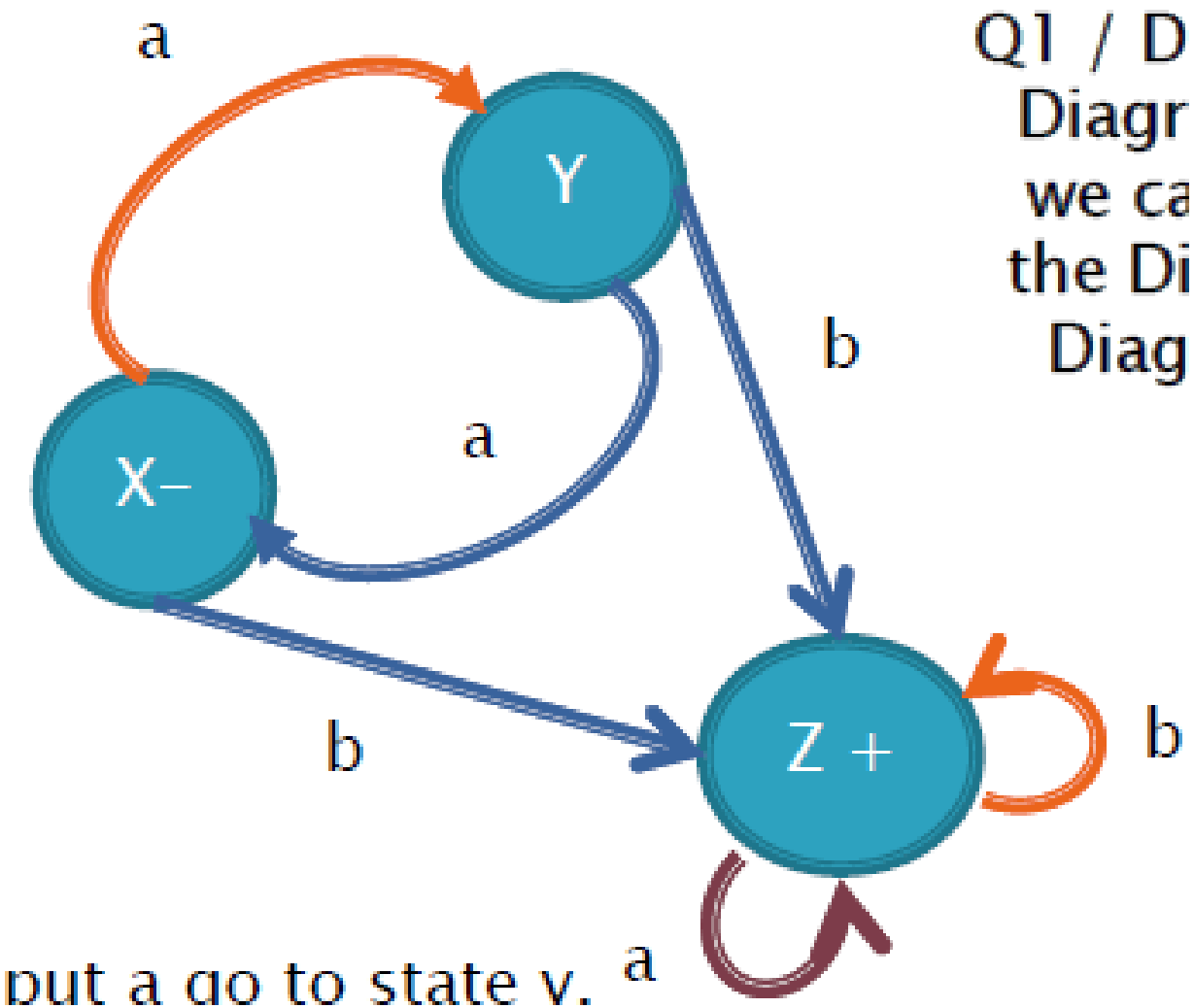
In summary, the words accepted by this machine are exactly those strings that have a double letter in them. This language, as we have seen, can also be defined by the regular expression

$$(a + b)^*(aa + bb)(a + b)^*$$

abba

babaa

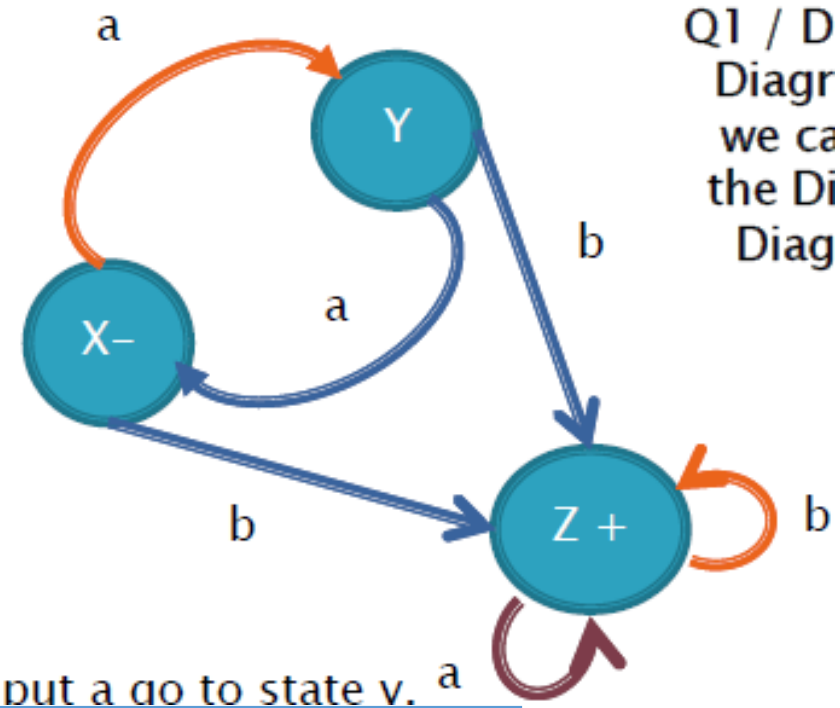
abbaa



Q1 / D
Diagr
we ca
the Di
Diag

nd input a go to state v.

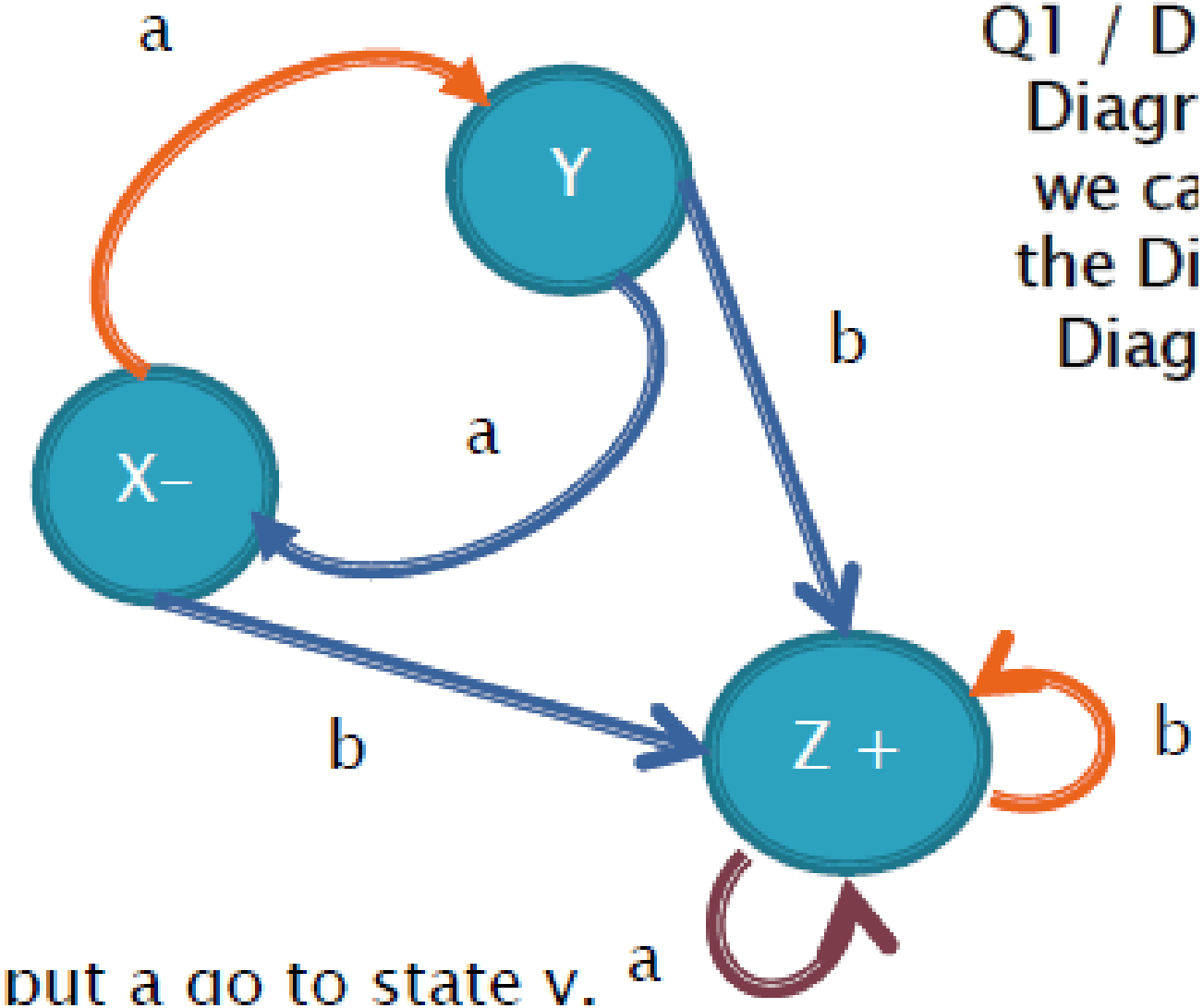
1. From state x and input a go to state y.
2. From state x and input b go to state z.
3. From state y and input a go to state x.
4. From state y and input b go to state z.
5. From state z and any input stay at state z.



Q1 / D
Diagr
we ca
the Di
Diag

ıd input a go to state y.

Previous state	a	b
x	y	z
y	x	z
z	z	z



Q1 / D
Diagr
we ca
the Di
Diag

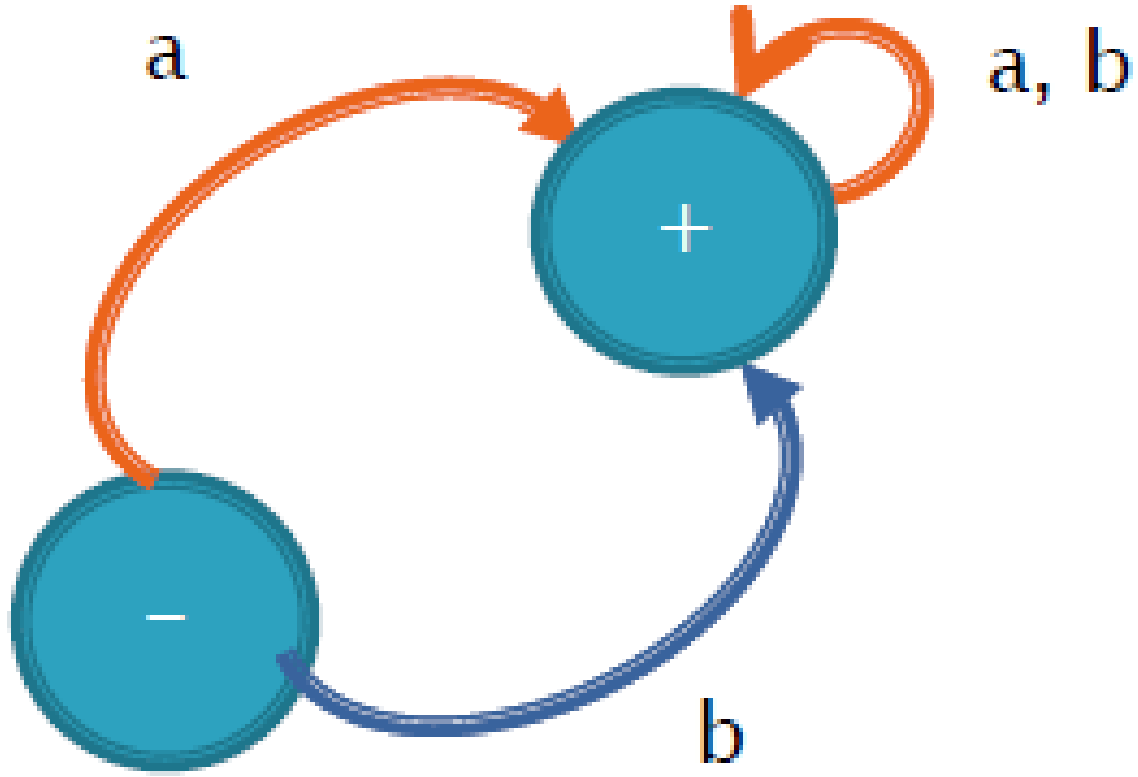
nd input a go to state v.

b

ab

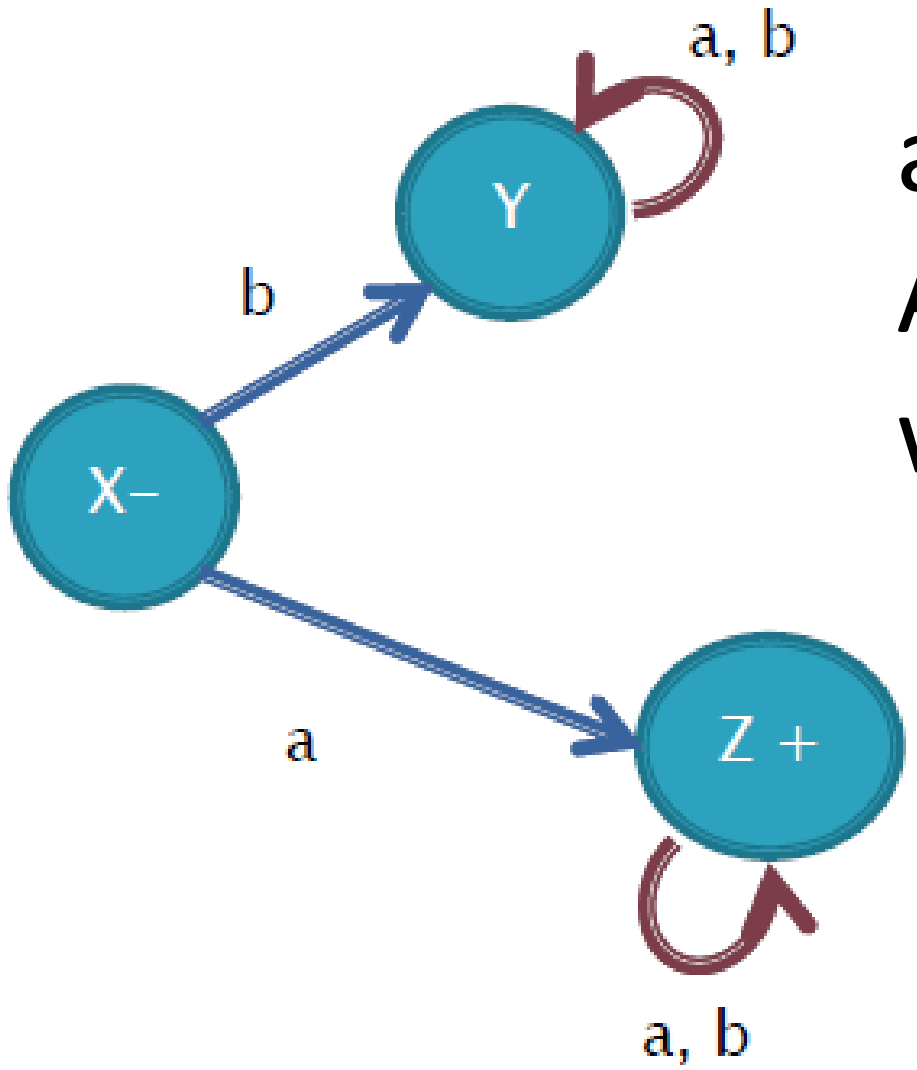
aab

aaab



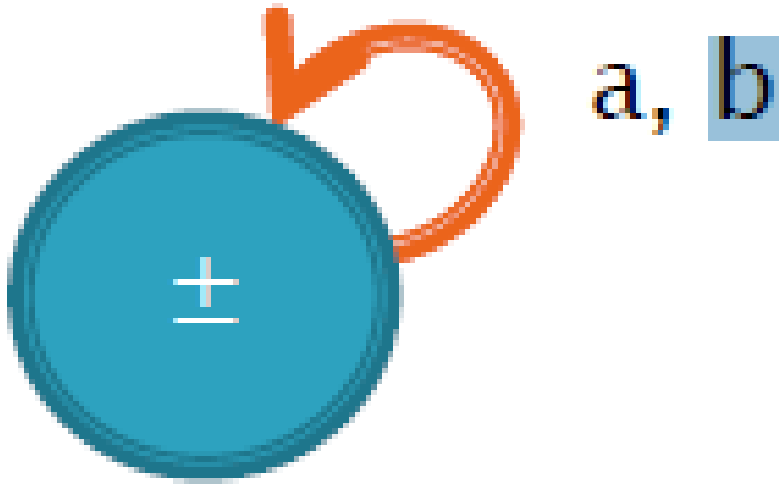
a(a+b)*

b(a+b)*

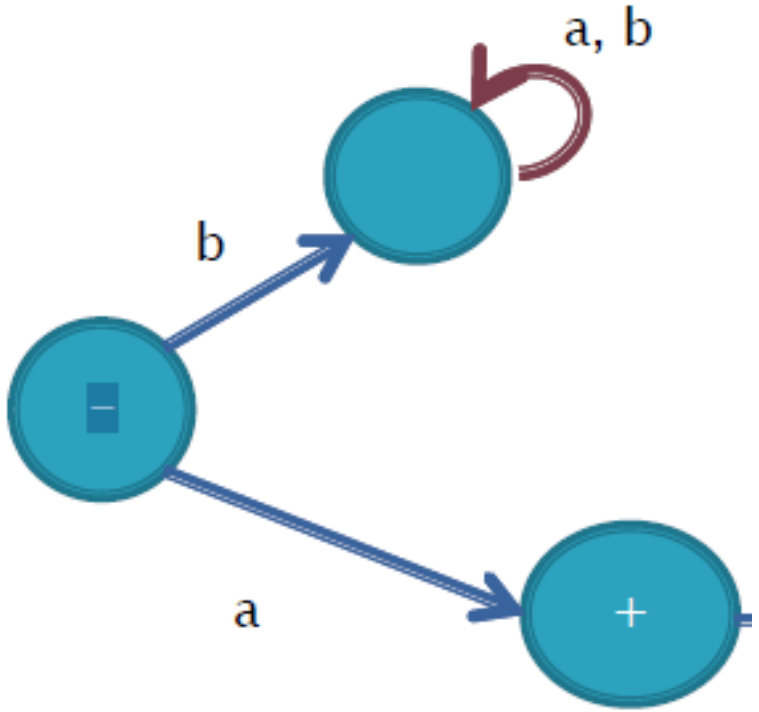


a(a+b)*

All words that start with a is accepted



(a+b)*

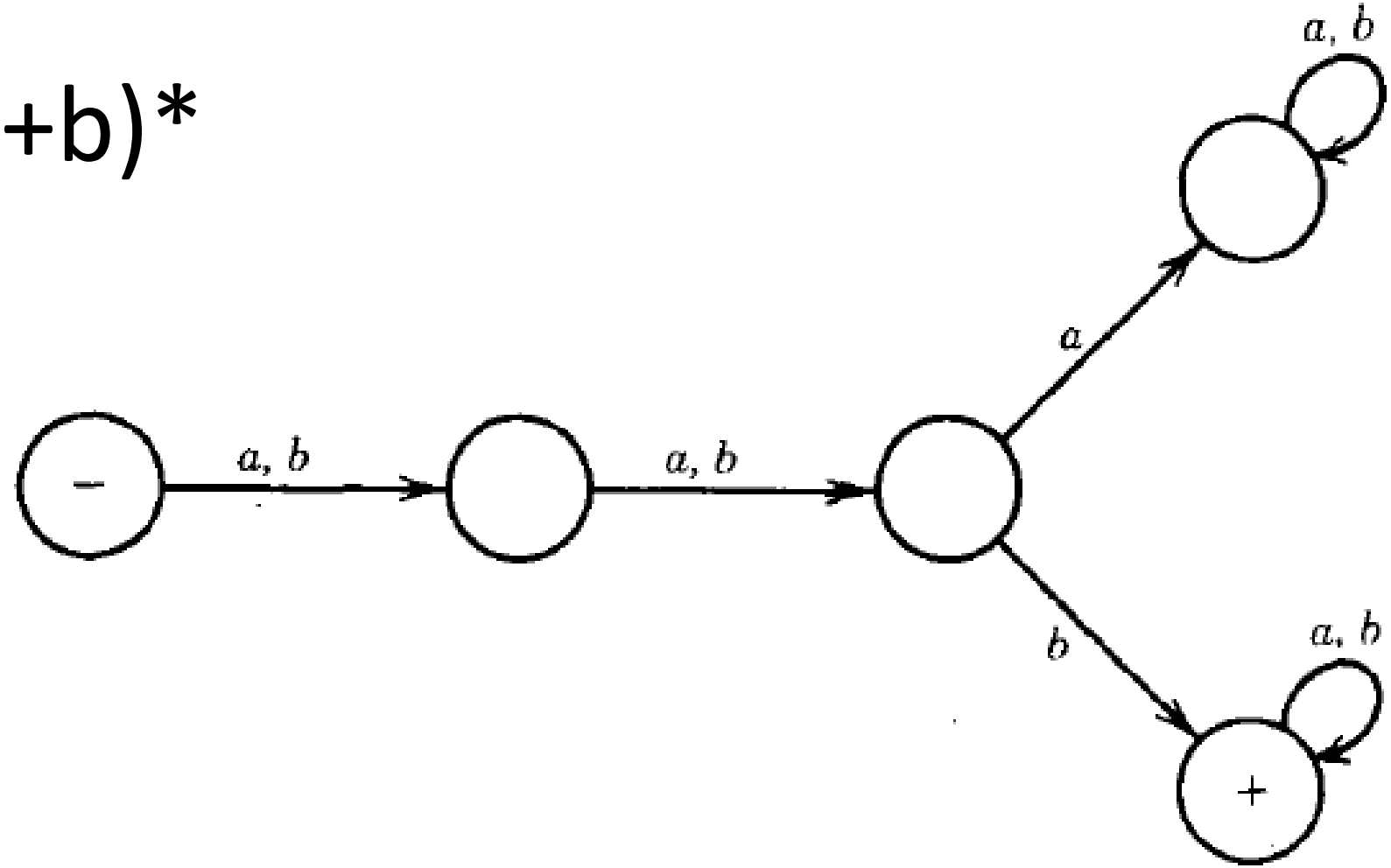


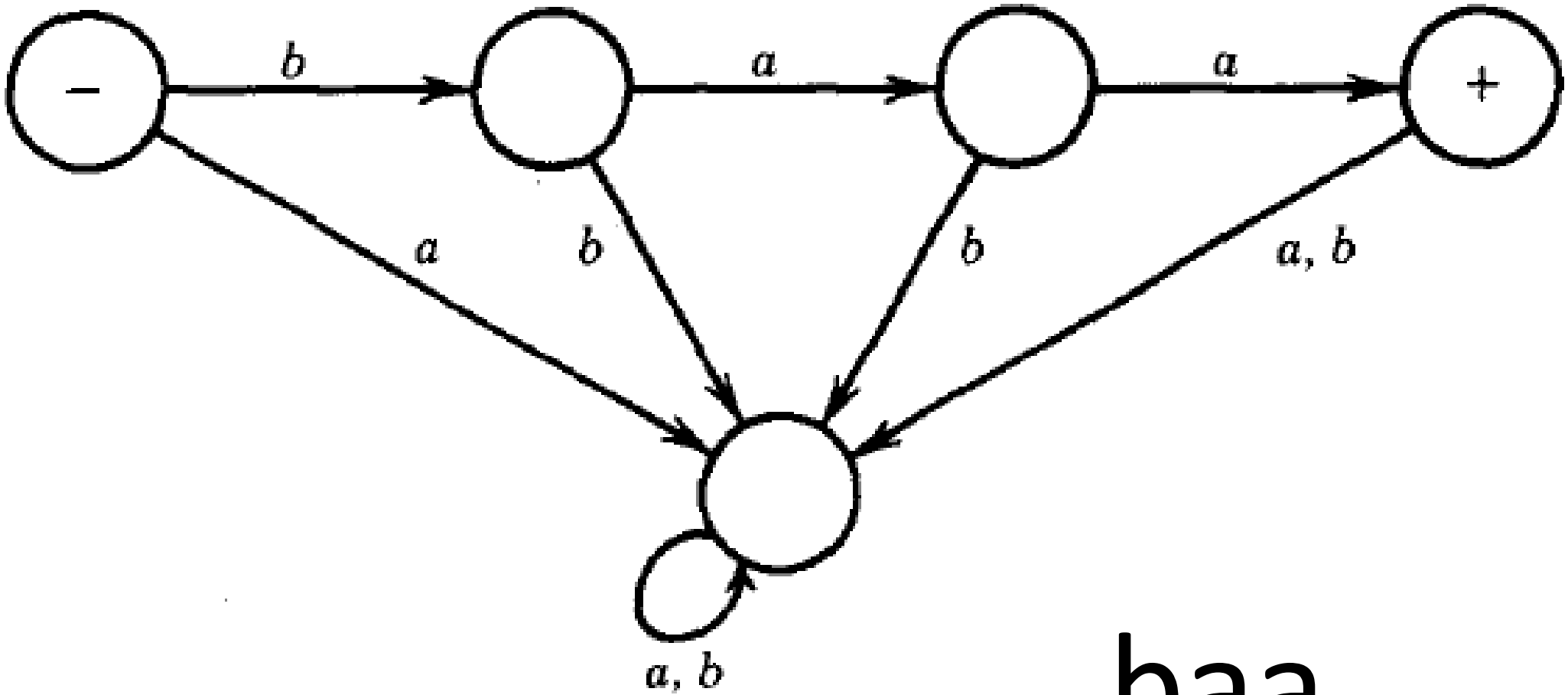
b(a+b)*

a

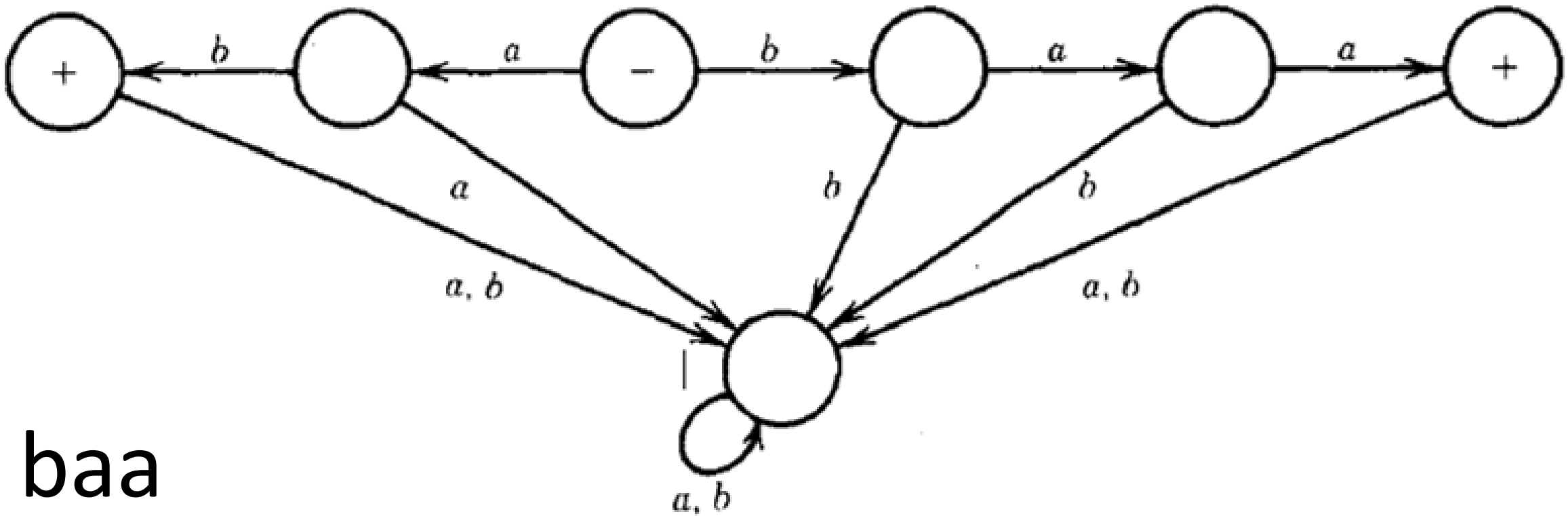
A

(a+b)(a+b)b(a+b)*





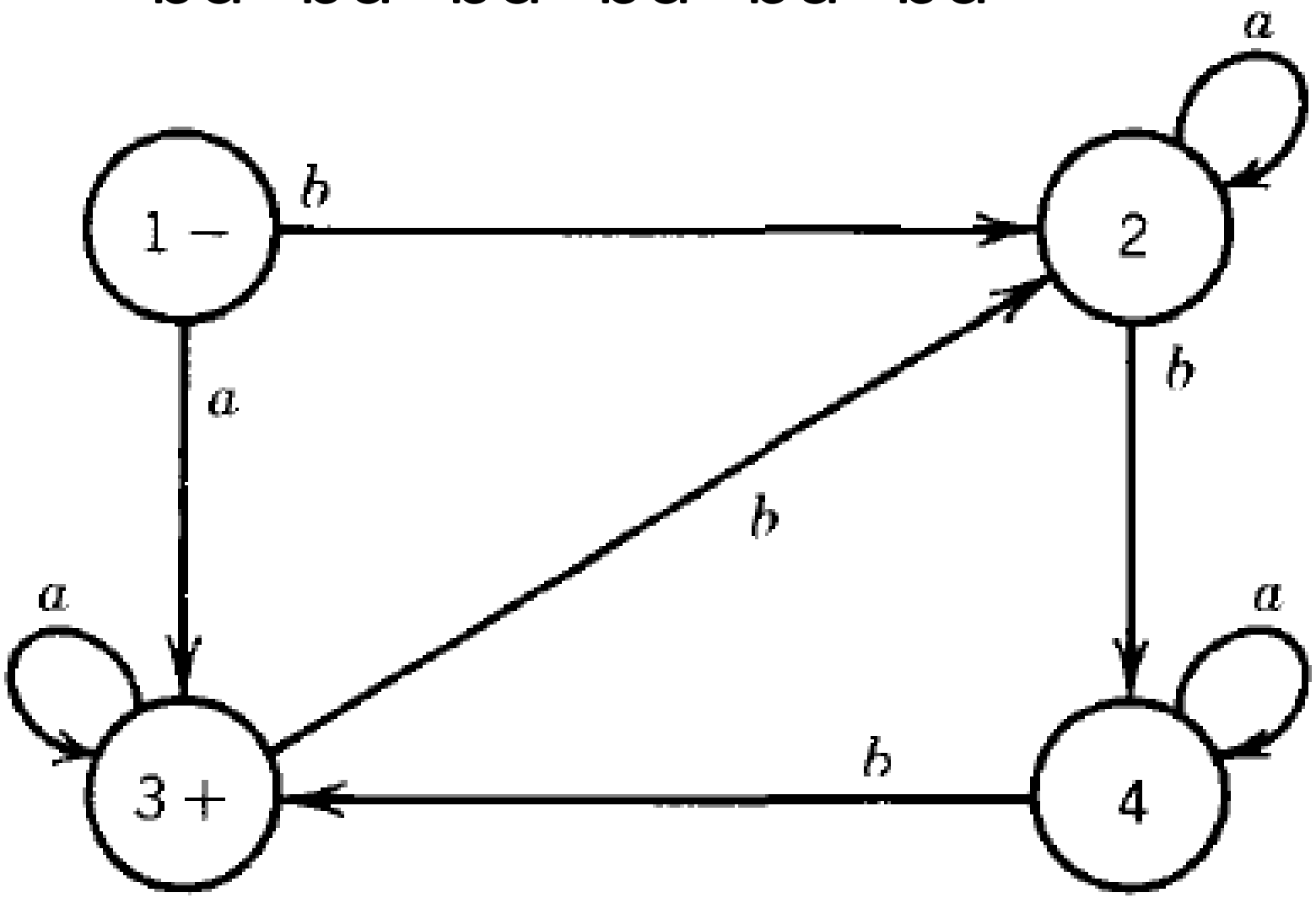
baa



baa

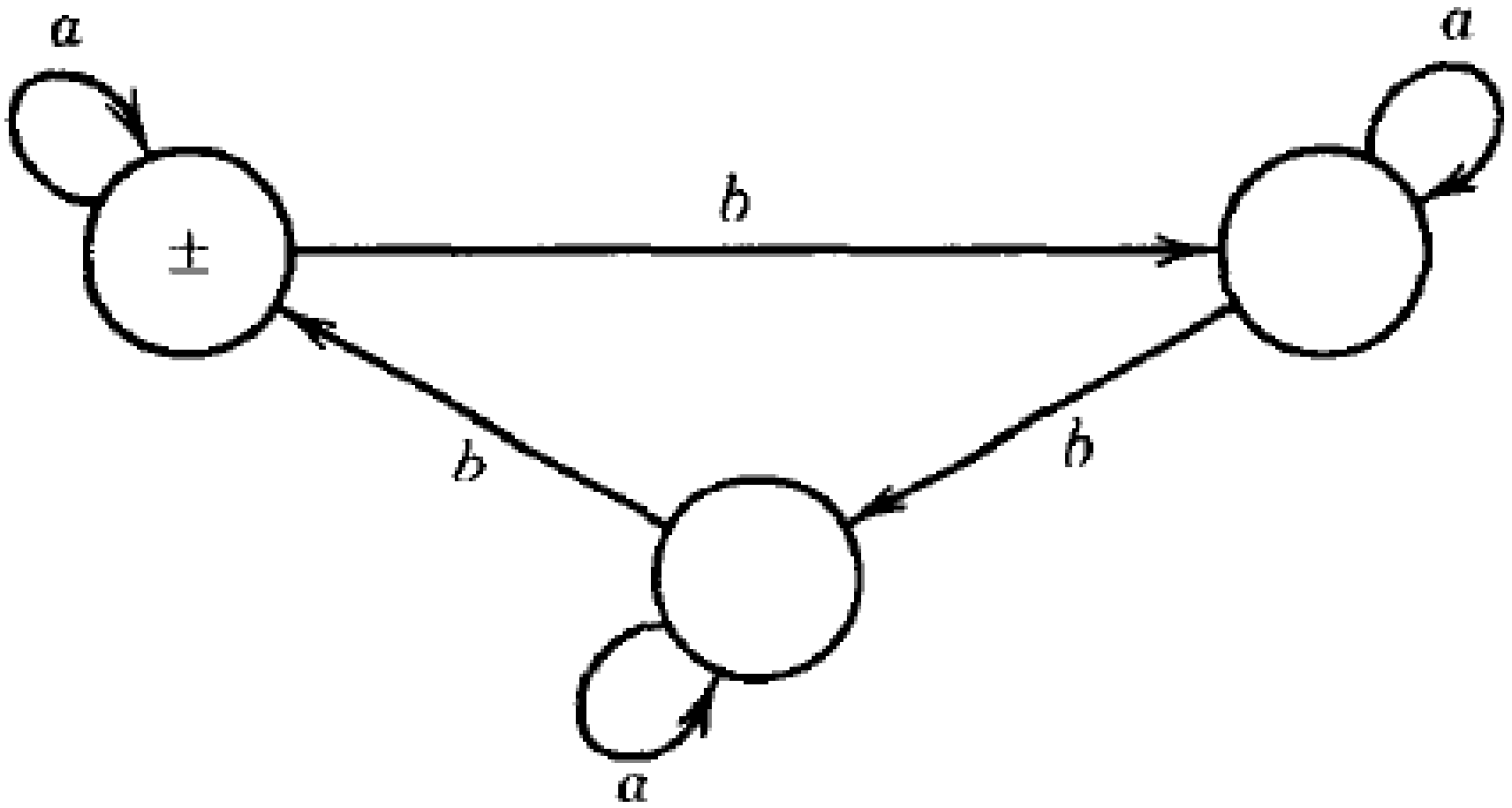
ab

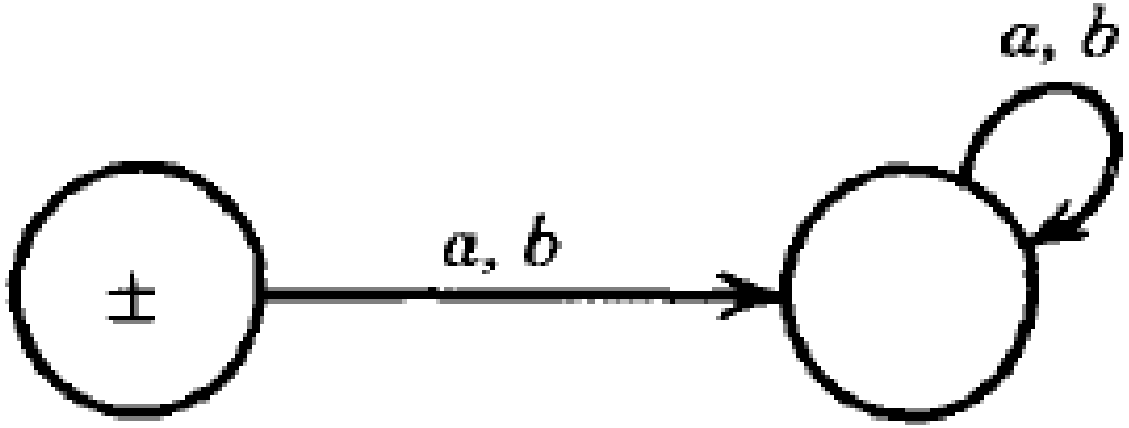
ba*ba*ba*ba*ba*ba*



a*

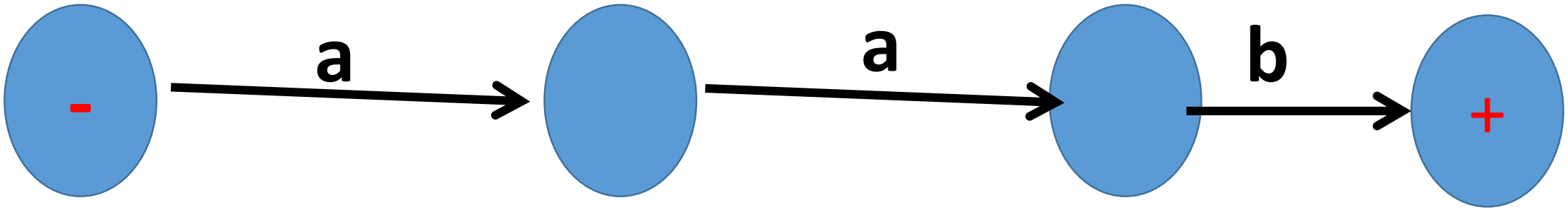
a*ba*ba*ba*

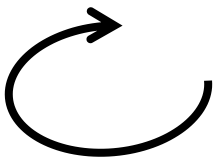




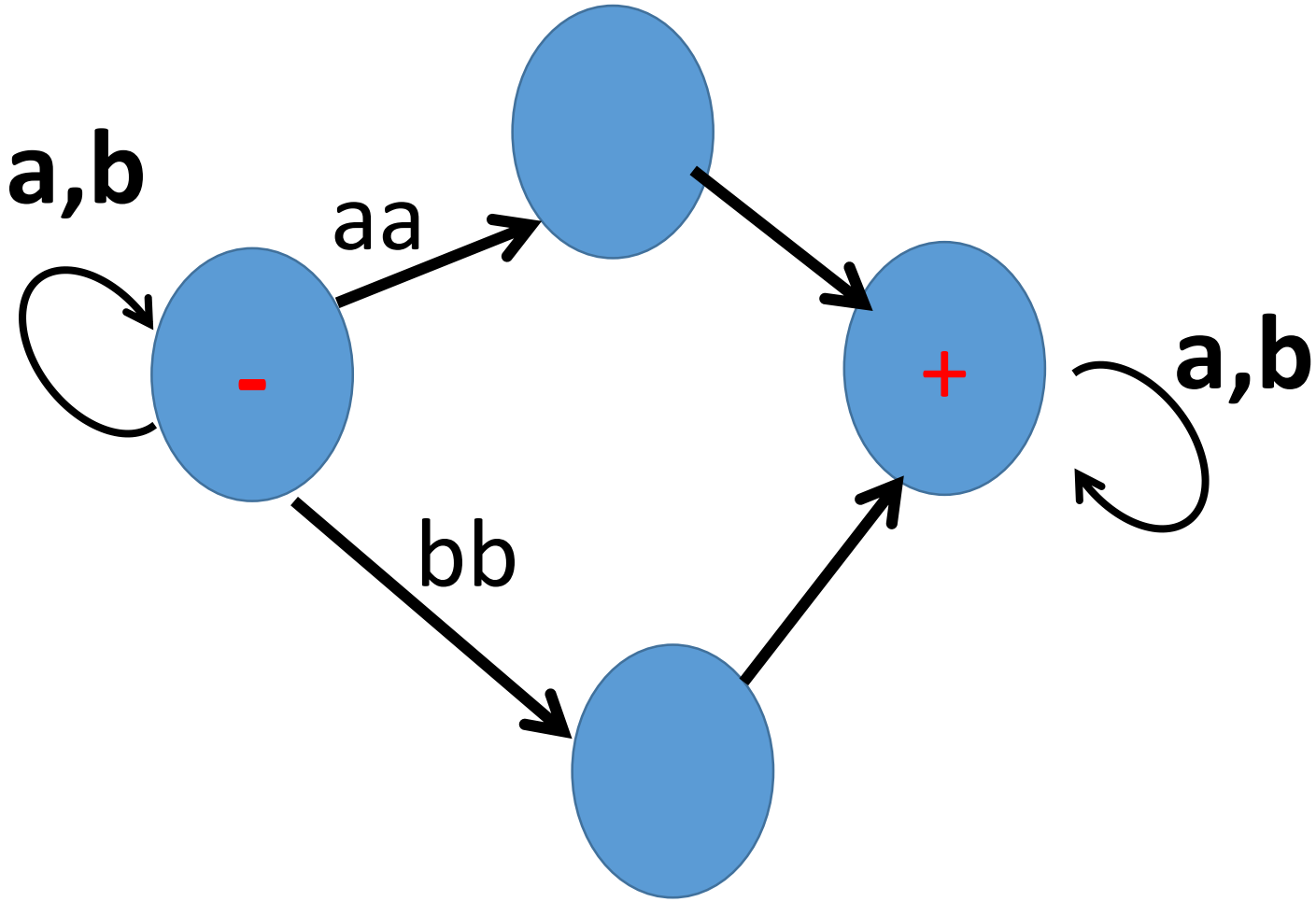
A

aab





$$(a + b)^*(aa + bb)(a + b)^*$$



b*a(a+b)*(ab)(b*)

Theory of Computation

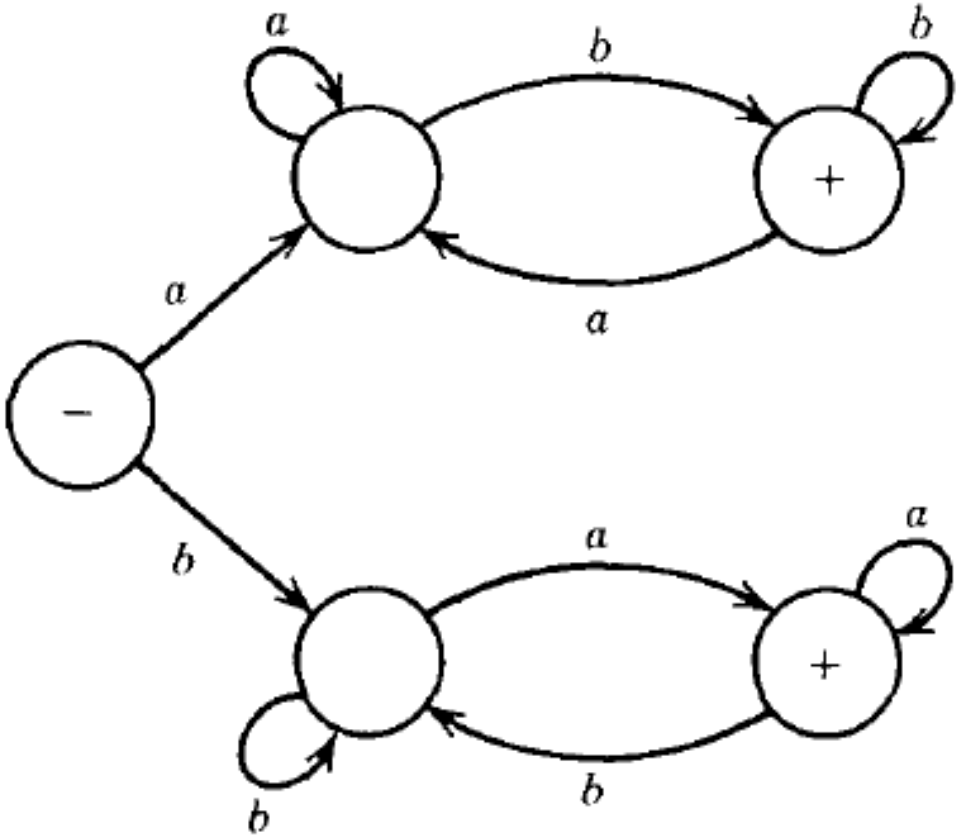
النظرية الاحتسابية
المحاضرة التاسعة

كلية التربية للعلوم الصرفة / جامعة ديالى

اعداد
م.د. محمد سامي محمد

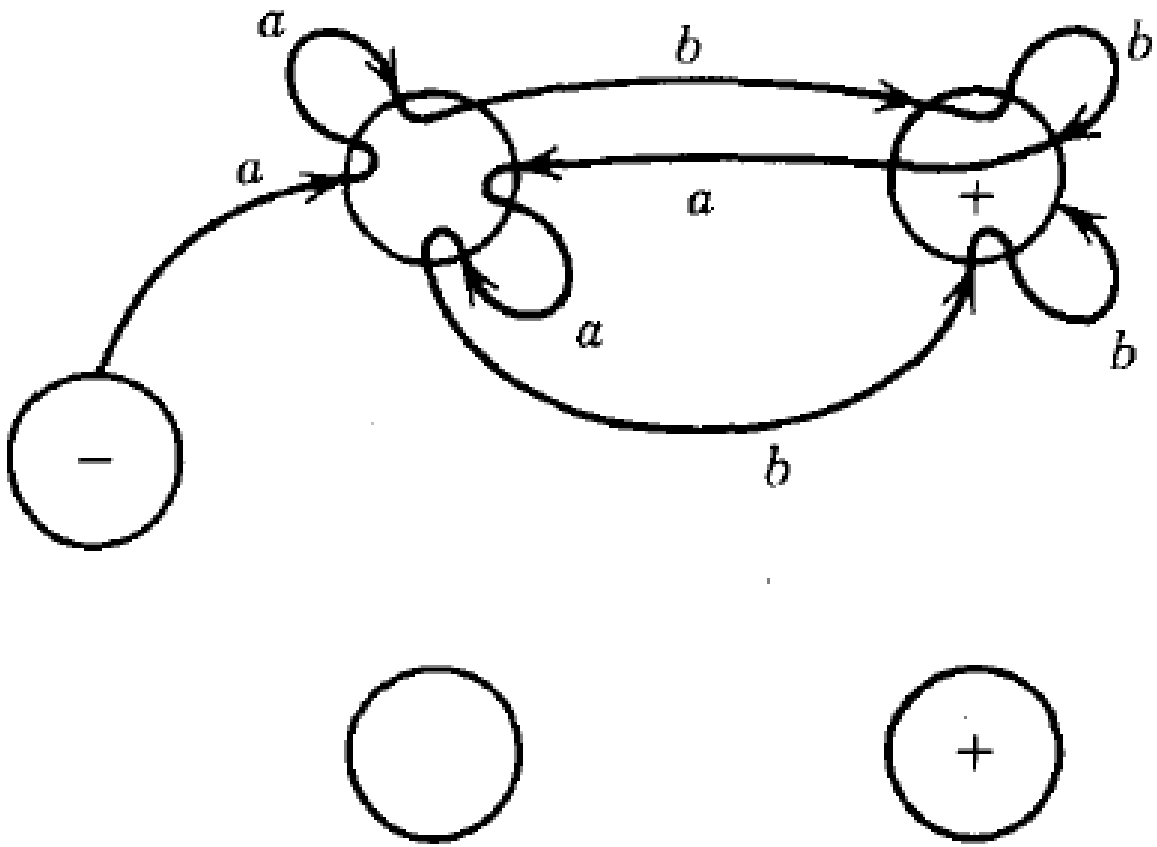
قسم علوم الحاسوب
المرحلة الثانية

EXAMPLE: The following FA accepts all words that have different first and last letters. If the word begins with an a, to be accepted it must end with a b and vice versa.

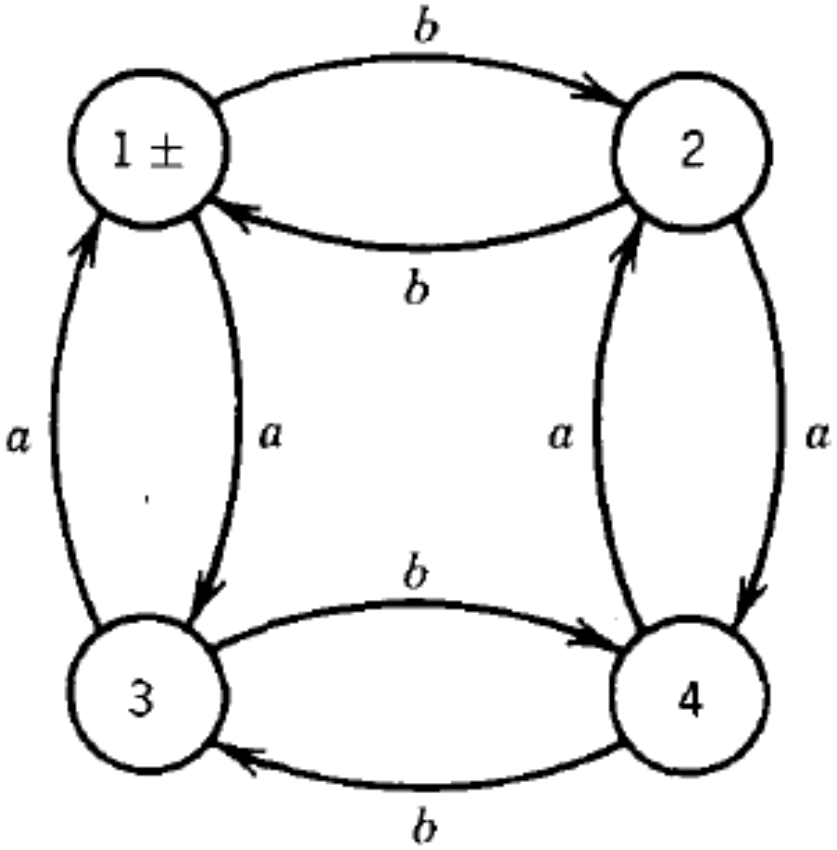


الدكتورالمهندس محمد سامي محمد

This can be better understood by examining the path through the FA of the input string *aabbaabb*, as shown below:



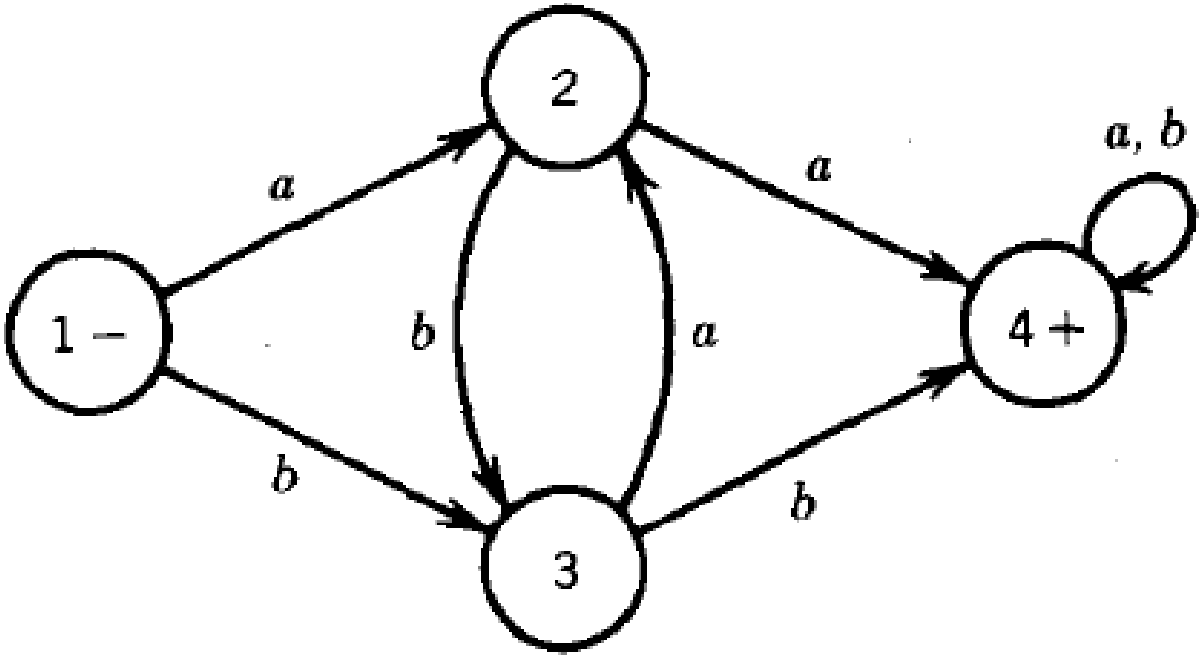
EXAMPLE As the last example of an FA in this chapter, let us consider the picture below:



الدكتورالمهندس محمد سامي محمد

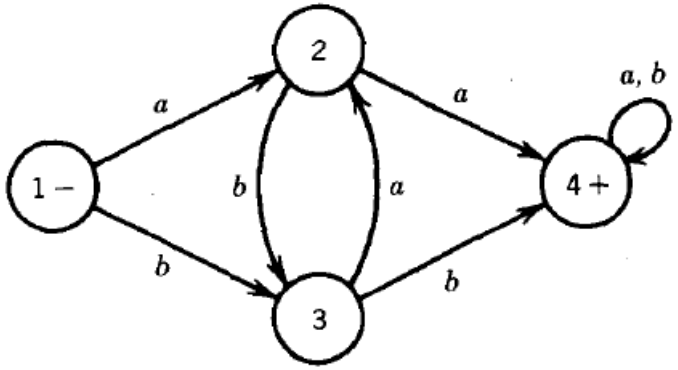
PROBLEMS

Write the transition table of



PROBLEMS

Write the transition table of

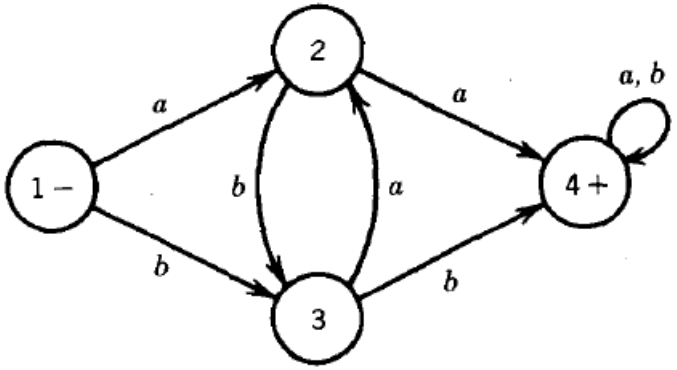


What is the mistake here ?

State	a	b
1	4	3
2	5	2
3	3	4
4	2	1

PROBLEMS

Write the transition table of

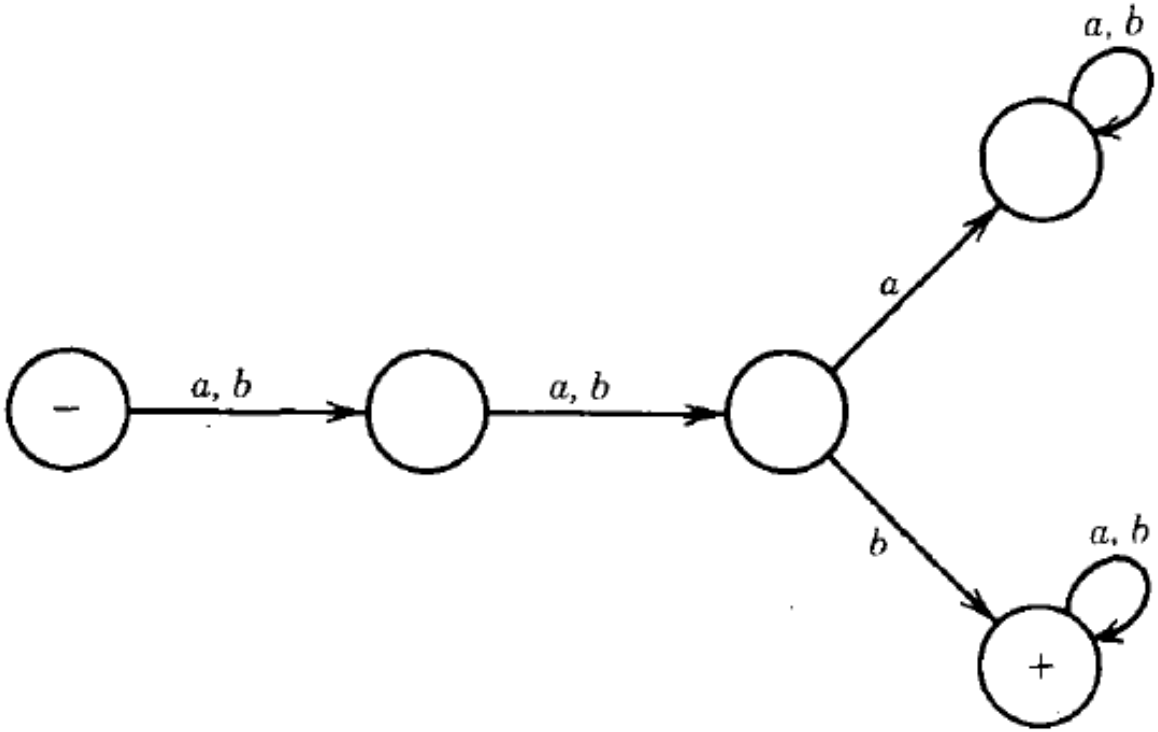


Correct Solution is

State	a	b
1	2	3
2	4	3
3	2	4
4	4	4

PROBLEMS

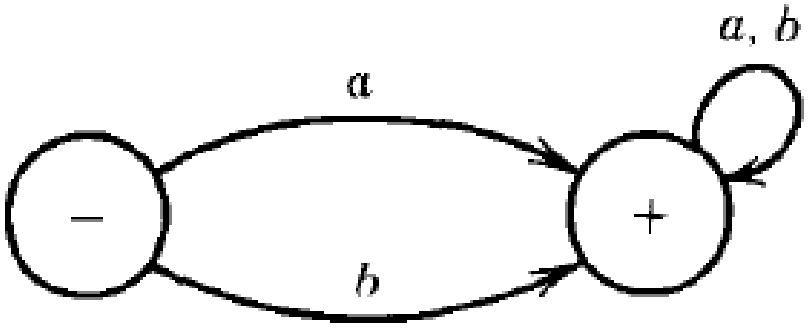
Write the transition table of



الدكتور المهندس محمد سامي محمد

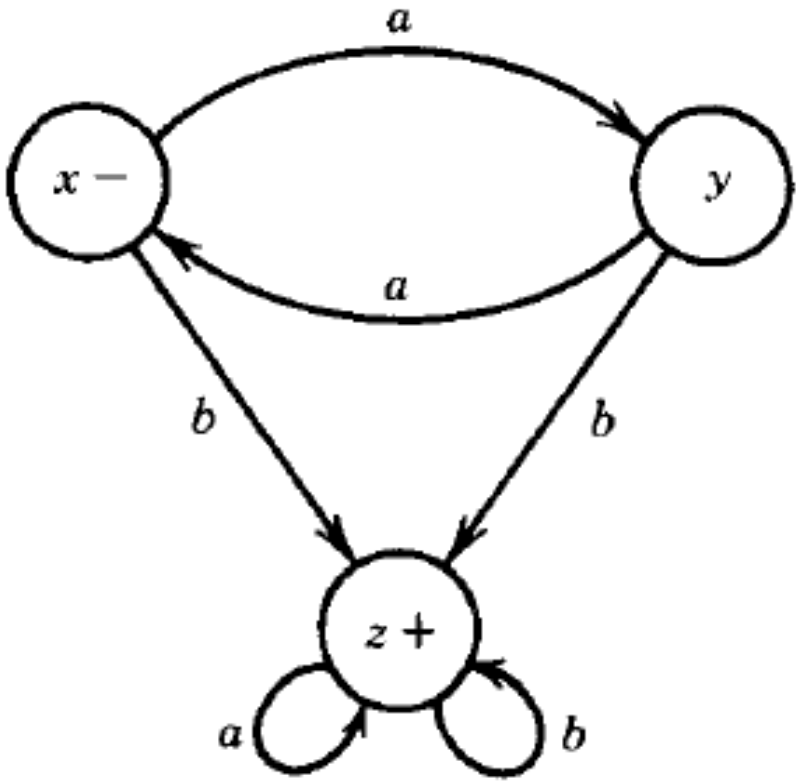
PROBLEMS

Write the transition table of



PROBLEMS

Write the transition table of



الدكتور المهندس محمد سامي محمد

Q/ Build an FA that accepts only the language of all words with b as the second letter. Show both the picture and the transition table for this machine and find a regular expression for the language.

Q/ Build an FA that accepts only the words *baa*, *ab*, and *abb* and no other strings longer or shorter.

Q/ Build an FA that accepts only those words that have an even number of letters total.

Q/ Build an FA that accepts only those words that do not end with *ba*.

Q/ Build an FA that accepts only those words that begin or end with a double letter.

الدكتور المهندس محمد سامي محمد

Q/ Build an FA that accepts only those words that have more than four letters.

Q/ Build an FA that accepts only those words that have fewer than four letters.

Q/ Build an FA that accepts the language of all words with only a's or only b's in them. Give a regular expression for this language.

الدكتور المهندس محمد سامي محمد

Theory of Computation

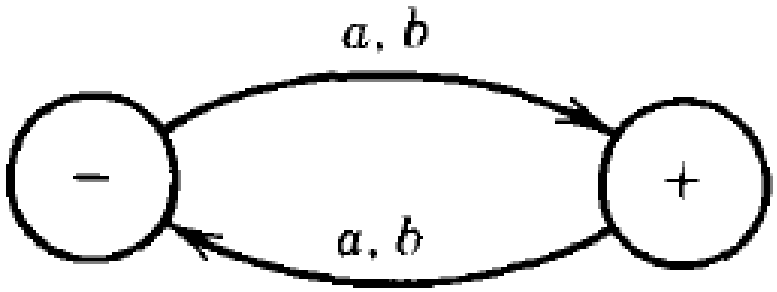
النظرية الاحتسابية
المحاضرة العاشرة

كلية التربية للعلوم الصرفة / جامعة ديالى

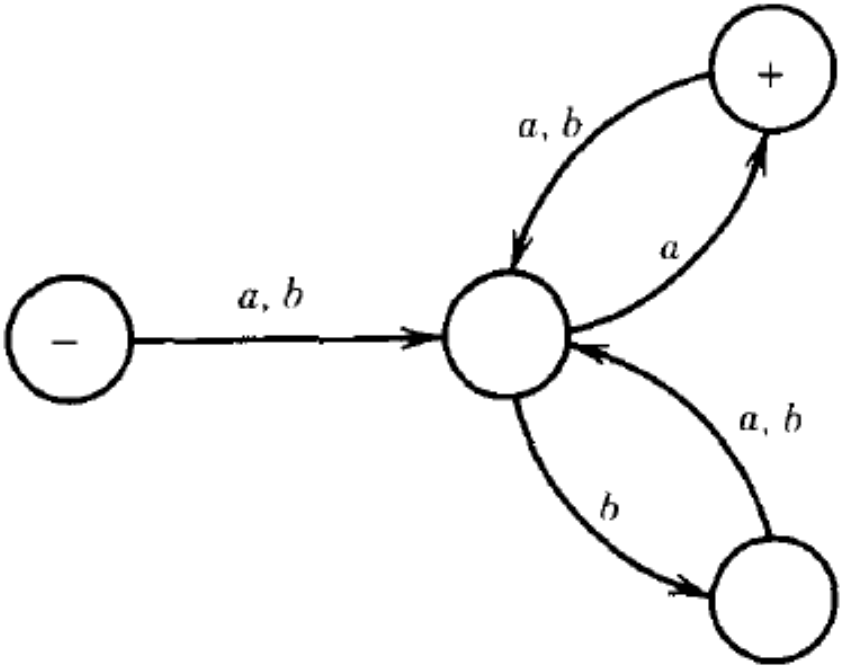
اعداد
م.د. محمد سامي محمد

قسم علوم الحاسوب
المرحلة الثانية

Q/ Describe the languages accepted by the following FA's.

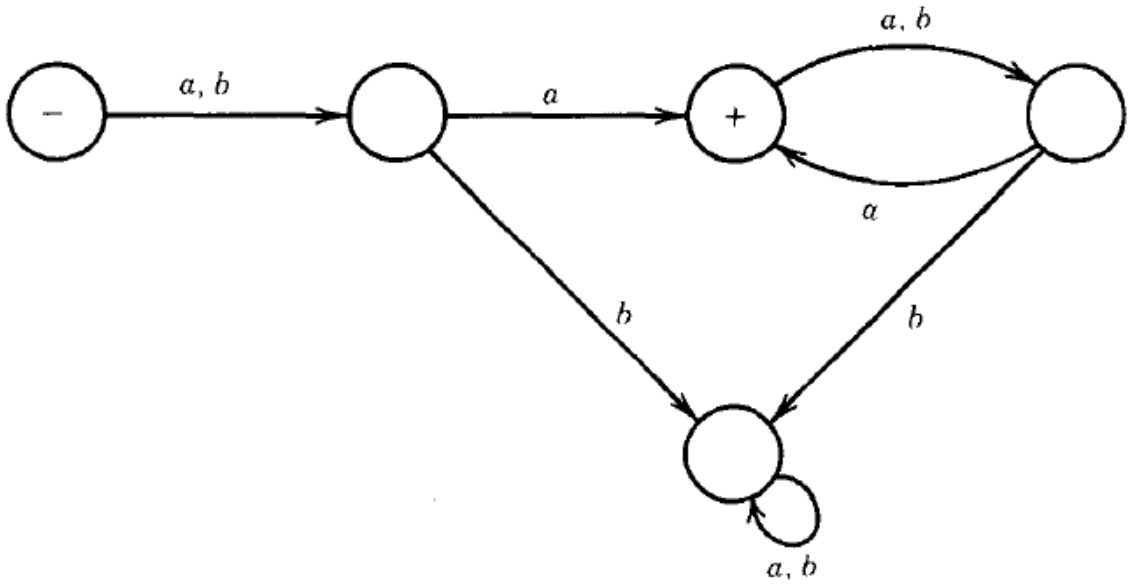


Q/ Describe the languages accepted by the following FA's.



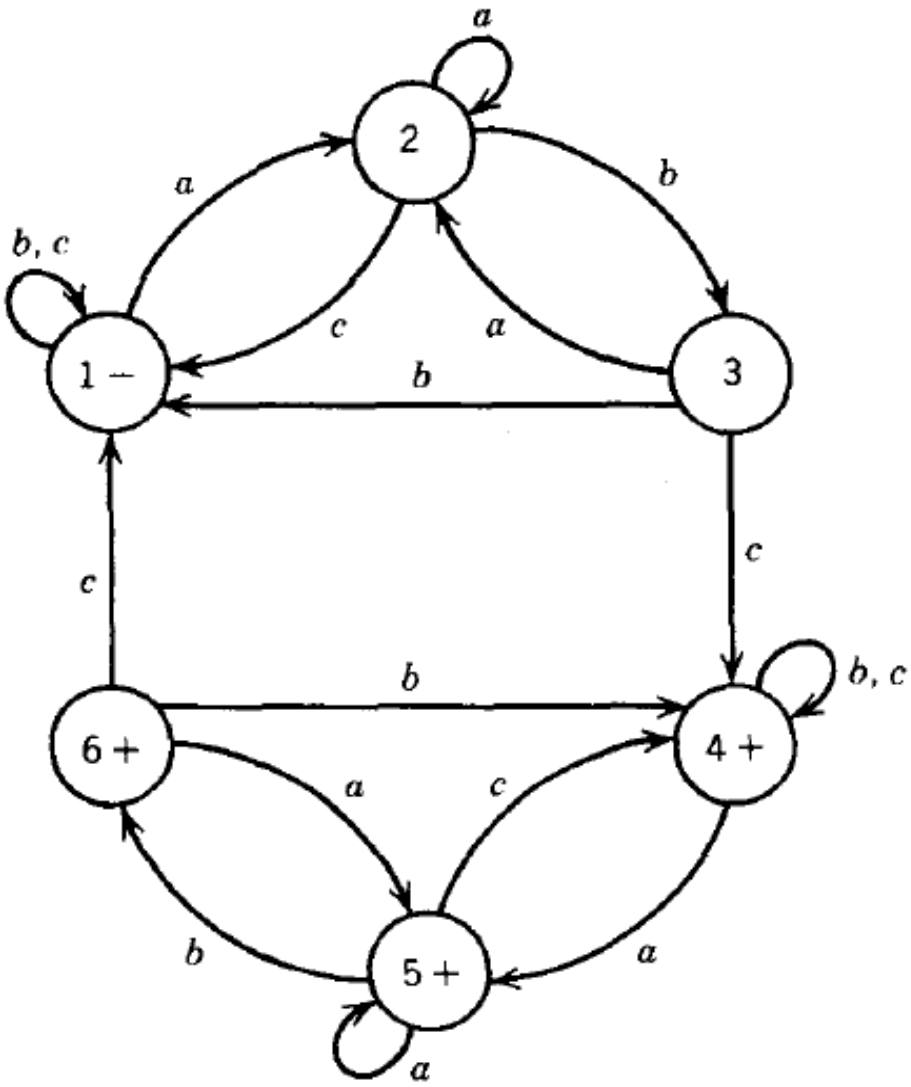
الدكتورالمهندس محمد سامي محمد

Q/ Describe the languages accepted by the following FA's.



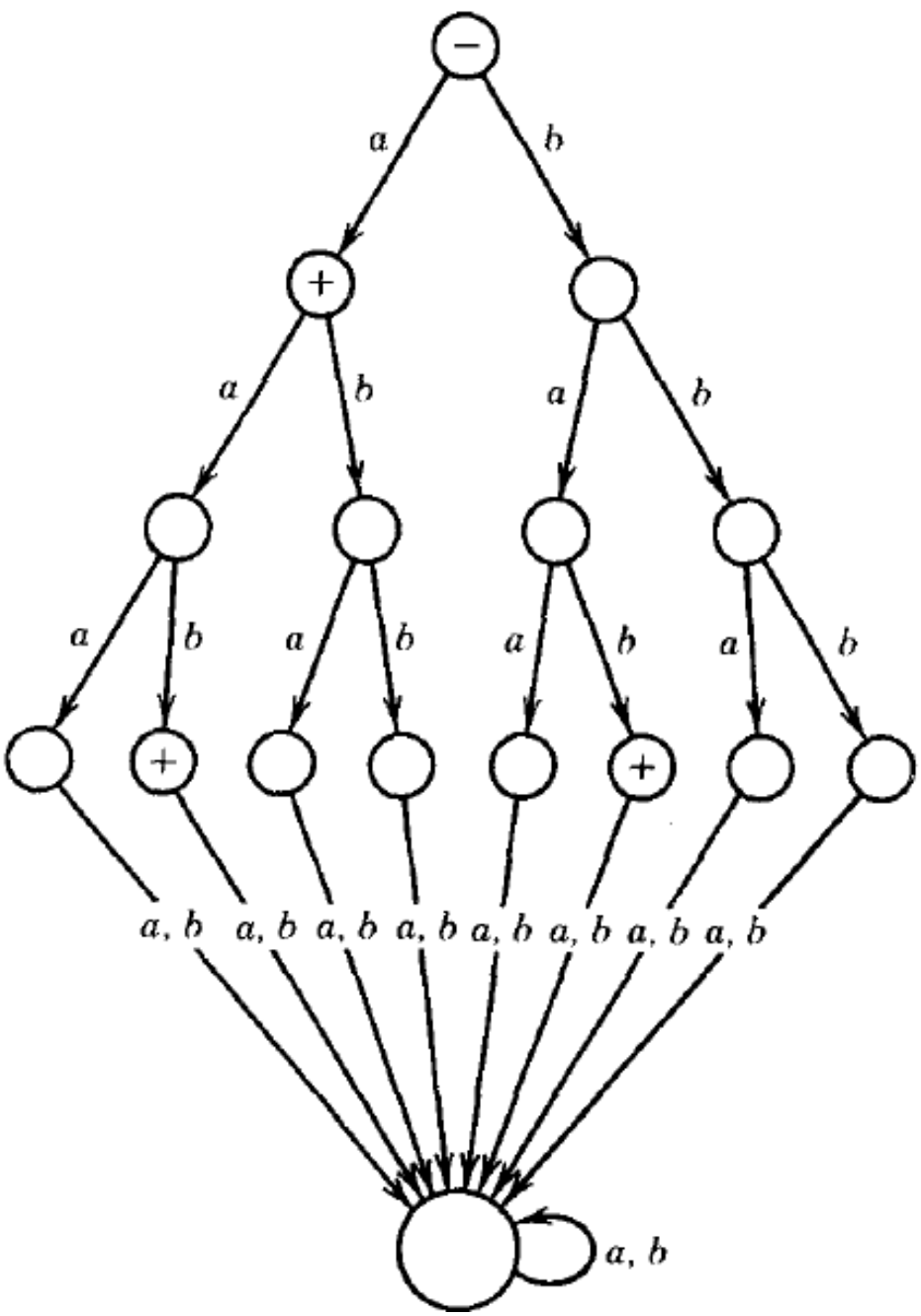
الدكتورالمهندس محمد سامي محمد

Q/ The following is an FA over the alphabet I = {a, *b*, *c*}. Prove that it accepts all strings that have an odd number of occurrences of the substring *abc*.



الدكتور المهندس
محمد سامي محمد

Q/ Consider the following FA:



الدكتور المهندس
محمد سامي محمد

(i) Show that any input string with more than three letters is not accepted by this FA.
(ii) Show that the only words accepted are *a*, *aab*, and *bab*.
(iii) Show that by changing the + signs alone we can make this FA accept the language *{bb, aba, bba}*
(iv) Show that any language in which the words have fewer than four letters can be accepted by a machine that looks like this one with the + signs in different places.
(v) Prove that if L is a finite language, then there is some FA that accepts *L*.

الدكتورالمهندس محمد سامي محمد

Theory of Computation

النظرية الاحتسابية
المحاضرة الحادية عشر

كلية التربية للعلوم الصرفة / جامعة ديالى

قسم علوم الحاسوب
المرحلة الثانية

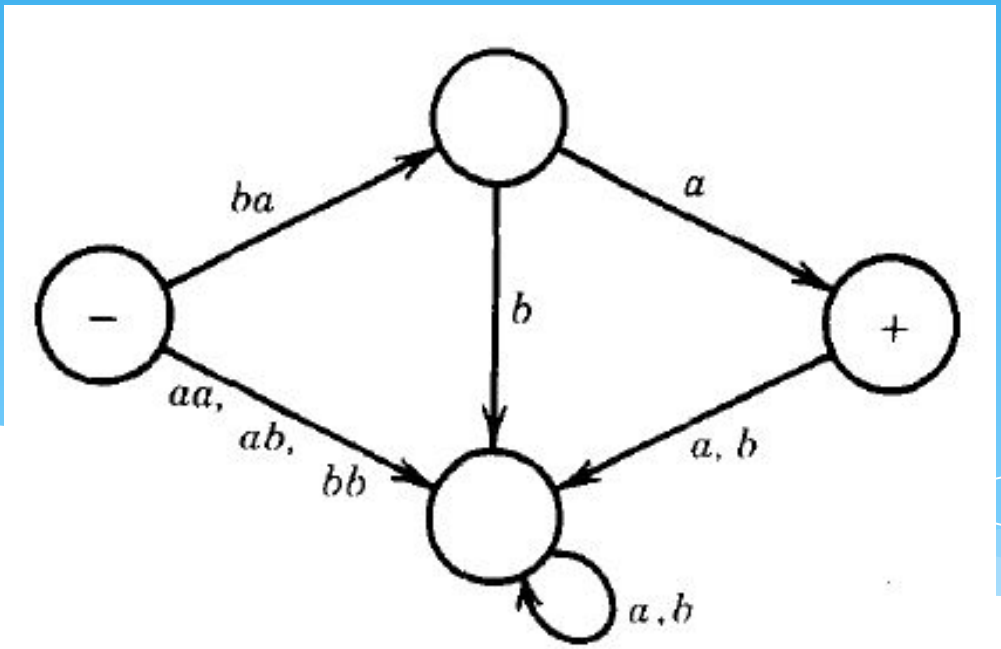
اعداد
م.د. محمد سامي محمد

TRANSITION GRAPHS

We saw in the last chapter that we could build an FA that accepts only the word baa.

Suppose we designed a more powerful machine that could read either one or two letters of the input string at a time and could change its state based on this information.

We might design a machine like the one below:



TRANSITION GRAPHS

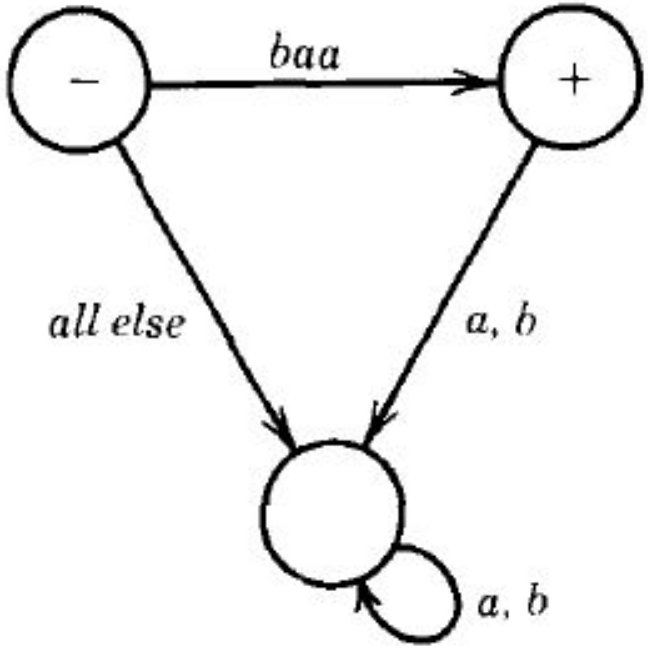
The objects we deal with in this book are mathematical models

Which we shall discover are abstractions and simplifications of how certain actual machines do work.

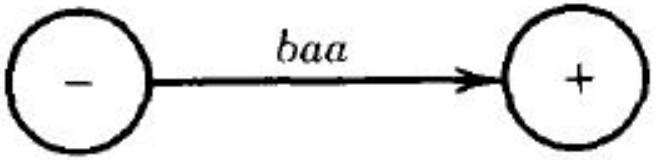
This new rule for making models will also turn out to be practical. It will make it easier for us to design machines that accept certain different languages.

If we are interested in a machine that accepts only the word baa, why stop at assuming that the machine can read just two letters at a time? A machine that accepts this word and that can read up to three letters at a time from the input string could be built with even fewer states.

TRANSITION GRAPHS



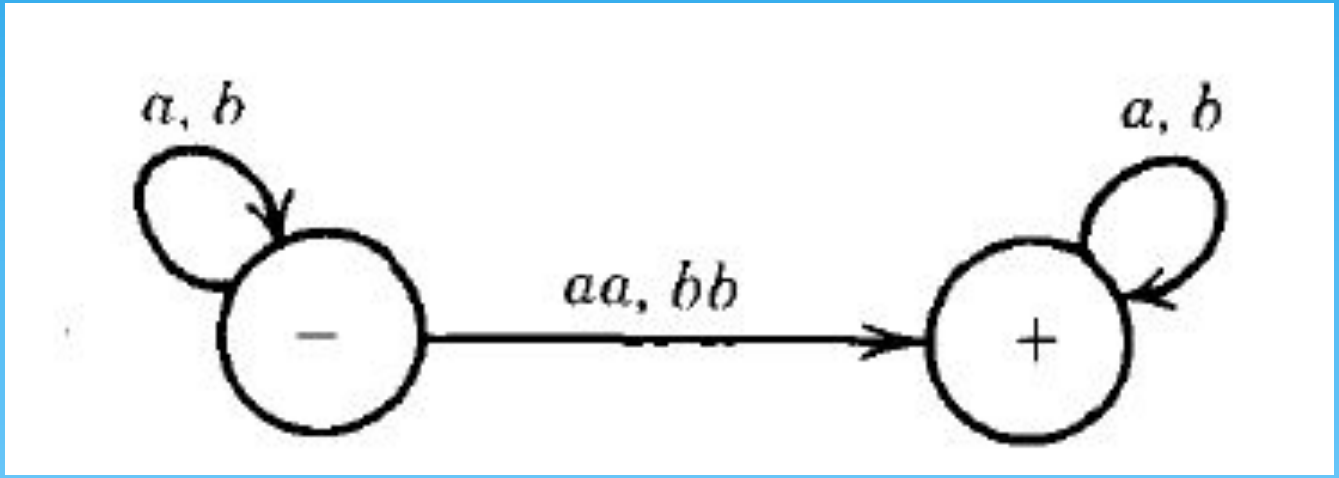
or even



TRANSITION GRAPHS

Example

We can be built using only two states that can recognize all words that contain a doubled letter.



Last Example Notes

This last machine makes us exercise some choice in its running. We must decide how many letters to read from the input string each time we go back for more.

This decision is quite important. Let us say, for example, that the input string is baa. It is easy to see how this string can be accepted by this machine. We first read in the letter b, which leaves us back at the start state by taking the loop on the left.

Then we decide to read in both letters aa at once, which allows us to take the highway to the final state where we end. However, if after reading in the single character b we then decided to read in the single character a, we would loop back and be stuck at the start state again.

Last Example Notes

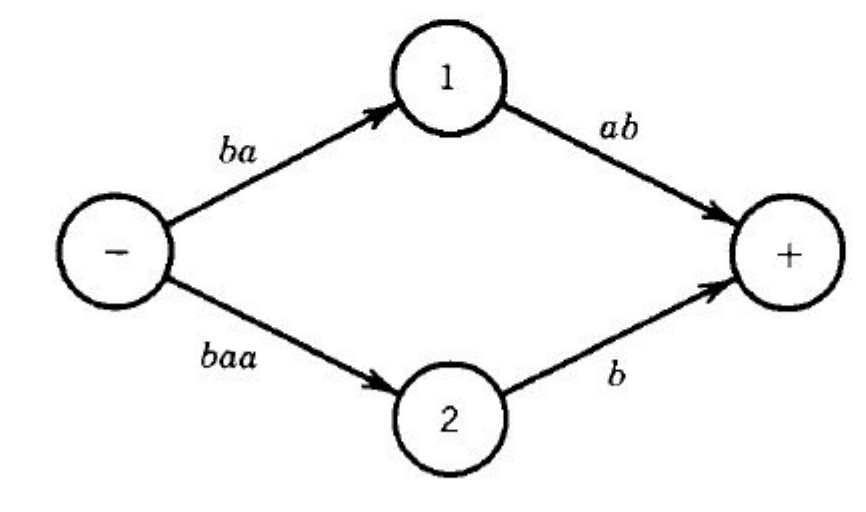
When the third letter is read in, we would still be at the starting post. We could not then accept this string.
There are two different paths that the input *baa* can take through the machine. This is totally different from the situation we had before, especially since one path leads to acceptance and one to rejection.

Another bad thing that might have happened is that we could have started processing the string *baa* by reading the first two letters at once. Since *ba* is not a double, we could not move to the final state.
In fact, when we read in *ba,* no edge tells us where to go, since ba is not the label of any edge leaving the start state. The processing of this string breaks down at this point. When there is no edge leaving a state corresponding to the group of input letters that have been read while in that state, we say that the input **crashes**.
The result of these considerations is that if we are going to change the definition of our abstract machine to allow for more than one letter to be read in at a time, we must also **change the definition of acceptance**. We have to say that a string is accepted by a machine if there is *some* way it could be processed so as to arrive at a final state. There may also be ways in which this string does not get to a final state, but we ignore all failures.

Last Example Notes

We are about to create machines in which any edge in the picture can be labeled by any string of alphabet letters, but first we must consider the consequences. We could now encounter the following additional problem:

baab



Last Example Notes

On this machine we can accept the word baab in two different ways. First, we could take ba from the start state to state 1 and then ab would take us to the final state.

Or else we could read in the three letters baa and go to state 2 from which the final letter, b, would take us to the final state.
Previously, when we were dealing only with FA's, we had a unique path through the machine for every input string. Now some strings have no paths at all while some have several.

at least be fully processed to arrive at some state. However, we are not yet interested in the different reasons for failure and we use the word "rejection“ for both cases.

We now have observed many of the difficulties inherent in expanding our definition of "machine" to allow word-labeled edges (or, equivalently, to reading more than one letter of input at a time). We shall leave the definition of finite automation alone and call these new machines **transition graphs** because they are more easily understood as graphs than as tables.

TRANSITION GRAPHS

DEFINITION

A transition graph, abbreviated TG, is a collection of three things:

1. A finite set of states at least one of which is designated as the start state (-) and some (maybe none) of which are designated as final states (+).

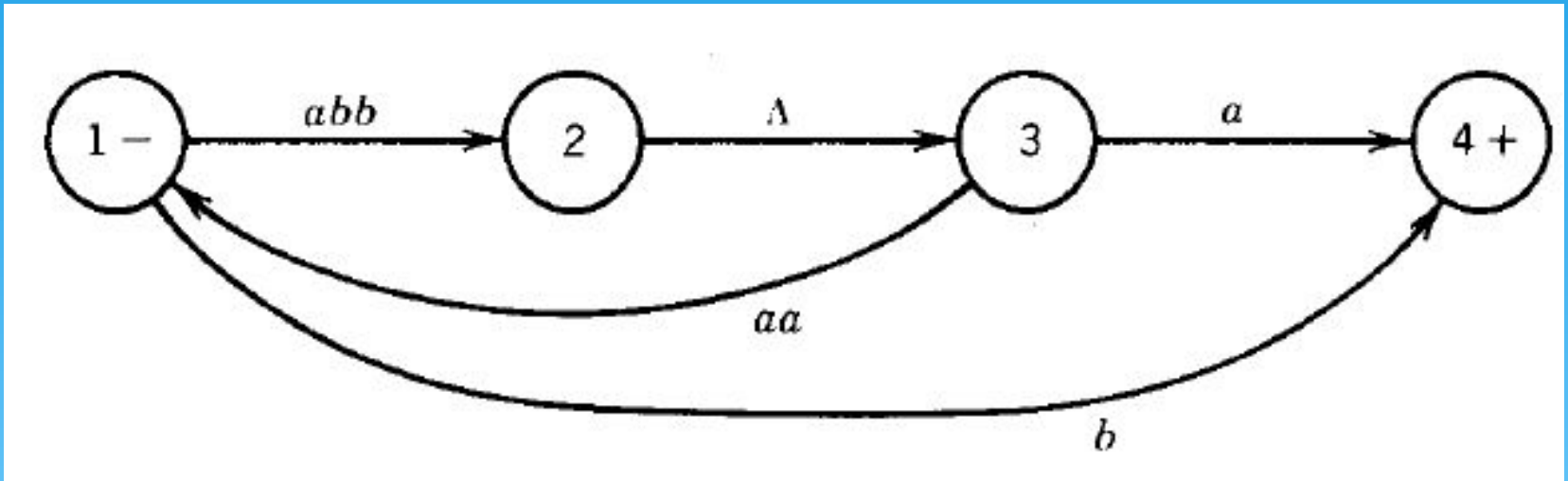
2. An alphabet I of possible input letters from which input strings are formed.

3. A finite set of transitions that show how to go from one state to another based on reading specified substrings of input letters (possibly even the null string A).

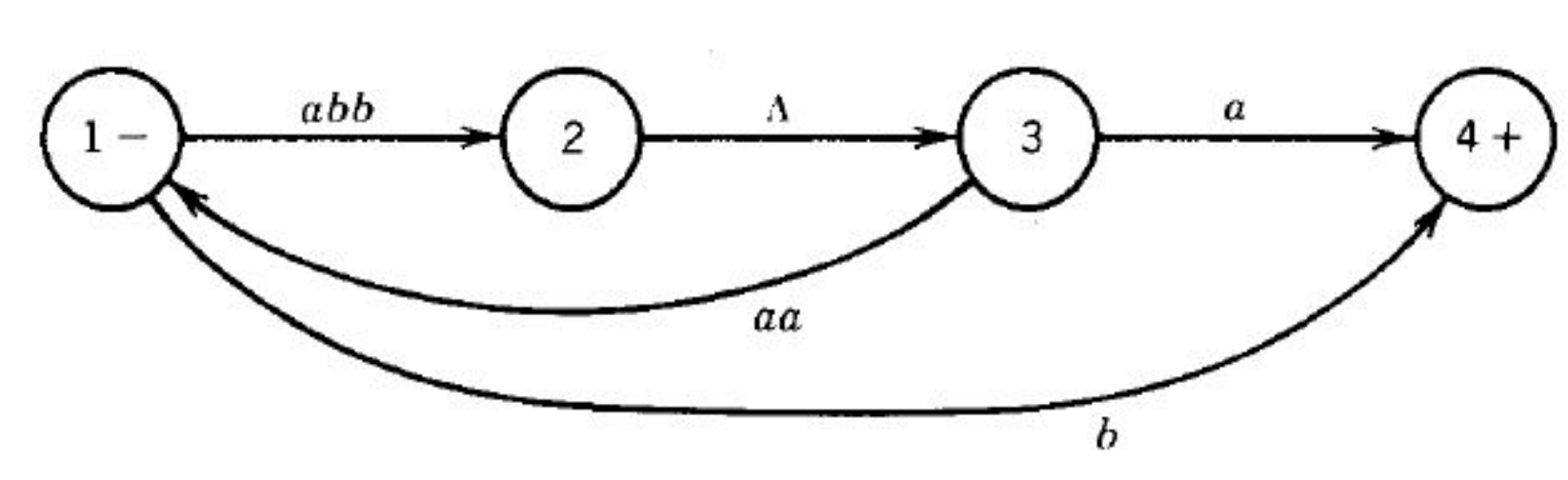
A successful path through a transition graph is a series of edges forming a path beginning at some start state (there may be several) and ending at a final state. If we concatenate in order the strings of letters that label each edge in the path, we produce a word that is accepted by this machine.

Another Example

For example, consider the following TG:



Example Continue

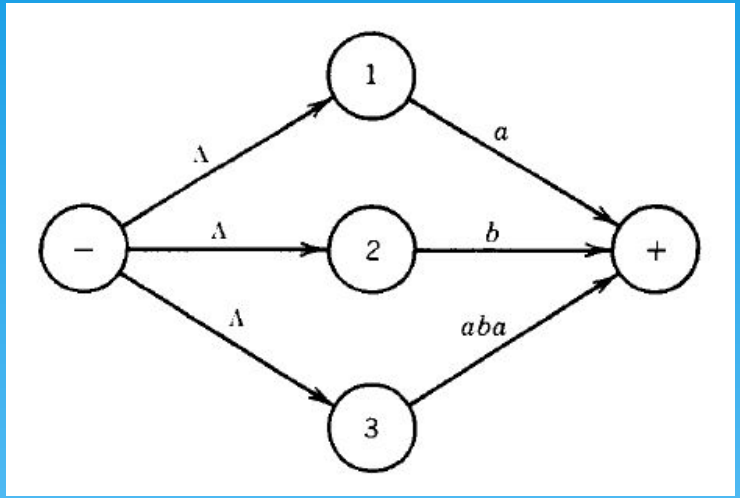


The path from state 1 to state 2 to state 3 back to state 1 then to state 4 corresponds to the string *(abb)(A)(aa)(b)*.

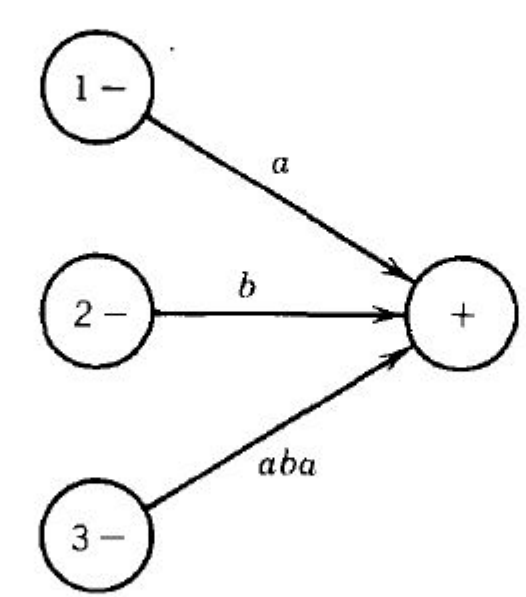
This is one way of factoring the word *abbaab*, which, we now see, is accepted by this machine. Some other words accepted are *abba*, *abbaaabba*, and b.

Another Example

This point is illustrated by the following example. There is no difference between the TG



and the TG



Example Continue

in the sense that all the strings accepted by the first are accepted by the second and vice versa. There are differences between the two machines such as **the number of total states** they have, but as *language acceptors* they are equivalent. It is extremely important for us to understand that every FA is also a TG. This means that any picture that represents an **FA can be interpreted as a picture of a TG**. Of course, **not every TG satisfies the definition of an FA**.

Theory of Computation

النظرية الاحتسابية
المحاضرة الثانية عشر

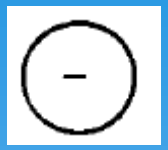
كلية التربية للعلوم الصرفة / جامعة ديالى

قسم علوم الحاسوب
المرحلة الثانية

اعداد
م.د. محمد سامي محمد

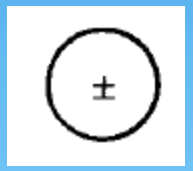
Some notes

Let us consider some more examples of TG's.



The picture above represents a TG that accepts nothing, not even the null string A. To be able to accept anything, it must have a final state.

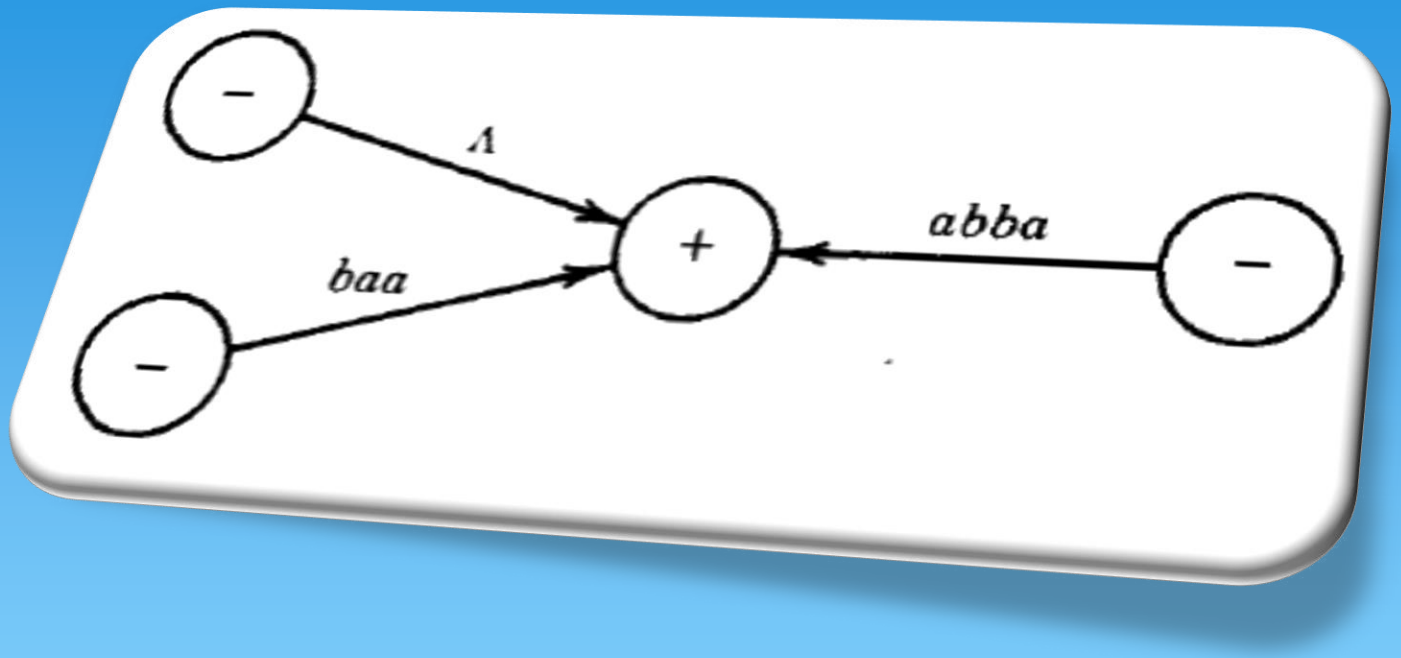
The machine



accepts only the string A. Any other string cannot have a successful path to the final state through labels of edges since there are no edges (and hence no labels).

Any TG in which some start state is also a final state will always accept the string A; this is also true of FA's. There are some other TG's that accept the word A, for example:

Another Example



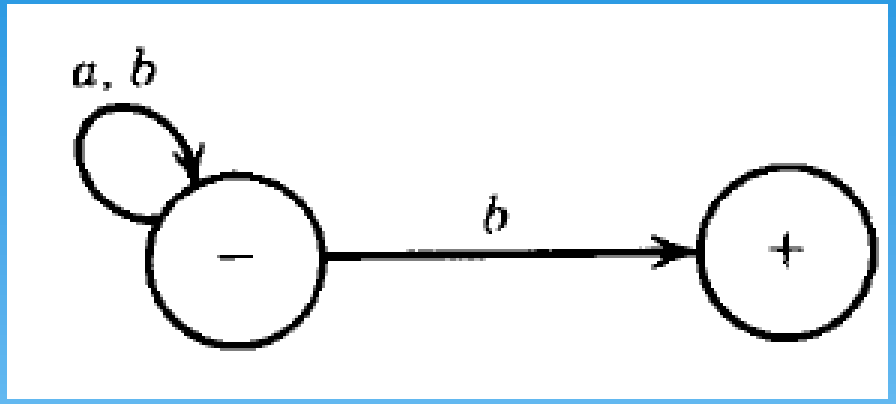
This machine accepts only the words A, *baa*, and *abba*. Anything read while in the + state will cause a crash, since the + state has no outgoing edges.

Explain

Design the previous example with more than one state

Another Example

Consider the following TG:



We have two conditions for b so ???????

It must be the very last letter, since once in the right-side state if we try to read another letter we crash.

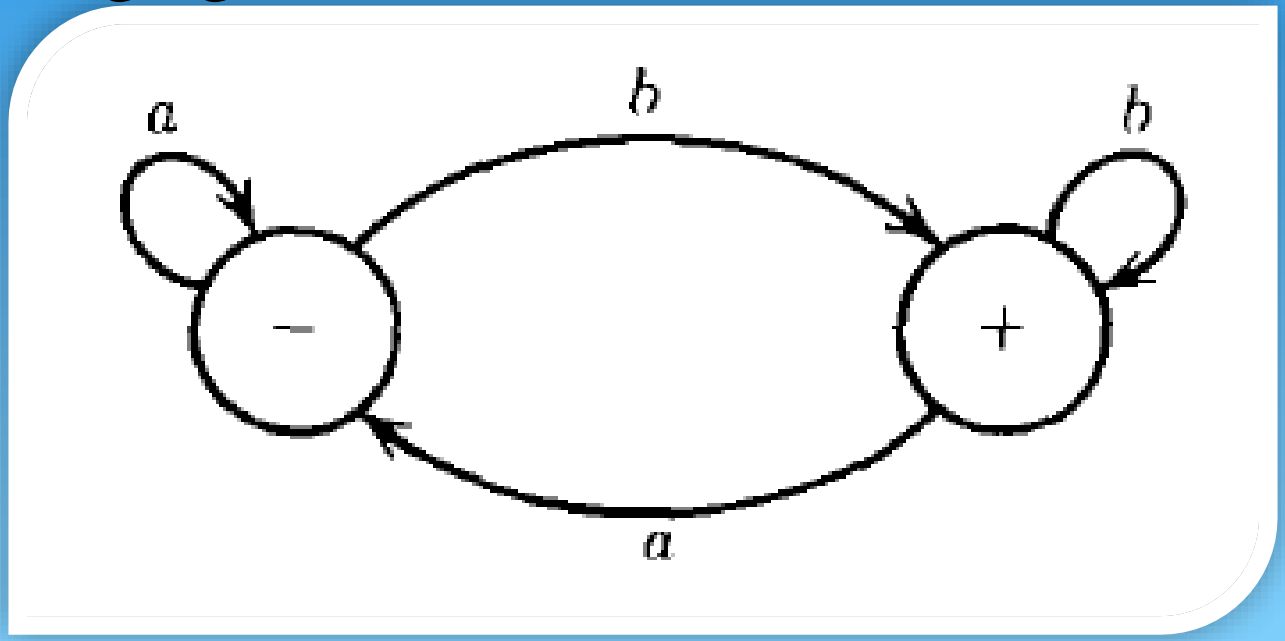
Previous Example Notes

Notice that it is also possible to start with a word that does end with a b but to follow an unsuccessful path that does not lead to acceptance.

We could either make the mistake of following the non-loop b-edge too soon (on a non-final b) in which case we crash on the next letter; or else we might make the mistake of looping back to – when we read the last b, in which case we reject without crashing. But still, all words that end in *b can* be accepted by some path, and that is all that is required.

Another Example

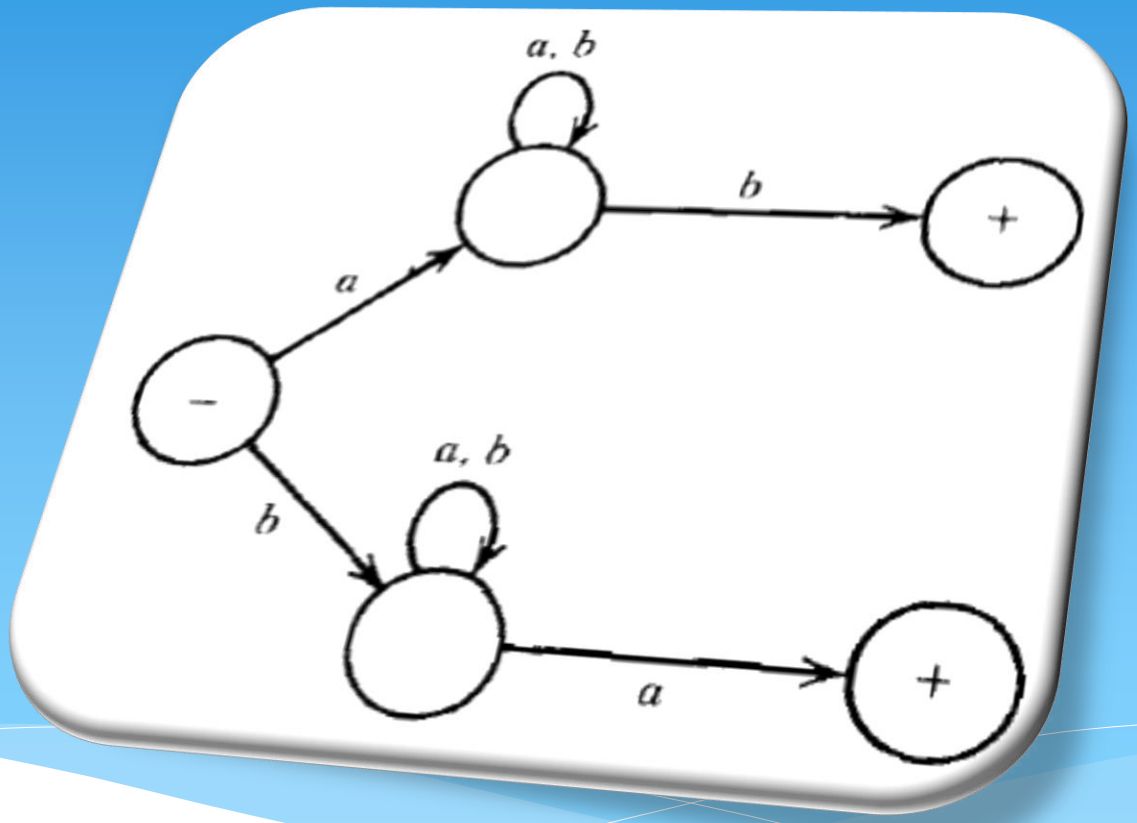
The language accepted by this TG is all words ending in b. One regular expression for this language is (a + b)*b and an FA that accepts the same language is:



Another Example

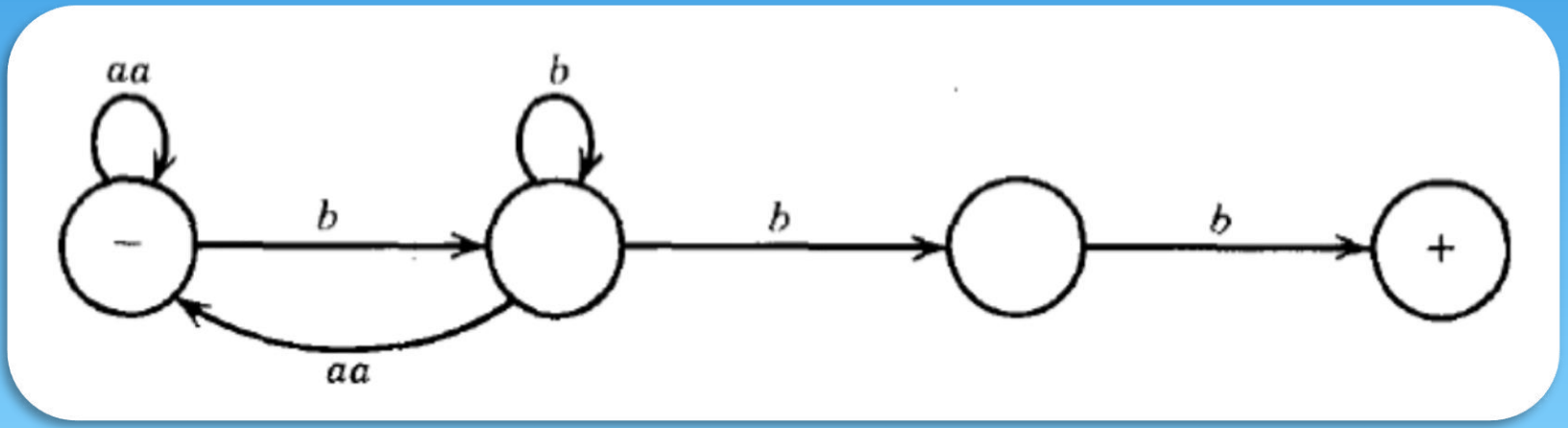
The following TG:

accepts the language of all words that begin and end with different letters.



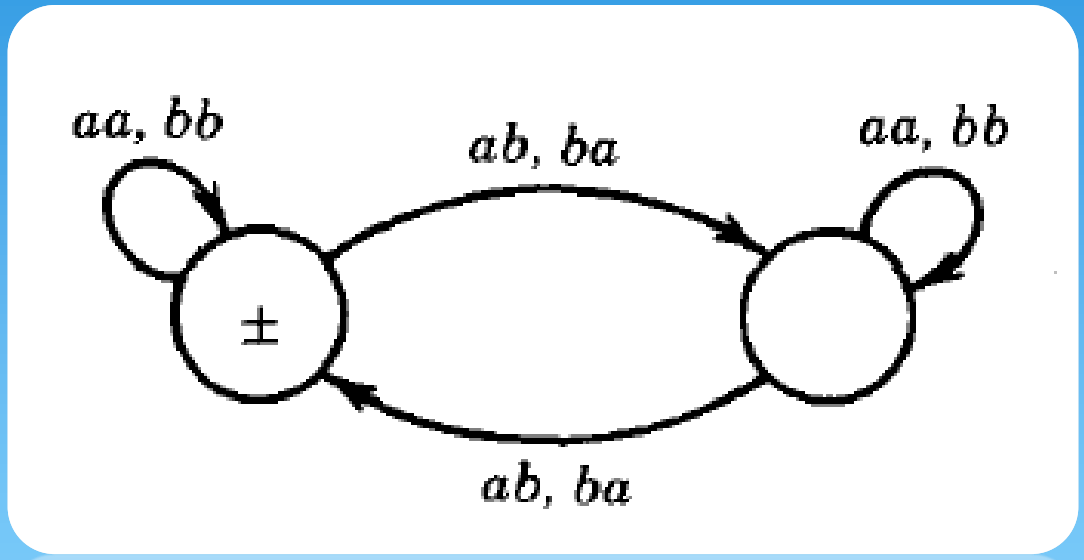
Another Example

accepts the language of all words in which the a's occur only in even clumps and that end in three or more *b's.*



Another Example

Consider the following **TG:**

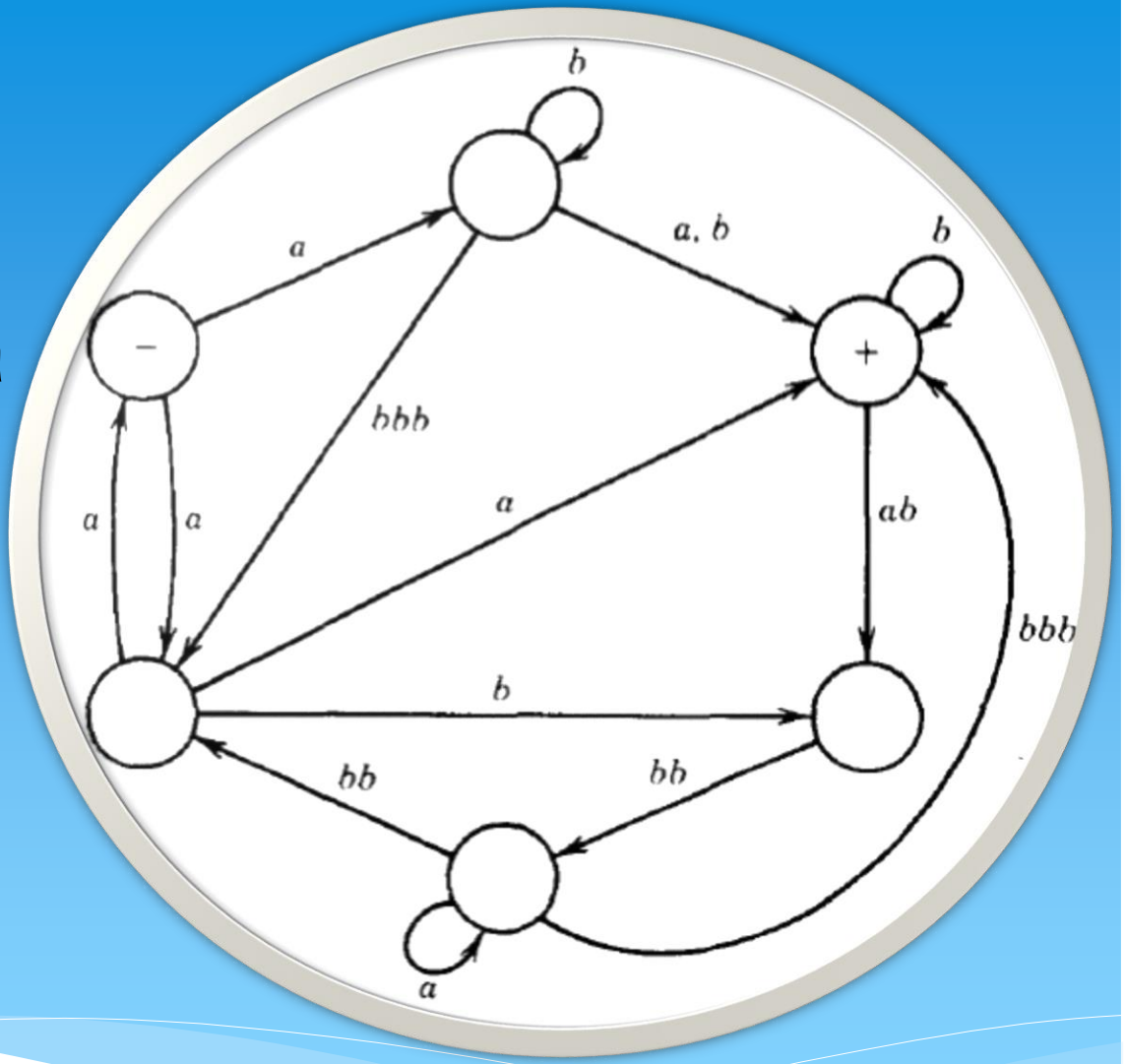


There is a practical problem with TG's. There are occasionally so many possible ways of grouping the letters of the input string that we must examine many possibilities before we know whether a given string is accepted or rejected.

EXAMPLE

Consider this TG:

abbbabbbbabba



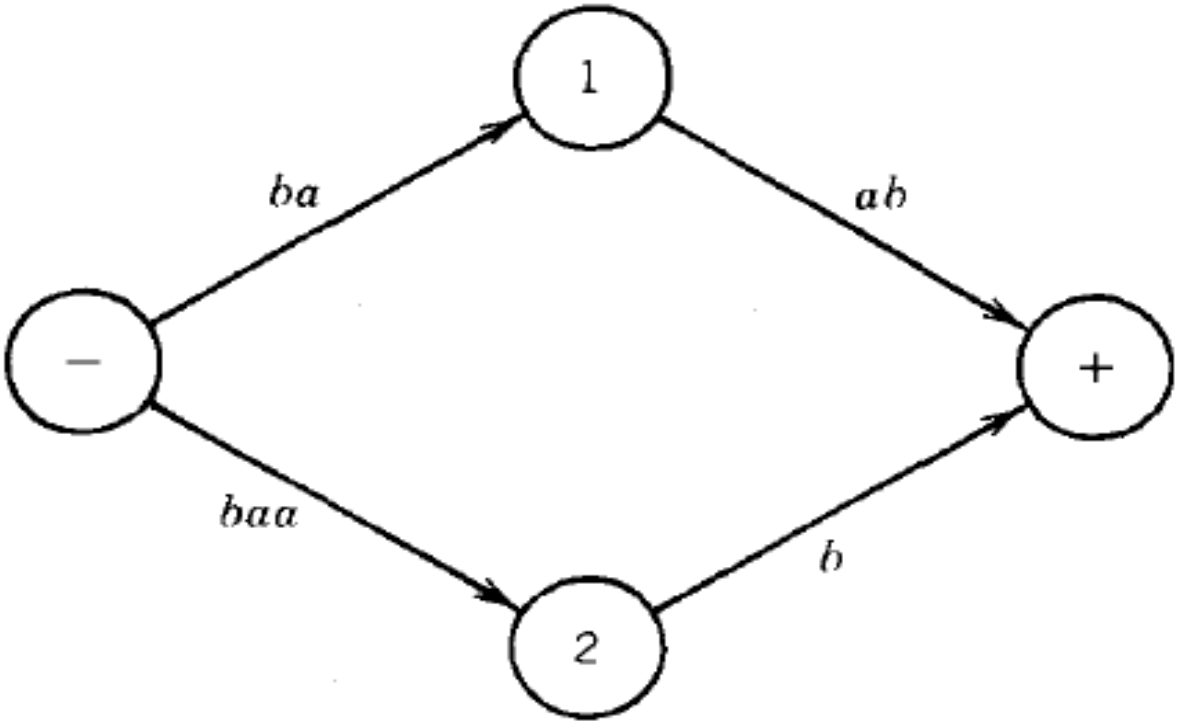
Is the word *abbbabbbbabba* accepted by this machine?

Is the word *abbbabbbabba* accepted by this machine? (Yes, in two ways.)

When we allow A-edges we may have an infinite number of ways of grouping the letters of an input string. For example, the input string *ab* may be factored as:

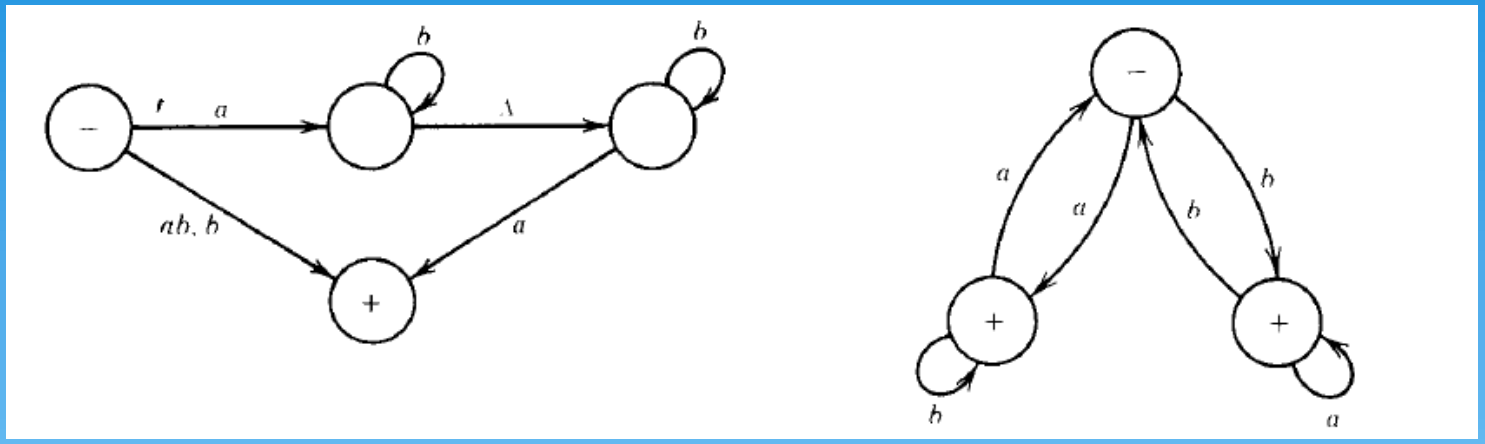
(a) (b)
(a) (A) *(b)*
(a) (A) (A) *(b)*
(a) (A) (A) (A) *(b)*

Problems

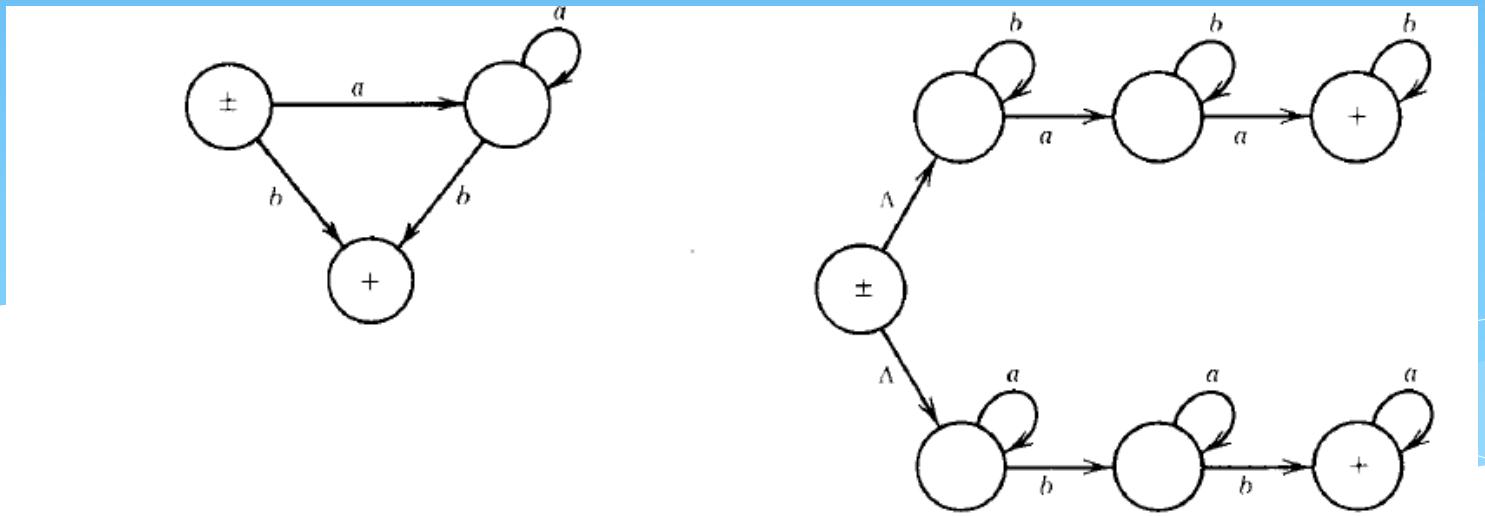


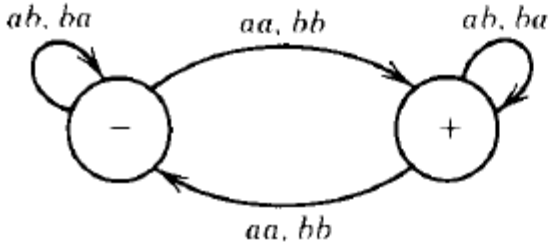
	b	*ab*	*ba*	*baa*
−			1	2
1		+		
2	+			
+				

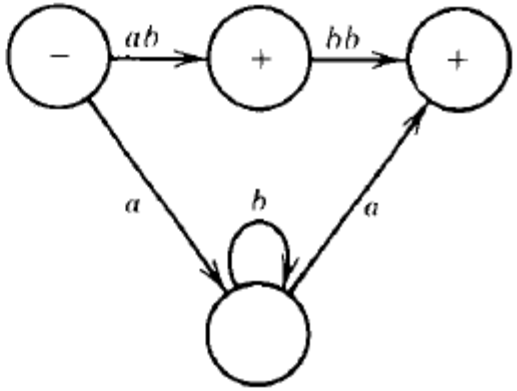
Here are six TG's. For each of the next 10 words decide which of these machines accepts the given word.



Find regular expressions defining the language accepted by each of the six TG's above.







(i) Λ
(ii) *a*
(iii) *b*
(iv) *aa*
(v) *ab*
(vi) *aba*
(vii) *abba*
(viii) *bab*
(ix) *baab*
(x) *abbb*

Computational Theory

كلية التربية للعلوم الصرفة / جامعة ديالى

Lecture 13

اعداد
م.د. محمد سامي محمد

قسم علوم الحاسوب
المرحلة الثانية

KLEENE'S THEOREM

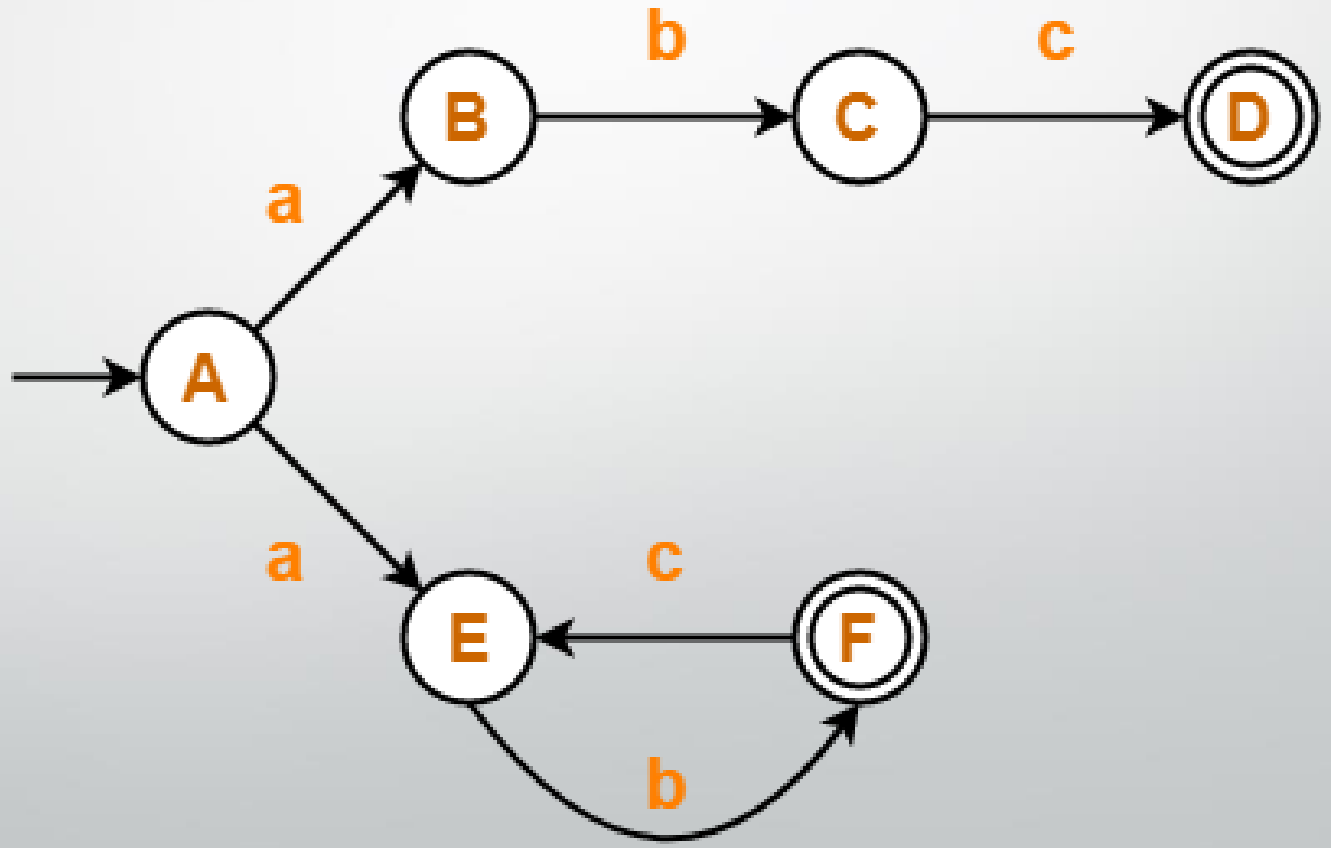
Part 1 Every language that can be defined by a finite automaton can also be defined by a transition graph.

Part 2 Every language that can be defined by a transition graph can also be defined by a regular expression.

Part 3 Every language that can be defined by a regular expression can also be defined by a finite automaton.

The Proof of Part 1

This is the easiest part. Every finite automaton *is* itself a transition graph. Therefore, any language that has been defined by a finite automaton has already been defined by a transition graph.

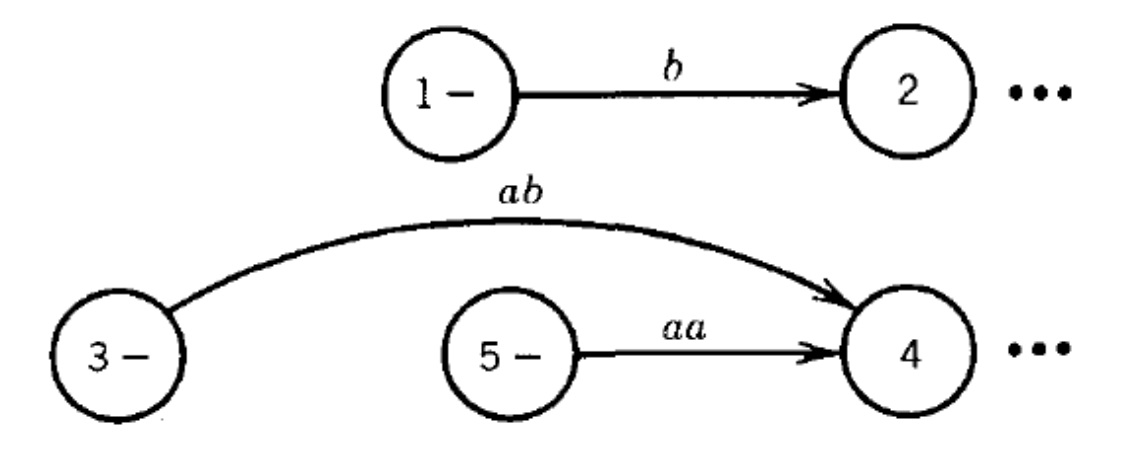


The Proof of Part 2

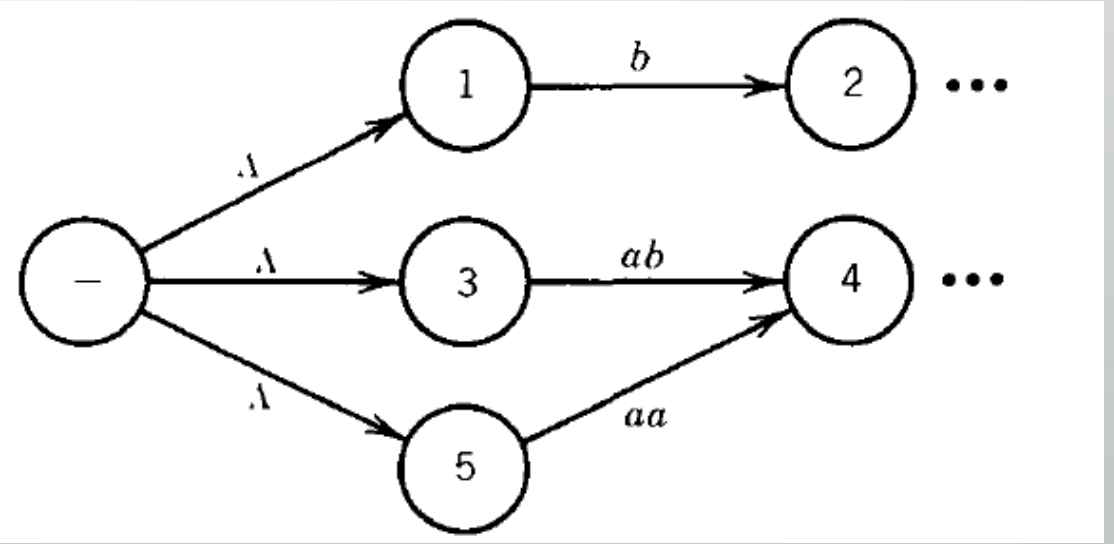
This means that we present a procedure that starts out with a transition graph and ends up with a regular expression that defines the same language.

To be acceptable as a method of proof, any algorithm must satisfy two criteria. It must work for every conceivable TG, and it must guarantee to finish its job in a finite time (a finite number of steps).

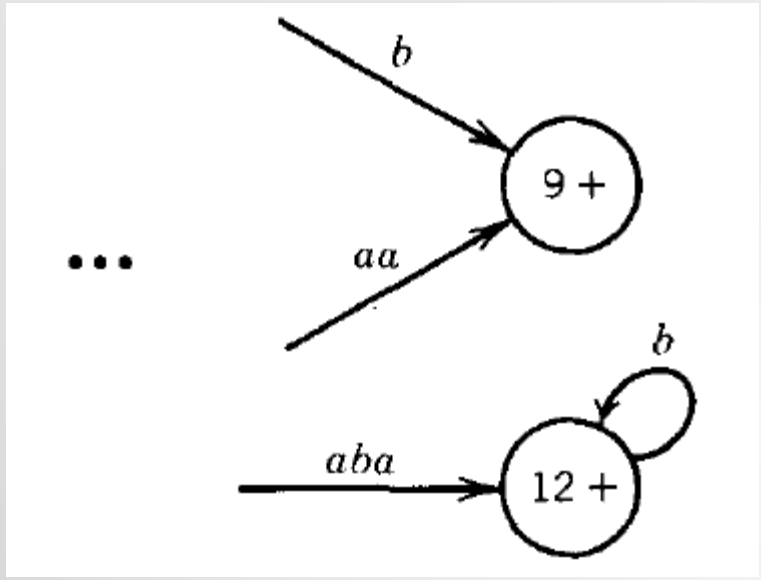
The Proof of Part 2



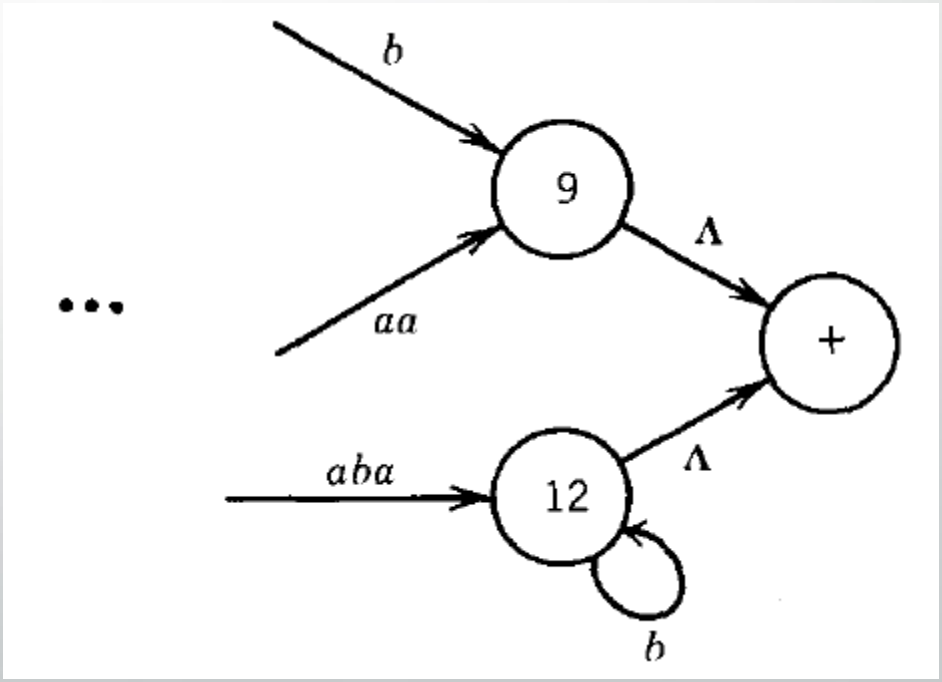
Make It one Start



The Proof of Part 2



Make It one
Final



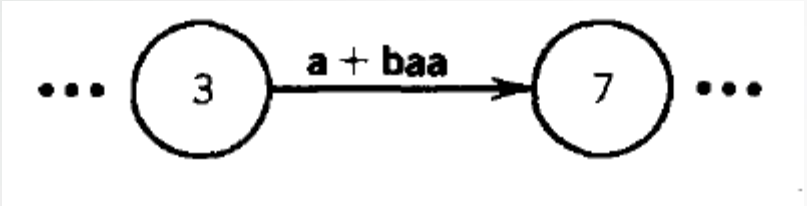
The Proof of Part 2

It should be clear that the addition of these two new states does not affect the language that T accepts. Any word accepted by the old T is also accepted by the new T, and any word rejected by the old T is also rejected by the new T.

We are now going to build the regular expression that defines the same language as T piece by piece. To do so we extend our notion of transition graph. We previously allowed the edges to be labeled only with strings of alphabet letters. For the purposes of this algorithm, we allow the edges to be labeled with regular expressions.

The Proof of Part 2

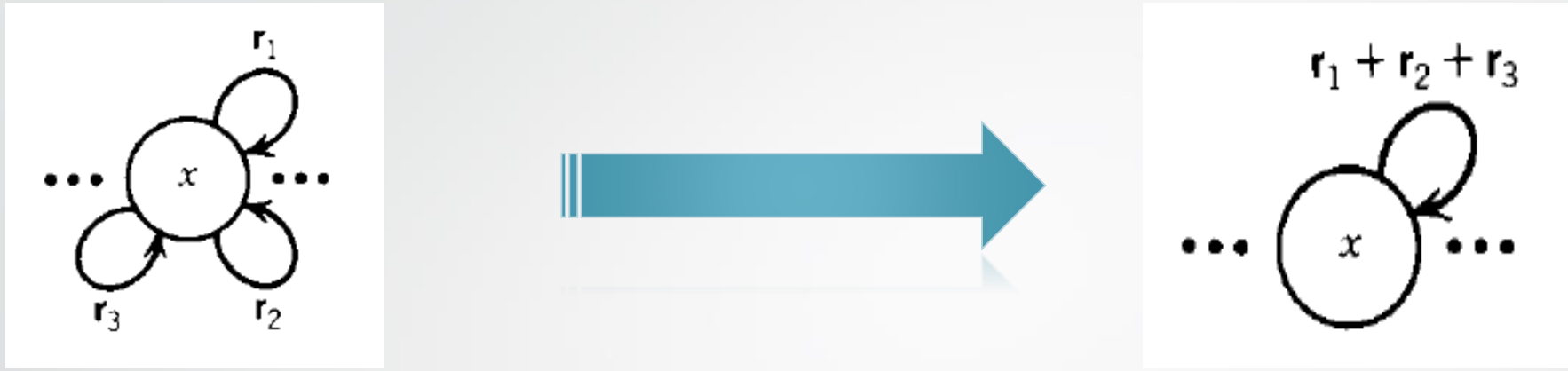
For example



we can cross from state 3 to state 7 by reading from the input string either the letter a alone or else the sequence baa.

Labeling an edge with the regular expression (ab)* means that we can cross the edge by reading any of the input sequences
A, ab, abab, ababab . . .

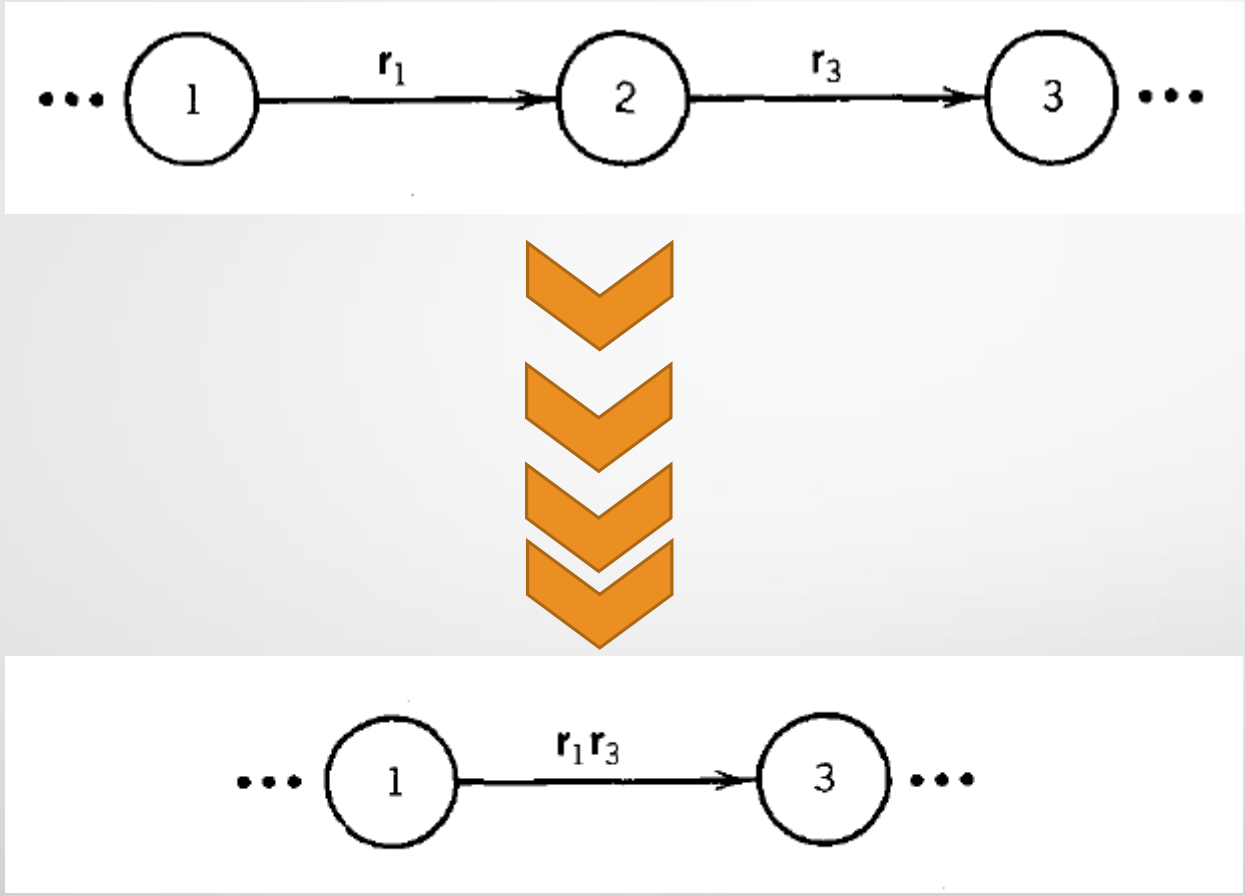
The Proof of Part 2



Bypass Operation

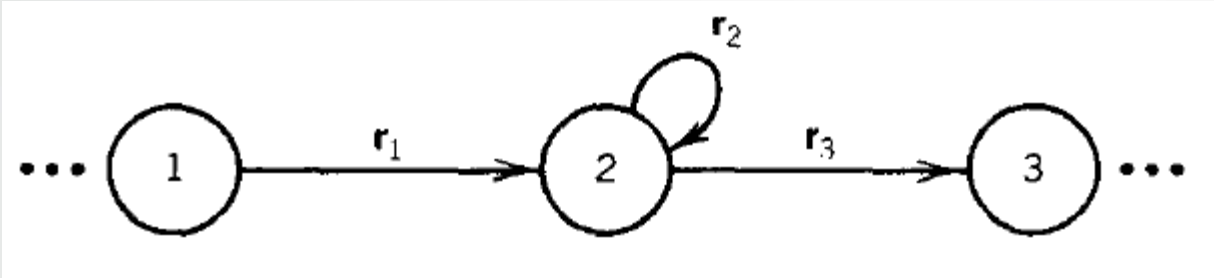
The Proof of Part 2

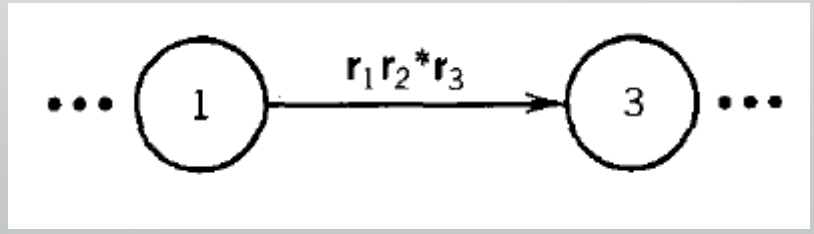
Example



The Proof of Part 2

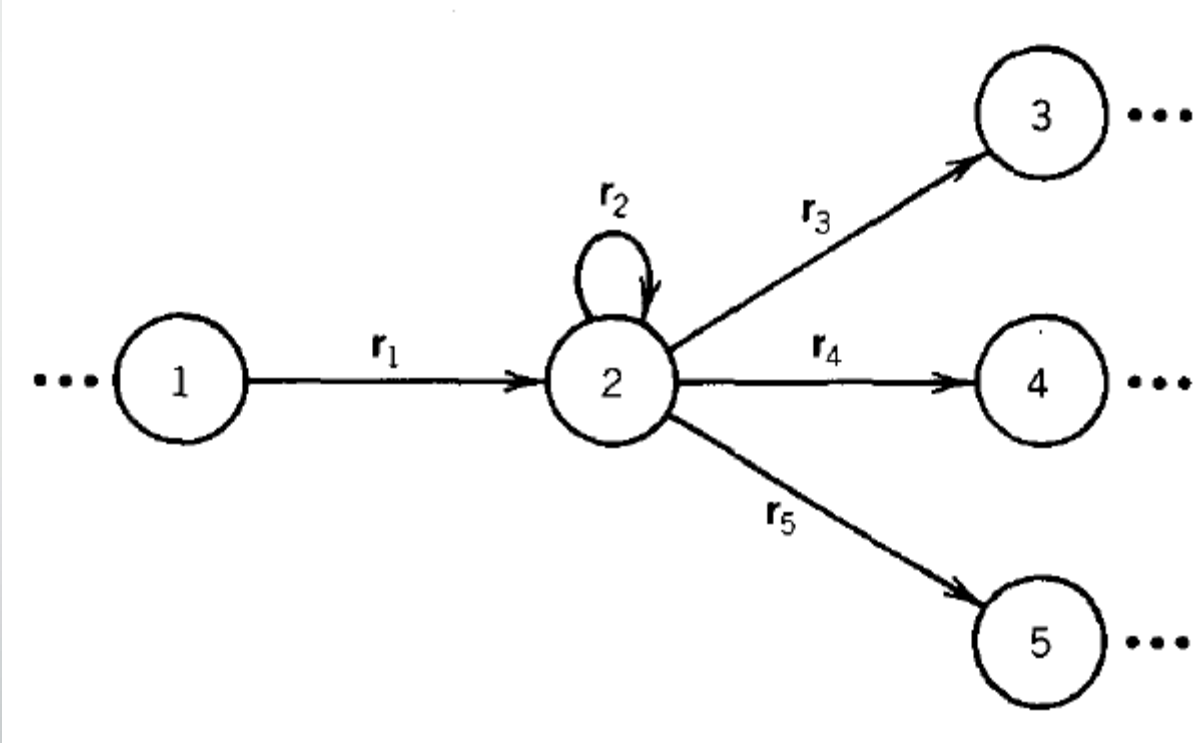
We can do the previous trick only as long as state 2 does not have a loop going back to itself. If state 2 does have a loop, we must use this model:



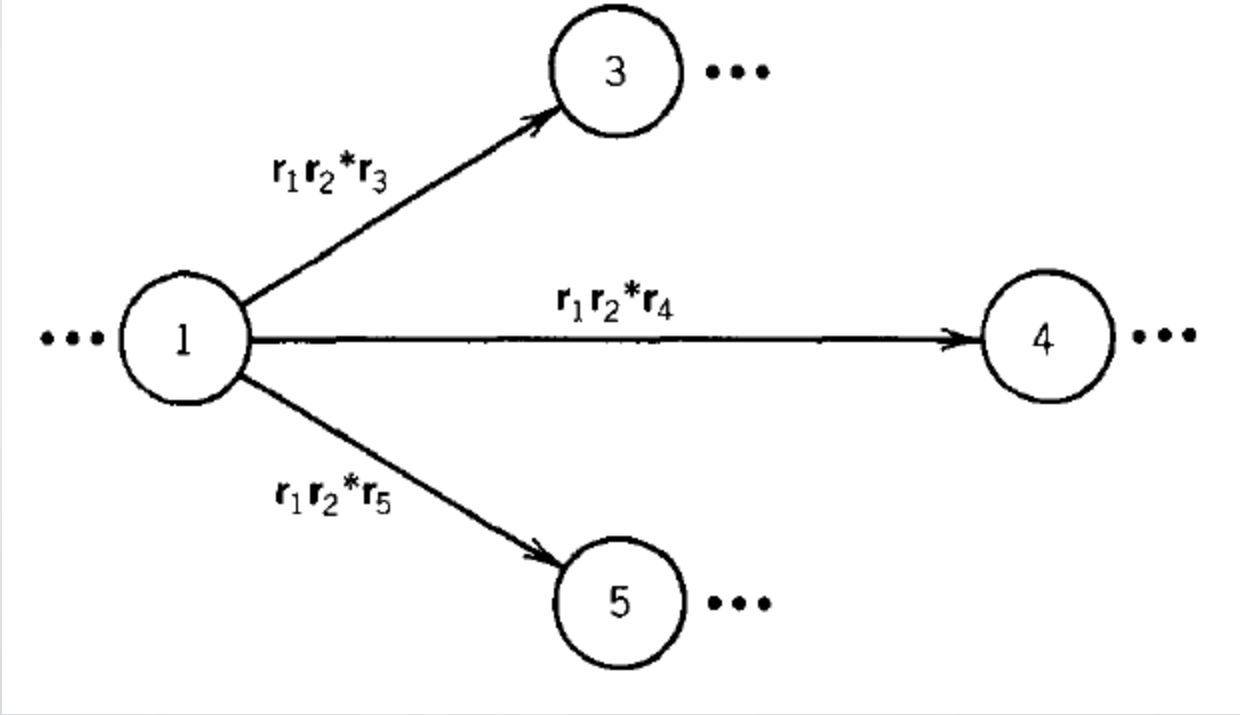


The Proof of Part 2

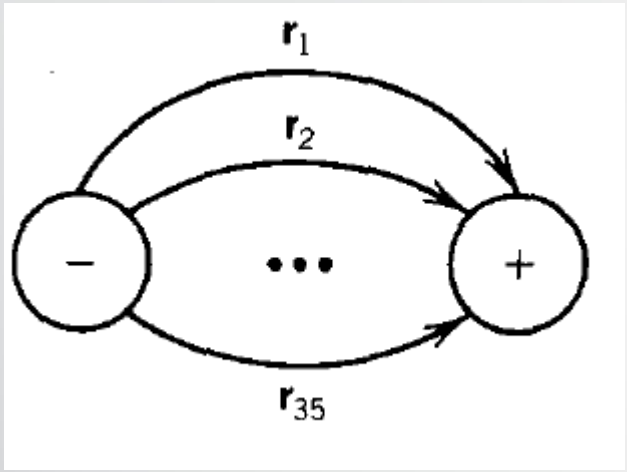
If state I is connected to state 2 and state 2 is connected to more than one other state (say to states 3, 4, and 5), then when we eliminate the edge from state 1 to state 2 we have to add edges that show how to go from state I to states 3, 4, and 5. We do this as in the pictures below.

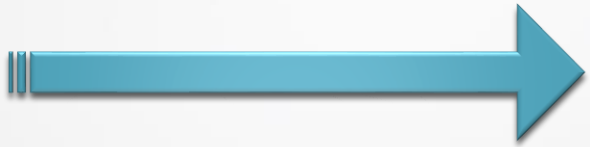


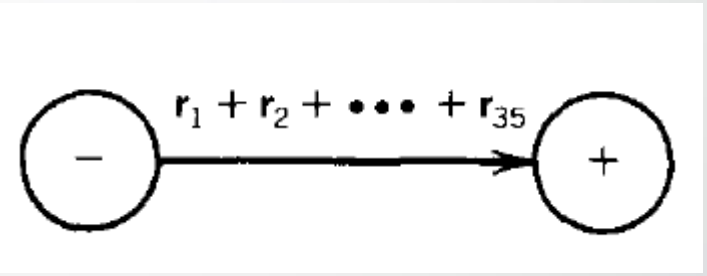
The Proof of Part 2



The Proof of Part 2







Computational Theory

كلية التربية للعلوم الصرفة / جامعة ديالى

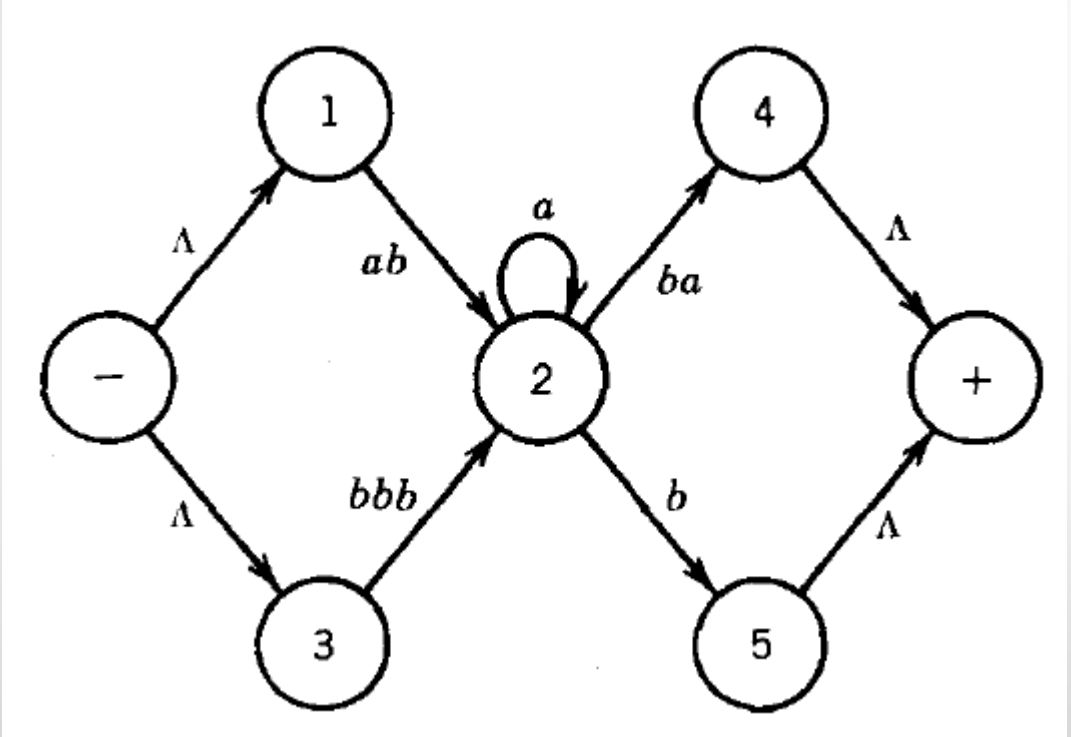
Lecture 14

اعداد
م.د. محمد سامي محمد

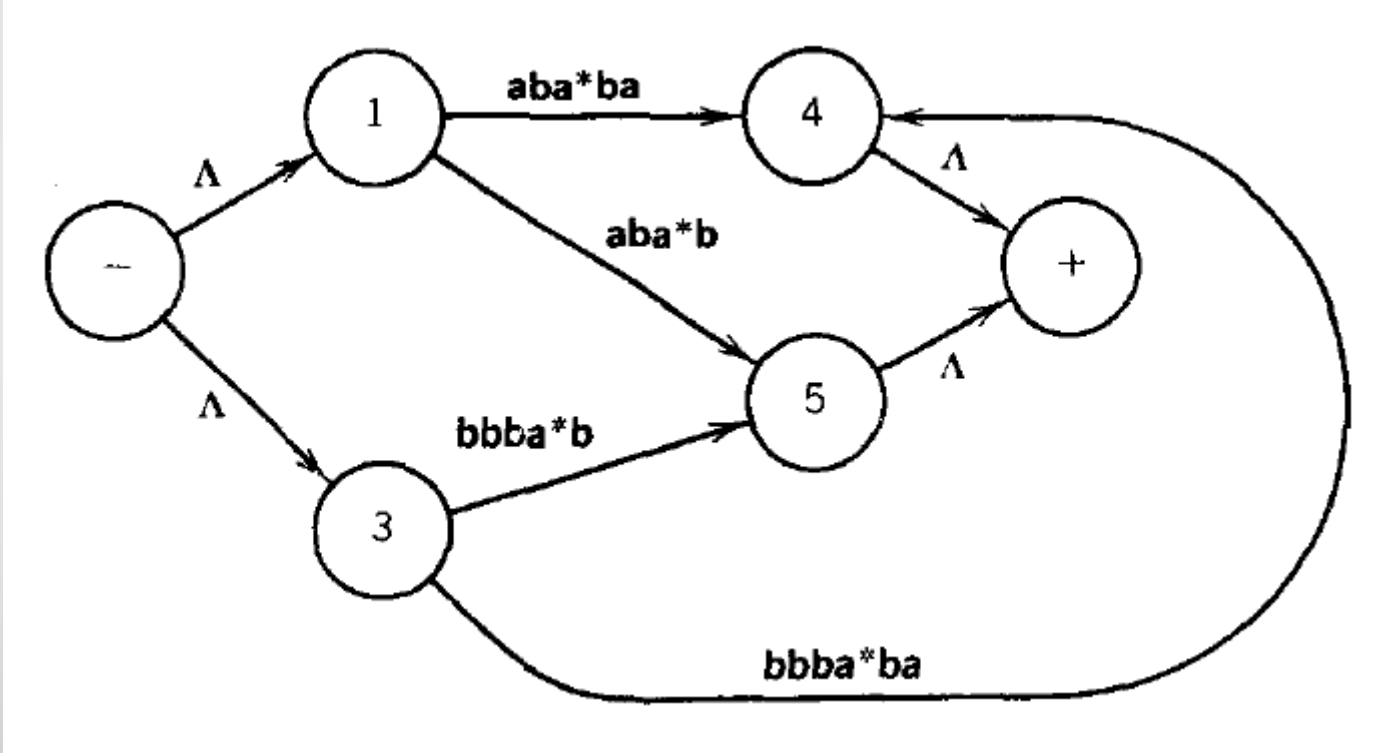
قسم علوم الحاسوب
المرحلة الثانية

The Proof of Part 2

For example,

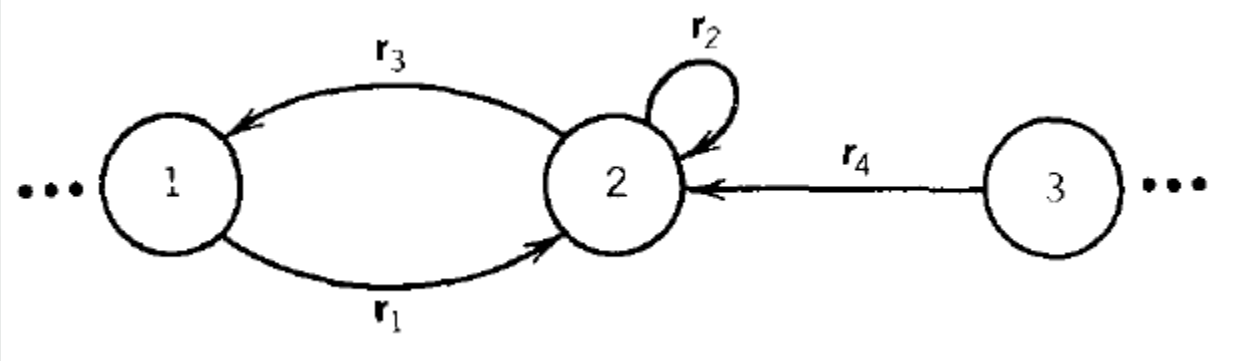


The Proof of Part 2



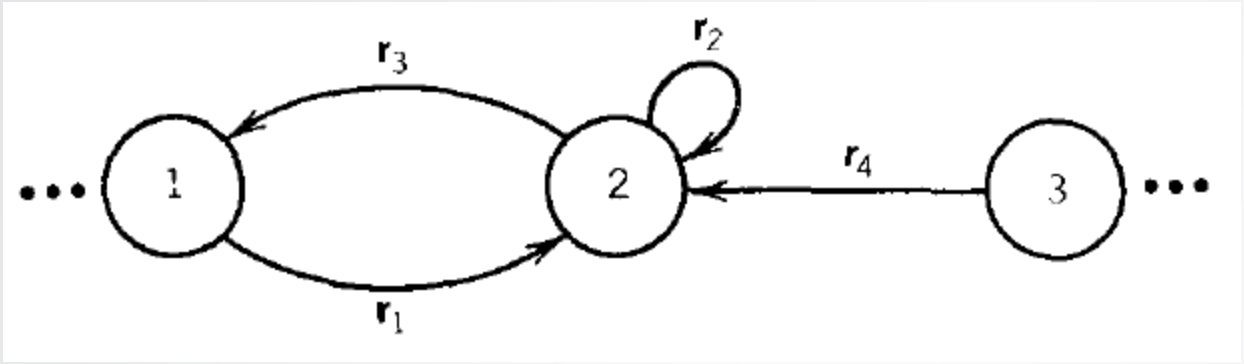
The Proof of Part 2

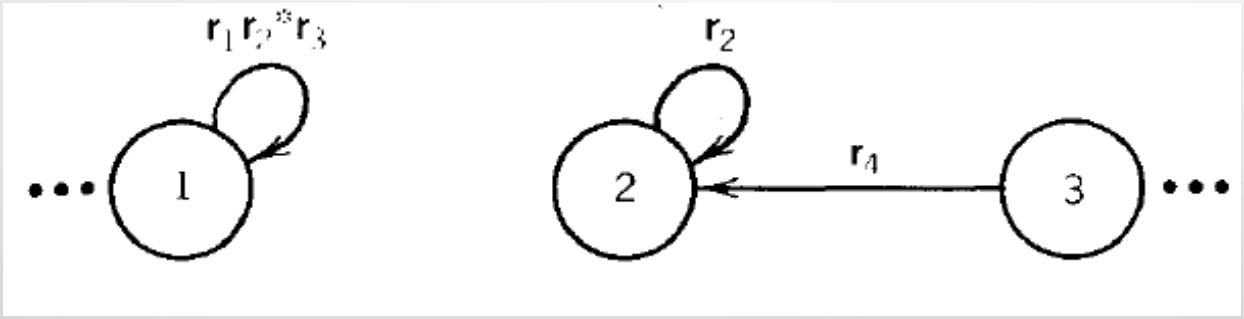
Before we claim to have finished describing this algorithm, there are some special cases that we must examine more carefully. In this picture



goes nowhere except into state 1, we might think that we can rewrite this part of T as:

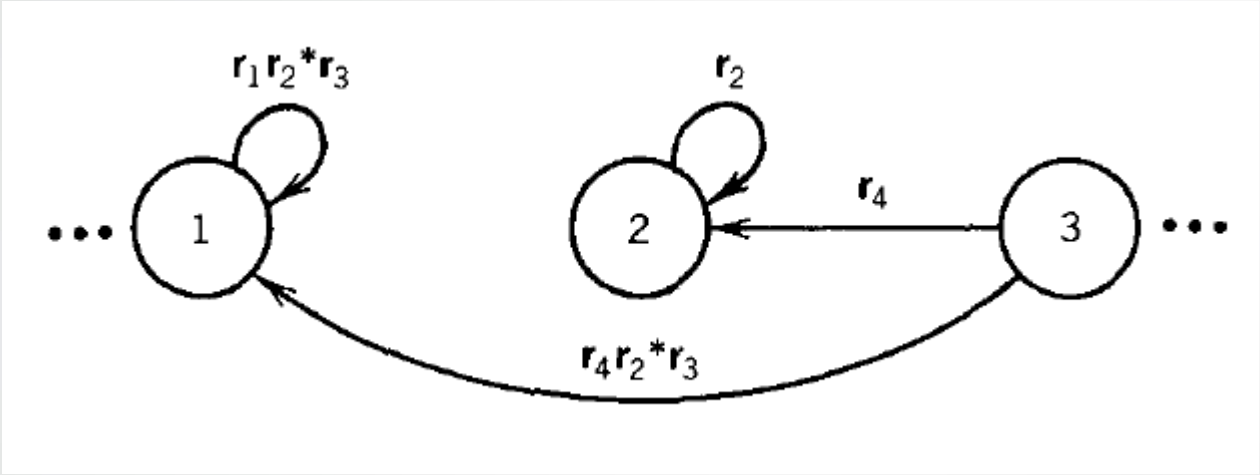
The Proof of Part 2





but this is wrong. In the original picture, we could go from state 3 to state 1, while in the modified picture that is impossible. Therefore, we must introduce an edge from state 3 to state 1 and label it as next slide show.

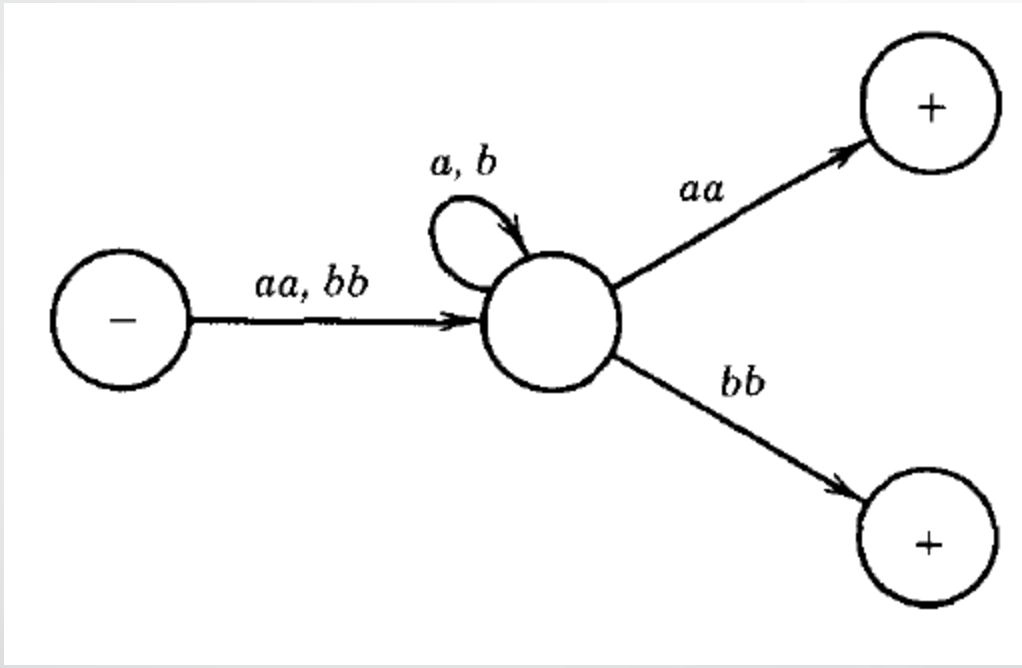
The Proof of Part 2



Whenever we remove an edge or a state we must be sure that we have not destroyed any paths through T that may previously have existed. Destroying paths could change the language of words accepted, which we do not want to do.

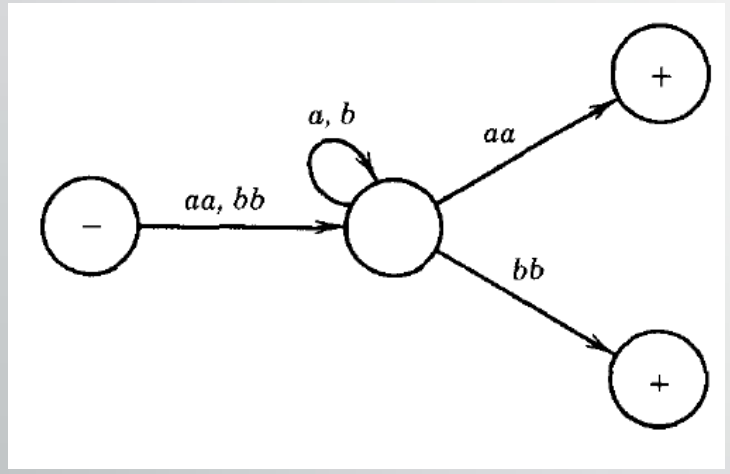
The Proof of Part 2

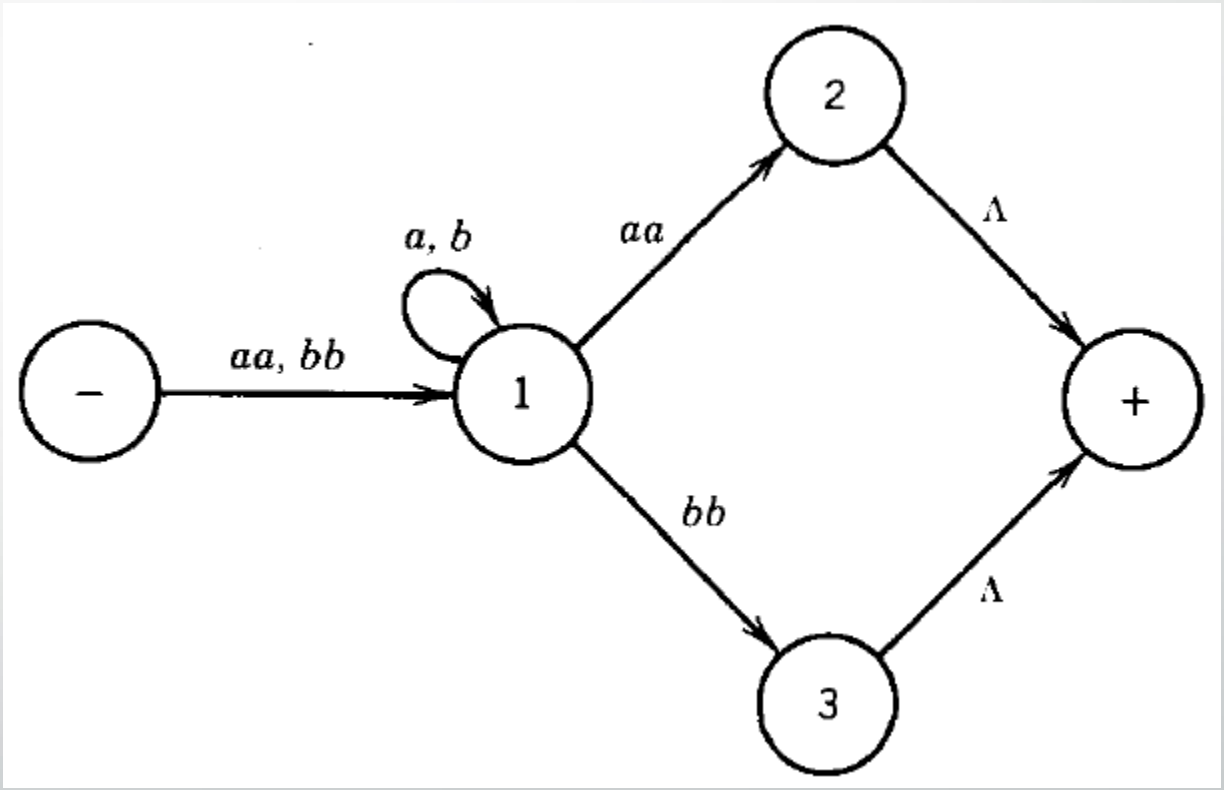
let us illustrate the algorithm above on a particular example.



The Proof of Part 2

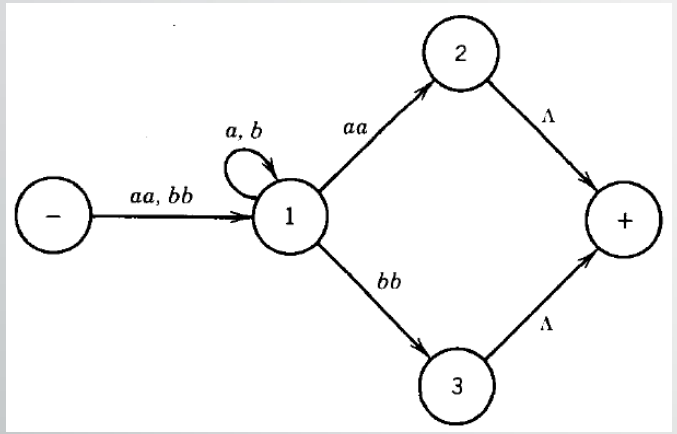
As it stands, this machine has only one start state, but it has two final states, so we must introduce a new unique final state following the method prescribed by the algorithm.

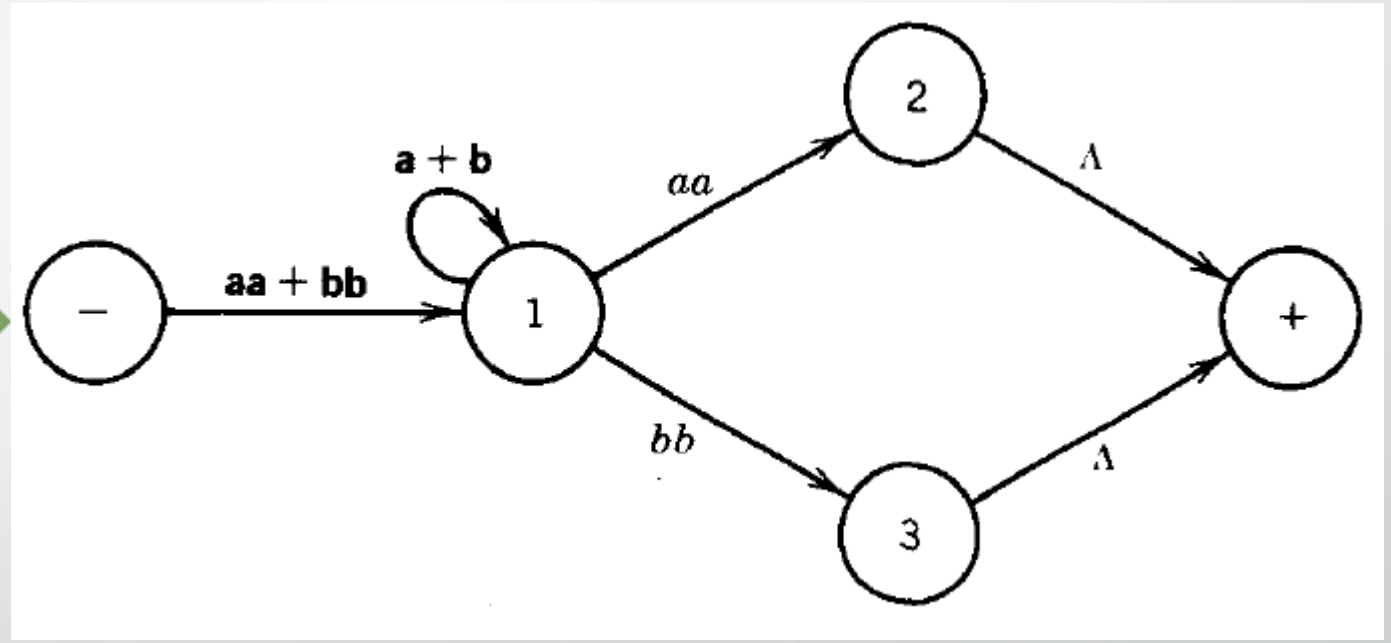




The Proof of Part 2

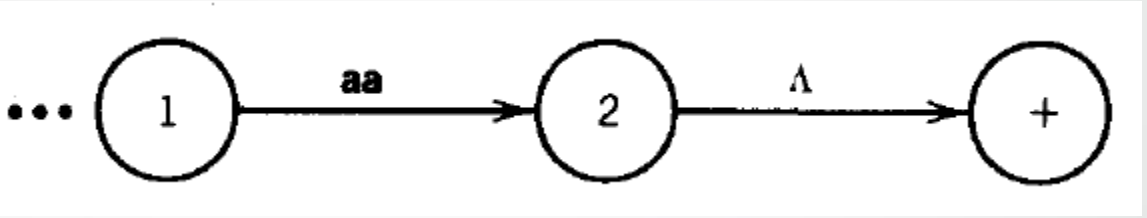
The algorithm says we are supposed to replace this double loop by a single loop labeled with the regular expression a + b. The picture of the machine has now become:

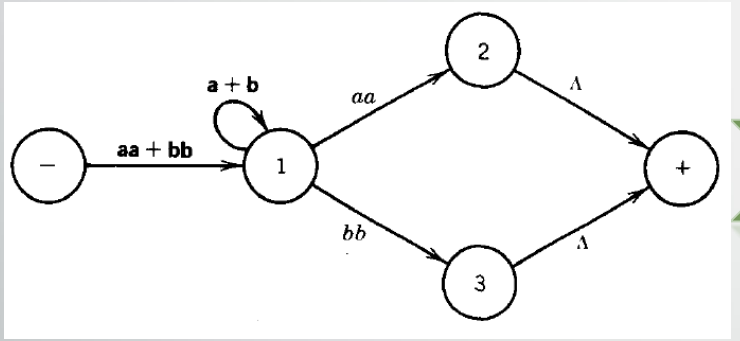


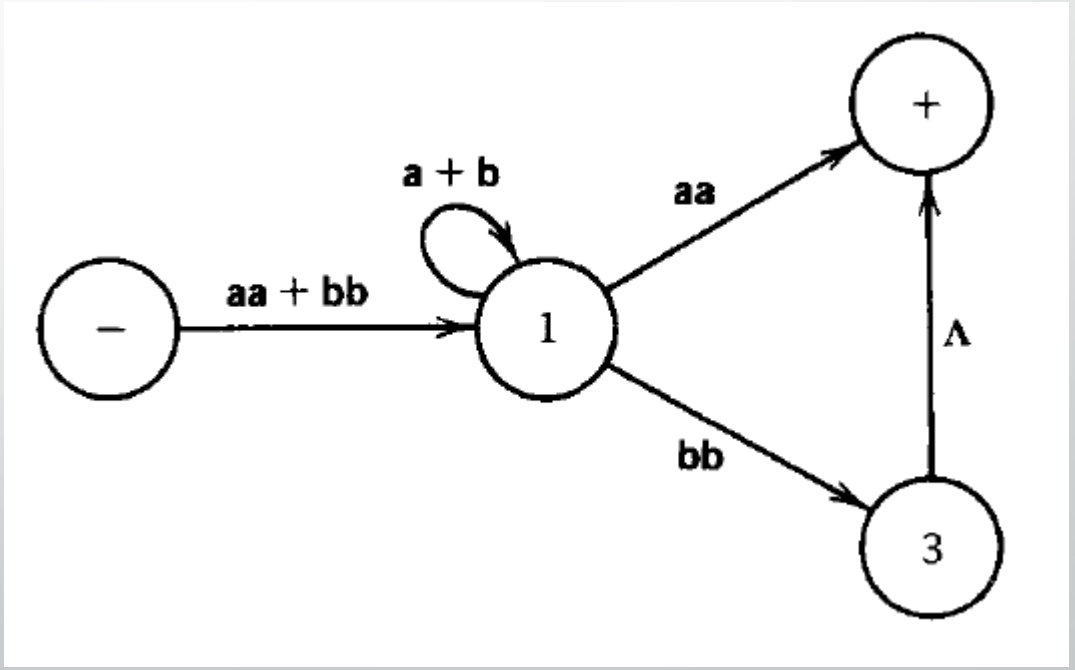


The Proof of Part 2

Let us choose for our next modification the path from state 1 to state 2 to state +.

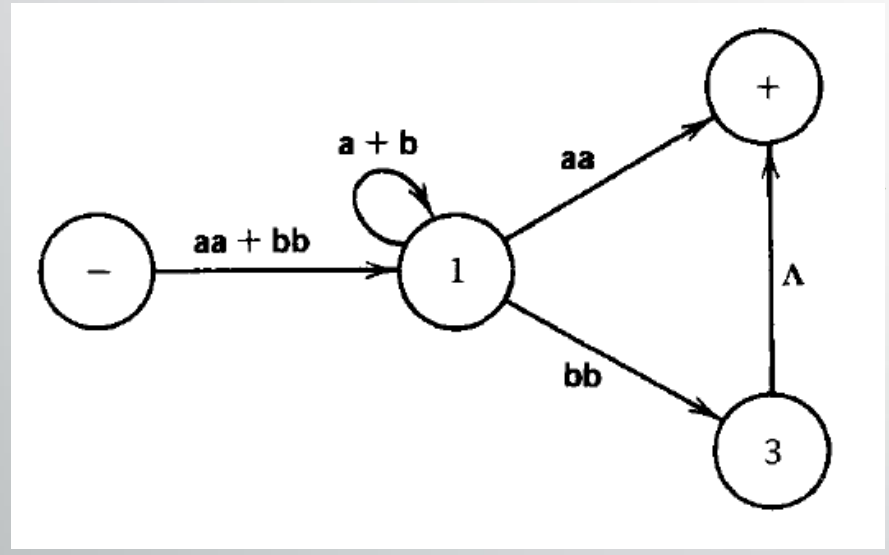


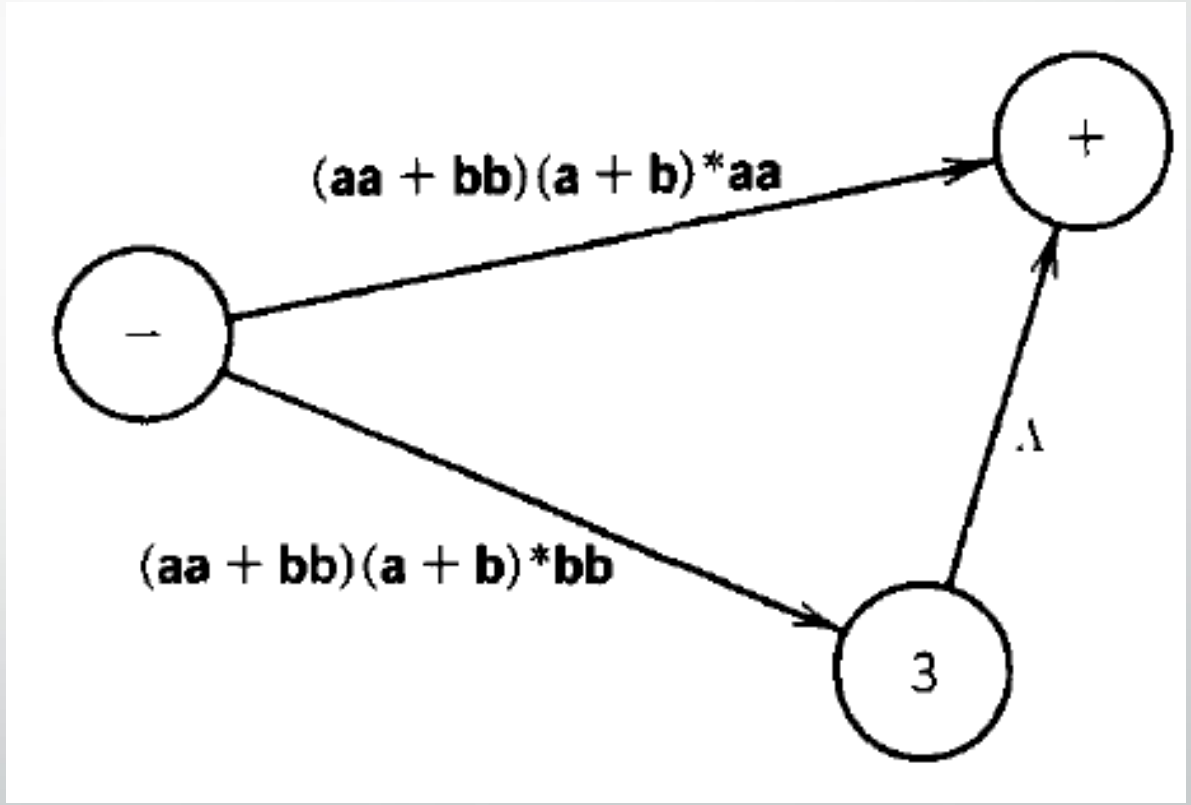




The Proof of Part 2

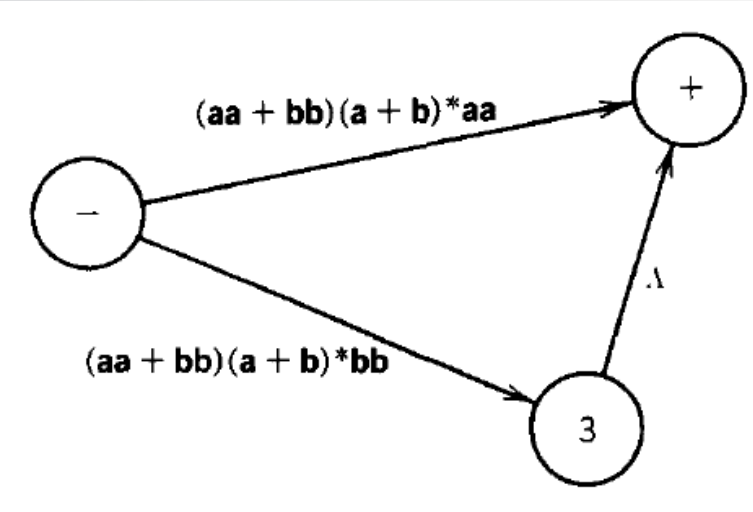
Let us try to bypass state 1. Only one edge comes into state 1 and that is from state –. There is a loop at state 1 with the label (a + b). State 1 has edges coming out of it that lead to state 3 and state +.

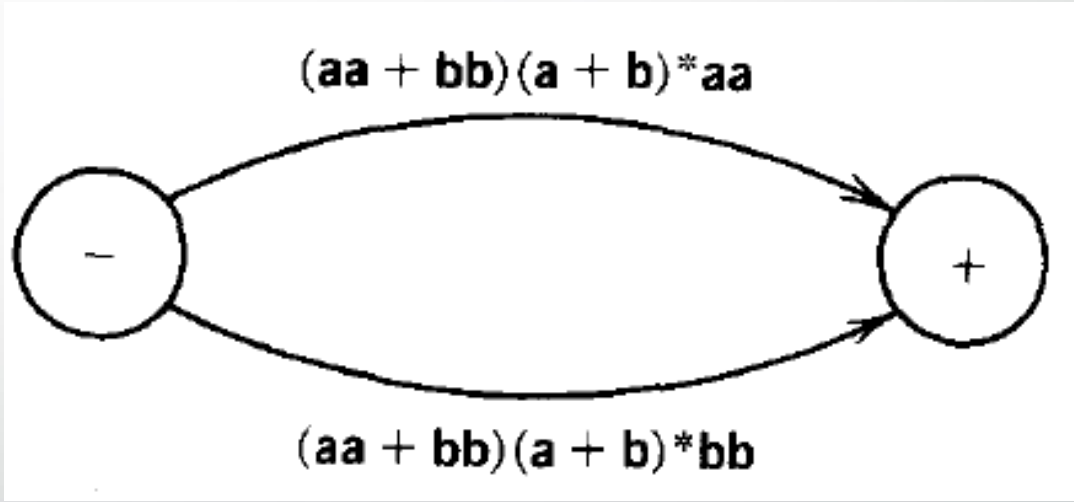




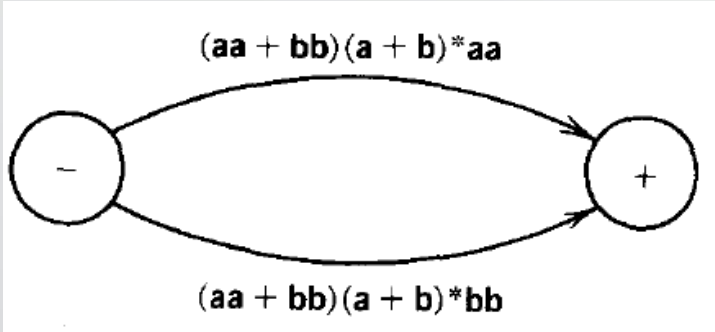
The Proof of Part 2

It is obvious that we must now eliminate state 3, since that is the only by passable state left. When we concatenate the regular expression from state - to state 3 with the regular expression from state 3 to state +, we are left with the machine:

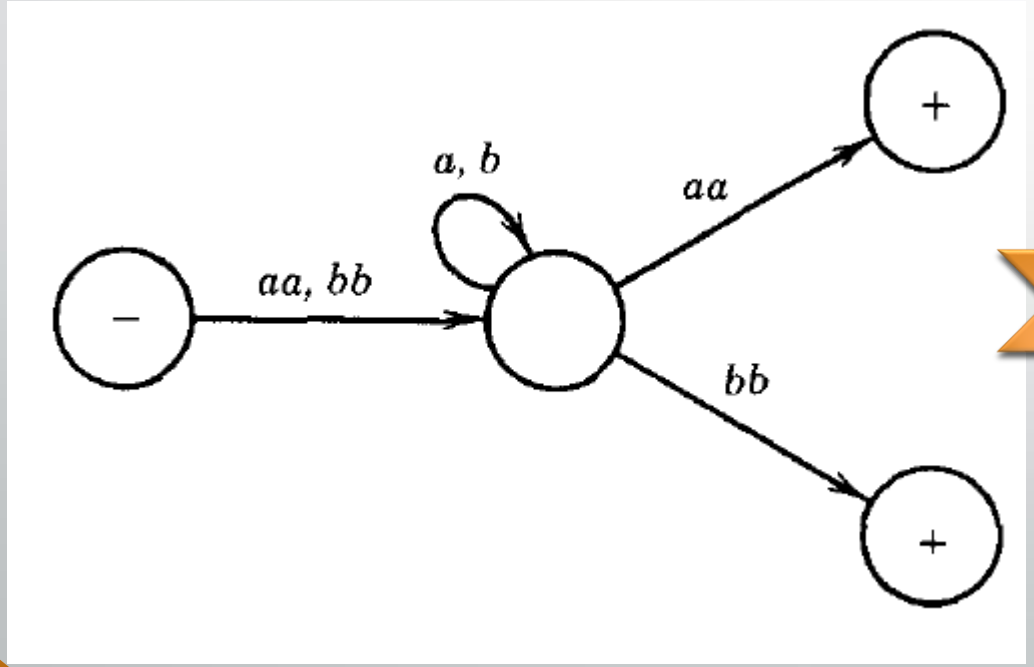




The Proof of Part 2

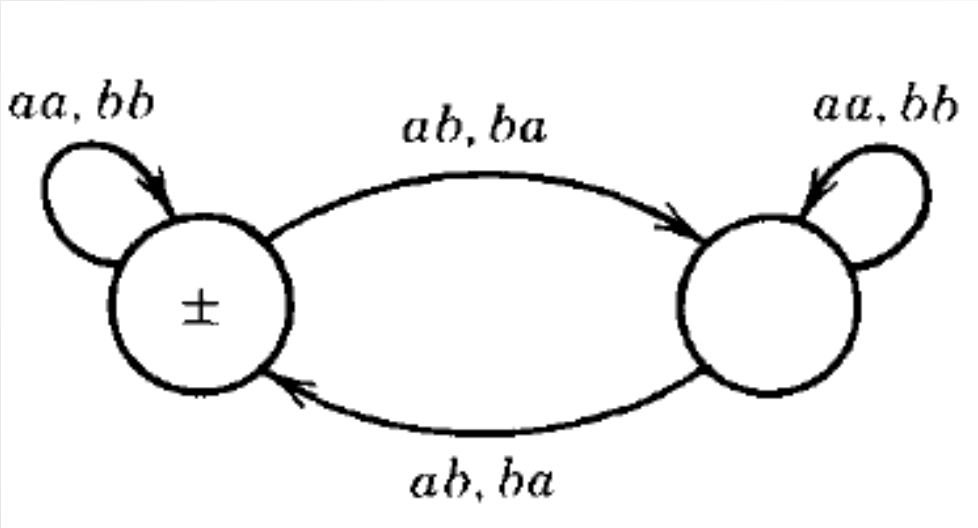


(aa + bb)(a + b)*(aa) + (aa + bb)(a + b)*(bb)

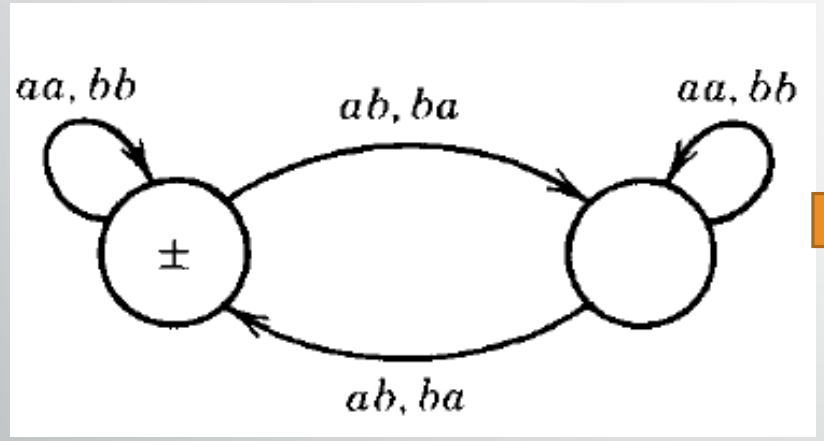


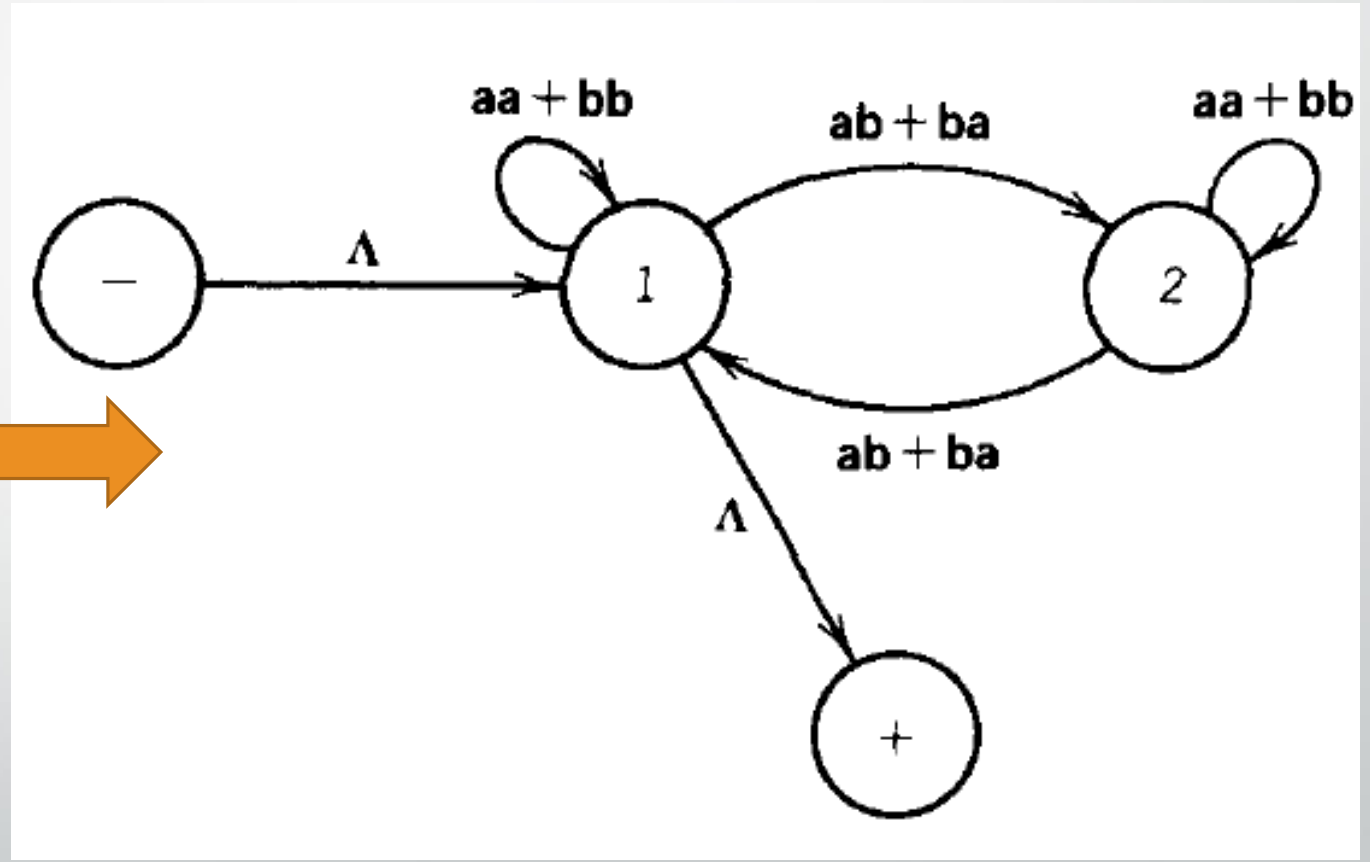
(aa + bb)(a + b)*(aa) + (aa + bb)(a + b)*(bb)

The Proof of Part 2

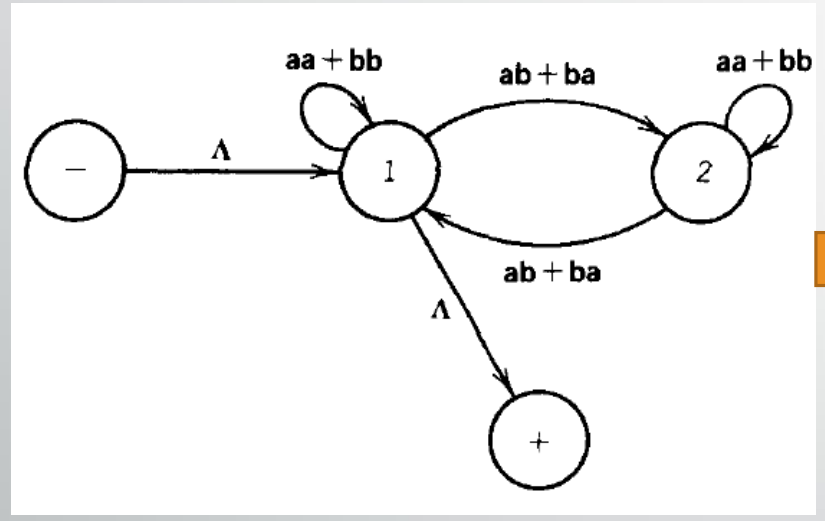


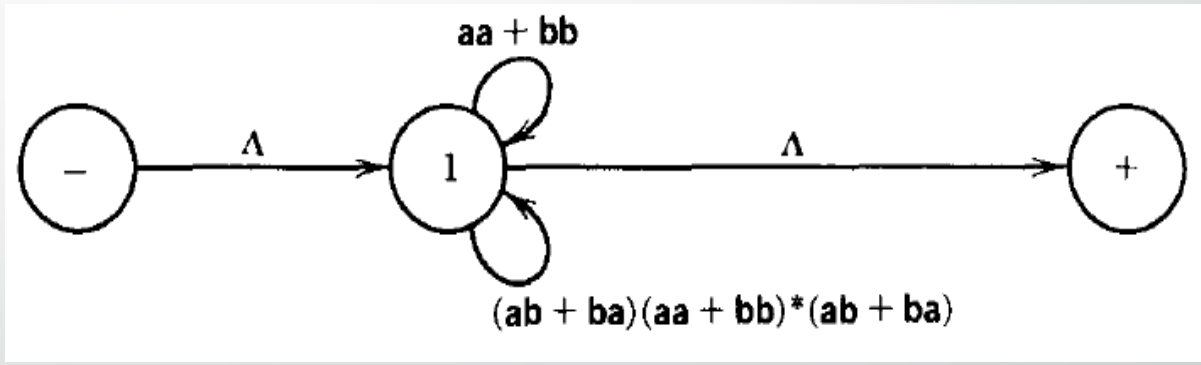
The Proof of Part 2



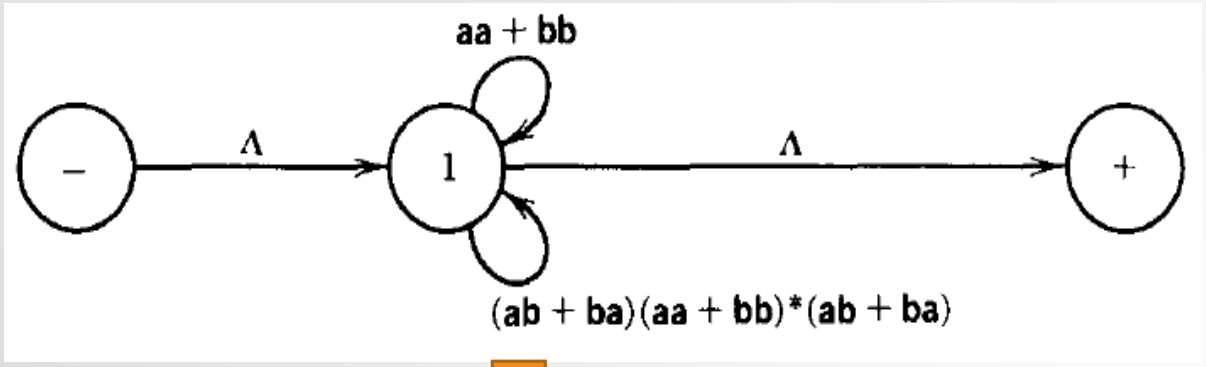


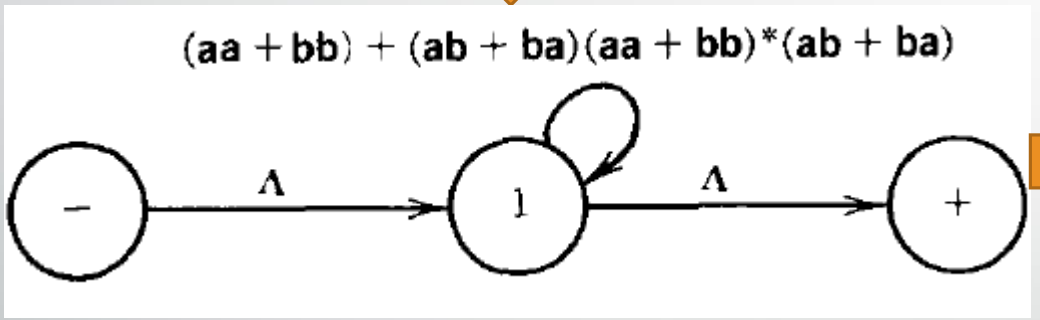
The Proof of Part 2

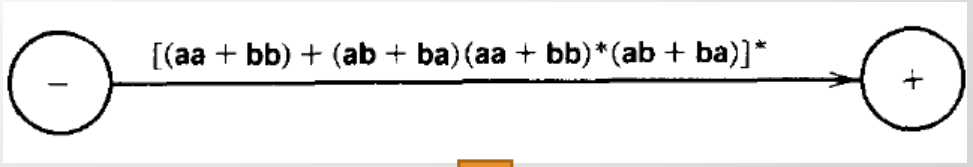




The Proof of Part 2

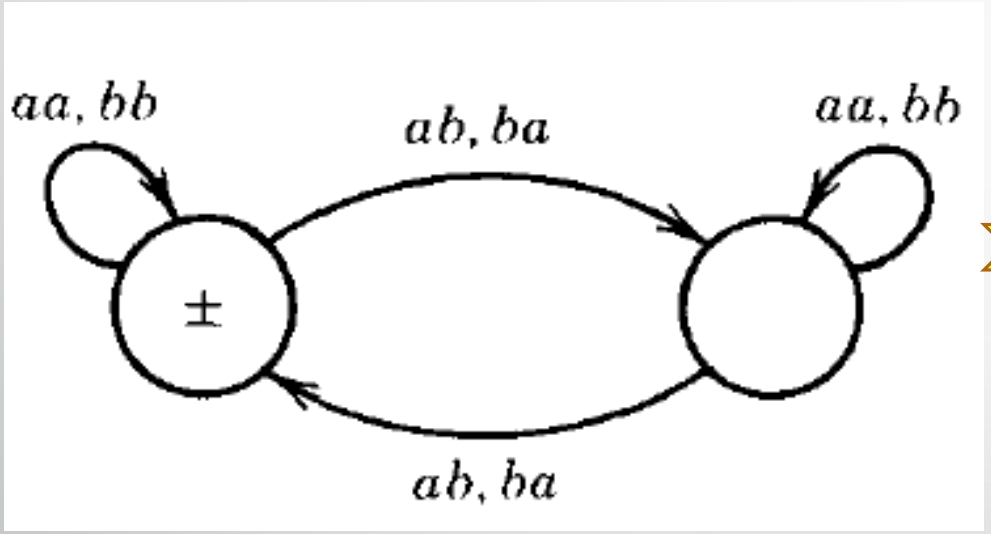






[(aa + bb) + (ab + ba) (aa + bb)*(ab + ba)]*

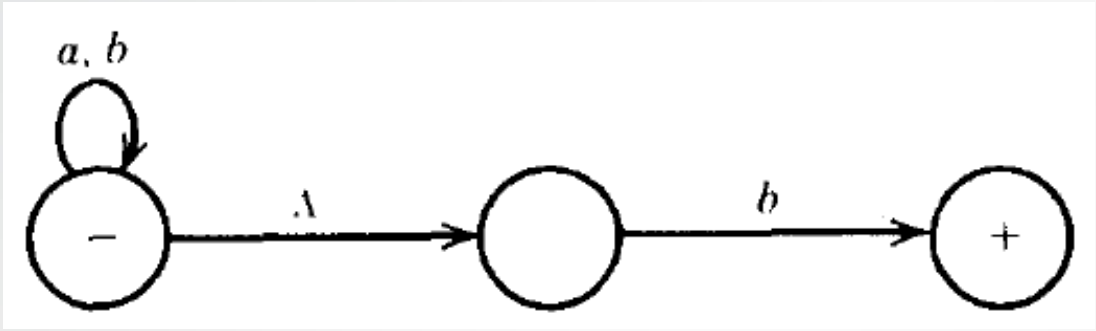
The Proof of Part 2

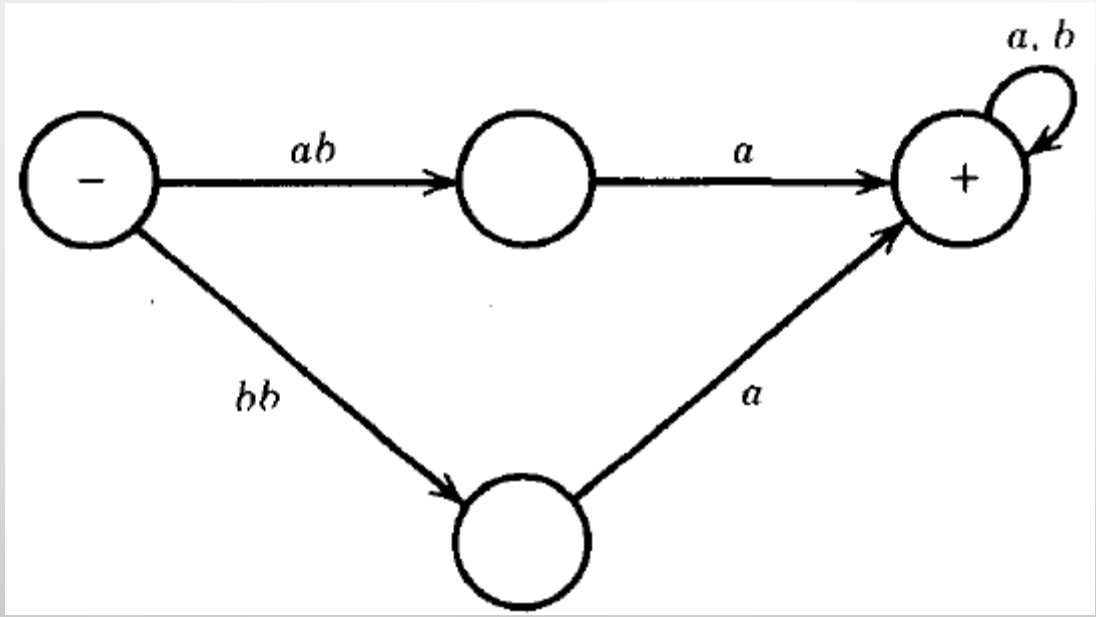


[(AA + BB) + (AB + BA) (AA + BB)*(AB + BA)]*

The Proof of Part 2

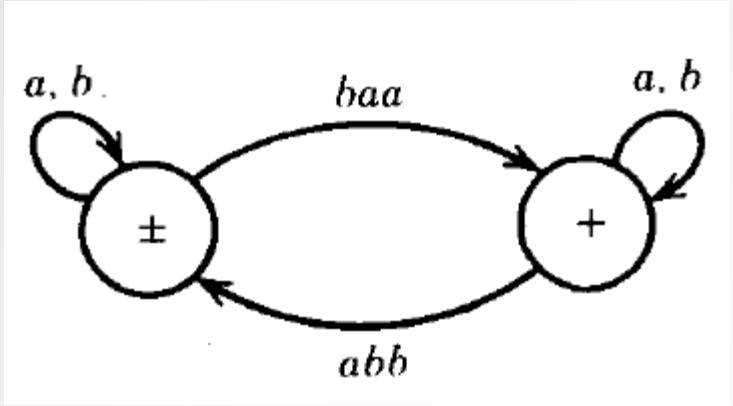
PROBLEMS

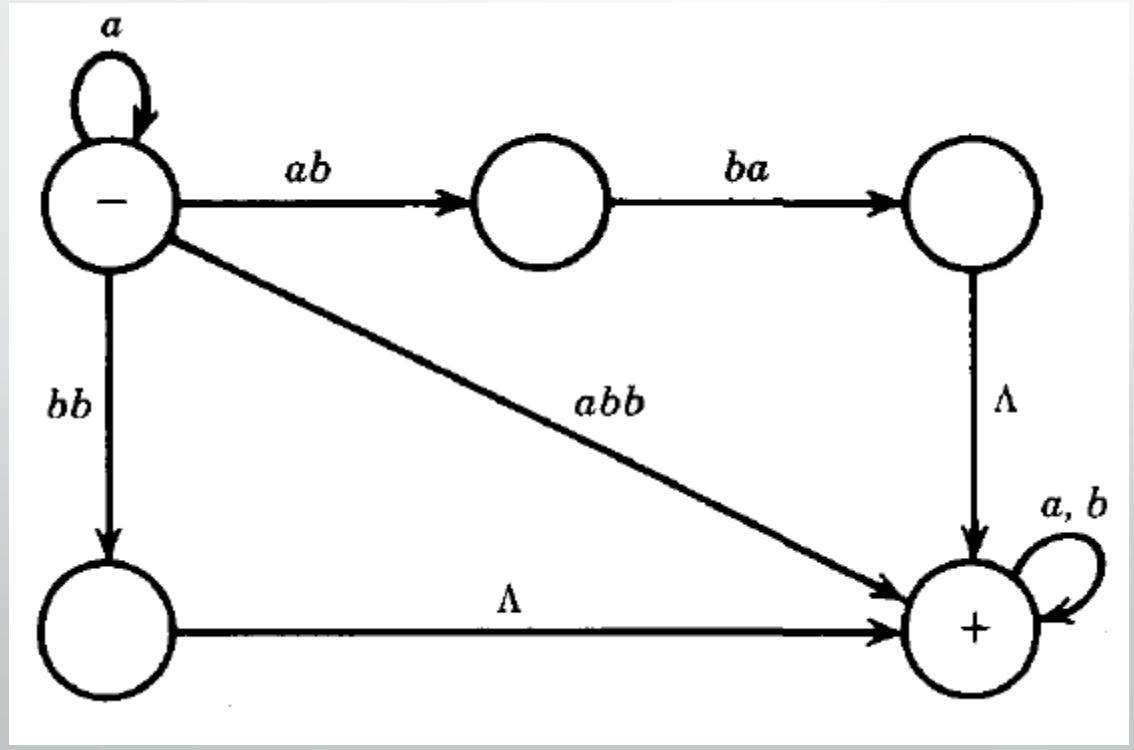




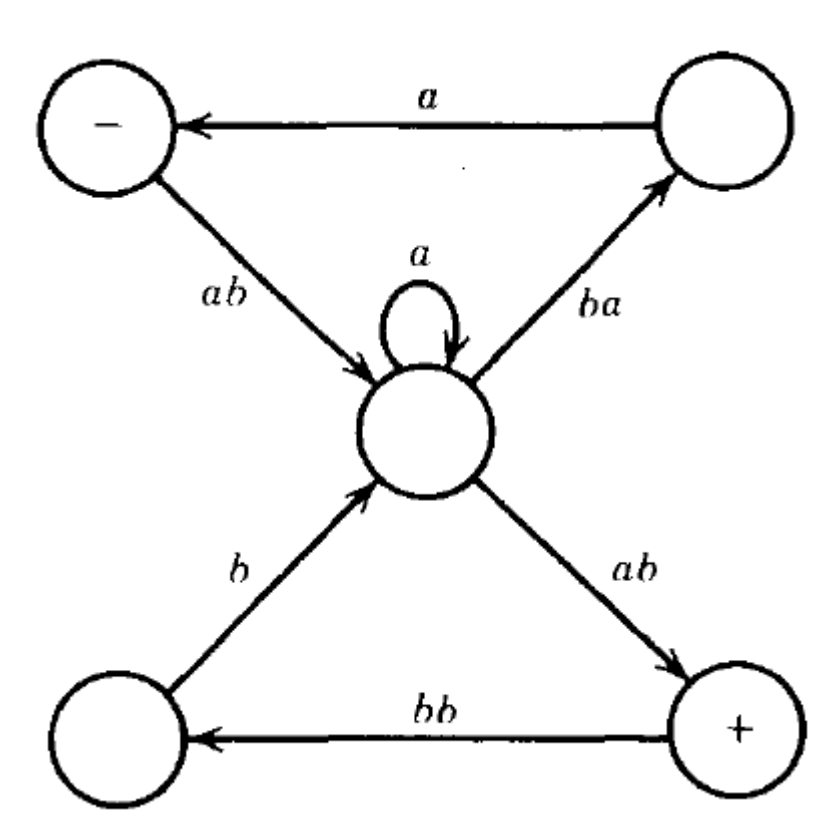
The Proof of Part 2

PROBLEMS





The Proof of Part 2



Theory of Computation

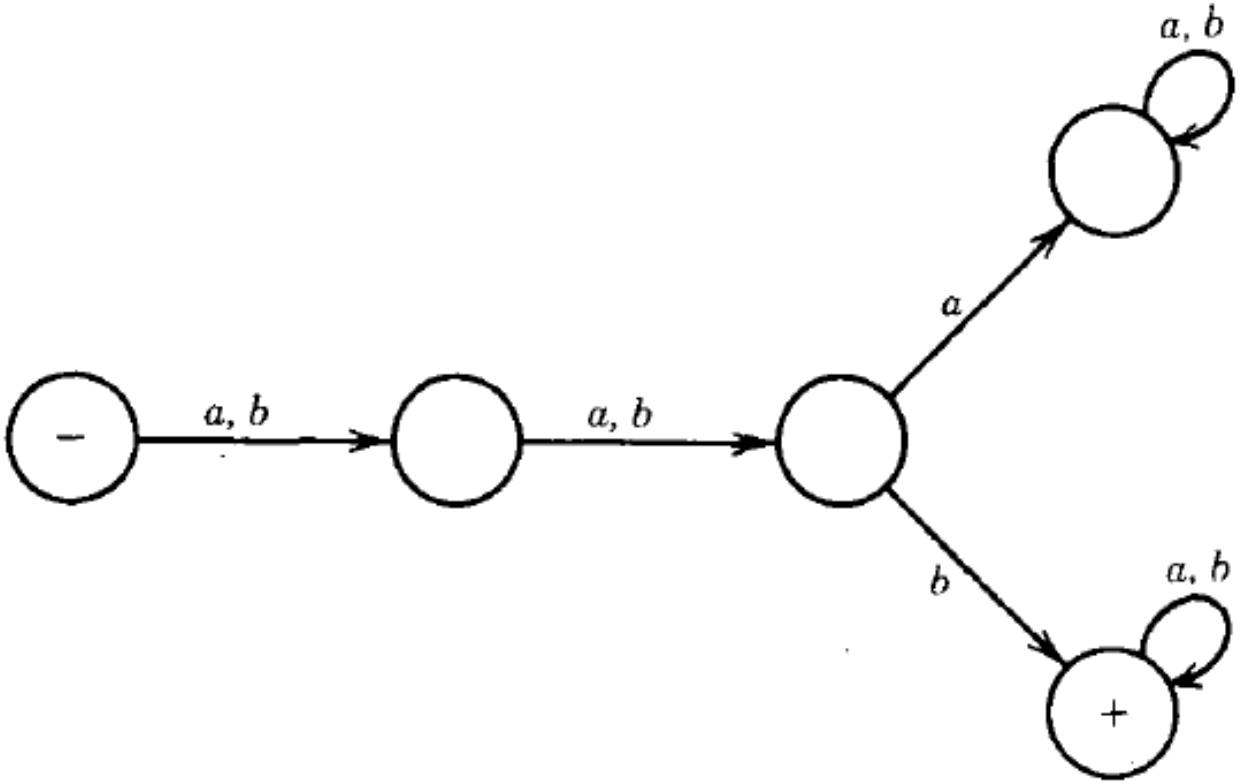
النظرية الاحتسابية
المحاضرة الثامنة

كلية التربية للعلوم الصرفة / جامعة ديالى

اعداد
م.د. محمد سامي محمد

قسم علوم الحاسوب
المرحلة الثانية

EXAMPLE: Let us consider the FA pictured below:



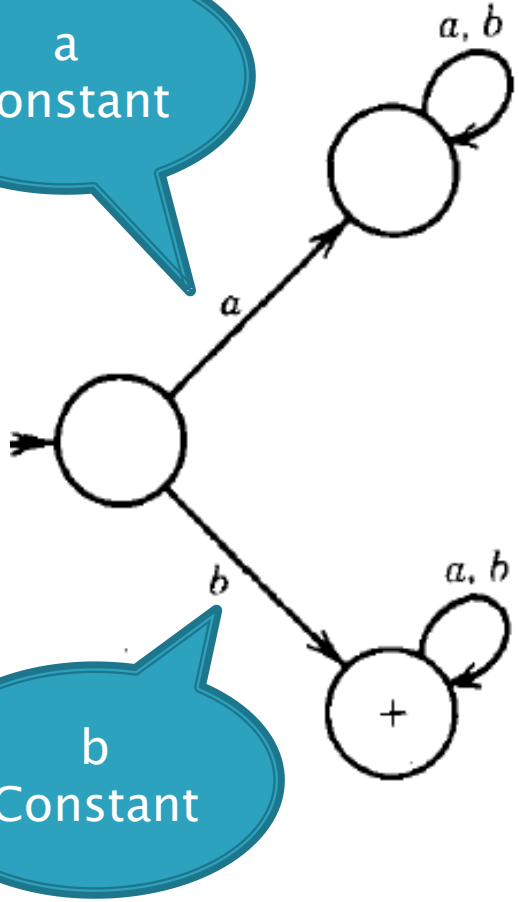
(aab + abb + bab + bbb)(a + b)*

الدكتور المهندس محمد سامي محمد

Notes before solve:-

Note 1

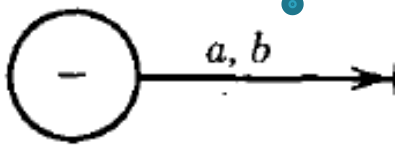
a
Constant



b
Constant

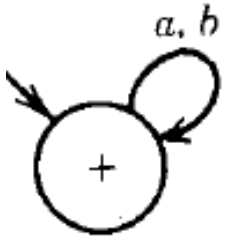
Note 2

(a+b)



Note 3

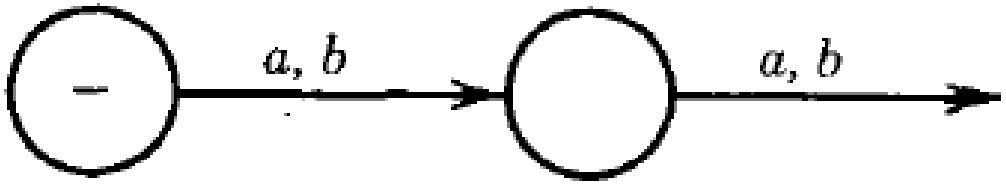
(a+b)*



الدكتور المهندس محمد سامي محمد

Notes before solve:-

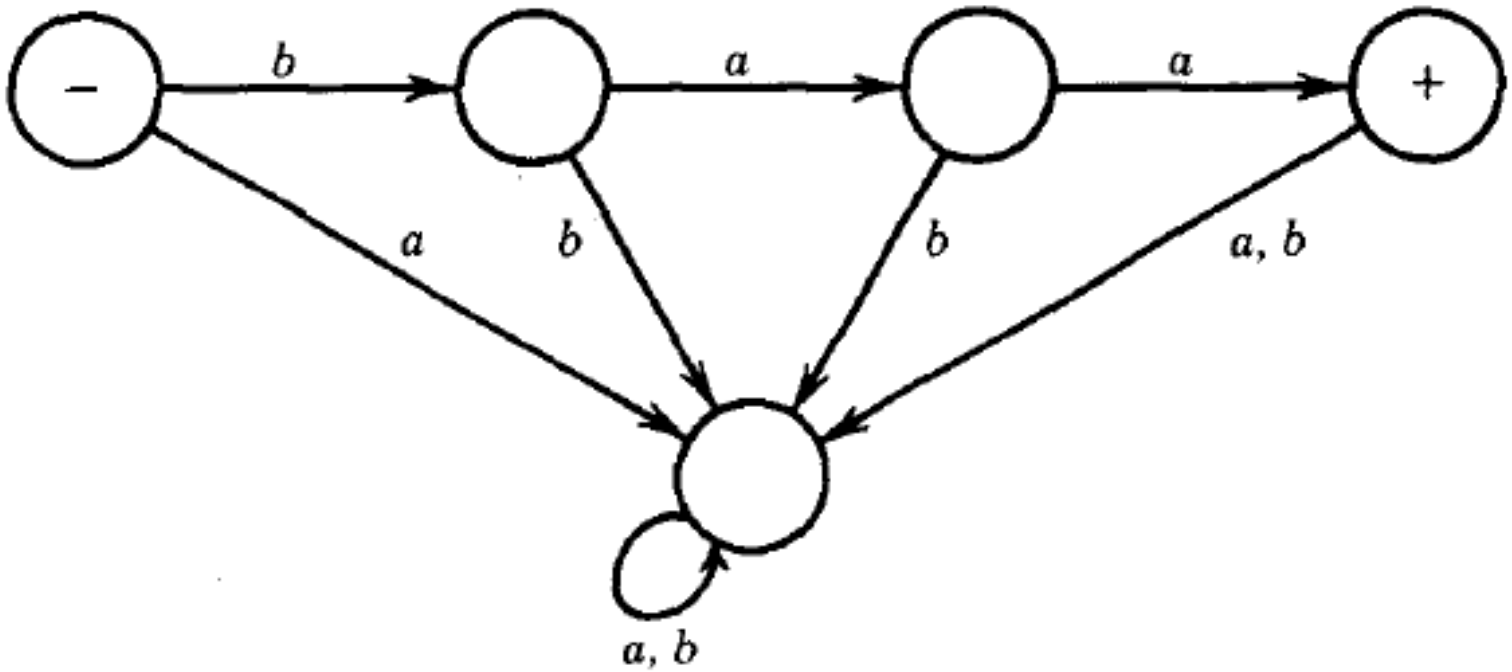
Note 4



(a+b) (a+b) = (aa+ab+ba+bb)

الدكتور المهندس محمد سامي محمد

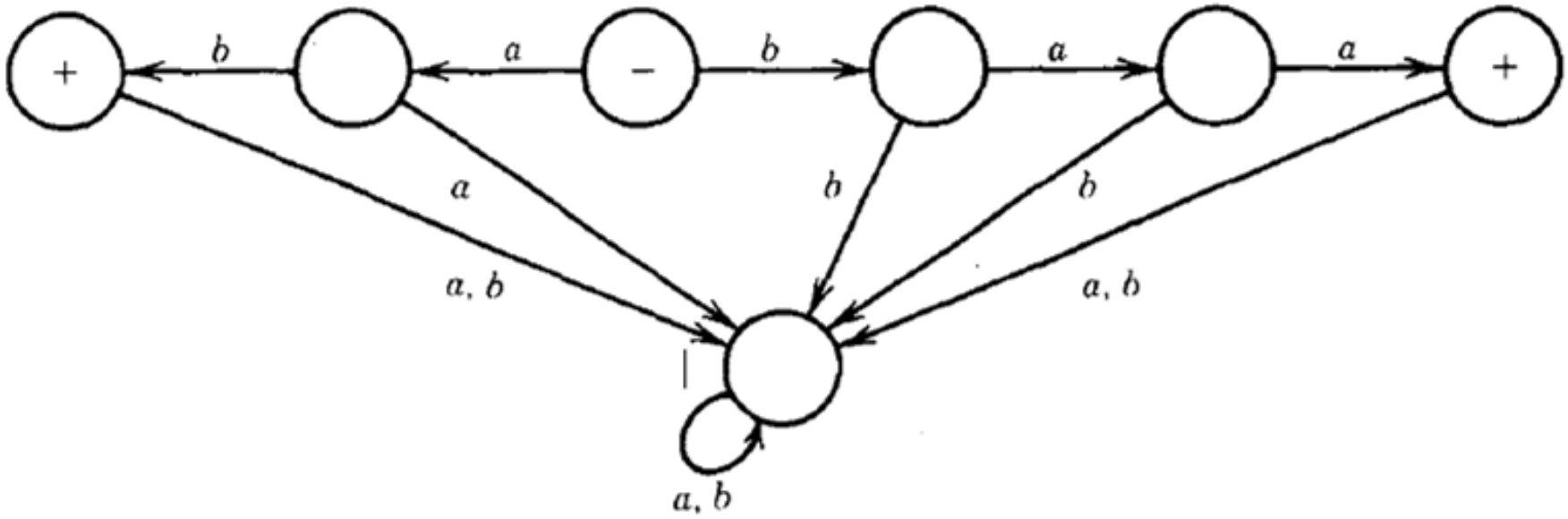
EXAMPLE: Let us consider a very specialized FA, one that accepts only the word *baa*.



$L = \{baa\}$

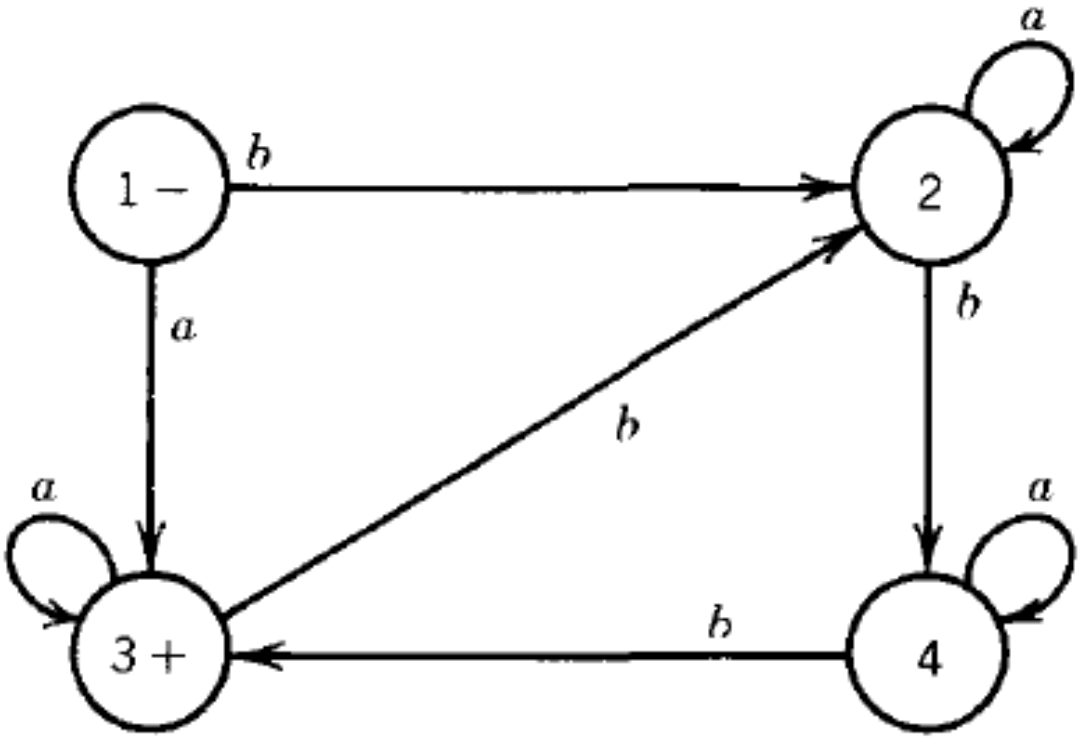
الدكتور المهندس محمد سامي محمد

EXAMPLE: The FA below accepts exactly the two strings *baa* and *ab*.



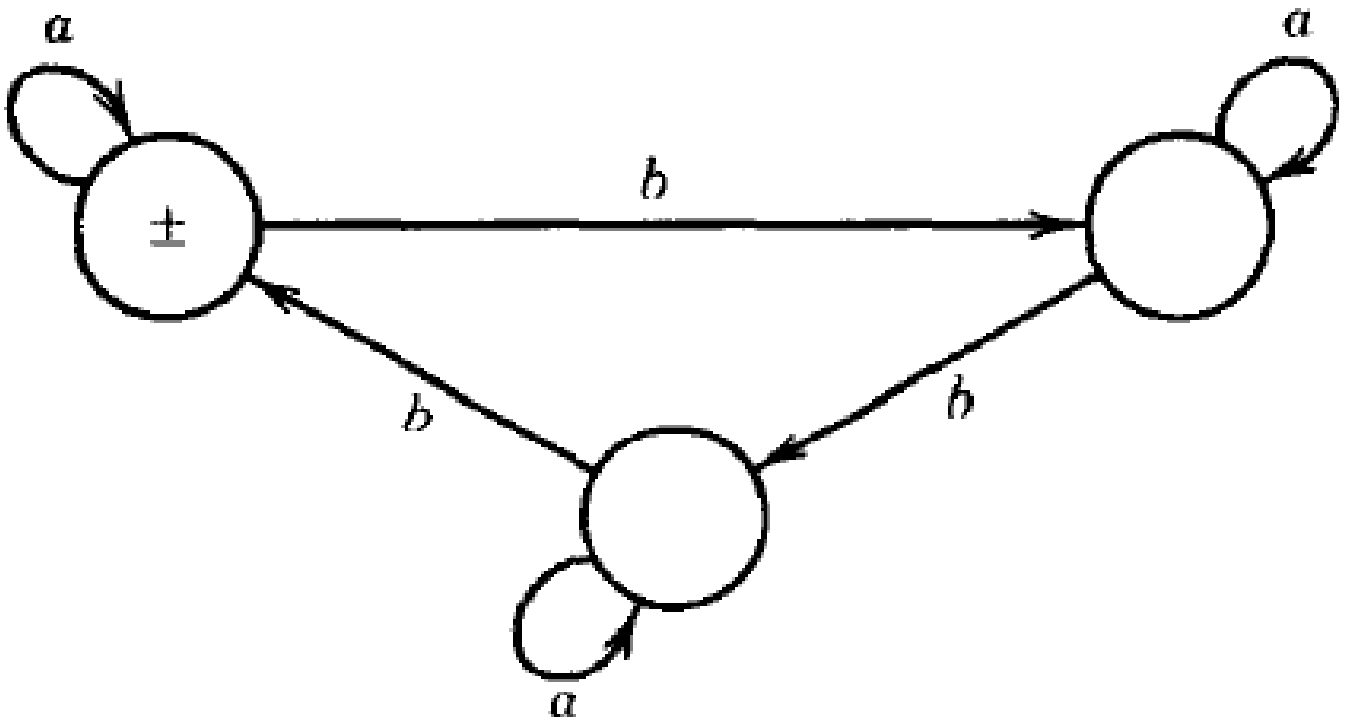
الدكتور المهندس محمد سامي محمد

EXAMPLE: Let us take a trickier example. Consider the FA shown below:



This is not the only possible Solution ⇒ **aa***

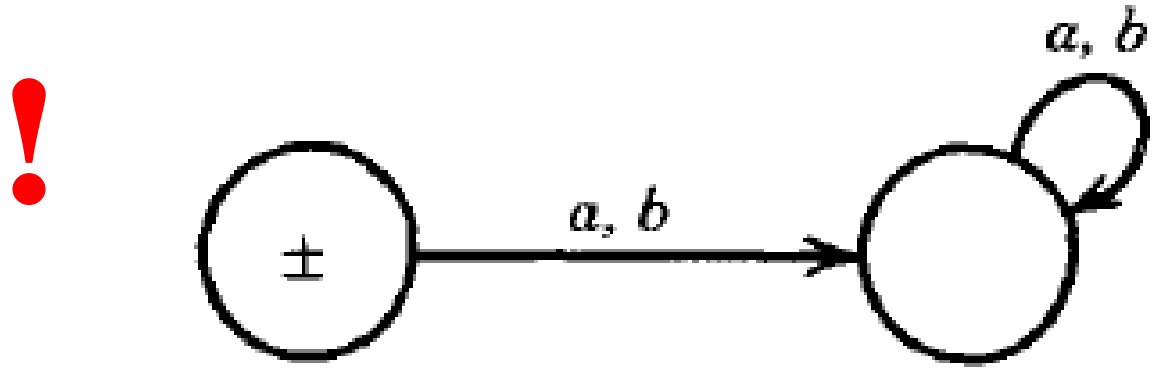
EXAMPLE



(a* + a* ba*ba*b a*)

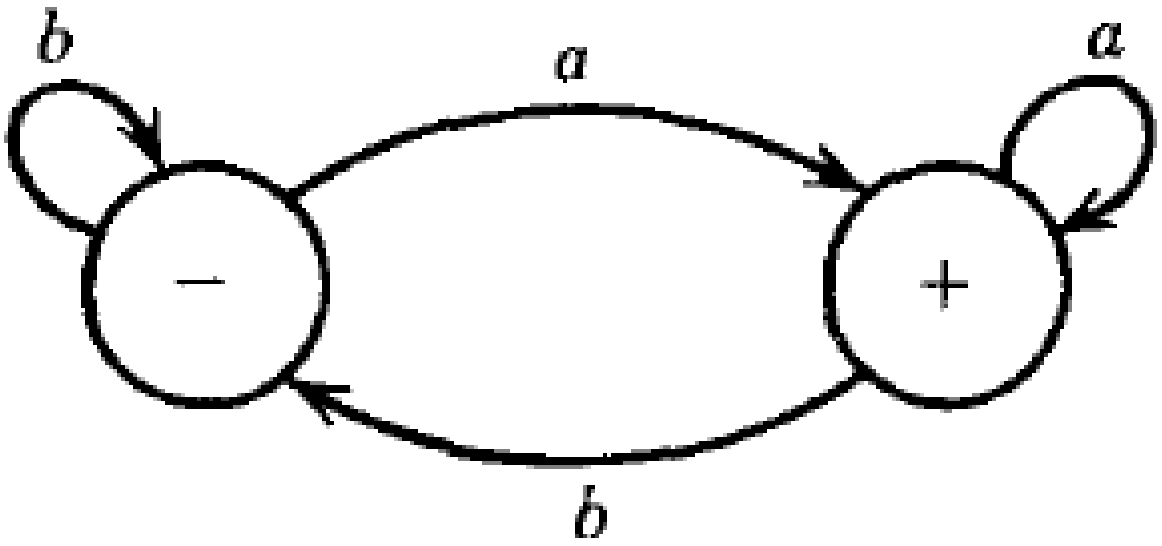
الدكتورالمهندس محمد سامي محمد

EXAMPLE: The following FA accepts only the word A



Which words can lead to final state ?

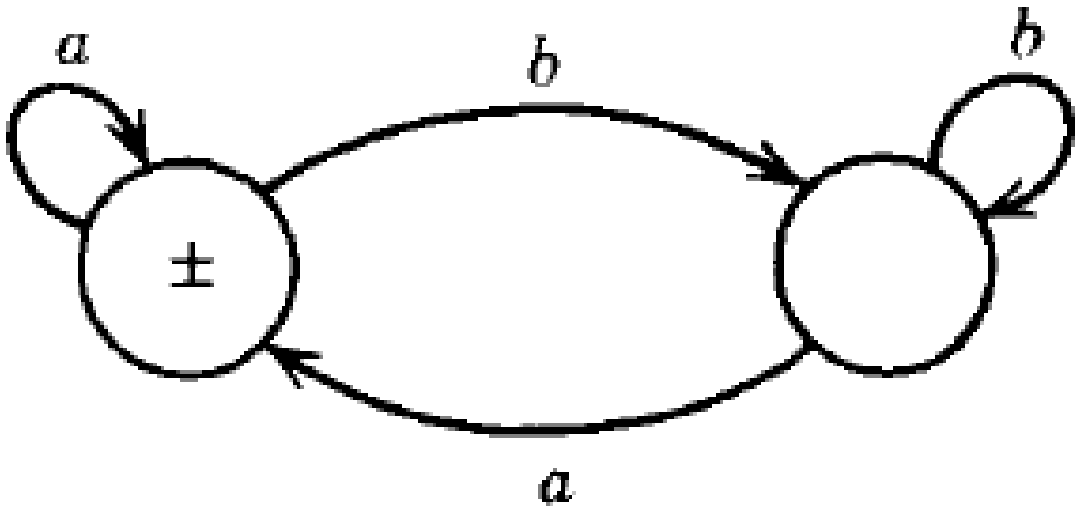
EXAMPLE: Consider the following FA:



(a + b)*a

الدكتور المهندس محمد سامي محمد

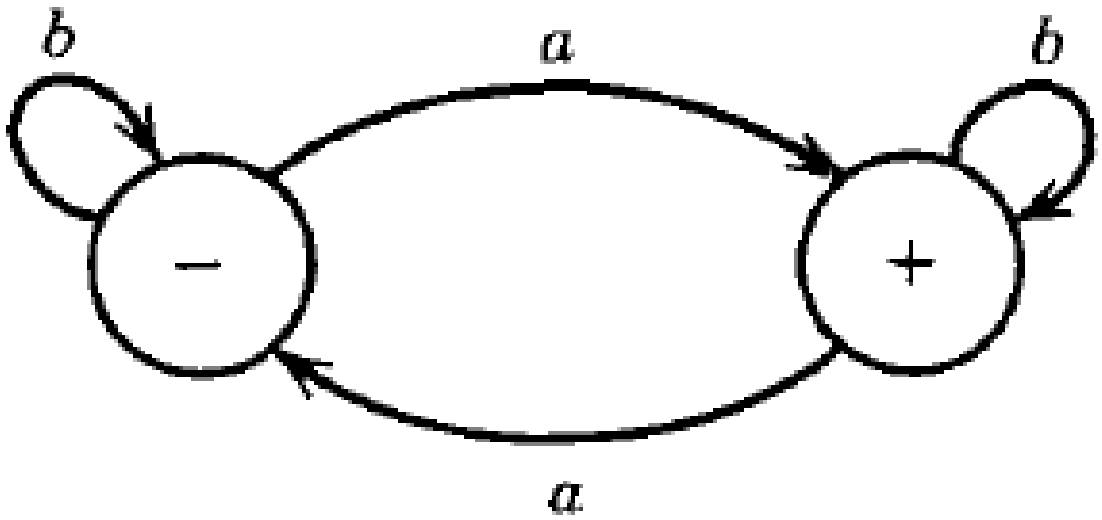
EXAMPLE: The language in the example above does not include A. If we add A we get the language of all words that do not end in b. This is accepted by the FA below.



الدكتورالمهندس محمد سامي محمد

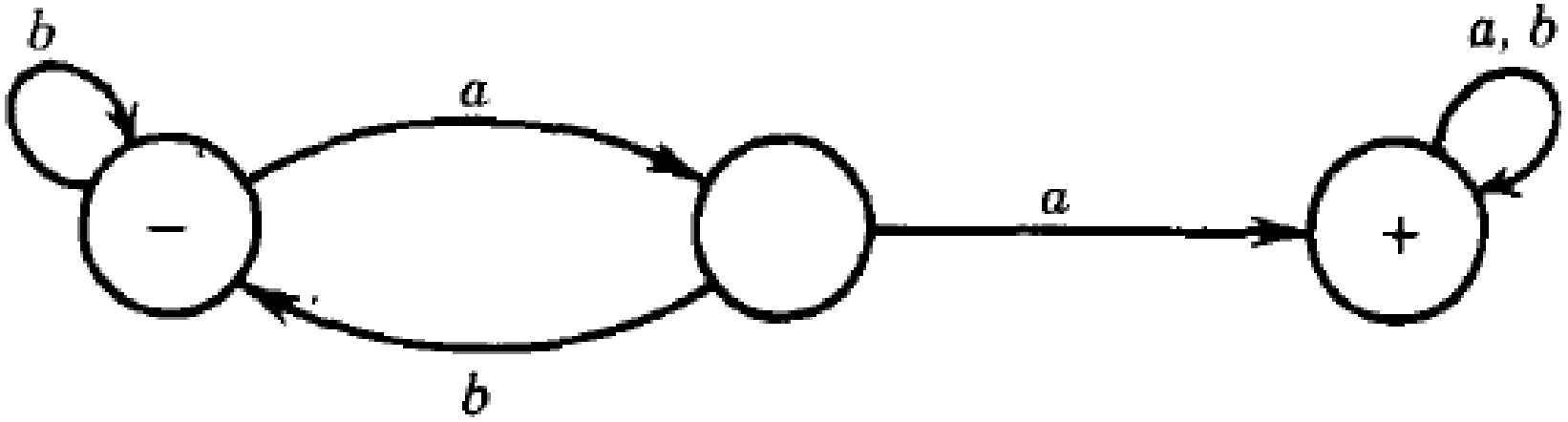
EXAMPLE: Consider the following FA:

If we read a first a, we go to +. A second a takes us back. A third a takes us to + again. We end at + after the first, third, fifth, seventh, . . . a. The language accepted by this machine is all words with an odd number of a's.



b*a(b*ab*ab*)*

EXAMPLE Consider the following FA:



(a + b)*aa(a + b)*

الدكتور المهندس محمد سامي محمد