Theory of Computation

النظرية الاحتسابية
المحاضرة الحادية عشر

كلية التربية للعلوم الصرفة / جامعة ديالى

قسم علوم الحاسوب
المرحلة الثانية

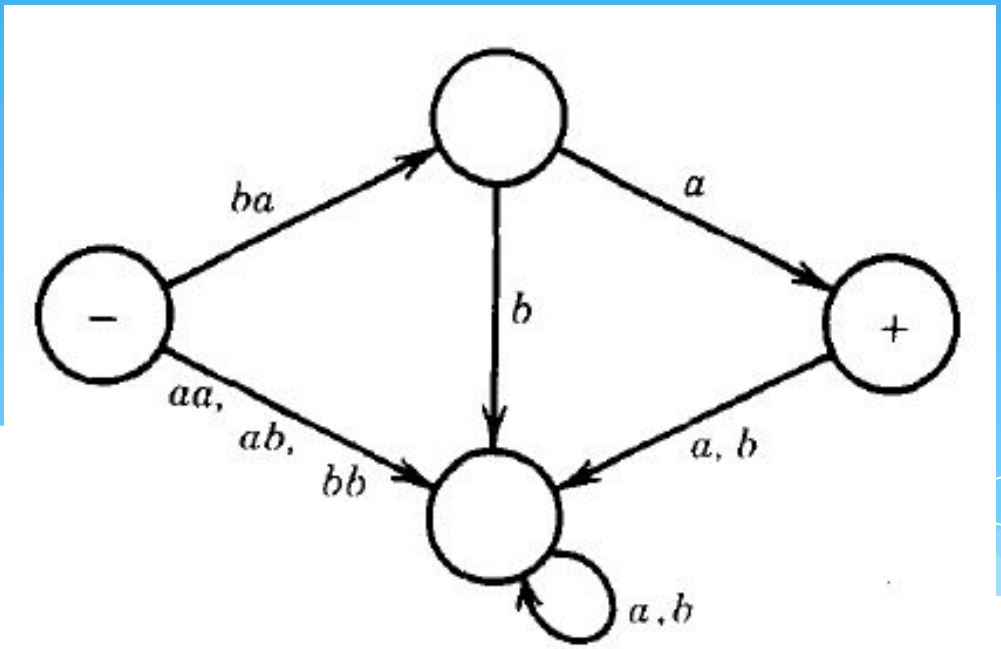
اعداد
م.د. محمد سامي محمد

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

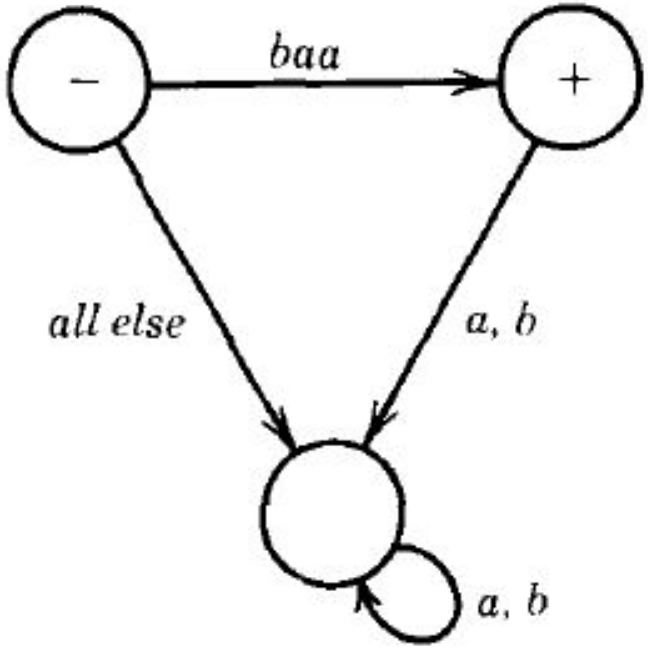
The objects we deal with in this book are mathematical models

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

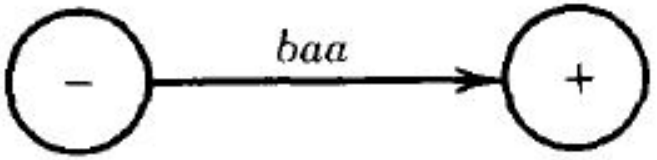
This new rule for making models will also turn out to be practical. It will make it easier for us to design machines that accept certain different languages.

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



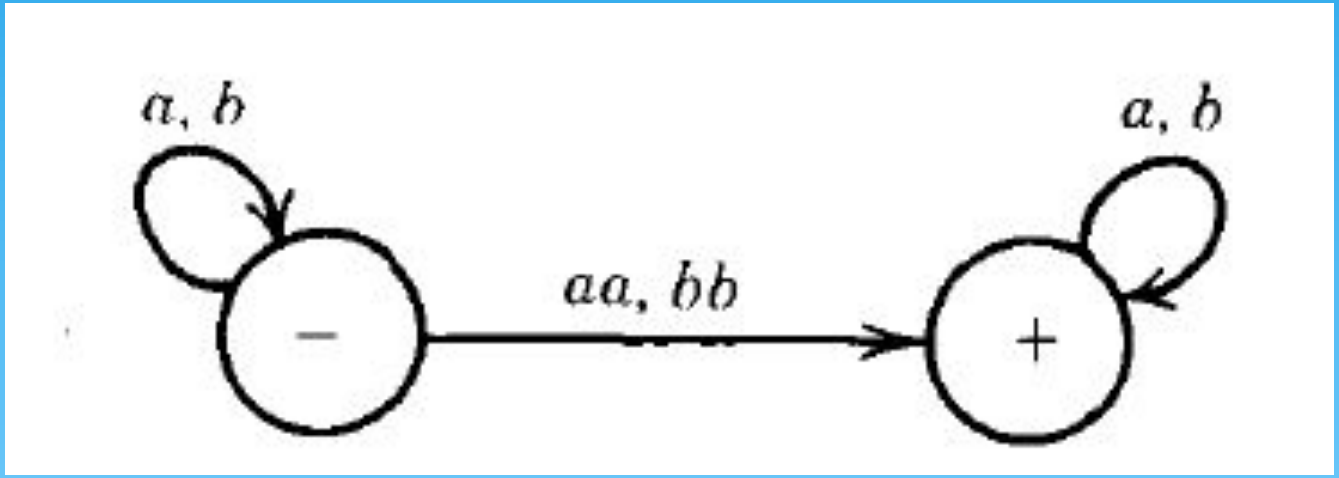
or even



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

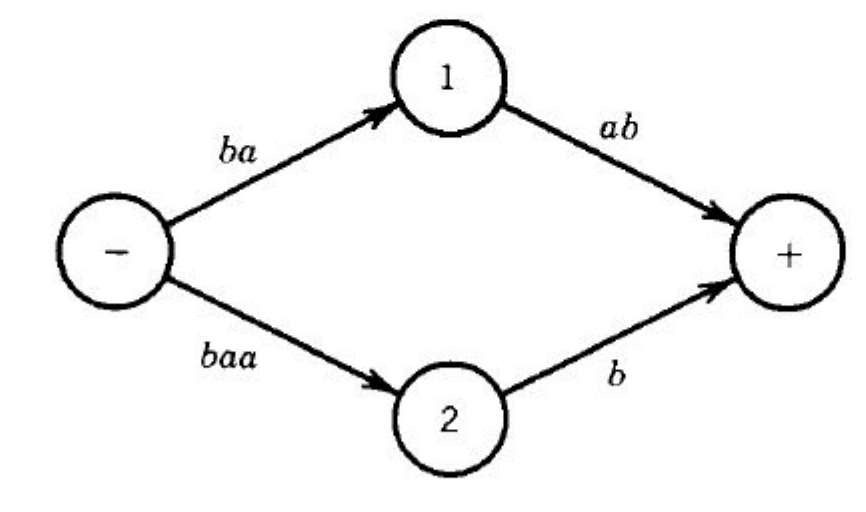
When the third letter is read in, we would still be at the starting post. We could not then accept this string.
There are two different paths that the input *baa* can take through the machine. This is totally different from the situation we had before, especially since one path leads to acceptance and 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 (+).

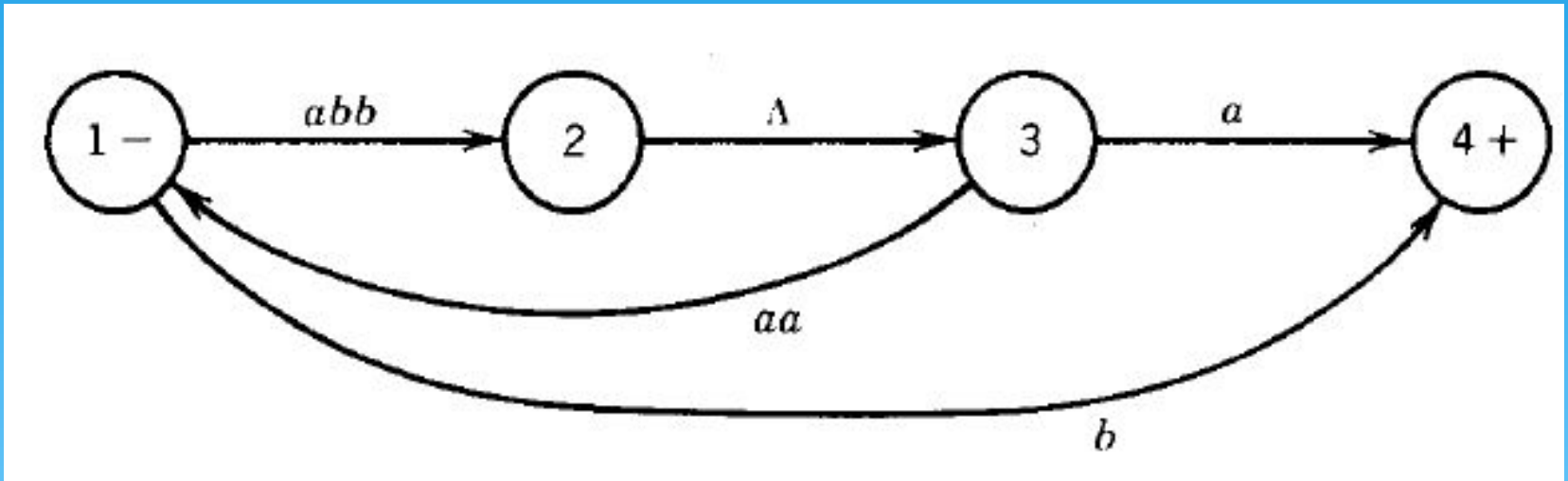
2. An alphabet I of possible input letters from which input strings are formed.

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).

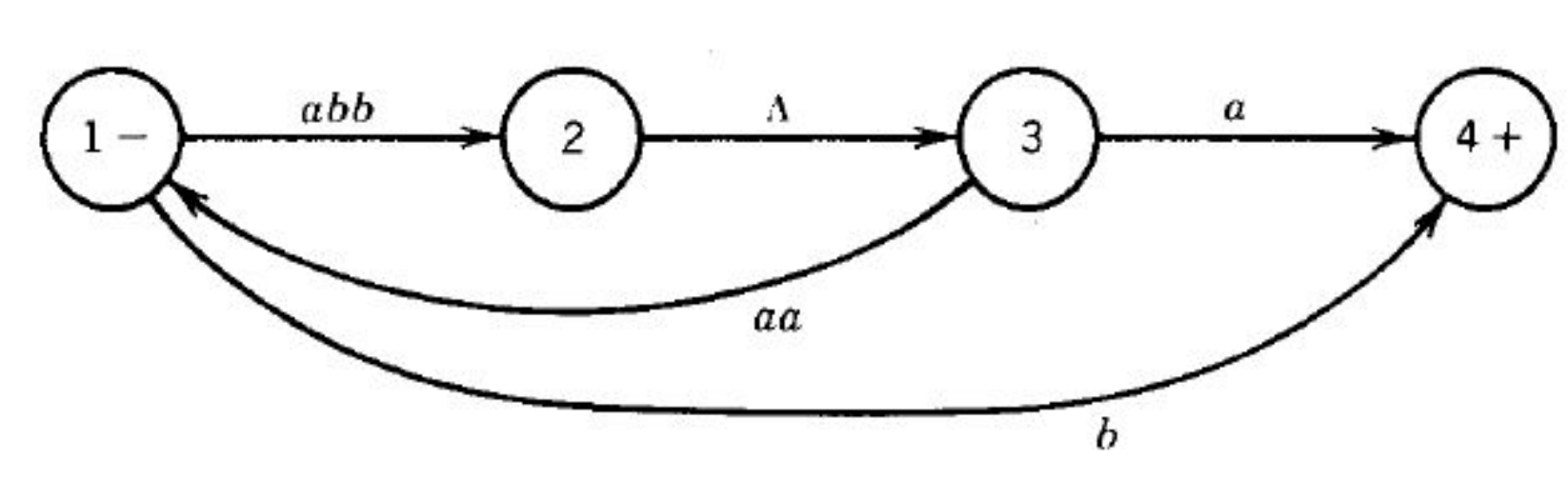
A successful path through a transition graph is a series of edges forming a path beginning at some start state (there may be several) and ending at a final state. If we concatenate in order the strings of letters that label each edge in the path, we produce a word that is accepted by this machine.

Another Example

For example, consider the following TG:



Example Continue

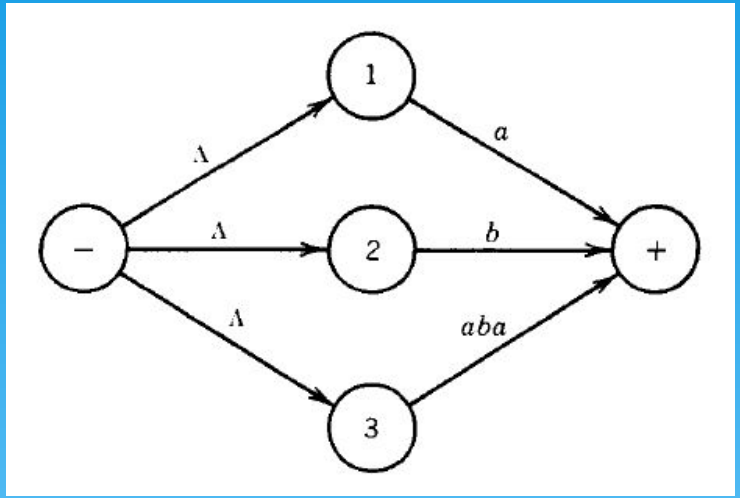


The path from state 1 to state 2 to state 3 back to state 1 then to state 4 corresponds to the string *(abb)(A)(aa)(b)*.

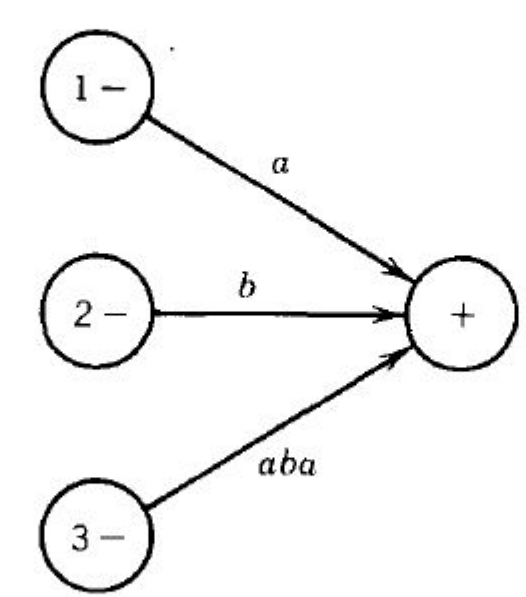
This is one way of factoring the word *abbaab*, which, we now see, is accepted by this machine. Some other words accepted are *abba*, *abbaaabba*, and b.

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**.