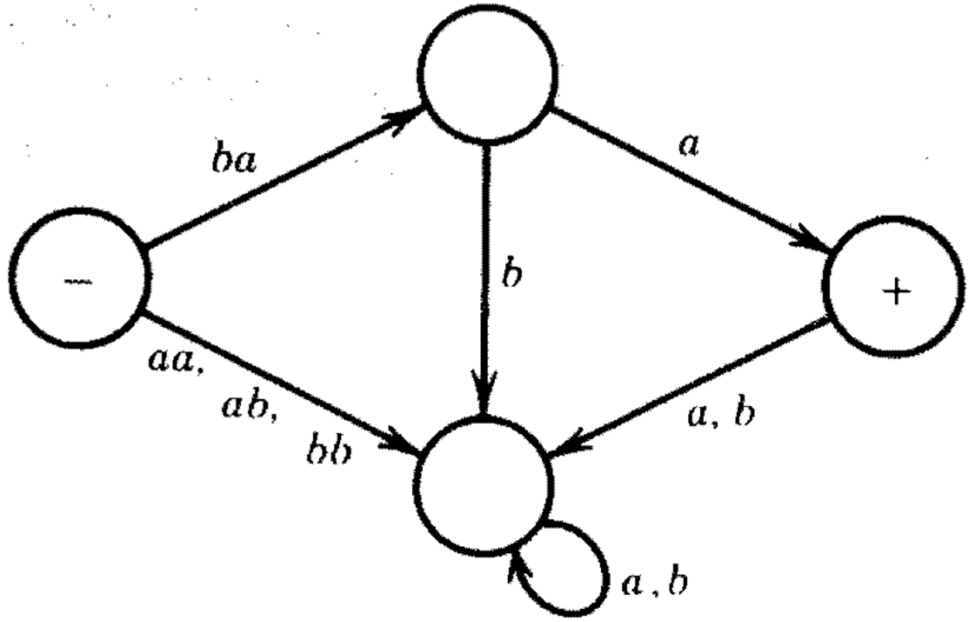
Biçimsel Diller ve Otomata Teorisi

Sunu V
Geçiş Çizgeleri

İZZET FATİH ŞENTÜRK

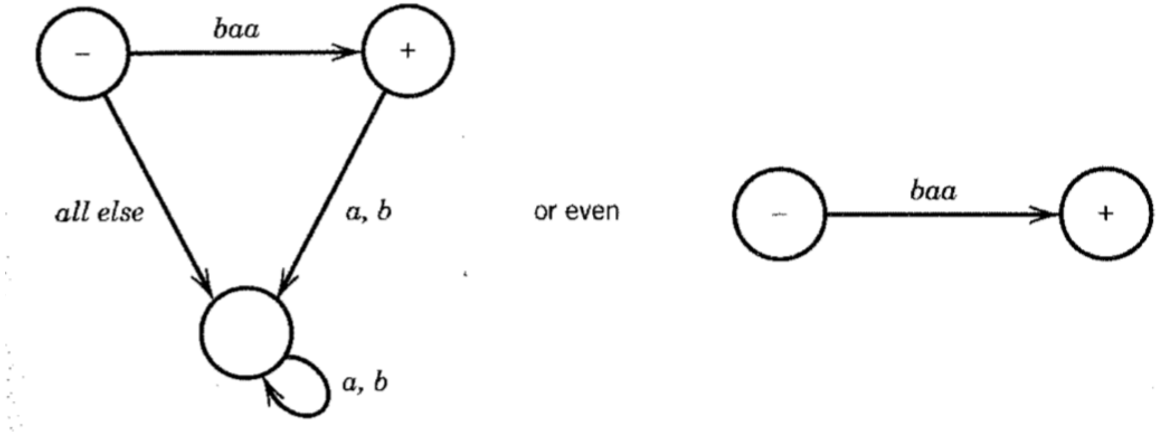
Relaxing the Restiction on Inputs

- In the last chapter, we saw an FA
  - That accepts only the word baa
  - Required five states because an FA can read <u>one letter at a time</u>
- Suppose we design a <u>more powerful machine</u> that can read either one or two letters at a time



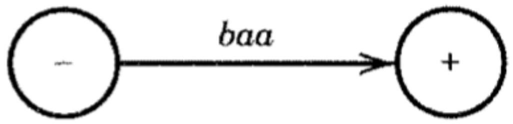
Relaxing the Restiction on Inputs

- A variation of FAs
  - We abondon the requirement that the edges eat just one letter at a time
- If we are interested in a machine that accepts only the word baa
  - Why just read two letters at a time?



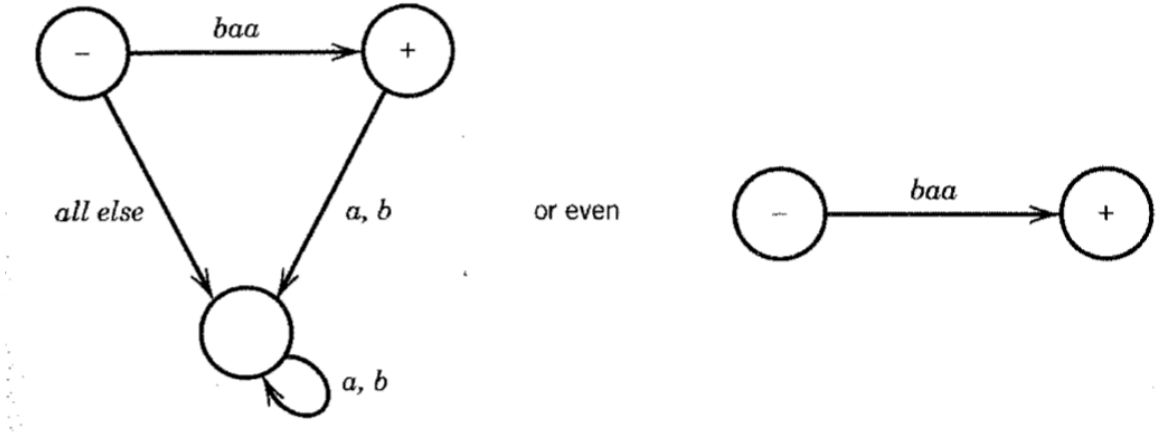
Relaxing the Restiction on Inputs

- Interprete the picture as an FA-like machine
  - baa get to the final state
  - All other input strings end up nowhere
- If we start in the minus state and the first letter of the input is an a
  - We have not direction as to what to do
- If the input is baabb
  - The first three letters take us to the accept state
  - When we read more of the input letters?
  - According to the rules of FAs, we cannot stop reading input letters until the input string completely runs out



Trash-can State

- When we fail to be able to make any of the allowable edge crossings
  - We assume a trash-can state that we must go
- We consider two pictures to be equivalent
  - They accept the exact same language



Trash-can State

- We introduce a new term to describe what happens when an input is running on a machine and gets into a state from which it cannot escape though it has not yet been fully read
- When an input string that has not been completely read reaches a state (final or otherwise) that it cannot leave because there is no outgoing edge that it may follow
  - We say that the input (or machine) **crashes** at that state
  - Execution terminates
  - The input is rejected

FAs and Crashes

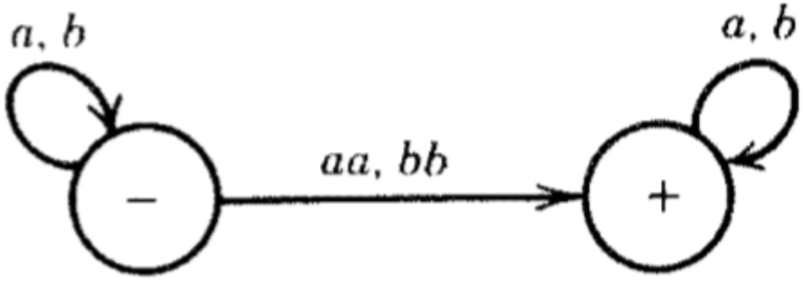
- It is not possible for any input to crash
  - Because there is always an outgoing a-edge and an outgoing b-edge from each state
  - As long as there remains letters unread, progress is possible

TGs and Rejects

- There are two different ways that an input can be rejected
  - Trace a path ending a nonfinal state
  - Crash while being processed

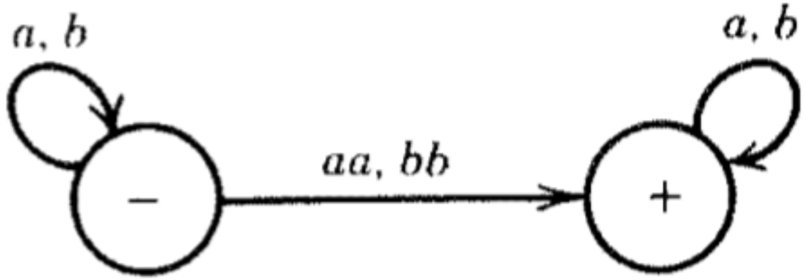
Decisions to Make

- Assume a machine that can read **one** or **two** letters at a time
  - Recognize all words that can contain a double letter



- We have changed a fundamental rule in this machine
  - We must decide **how many letters** to read from the input string **each time**

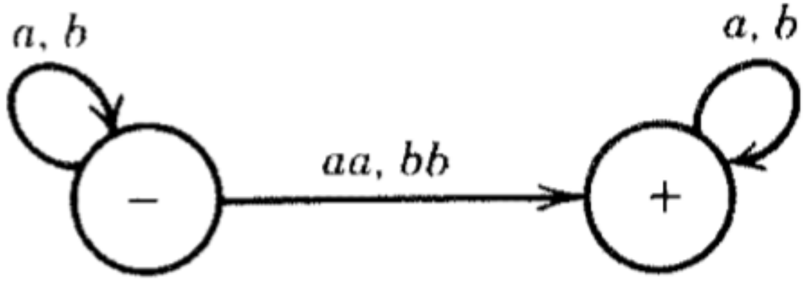
Decisions to Make



- Assume that the input string is baa
- It is easy to see how this string is **accepted**: (b)(aa)
- We **reject** the same string if read (b)(a)(a)
- What if we read (ba)(a)? The machine **crashes**

- This situation is totally different than what we had before
    - One path leads to **acceptance** and one to **rejection** for the <u>same input</u>

Decisions to Make



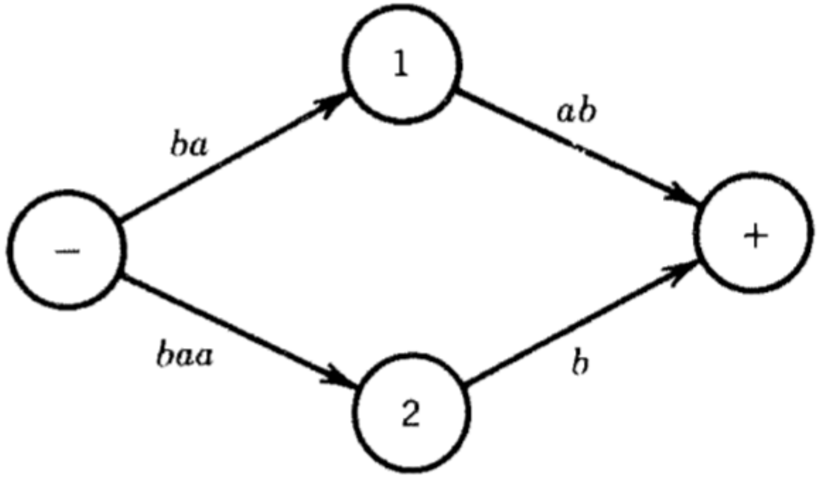
- Is this input string part of the language of this machine or not?
  - It cannot depend on the cleverness of the machine operator
  - It must be an absolute yes or no
  - Otherwise, the language is not <u>well defined</u>

Change the Definition of the Abstract Machine

- If we are going to change the definition of the abstract machine
  - To allow for more than one letter to be read at a time
  - We must also change the definition of acceptance

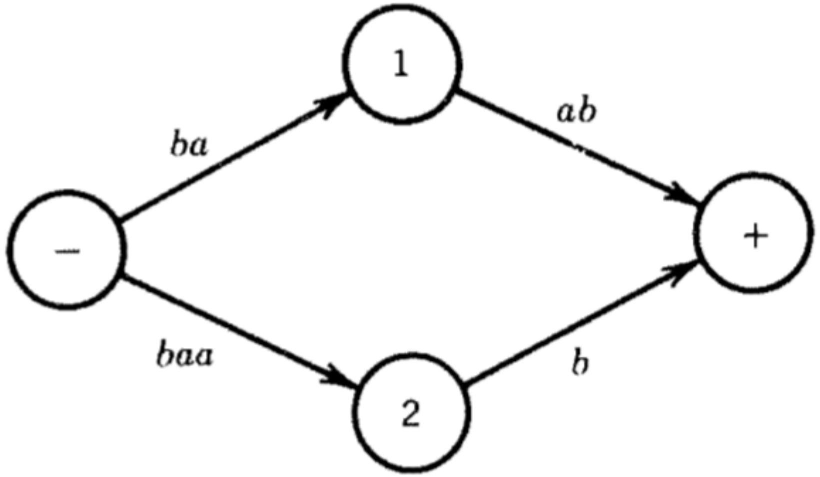
- We shall say that a string is **accepted** by a machine if there is <u>some</u> way it could be processed as as to **arrive at a final state**
  - There may also be ways in which this string does not get to a final state, but we ignore all failures

Change the Definition of the Abstract Machine



- We are about to design machines in which
  - <u>any edge</u> in the picture can be **labeled** by <u>any string of alphabet letters</u>
- We must consider some additional consequences
  - We can encounter the following problem..

Change the Definition of the Abstract Machine



- We can accept the baab in two different ways
  - (baa)(b)
  - (ba)(ab)
- In FAs, we had a unique path for every input string
  - Now, some strings have no paths at all, while some have several

Transition Graphs

- We have observed many of the difficulties when we expand the definition of "machine" to allow reading more than one letter of input at a time
- We leave the definition of the FA
- We call these <u>new</u> machines **transition graphs**

Definition of Transition Graphs

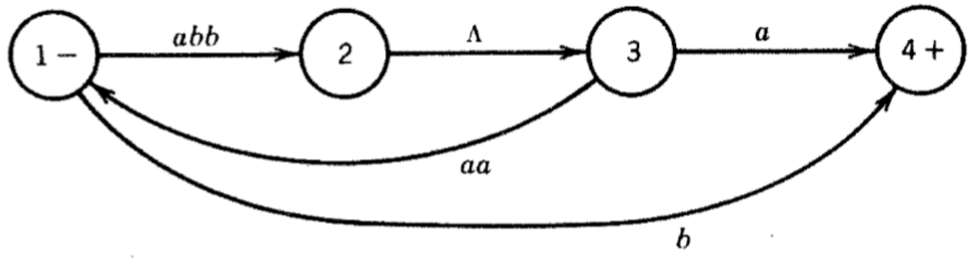
- TG is a collection of three things
  1. A finite set of states, at least one of which is designated as the start state (**-**) and some (maybe none) of which are designated as final states (**+**)
  2. An alphabet **Σ** of possible input letters from which input strings are formed
  3. A finite set of transitions (edge labels) that show how to go from some states to some others, based on reading specified substrings of input letters (possible even the null string **Λ**)

Transition Graphs

- Some states may have no edge coming out of them at all and some may have thousands (a, aa, aaa, aaaa, …)
- Transition graphs were invented in 1957
- 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

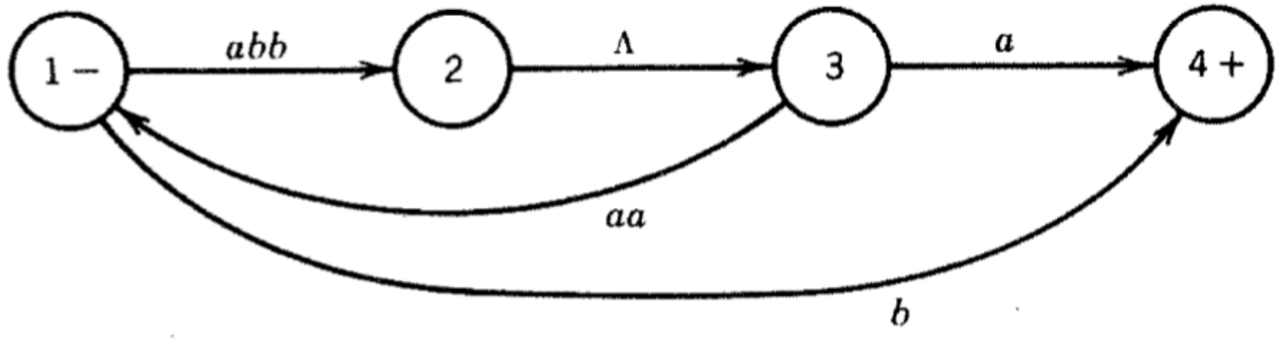
Transition Graphs

- To produce a word accepted by this machine, concatenate edge labels in the path
  - (abb)(Λ)(aa)(b): abbaab
- Λ means a free ride
  - Without consuming letters
  - We do not have to follow that edge but we can if we want to



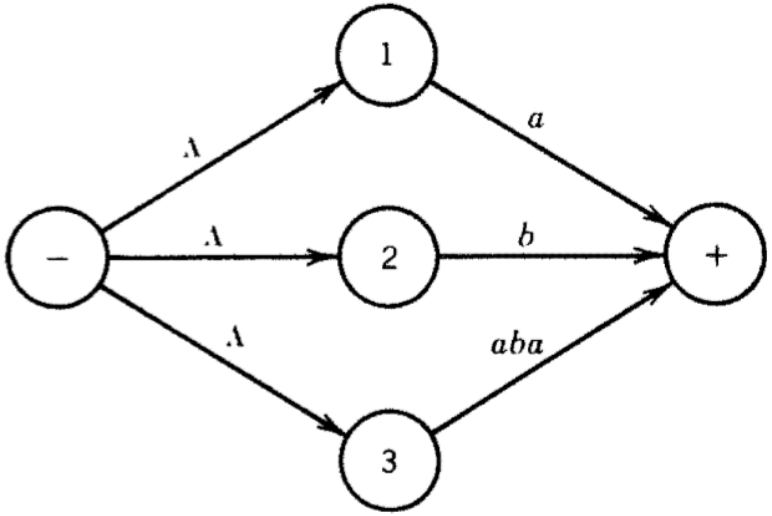
Transition Graphs

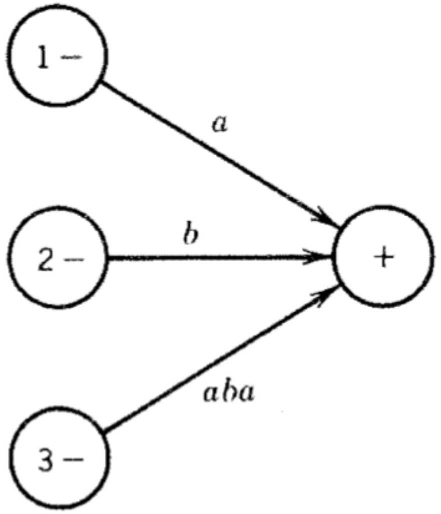
- If we are presented with a particular string of a’s and b’s to run on a given TG
  - We must decide how to break the word into substrings that might correspond to labels of edges in a path
- Consider the input string abbab on the following machine



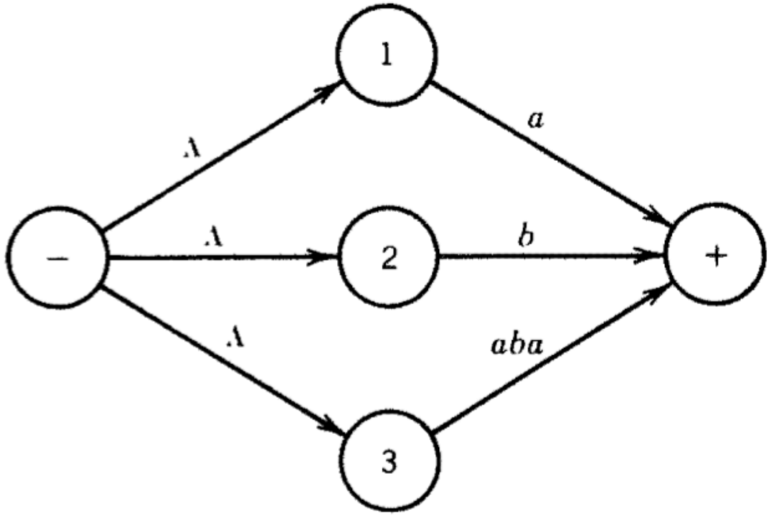
Transition Graphs

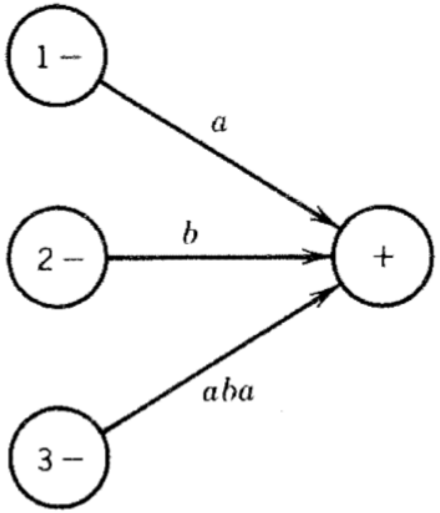
- The possibility of more than one start state?
- If we allow some edges to be traversed for free (Λ)
  - We can always introduce more start states and connect them to the original start state by edges labeled Λ





Transition Graphs





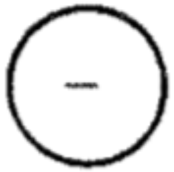
- All the strings accepted by the first are accepted by the second and vice versa
- They are equivalent despite differences such as the number of states they have

TGs and FAs

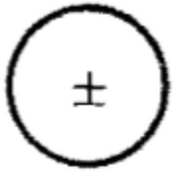
- Every FA is also a TG
- Not every TG satisfies the definition of an FA

Example

- A TG that accepts nothing, not even the null string **Λ**



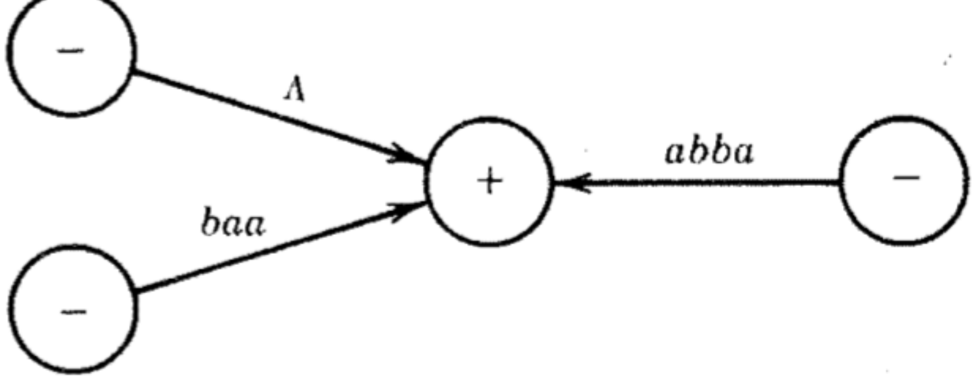
- To be able to accept anything, it must have a final state



- This machine accepts only the string **Λ**

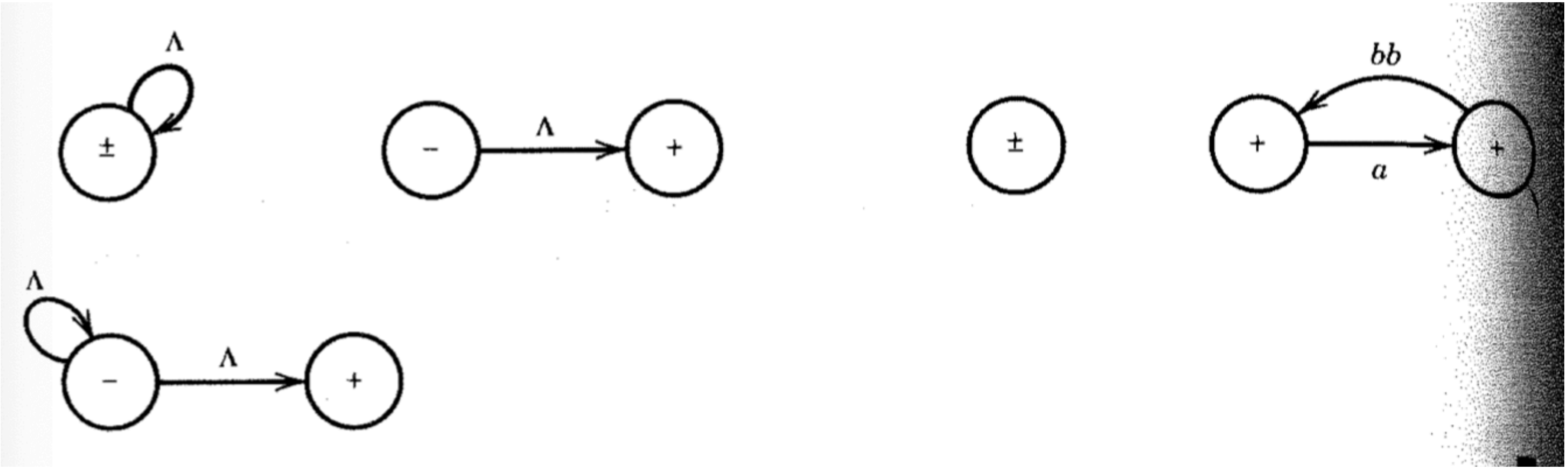
Example

- Any TG in which some start state is also a final state will always accept the string **Λ**
  - This is also true for Fas
- There are some other TGs that accept the word **Λ**
  - Also accepts baa and abba



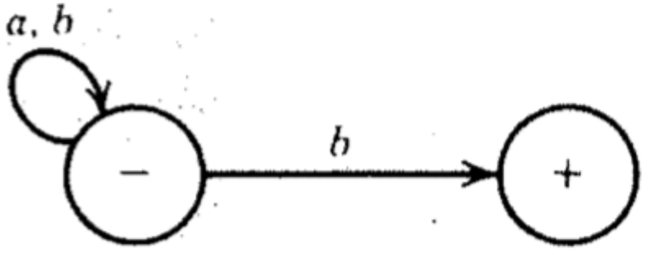
Example

- The following TGs only accept **Λ**

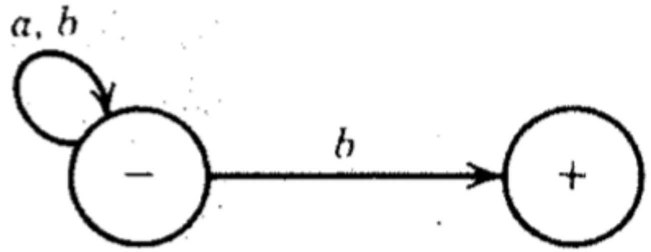


Example

- We can read all input letters one at a time and stay in the left-side state
  - When we read a *b* in the – state, there are two edges we can follow
  - If the last letter is a *b* we can use it to go to the + state
- If we try to read another letter in the right state, we crash



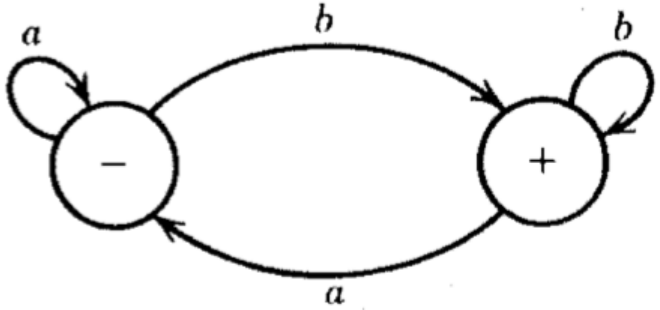
Example



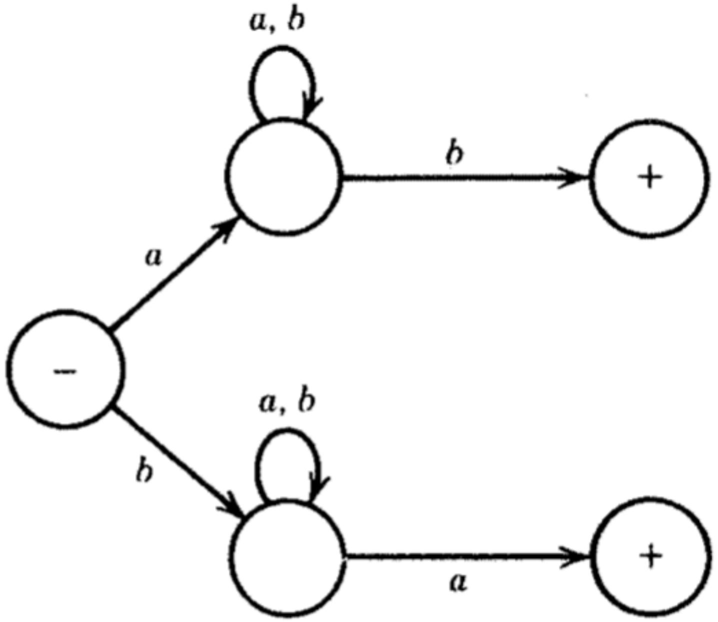
- All words that end in *b* can be accepted by some path
  - The language accepted by this TG is all words ending in *b*
- One regular expression for this language:

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

- An FA that accepts the same language:

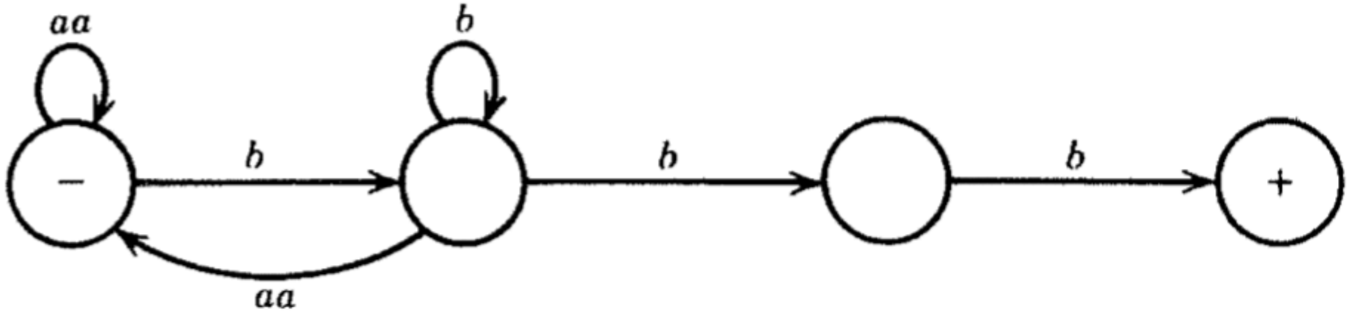


Example



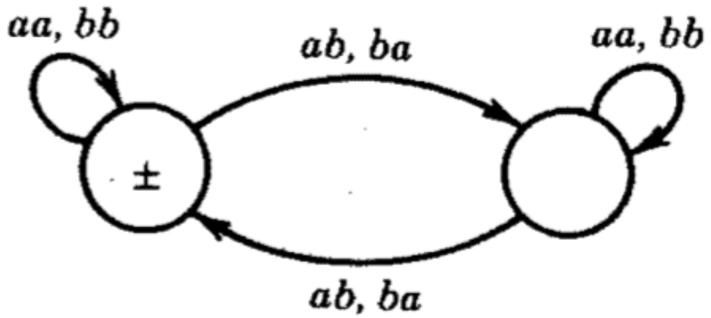
- The language of all words that begin and end with different letters

Example



- The language of all words in which the *a*'s occur only in even clumps and that end in three or more *b*'s

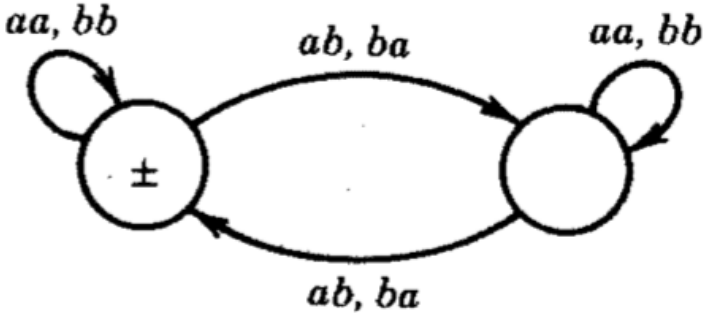
Example

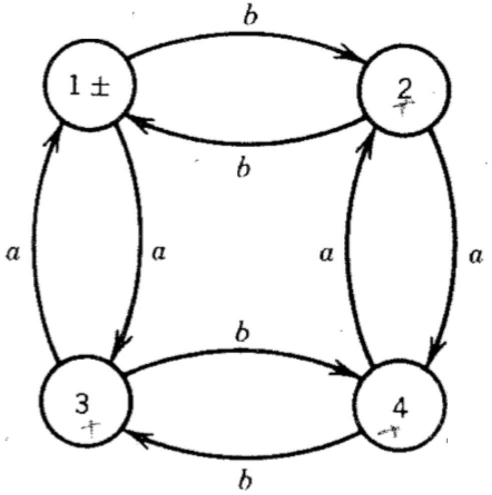


- **EVEN-EVEN:** The language of all words with an even number of *a*'s and an even number of *b*'s

Even-Even: TG vs FA vs RE

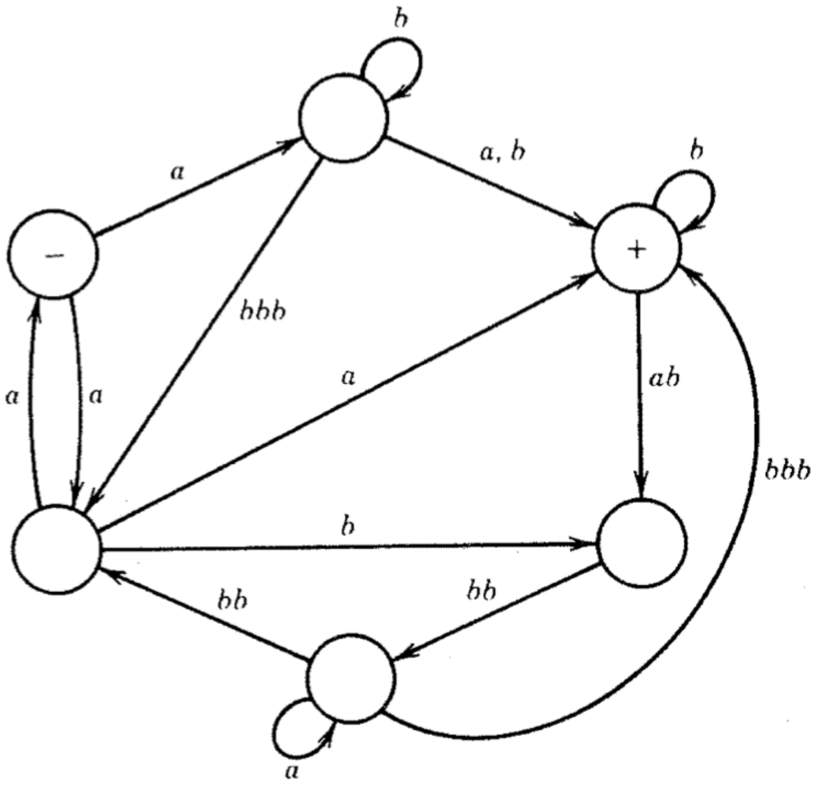
- We have reviewed three examples of definitions of this language
    - TG is the most understandable
    - A practical problem with TGs? So many possible ways of grouping the letters of the input string. More complicated to decide whether the given string is accepted or rejected
- [**aa** + **bb** + (**ab** + **ba**)(**aa** + **bb**)*(**ab** + **ba**)]*



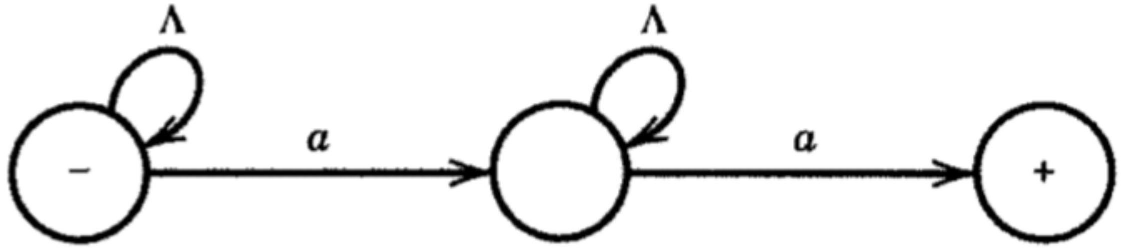


Example

- Is the word abbbabbbabba accepted by this machine?



Example



- The only word accepted by this machine is the single word *aa*
  - But it can be accepted by infinitely many different paths
- Λ-loop-edges can make life difficult
  - When we trim away Λ-loops, the graph is still a TG and accepts the same set of input strings
  - Why did we ever allow Λ-loops in the first place?

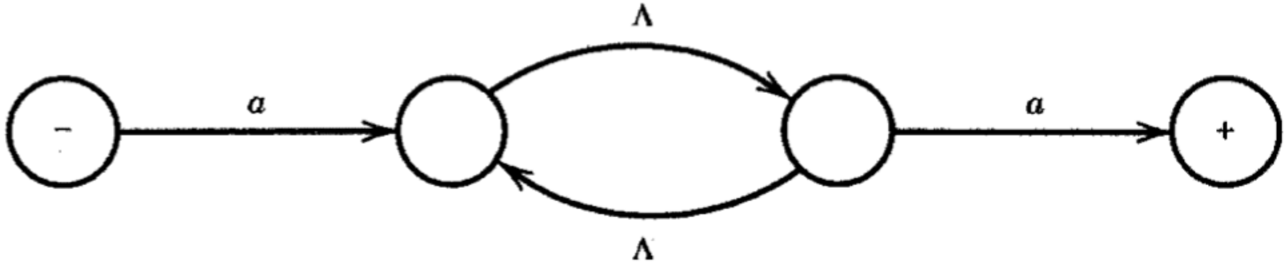
Λ Loops

- The definition is simple and universal with Λ-loops
  - Any edges, anywhere, with any labels

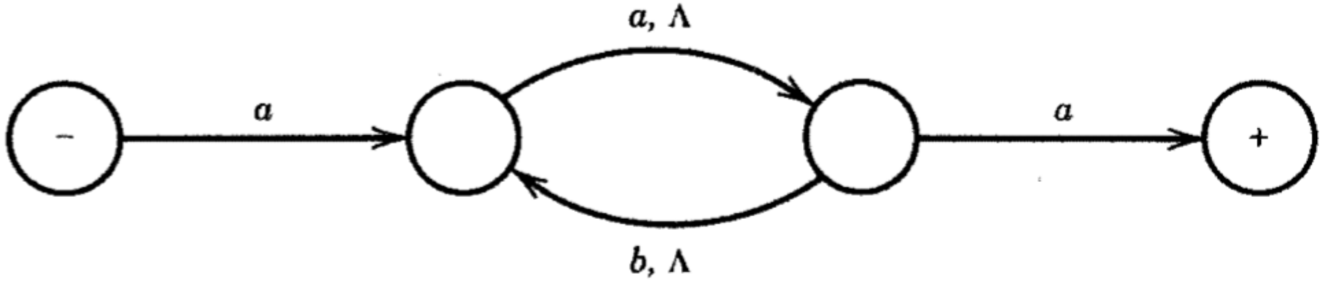
- We will see in Chapter 7 that **Λ-edges** are <u>never</u> necessary at all
  - Any language that can be accepted by a TG with Λ-edges can be accepted by some different TG without Λ-edges

Λ Loops

- It is obvious how to eliminate the Λ-loop



- But in this machine, if any Λ option is erased, the resultant language is changed

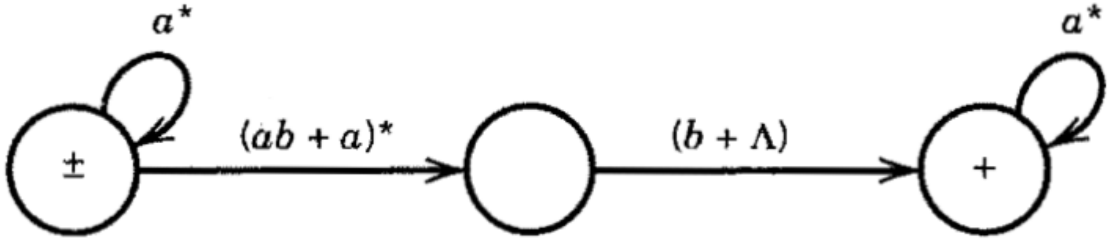


Generalized Transition Graph (GTG)

- GTG is a collection of three things
  1. A finite set of states of which at least one is a start state and some (maybe none) are final states
  2. An alphabet Σ of input letters
  3. Directed edges connecting some pairs of states each labeled with a regular expression

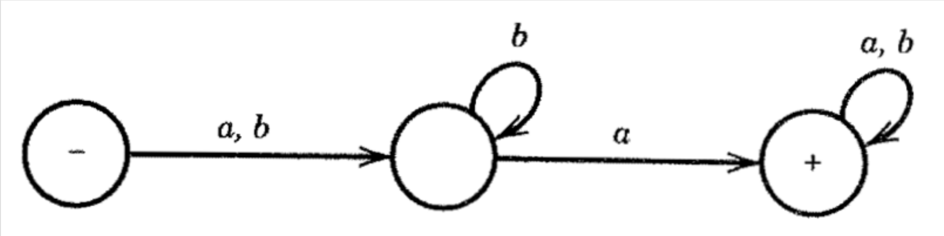
Example

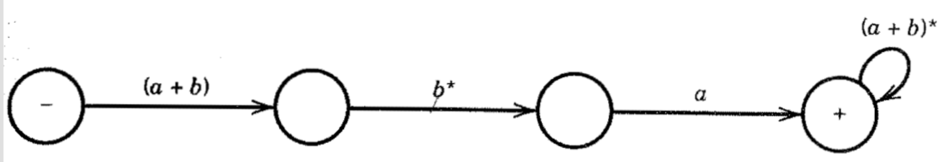
- The machine that accepts all strings <u>without</u> a **double *b***



Kleene Star vs Loop

Choose **Λ** from **b*** for no loop in the middle





Nondeterminism

- The ultimate path through the machine is not determined by the input alone.
- Human choice becomes a factor in selecting the path
- This machine is **nondeterministic**
- The machine does not make all its own determinations