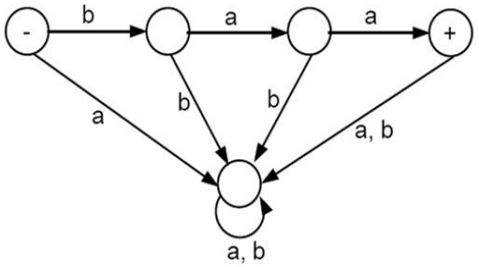
Theory of Computational

Transition Graphs(TG):

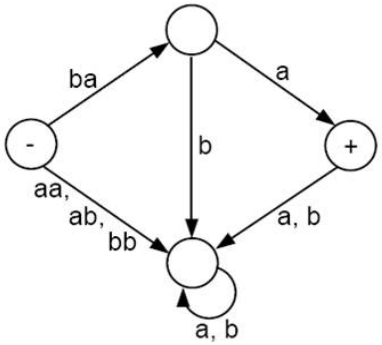
Relaxing the Restriction on Inputs
Looking at TGs
Generalized Transition Graphs
Nondeterminism

Relaxing the Restriction on Inputs:

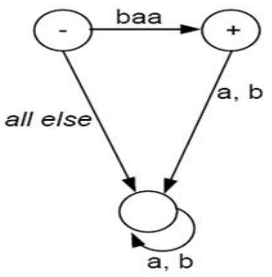
- Let's consider a very specialized FA that accepts only the word baa:

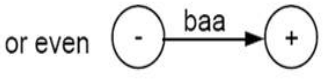


- Beginning at the start state, anything but the string *baa* will drop down into the garbage collecting state and never be seen again. Even the string *baabb* will fail.
- Hence the language accepted by this FA is L = {baa}
- Let us relax the restriction that *an FA can only read one letter from the input string at a time*
- Suppose we now allow a machine to read *either one or two letters* of the input string at a time. Then we may design a machine that accepts only the word *baa* with fewer states as the one below:



- If we go further to allow a machine to read *up to three letters* of the input string at a time, then we may design the machine accepting only the word baa with even fewer states as follows:



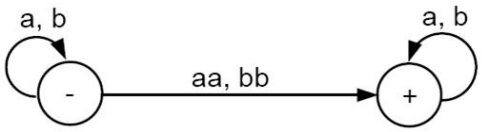


- The picture on the left tells us that when the input fails to be of the desired form, we must go to the garbage collection state and read through the rest of the input, knowing that we can never leave there.
- The picture on the right introduces some problems:
  – If we begin in the start (-) state and the first letter of the input is an a, we have no direction as to what to do.
  – We also have problem even with the input baaa. The first three letters take us to the accept state, but then the picture does not tell us where to go when the last a is read. (Remember that according to the rules of FAs, one cannot stop reading input letters until the input string completely runs out.)
- We may assume, as a convention, that there is some garbage collection state associated with the picture on the right, but we do not draw it; and that we must go and stay there when the input string fails to be of the desired form.
- With this assumption, we can consider the two pictures above to be equivalent, in the sense that they accept the exact same language.
- Rather than trying to imagine a garbage collection state as described above, it is more standard to 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, even though it has not yet been fully read.

Definition: 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 the machine) crashes at that state. Execution then terminates and the input must be rejected.

Note that on an FA it is not possible for any input to crash because there are always an outgoing a-edge and an outgoing b-edge from each state. As long as there remains input letters unread, progress is possible.

- There are now two different ways that an input can be rejected: (i) It could peacefully trace a path ending in a non-final state, or (ii) it could crash while being processed.
- If we hypothesize that a machine can read *one or two* letters at a time, then one may build a machine using only two states that can recognize all words containing a double letter (aa or bb) as follows:



- We must now realize that we have changed something more fundamental than just the way the edges are labeled or the number of letters read at a time: This

machine makes us **exercise some choice**, i.e., we must **decide how many letters to read** from the input string each time.

- As an example for the problems of making choices, let us say that the input is baa.
- If we first read b and then read aa we will go to the final state. Hence, the string is accepted.
- If we first read b, then read a, and then read a, we will loop back and be stuck at the start state. Hence, the string is rejected in this case.
- If we first read two letters ba at once, then there is no edge to tell us where to go. So, the machine crashes and the input string is rejected.
- What shall we say? Is this input string a word in the language of this machine or not?
- The above problems tell us that if we change the definition of our 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 if there is some way it could be processed so as to arrive at a final state.**
- Due to many difficulties inherent in expanding our definition of machine to reading more than one letter of input at a time, we shall leave the definition of finite automaton alone and call these new machines **transition graphs**.
- Transition graphs were invented by John Myhill in 1957 for reasons that will be revealed in the next chapter.

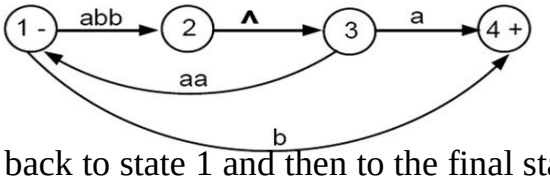
Definition of A Transition Graph:

- 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 $\sum$ 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 (possibly even the null string **Λ**).
- We should note the following from the definition of a TG:
  1. Clause 3 in the definition means that every edge is labeled by **some string** or **strings of letters**, not necessarily only one letter.
  2. We are NOT requiring that there be any specific number of edges emanating from any state: Some states may have no edge coming out at all, and some may have thousands (e.g., edges labeled a, aa, aaa, aaaa, ...).

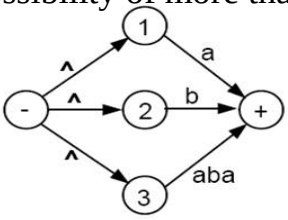
3. A **successful path** through a TG is a series of edges forming a path beginning at some start state (there may be several) and ending at a final state.
4. If we concatenate in order the strings of letters that label each edge in a successful path, we produce a word that is accepted by this TG.

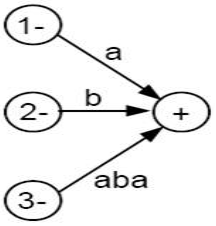
Example:

- For example, consider the following TG:



- The path from state 1 to state 2 to state 3 back to state 1 and then to the final state 4 corresponds to the string (abb)(Λ)(aa)(b) = abbaab.
- Some other accepted words are abba, abbaaabba, and b.
- When an edge is labeled with **Λ**, it means that we can take the ride it offers for free (without consuming any letter from the input string).
- If we are presented with a particular string to run on a given TG, we must decide how to break the string into substrings that may correspond to the labels of edges in the TG.
- Let's run the input string abbab on the machine in previous:
  – The substring abb takes us from state 1 to state 2.
  – We move to state 3 along the **Λ**-edge without any substring being consumed.
  – We are now in state 3 and what is left of the input string is the substring ab. We cannot read aa, so we must read only a and go to state 4.
  – At state 4, we have b left in the input string but no edge to follow, so we must crash and reject the input string abbab.
- Because we allow some edges to be traversed for free, it is logical to allow for the possibility of more than one start state, as illustrated below:





- These two machines are equivalent, in the sense that all the strings accepted by the first are accepted by the second and vise versa.
- Important Note: **Every FA is also a TG. However, NOT every TG satisfies the definition of an FA.**