Biçimsel Diller ve Otomata Teorisi

Sunu XIII
Pushdown Automata

İZZET FATİH ŞENTÜRK

New Format for FAs

- The class of languages generated by CFGs is larger than the class of languages defined by REs
  - All regular languages can be generated by CFGs, and so can some nonregular languages (a^nb^n, PALINDROME)
- We are looking for a mathematical model of some class of machines that correspond analogously to CFGs
  - There should be at least one machine that accepts each CFL
  - The language accepted by each machine is context-free
- We want CFL-recognizers (acceptors) just as FAs are regular language-recognizers

New Format for FAs

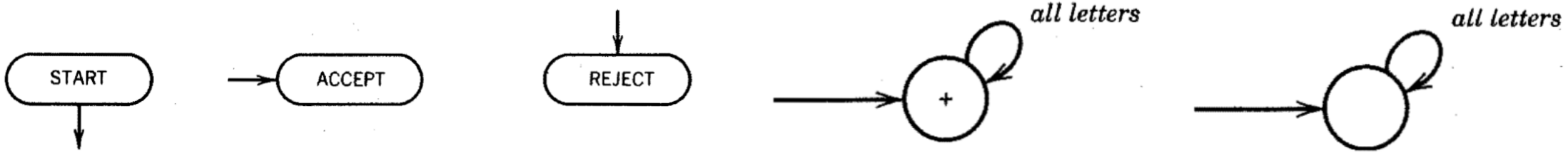
- We have, so far, not given a name to the part of the FA where the input string lives while it is being run
  - Let us call this the Input Tape
  - Input tape is long enough for any possible input
  - Any word (**a***) is a possible input, the tape must be infinitely long
  - The tape has a first location for the first letter, and so on. The tape is infinite in one direction only
  - The locations into which we put the input letters are called cells
  - Δ indicates a blank in a tape cell

New Format for FAs

- As we process this tape on the machine, we read one letter at a time and eliminate each as it is used
- We stop when we reach the first blank cell (we assume that once the first blank is encountered, the rest of the tape is also blank)
- We read from left to right and never go back to a cell that was read before

New Format for FAs

- As part of the new representations, let us introduce the symbols
- The arrows into or out of these states can be drawn at any angle
- The start state is like a – state connected to another state in a TG by a Λ edge
  - We begin the process there, but we read no inout letter
  - We just proceed immediately to the next state
  - A start state has no arrows coming into it
- An accept state is a dead-end final state – once entered, it cannot be left. A reject state is a dead-end state that is not final

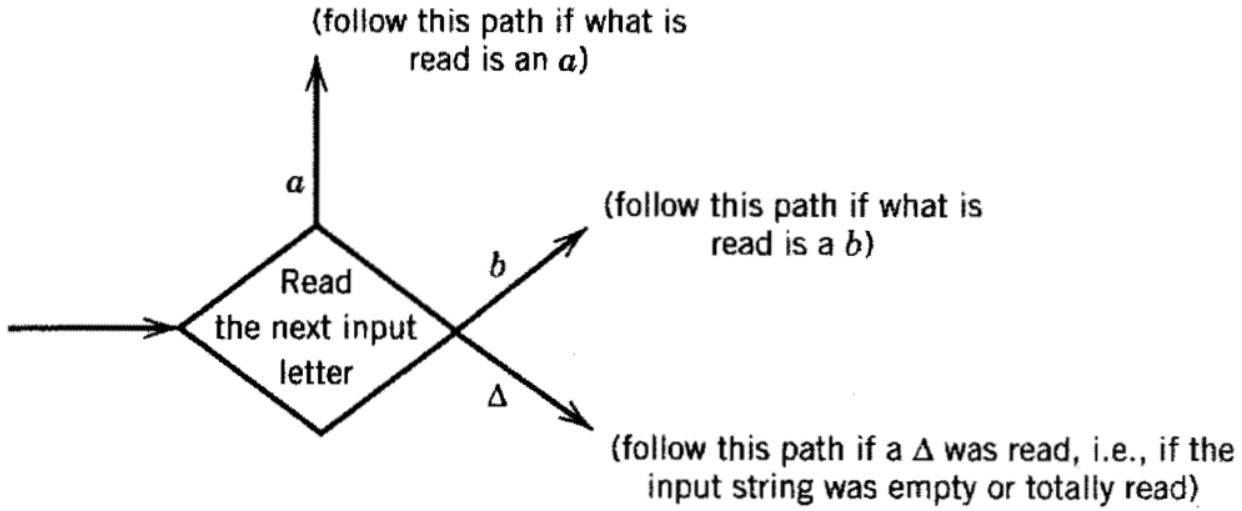


New Format for FAs

- Bacause we have used “final” to apply only to accepting states in FAs, we call the accept and reject states “halt states”
- Previoulsy, we could pass through a final state if we were not finished reading the input data. Halt states cannot be traversed
- Every function a state performs is done by a separate box in the picture
  - The most important job performed by a state in an FA is to read an input letter and branch to other states depending on what letter has been read

New Format for FAs

- We introduce read states
  - Depicted as diamond-shaped boxes
  - The edges show only one of the many possibilities
  - When the Δ is read from the tape, it means we are out of input letters

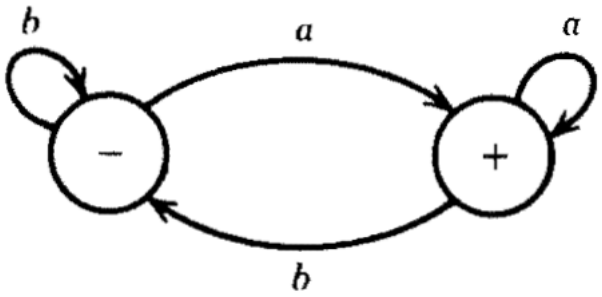


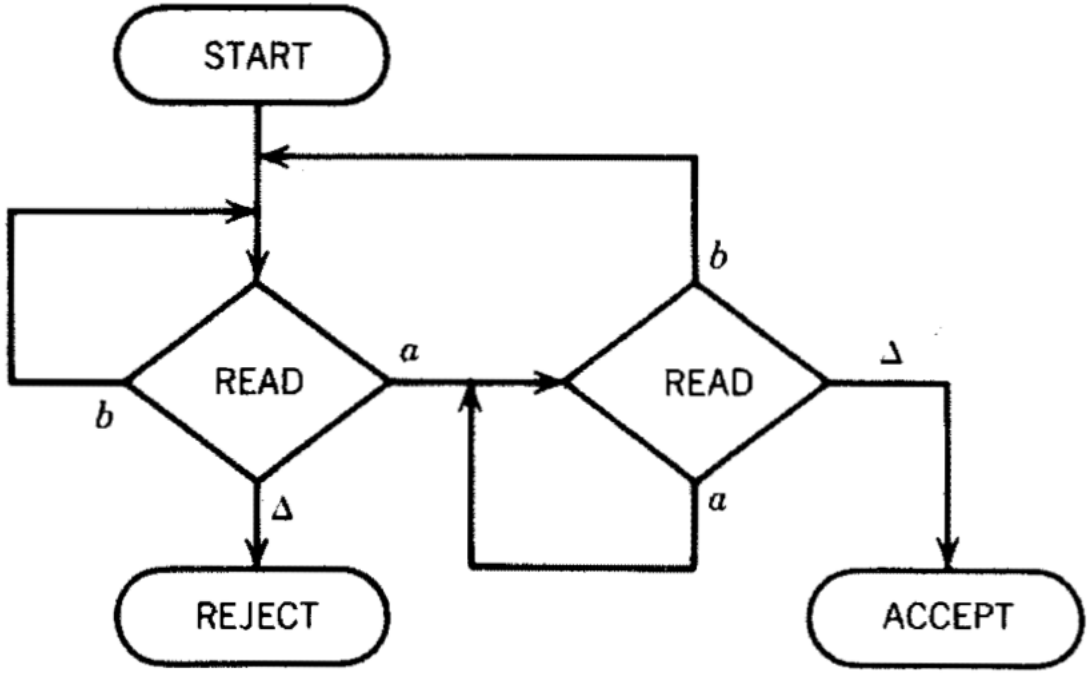
New Format for FAs

- The Δ-edge leads to accept if the state we have stopped in is a final state and reject if the processing stops in a state that is not a final state
- Before, we never explained how we knew we were out of input letters. In these new pictures, we can recognize this by reading a blank from the tape
- These suggestions have not altered the power (language accepting abilities) of our machines

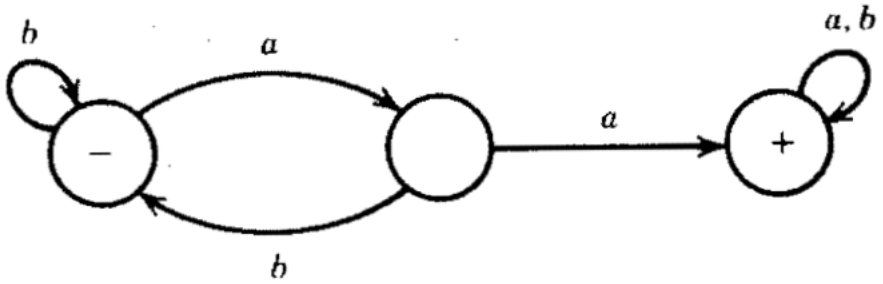
New Format for FAs

- The FA that accepts all words ending in the letter a becomes this new machine
- Notice that the edge from start needs no label because start reads no letter

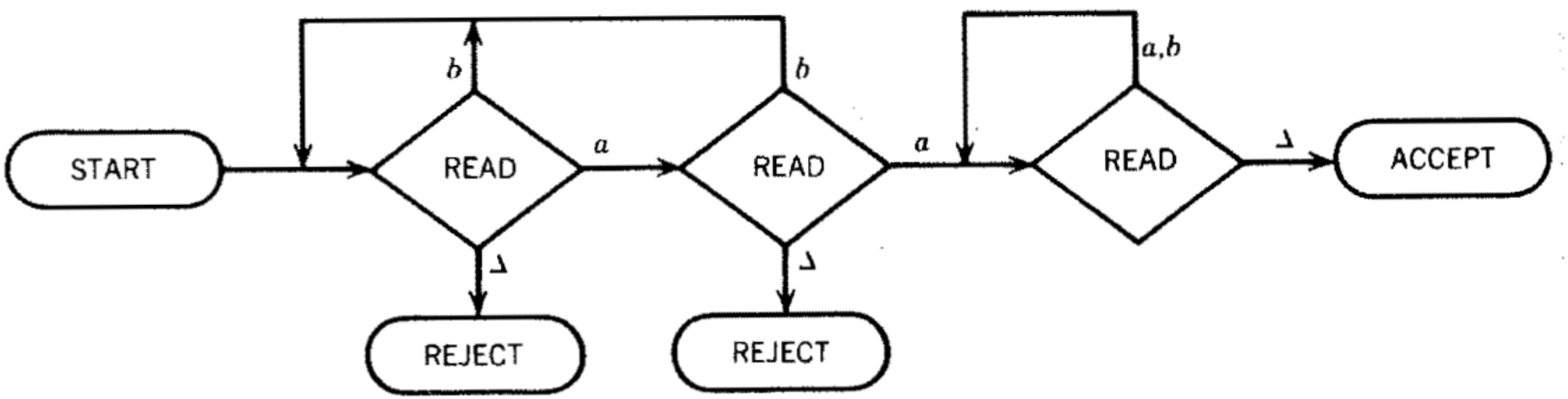




Example

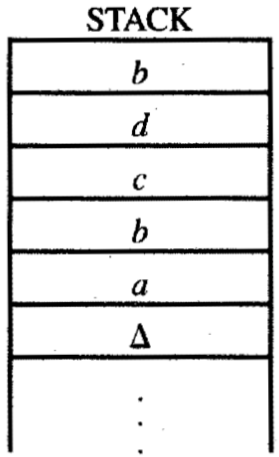


Becomes..



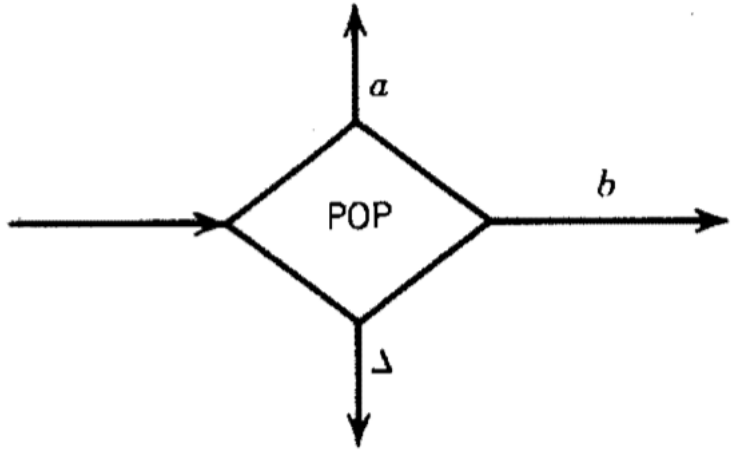
Adding a Pushdown Stack

- We construct new pictures for FAs because it is easier now to make an addition to our machines: push-down stack
- The operation push adds a new letter to the top of the stack and all the other letters are pushed back (or down)
- Before the machine begins to process an input string, the stack ,s assumed to be empty



Adding a Pushdown Stack

- The instruction to take a letter out of the stack is called pop
- The rest of the letters in the stack are moved up one location
- A push down stack is called a LIFO file
- A pushdown stack lets us read only the top letter
- To read the third letter, we must go pop, pop, pop
- We also have no simple instruction to determine the bottom letter, tell how many b's are in the stack, etc.
- The only stack operations are push and pop

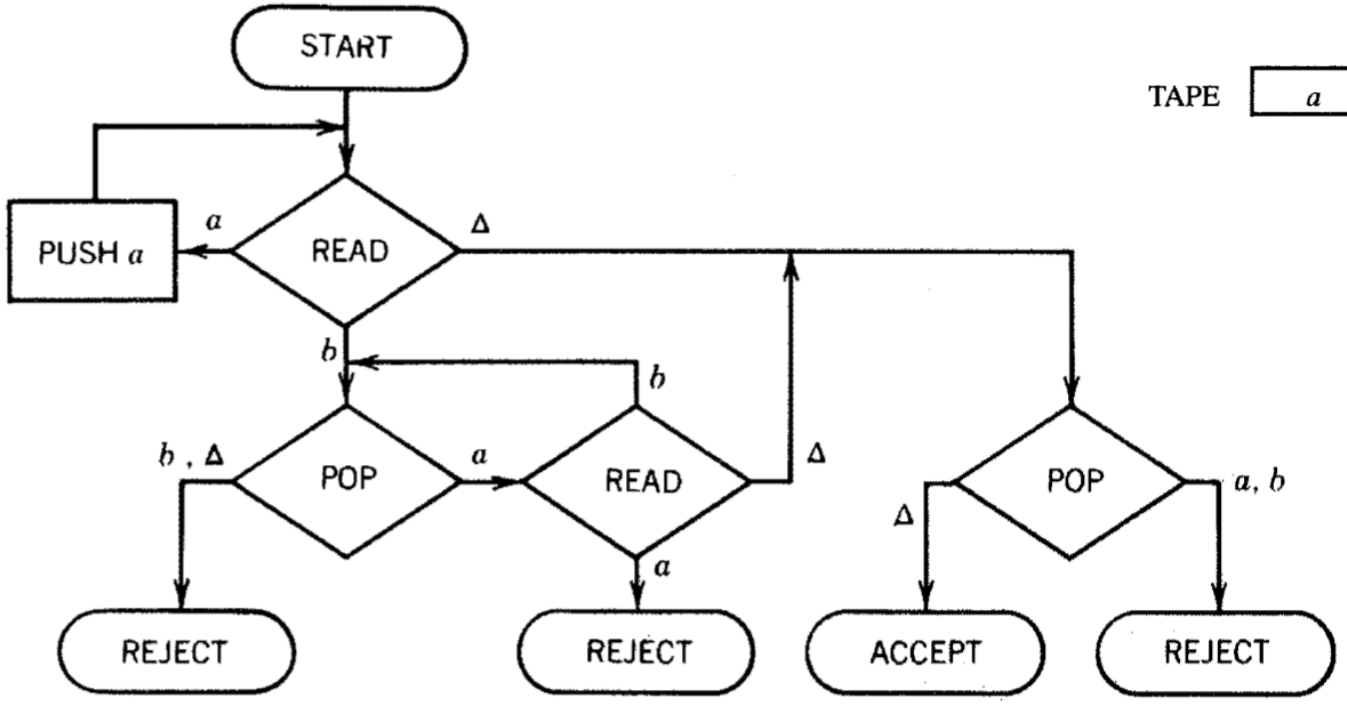


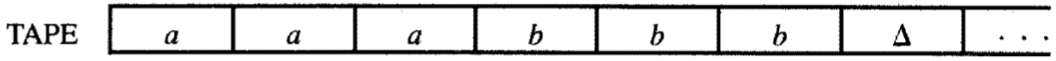
Adding a Pushdown Stack

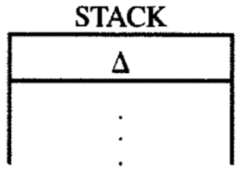
- The edges coming out of a pop state are labeled in the same way as the edges from a read state
- Branching can occur at pop states but not at push states
- We can leave push states only by the one indicated route
- We can enter a push state from any direction
- When FAs have been souped with a stack and pop and push states we call them pushdown automata (PDA)

Example

- Consider the following PDA on the input string aaabbb

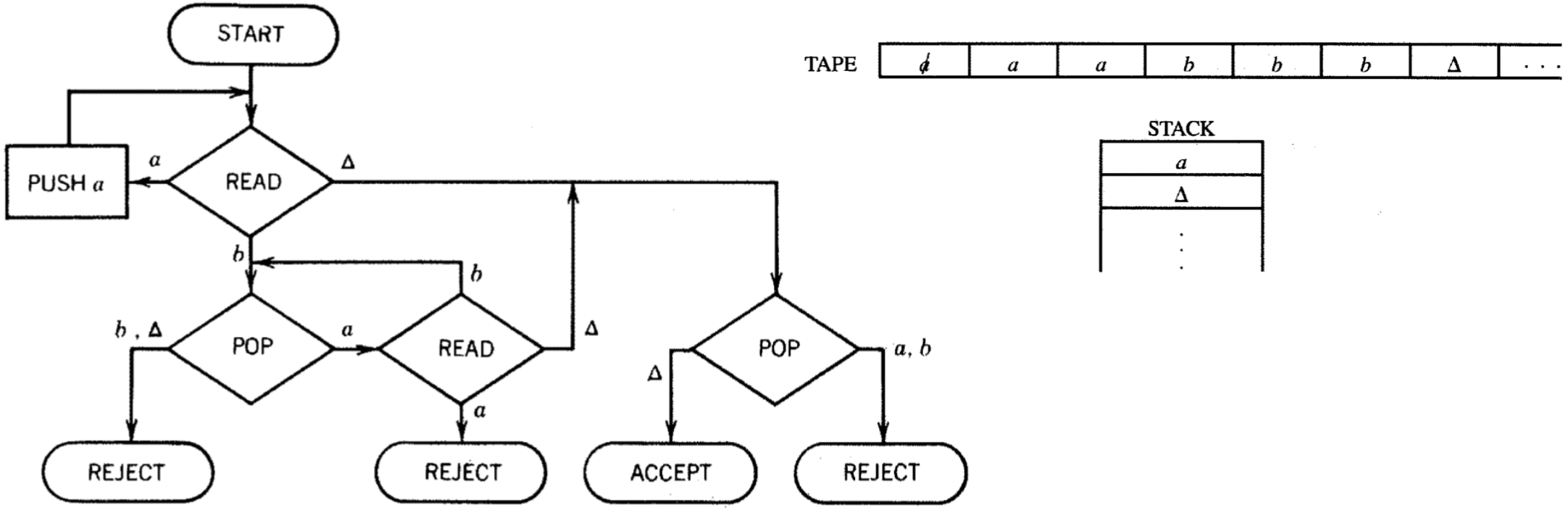






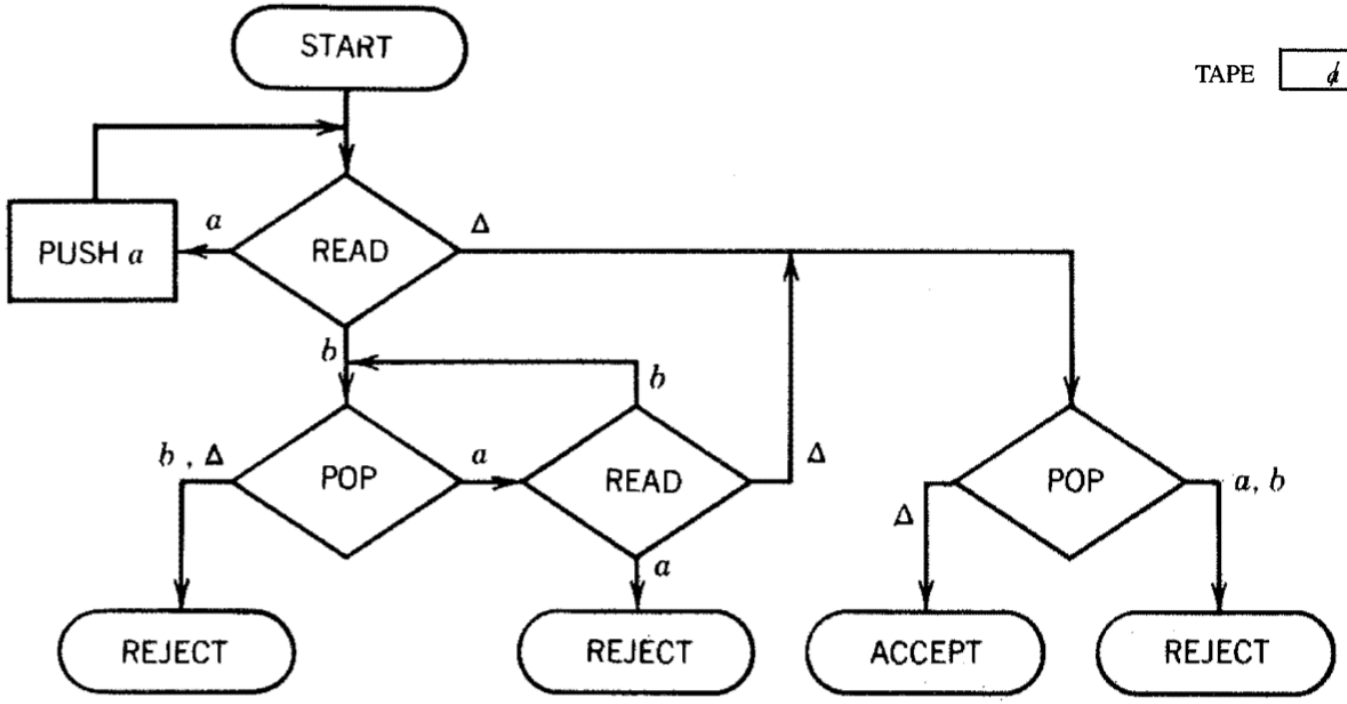
Example

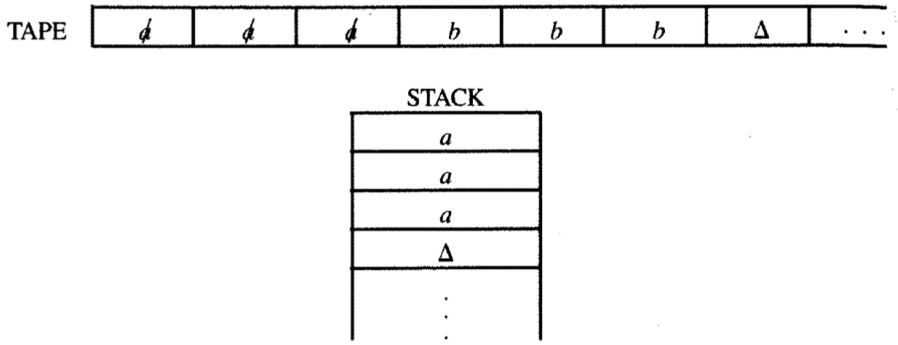
- Consider the following PDA on the input string aaabbb



Example

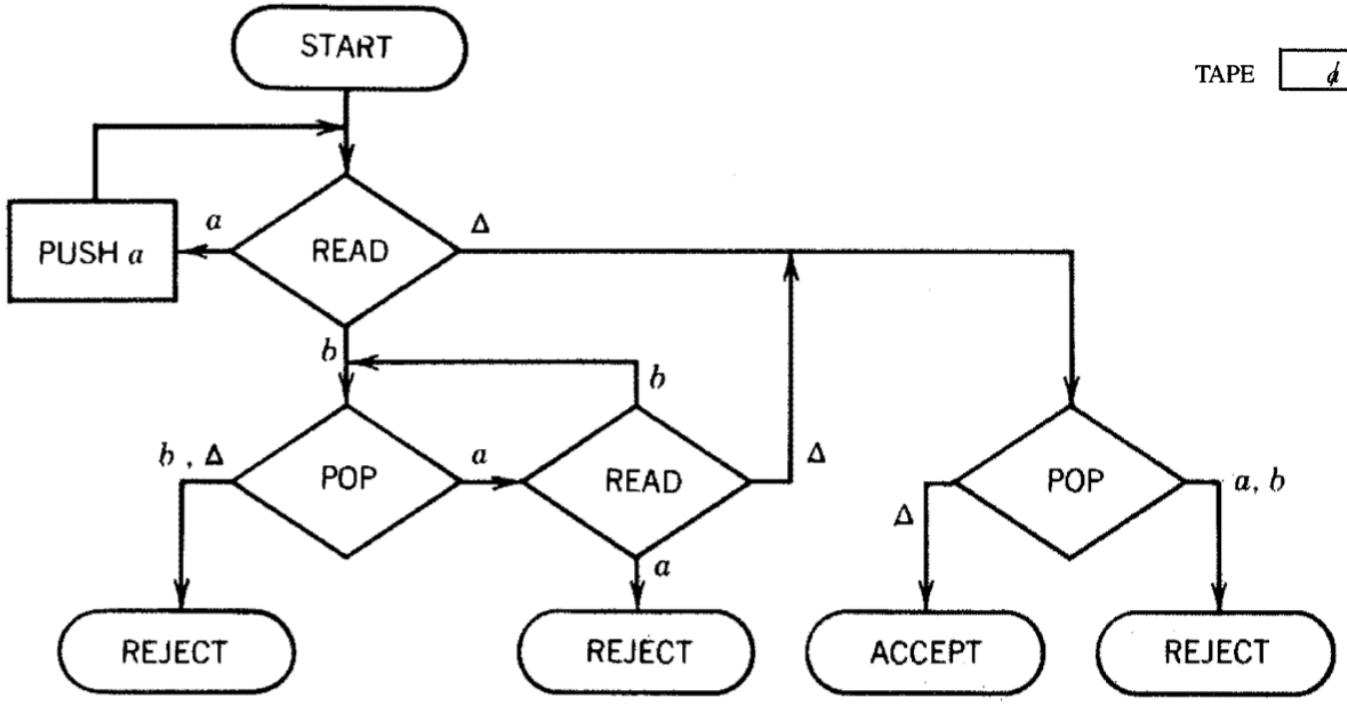
- Consider the following PDA on the input string aaabbb

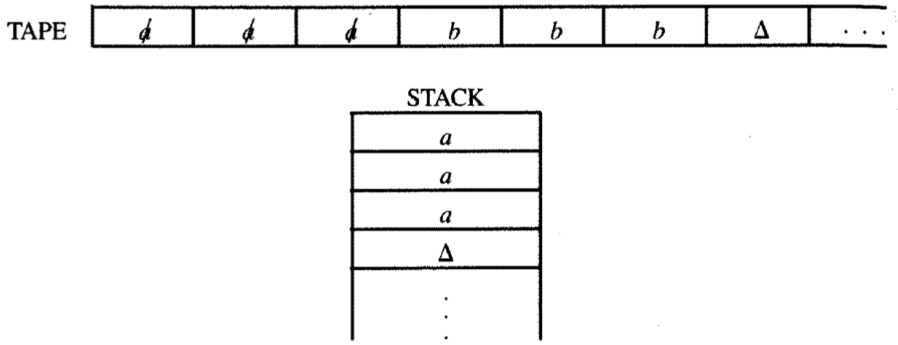




Example

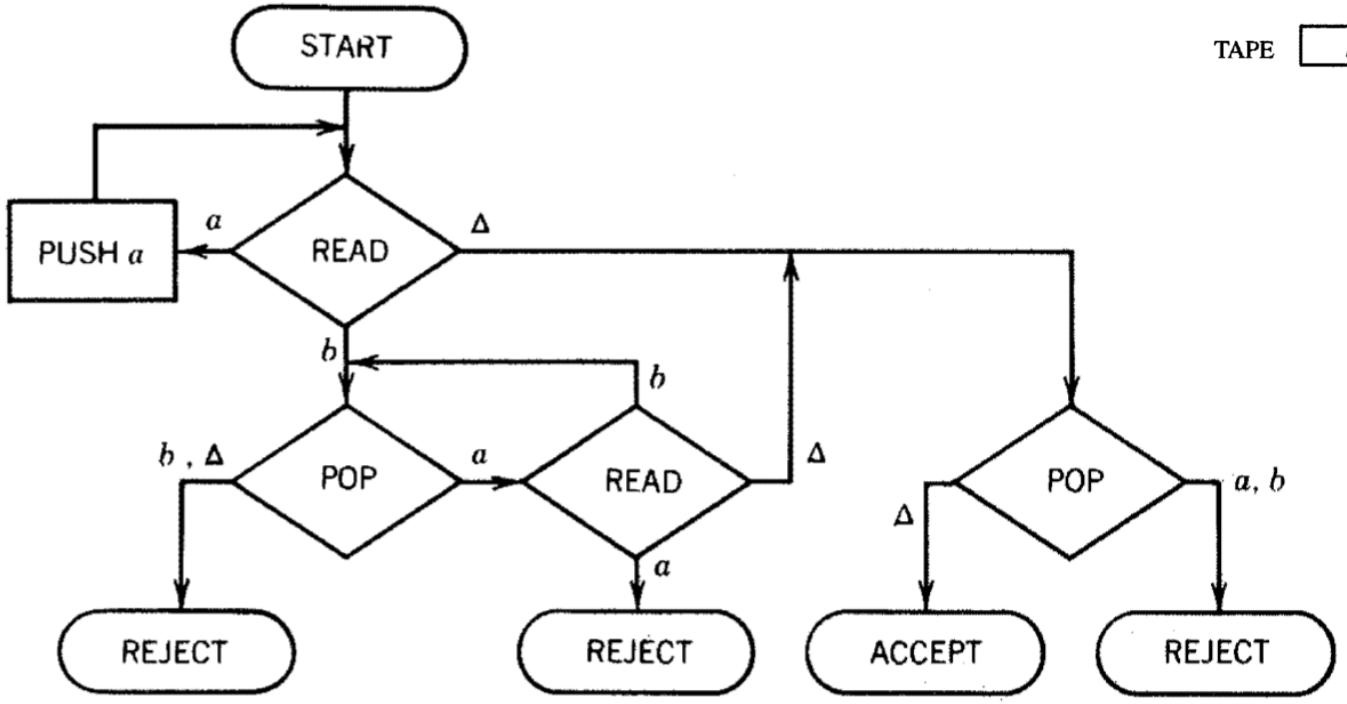
- Consider the following PDA on the input string aaabbb

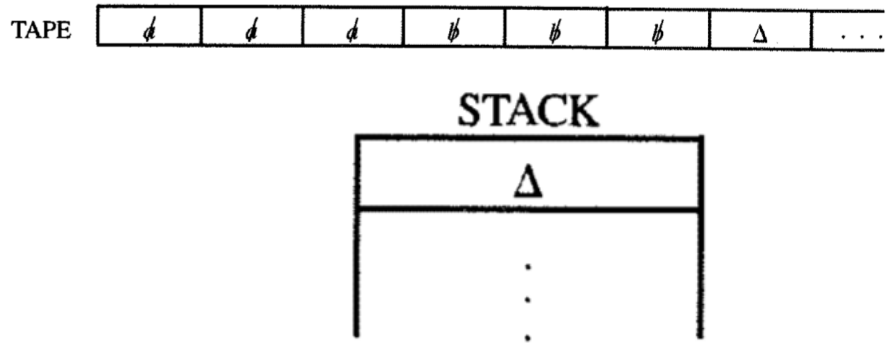




Example

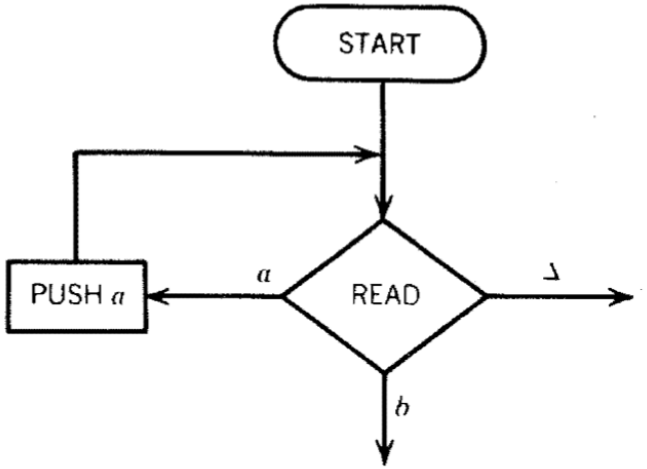
- aaabbb is accepted by this machine ($a^n b^n$, n= 0 1 2 …)





Example

- The first part of the machine is a circuit of states that reads from the tape some number of a's in a row and pushes them into the stack
- This is the only place in the machine where anything is pushed into the stack
- Once we leave this circuit, we cannot return and the stack contains everything it will contain



Example

- After we have loaded the stack with all the a's from the front of the input string, we read another letter from the tape
  - If this is a Λ, it means that the input word was of the form a^n (n might have been 0)
  - If this is the input we take the Δ-line to the right side pop state. This tests the stack to see whether it has anything in it. If it has, we go to reject. If the stack is empty, we accept.
  - If we read a b after reading a's, this takes us to a new section of the machine

Example

- On reading the first b, we immediately pop the stack. The stack can contain some a's or only Δ's.
- If the input started with a b, we would be popping the stack without ever having pushed anything onto it
  - We would the pop a Δ and go to reject
  - If we pop a b, something impossible has happened. We go to reject
  - If we pop an a, we go to the lower right read that asks us to read a new letter

PDA

- What makes these machines more powerful than FAs?
- FA cannot accept the language a^nb^n because a^n part had run run around in a circuit and FA cannot keep track of how many times it has looped around the circuit
- However, the PDA has a primitive memory unit. It can keep track of how many a's are read at the beginning
- We need not restrict ourselves to use the same alphabet for input strings as we use for the stack

PDA

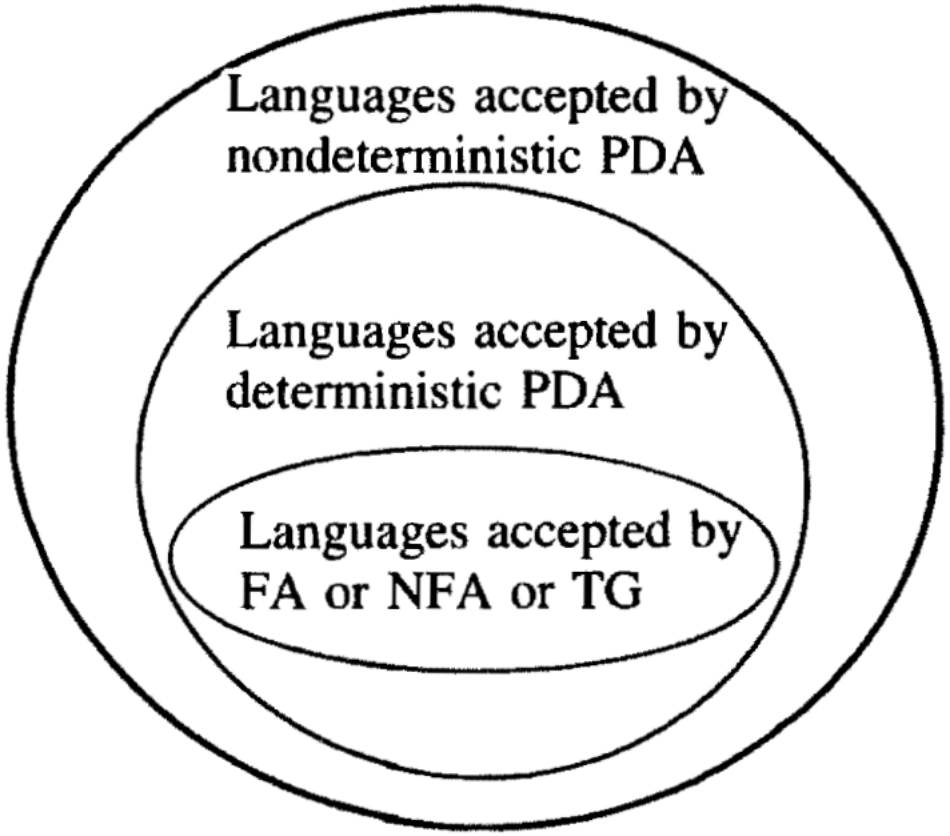
- PDAs will have to be nondeterministic as well if they are to correspond to CFGs (consider the language palindrome)
- A deterministic PDA is one like pictures we draw earlier. Every input string has a unique path through the machine
- A nondeterministic PDA is one for which at certain times we may have to choose among possible paths through the machine
  - We say that an inout string is accepted by such a machine if *some* set of choices leads us to an accept state
  - This is similar to the definition of acceptance for nondeterministic TGs

PDA

- We shall have freedom not to put a b-edge leading out of a particular read state. If a b is read from the tape, processing cannot continue and the machine crashes and the input is rejected
- PDAs that are equivalent to CFGs are the nondeterministic one
- For FAs , we found that nondeterminism did not increase the power of the machine to accept new languages
  - For PDAs, this is different

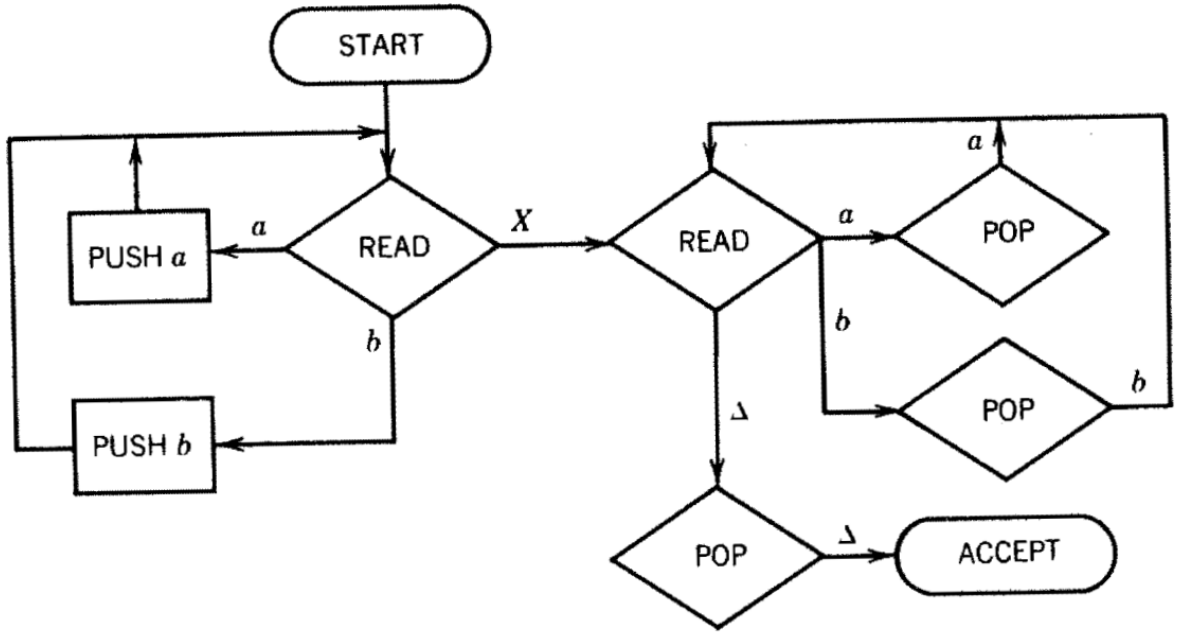
PDA

- The relative power of three types of machines



Example

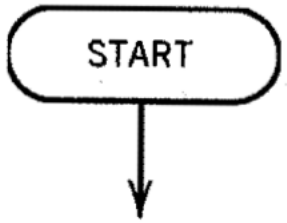
- The language PALINDROMEX
  - s X reverse(s)
  - S is any string in (**a** + **b**)*****
  - The PDA without rejects



Definition

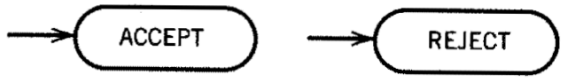
- A pushdown automata (PDA) is a collection of eight things

1. An alphabet Σ of input letters.
2. An input TAPE (infinite in one direction). Initially, the string of input letters is placed on the TAPE starting in cell i. The rest of the TAPE is blank.
3. An alphabet Γ of STACK characters.
4. A pushdown STACK (infinite in one direction). Initially, the STACK is empty (contains all blanks).
5. One START state that has only out-edges, no in-edges:

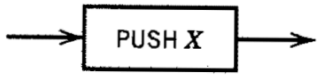


Definition

6. Halt states of two kinds: some ACCEPT and some REJECT. They have in-edges and no out-edges:

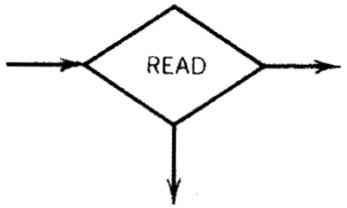


7. Finitely many nonbranching PUSH states that introduce characters onto the top of the STACK. They are of the form



where X is any letter in Γ.

8. Finitely many branching states of two kinds:
   (i) States that read the next unused letter from the TAPE



which may have out-edges labeled within letters from Σ and the blank character Δ, with no restrictions on duplication of labels and no insistance that there be a label for each letter of Σ, or Δ.

   (ii) States that read the top character of the STACK