Theory of Automata
Pushdown Automata

Hafiz Tayyeb Javed

Week 13

Lecture 02

Contents

- A New Format for FAs
- Adding a Pushdown Stack
- Defining the PDA

A new format for FA

- We have learned that all regular languages can be generated by CFGs, and so can some non-regular languages. In other words, the set of CFLs is larger than the set of regular languages.

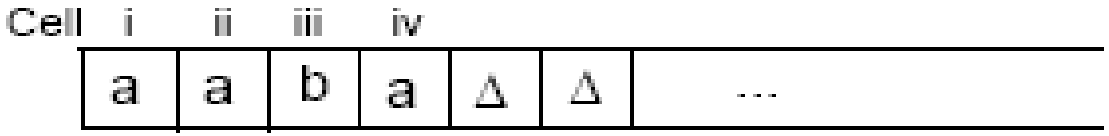
- Since for each regular language, there is an FA that accepts exactly the language, we expect that for each CFL, there should also be at least one machine that accepts exactly the CFL.

- In other words, we would like to have CFL-acceptors (or CFG recognizers) in the same manner as FAs are regular language-acceptors (or recognizers).

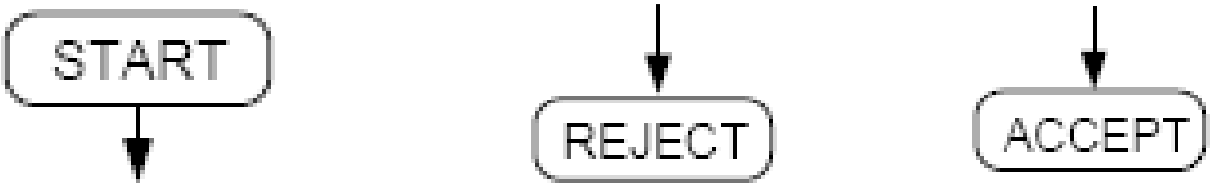
- In this lecture, we shall develop such a new type of machine. In the next lectures, we shall prove that these new machines do indeed correspond to CFLs in the way we desire.

INPUT TAPE

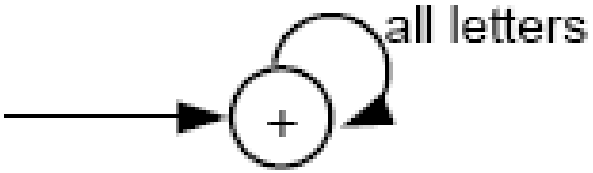
- The INPUT TAPE is part of the machine where the input string is placed. It **is infinitely long** to accommodate any possible input.
- The locations into which we put the input letters are called **cells**. We name cells with lowercase Roman numerals.
- The character Δ is used to indicate a **blank** in a TAPE cell.
- We read **one letter** at a time, from **left to right**, and never go back to a cell that was read before. When we reach the first blank cell, we stop. We always presume that once the first blank is encountered, the rest of the TAPE is also blank.



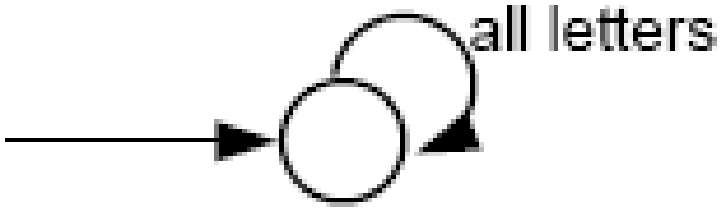
Other Symbols



- The START state is like a - state in an FA. We begin the process here, but we read no input letter and go immediately to the next state.
- The ACCEPT state is a dead-end final state + once entered, it cannot be left, such as



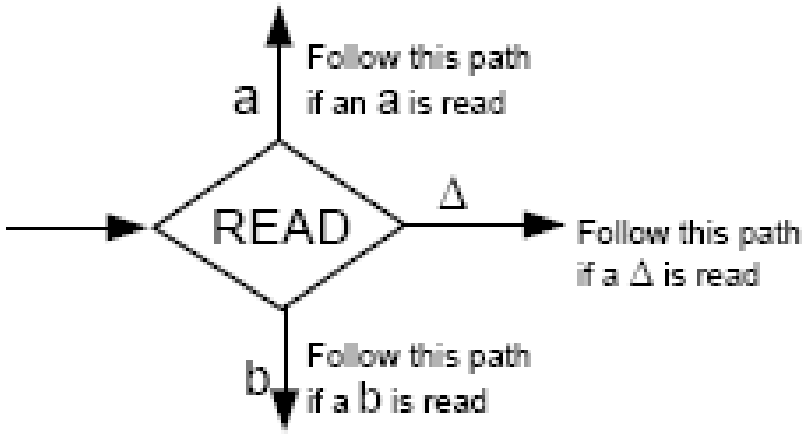
- The REJECT state is a dead-end state that is not final, such as



- The ACCEPT and REJECT states are called **halt** states. Halt states cannot be traversed (i.e., cannot be passed through like a final state in an FA).

READ states

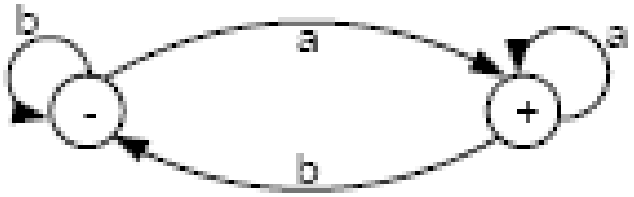
- READ states are depicted as diamond-shaped boxes as shown below:

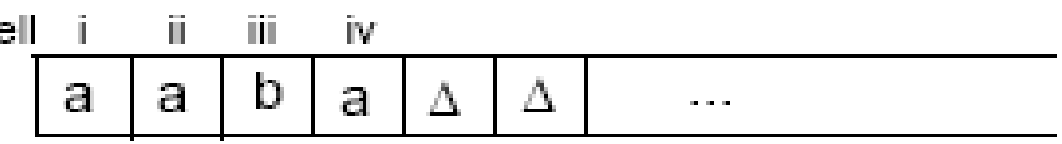


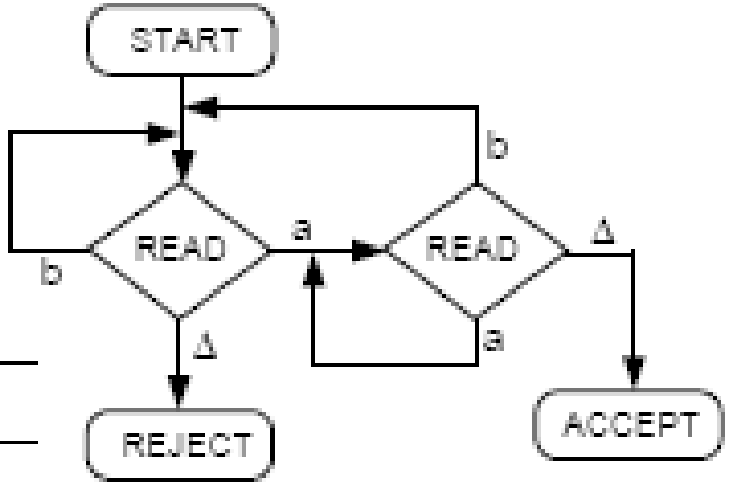
- When Δ is read from the TAPE, it means that we are out of input letters. We are then finished processing the input string. Hence, the Δ-edge will lead to ACCEPT or REJECT.

Example

- Below is an FA that accepts all words ending with letter a, and its equivalent new picture.



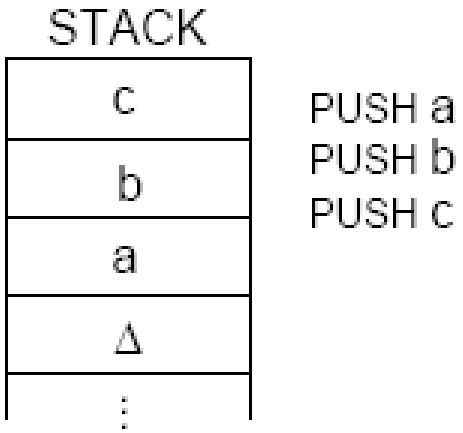




Adding a Pushdown Stack

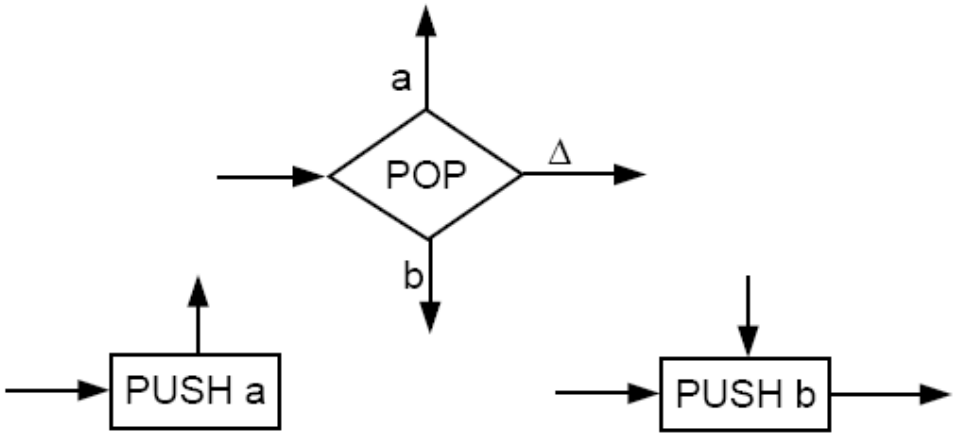
- We bothered to construct new pictures for FAs so that it is now easier to make an addition to the machine.
- We would like to add a **PUSHDOWN STACK** to our machine. A PUSHDOWN STACK is a place where input letters can be stored until we want to refer to them again.
- Before the machine begins to process an input string, the STACK is presumed to be empty.
- The operation PUSH adds a new letter to the top of the STACK. All the other letters are pushed down accordingly.
- The operation POP takes a letter out of the STACK. The rest of the letters are moved up one location each accordingly.

- For example, if we perform operations PUSH a, PUSH b, and PUSH c, then the stack will look like the following:



- Popping an empty STACK, like reading an empty TAPE, gives us the blank character Δ.

- We can add a PUSHDOWN STACK and the operations PUSH and POP to our machine by including the following symbols:

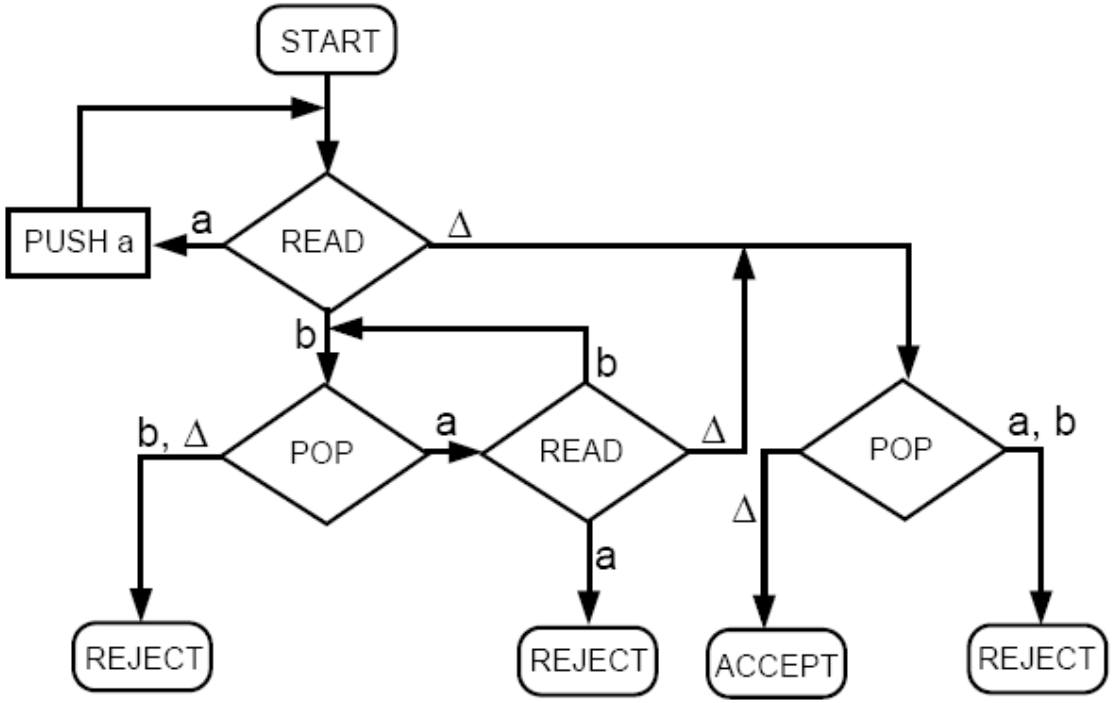


- The edges coming out of a POP state are labeled in the same way as the edges coming out of a READ state.
- Branching can occur at POP states but not at PUSH states: We can leave PUSH states only by the one indicated route, although we can enter a PUSH state from any direction.

- When FAs are equipped with a STACK and POP and PUSH states, we call them **pushdown automata** or **PDAs**.

- The notion of a PUSHDOWN STACK as a data structure has been around for a long time. However, when we incorporate this memory structure into an FA, its language-recognizing capabilities are increased considerably.

- Consider this PDA:



- What language is accepted by this machine?

- Let’s see the machine in operation on the input string ***aaabbb.***
- At the beginning the INPUT TAPE contains (from left to right) the letters a, a, a, b, b, b, Δ, Δ, Δ…; while the STACK is empty, that is, it contains (from top down) only the blank characters Δ, Δ, …
- After reading the first a, the remaining input letters to be read in the TAPE are a, a, b, b, b, Δ, … and the STACK looks like (from top down) a, Δ, …
- Similarly, after reading 3 letters a, we have

  TAPE = b, b, b, Δ, … STACK = a, a, a, Δ, …
- After reading the first b, STACK pops an a, so

  TAPE = b, b, Δ,… STACK = a, a, Δ, …

- Similarly, after reading the last b, STACK pops the last a, and we have

  TAPE = Δ, Δ, … STACK = Δ, Δ, …
- When we encounter the blank character Δ from the INPUT TAPE, the STACK pops a blank , and we are lead to the ACCEPT state.
- Hence, the string ***aaabbb*** is accepted by this machine.

- In fact, the language accepted by this machine is the **non-regular** language

  $\{a^n b^n \; n = 0, 1, 2, 3, \ldots\}$

  (see discussion on page 298 if you are not convinced)

- This machine can accept this non-regular language because it has a memory unit (the STACK) to keep track of how many a's were read at the beginning. Since FAs do not have such memory, they are not able to accept this language.

Remarks

- We need not restrict ourselves to using the same alphabet for input strings and for the STACK.
- In the example above, we could have read an a form the TAPE and then pushed an X into the STACK; that is, we let the ***X***'s count the number of a's.

- In this case, the READ state must provide branches for a, b, or Δ. The POP state must provide branches for X or Δ.

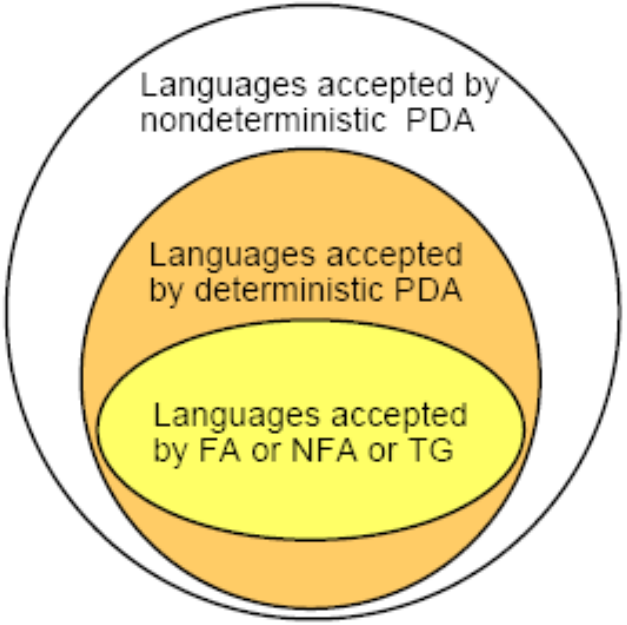
- Therefore, when we define PDAs later on, we shall require the specification of the TAPE alphabet ∑ and the STACK alphabet Γ, which may be different.

Deterministic VS Non-Deterministic PDA

- Since our goal is to produce a new type of machine that can recognize all CFLs (just as FAs can recognize all regular languages), the addition of a simple STACK may not be enough. We shall see that the new PDAs will have to be nondeterministic.

- A **deterministic PDA** is one for which every input string has a unique path through the machine (like the PDA in our example above).

- A **nondeterministic PDA** is one for which at certain times we may have to choose among possible paths through the machine.
  - An input string is accepted by a nondeterministic PDA if **some** set of choices leads us to an ACCEPT state.
  - If for **all possible** paths that a certain input string can follow, it always ends at a REJECT state, then the string must be rejected.



- Non-determinism allows for the possibility of too few (even zero) as well as too many edges leading out of a branch state (READ or POP).

- For example, we can have more than one edge with the same label leading out of a branch state. We can also have no edge, say no b-edge, leading out of a particular READ state. If a b is read at that state, processing cannot continue, the machine crashes, and the input is rejected.

- *Recall that for FAs, Non-determinism (e.g., NFAs and TG) does not increase the power of the machine to accept new languages. For PDAs, this is different:*

Example

- Let's consider what kind of PDA could accept the language of ODDPALINDROME. This is the language of all strings of a's and b's that are palindrome and have an odd number of letters. For instance, a word of this language is w = ***abbabba***.

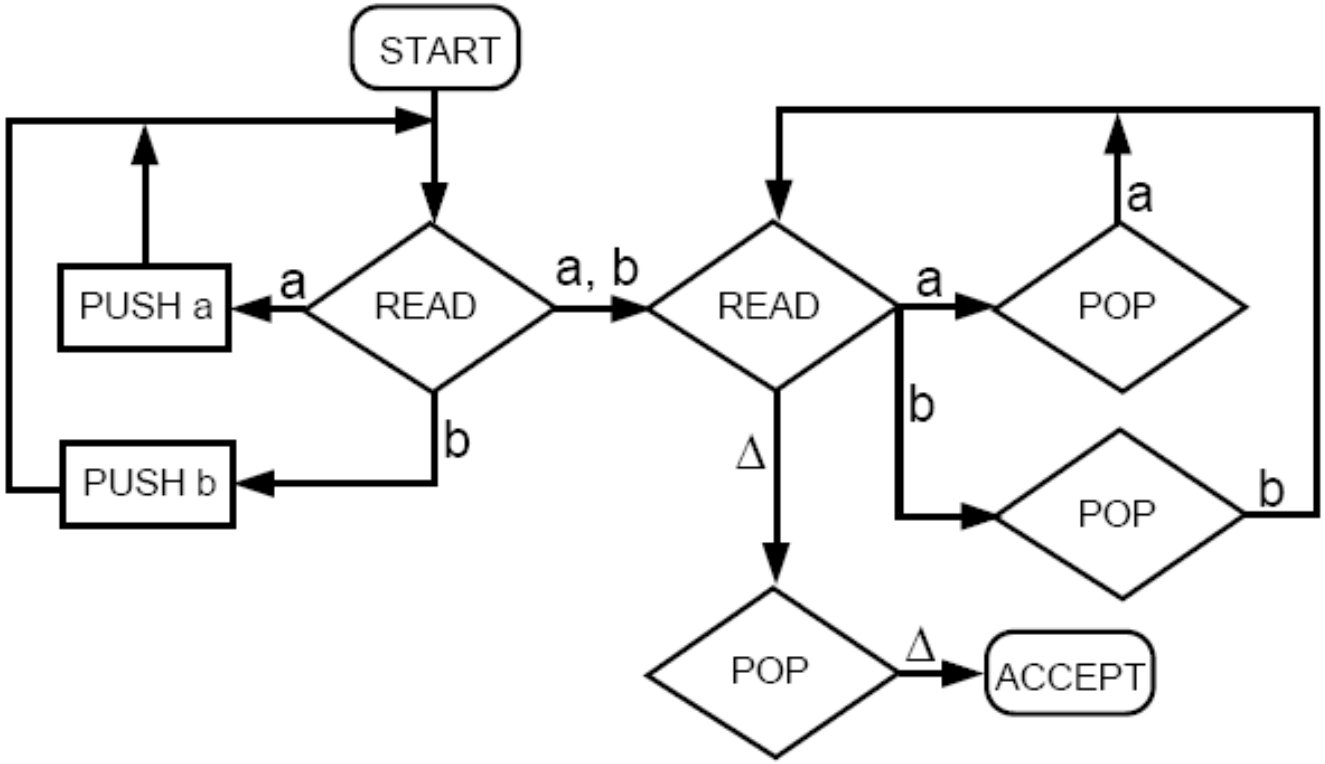
- Possible solution: When we process the front part of the word (***abb***), we also PUSH the input letters onto the STACK. When we process the back part of the string (***bba***), we can POP the STACK in order to compare the letters being read with those from the STACK.

- However, the problem here is that the middle letter does not stand out. It is therefore impossible for the machine to recognize where the front part ends and where the back part begins: A PDA, just like an FA, reads the input string from left to right and has no idea at any stage how many letters remain to be read.

- The PDA therefore should be nondeterministic: We need to make a choice to follow the right edge at the right time (i.e. when reading the middle letter.)

- The nondeterministic PDA that accept ODDPALINDROME is shown in the next slide.

- We shall not show REJECT states in the picture. This means that when an input string has no path to follow, it will **crash** and be rejected.

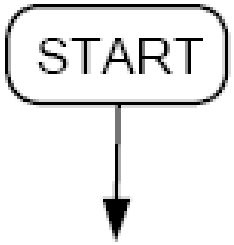


- Try the input string ***aba*** on this machine.

Defining the PDA

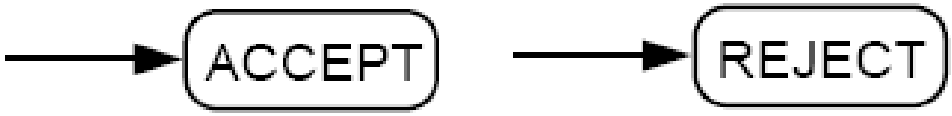
A **pushdown automaton, PDA**, is a collection of eight things:

1. An alphabet ∑ 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, and no in-edges:



Defining the PDA (contd.)

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

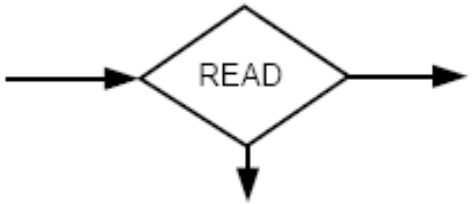


7. Finitely many non-branching PUSH states that put characters onto the top of the STACK. They are of the form

where X is any letter in Γ.

8. Finitely many branching states of two kinds:

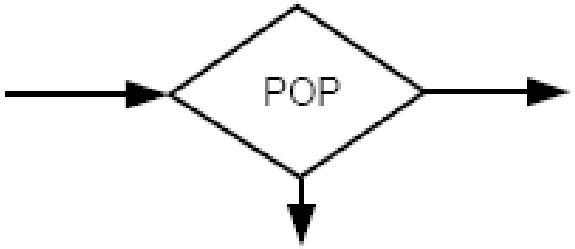
(i) States that read the next unused letter from the TAPE



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

Defining PDA (Contd.)

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



which may have out-edges labeled with the characters of Γ and the blank character Δ, again with **no restrictions**.

- We require that the states be connected to become a connected directed graph.
- To **run** a string of input letters on a PDA means that we begin from START and follow the unlabeled edges and those labeled edges, **making choices of edges when necessary**, to produce a path through the graph.

Defining the PDA (Contd.)

- This path will end either at a halt state or will crash in a branching state when there is no edge corresponding to the letter (character) being read (popped).
- When letters (characters) are read (popped) from the TAPE (STACK), they are used up and vanish.
- An input string with a path that ends in ACCEPT is said to be **accepted**.
- An input string that can follow a set of paths is said to be accepted if at least one of these paths leads to ACCEPT.
- The set of all input strings accepted by a PDA is called the **language accepted** by the PDA, or the **language recognized** by the PDA

Theorem 28

For every regular language L, there is some PDA that accepts it.

Proof:

- Because L is regular, there is an FA that accepts it.

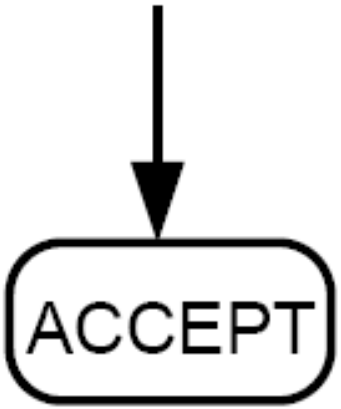
- We have shown how to convert an FA into an equivalent PDA at the beginning of this lecture.

Theorem 29

Given any PDA, there is another PDA that accepts exactly the same language with the additional property that whenever a path leads to ACCEPT, the STACK and the TAPE contain only blanks.

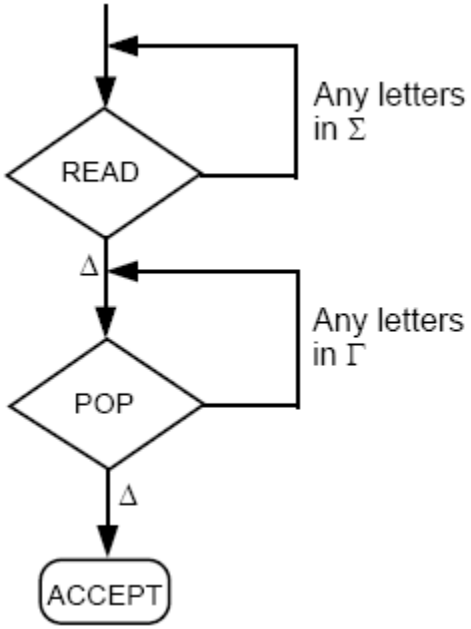
Proof of Theorem 29

- We present a constructive algorithm that will convert any PDA into a PDA with the property mentioned above.
- Whenever we have the machine part



Proof of Theorem 29 (Contd.)

- we replace it with the following diagram:



- The new PDA formed accepts exactly the same language and finishes all successful paths with empty TAPE and empty STACK.

Contents

- Theorem 30
- Theorem 31