جامـــــعة المـــــستقبل
AL MUSTAQBAL UNIVERSITY

قســـم الامـــــن الـــــــسيبرانـــــــي
DEPARTMENT OF CYBER SECURITY

SUBJECT: COMPUTATION THEORY
CLASS: 3rd

LECTURER: MSC :MUNTATHER AL-MUSSAWEE

LECTURE: (8)
PUSHDOWN AUTOMATA (PDA)

Lecture Six

PushDown Automata (PDA)

A **PDA** is a collection of eight things:

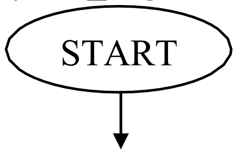
1- An alphabet Σ of input letters.

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 blanks.

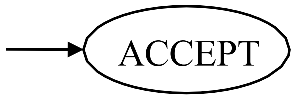
3- An alphabet Γ of STACK characters.

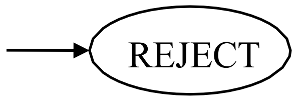
4- A pushdown STACK (infinite in one direction). Initially the STACK is empty (contains all blanks)

5- One START state that has only out_adges, no in-edges.

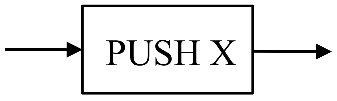


6- HALT states of two kinds: some ACCEPT and some REJECT they have inedges and no out-edges.





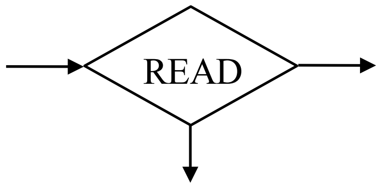
7- Finitely many non-branching PUSH states that introduce characters onto the top of the STACK. they are of the form:



Where X is any letter in Γ.

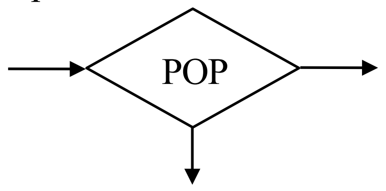
8- Finitely many branching states of two kinds:

a) States that read the next unused letter from the TAPE.



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

b) States that read the top character of STACK.



Which may have out-edges labeled with letters from Γ and the blank character Δ, again with no restrictions.

✍ **Note**: we require that the states be connected so as to become a connected directed graph.

Theorem

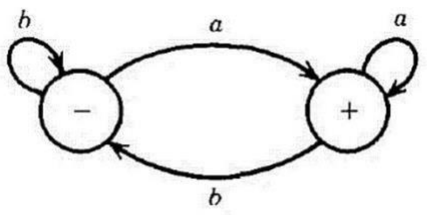
For every regular language L there is some PDA that accepts it.

Poof

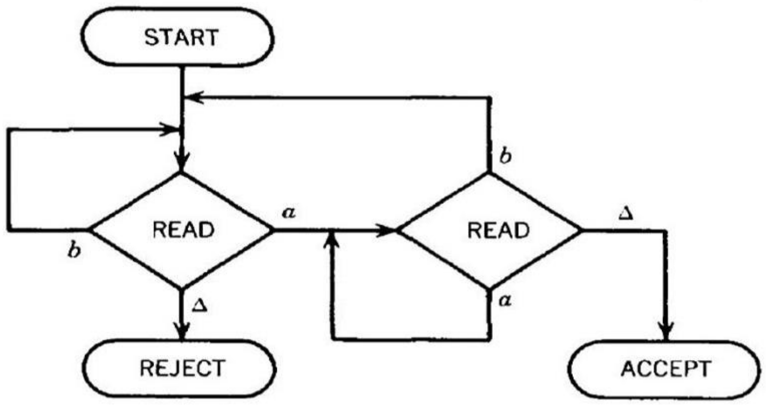
Since L is regular, so it is accepted by some FA, then we can convert FA to PDA (as in the following example).

Example:

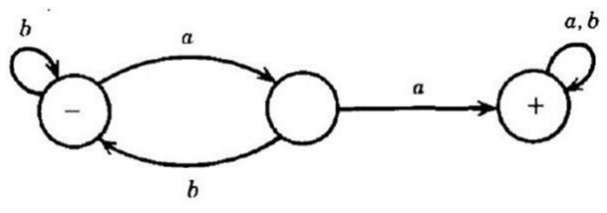
The FA that used to be drawn like this:



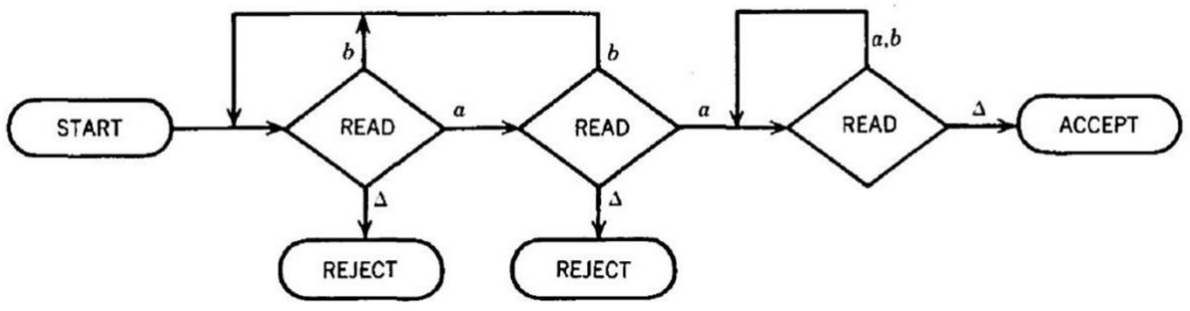
(the FA that accepts all words ending in the letter a) becomes, in the new symbolism, the machine below:



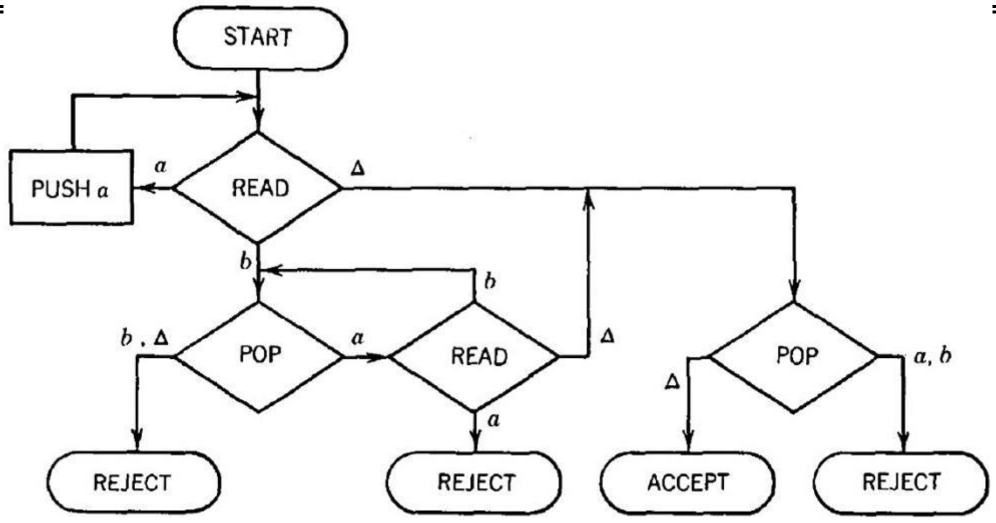
Example:



Becomes:



Example: The language accepted by this PDA is exactly: {a^nb^n, n = 0, 1, 2, …}



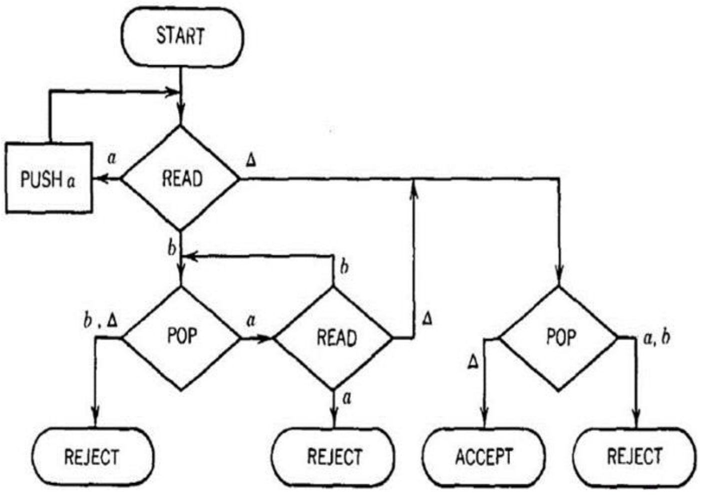
Before we begin to analyze this machine in general, let us see it in operation on the input string aaabbb. We begin by assuming that this string has been put on the TAPE. We always start the operation of the PDA with the STACK empty as shown:

TAPE	a	a	a	b	b	b	Δ	…

We have to PUSH the first part of string into the STACK and then POP contents of the STACK when we start reading the second part of the string. For example, aaabbbΔ We will PUSH all "aaa" into the stack then we will POP when we start reading "bbb".

We can divide string reading into two stages. When we get the first part "aaa" we will PUSH them into the STACK, and when we read the second part "bbb" we will POP from the stack. We should get "aaa" and the STACK will be empty and the string is reached to the space symbol, so if we read the space symbol we have to POP from the stack, if we get space symbol that is means the string is accepted.

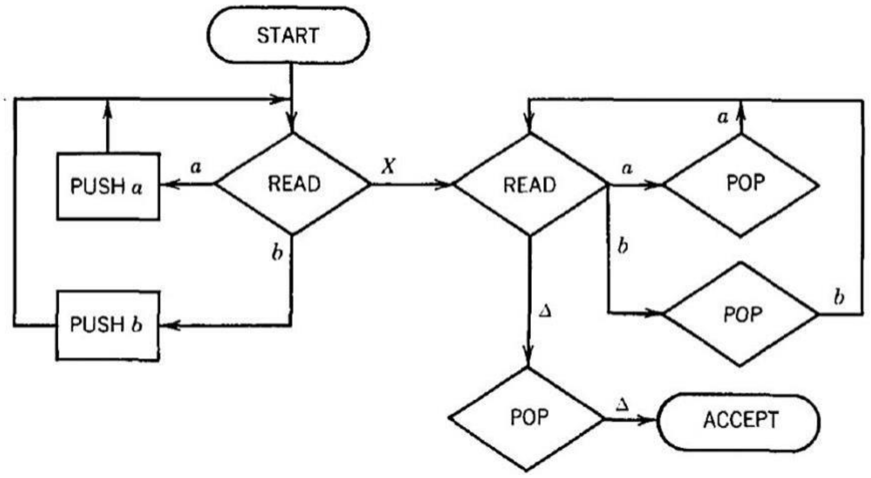
State	**Stack**	**Tape**
START	Δ	aaabbbΔ
READ	Δ	**a**aabbbΔ
PUSH a	aΔ	**a**aabbbΔ
READ	aΔ	**aa**abbbΔ
PUSH a	aaΔ	**aa**abbbΔ
READ	aaΔ	**aaa**bbbΔ
PUSH a	aaaΔ	**aaa**bbbΔ
READ	aaaΔ	**aaab**bbΔ
POP	aaΔ	**aaab**bbΔ
READ	aaΔ	**aaabb**bΔ
POP	aΔ	**aaabb**bΔ
READ	aΔ	**aaabbb**Δ
POP	Δ	**aaabbb**Δ
READ	Δ	**aaabbbΔ**
POP	-	**aaabbbΔ**
ACCEPT		



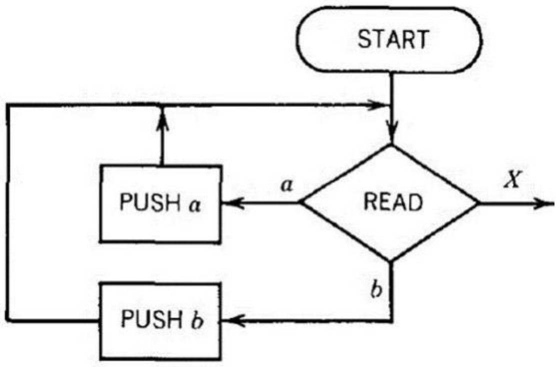
✍ **Note:** The Type of PDA above is called **deterministic PDA** is one (like the pictures we drew above) for which every input string has a unique path through the machine.

Example: Consider the palindrome X, language of all words of the form: sXreverese(s), where s is any string in (a+b)*, such as:

{X, aXa, bXb, aaXaa, abXba, aabXbaa, …}



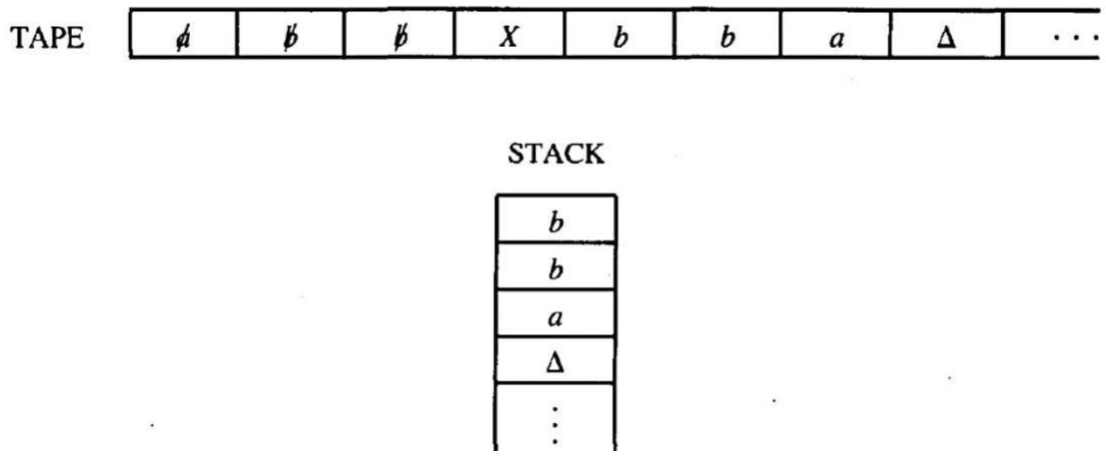
In the first part of machine, the STACK is loaded with the letters from the input string just as the initial a's from $a^n b^n$ were pushed onto the STACK. Conveniently for us, the letters go into the STACK first letter on the bottom, second letter on top of it, and so on till the last letter pushed in ends up on top. When we read the X we know we have reached the middle of the input. We can then begin to compare the front half of the word (which is reversed in the STACK) with the back half (still on the TAPE) to see that they match. We begin by storing the front half of the input string in the STACK with this part of the machine.



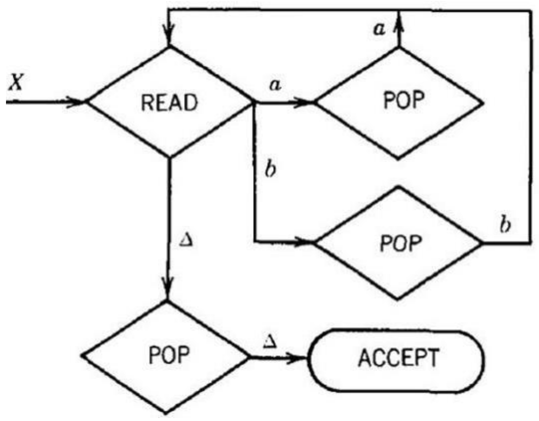
If we READ an a, we PUSH an a. If we READ a b, we PUSH a b, and on and on until we encounter the X on the TAPE.
After we take the first half of the word and stick it into the STACK, we have reversed the order of the letters and it looks exactly like the second half of the word. For example, if we begin with the input string

abbXbba

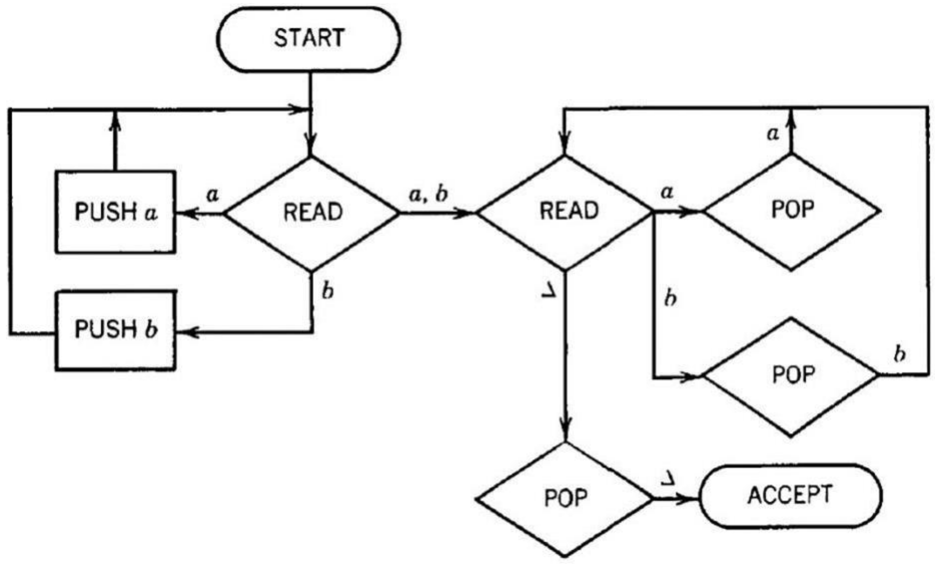
then at the moment we are just about to read the X we have:



When we read the X we do not put it into the STACK. It is used up in the process of transferring us to phase two. This is where we compare what is left on the TAPE with what is in the STACK. In order to reach ACCEPT, these two should be the same letter for letter, down to the blanks.



Example: ODDPALINDROME = {a, b, aaa, aba, bab, bbb, … }

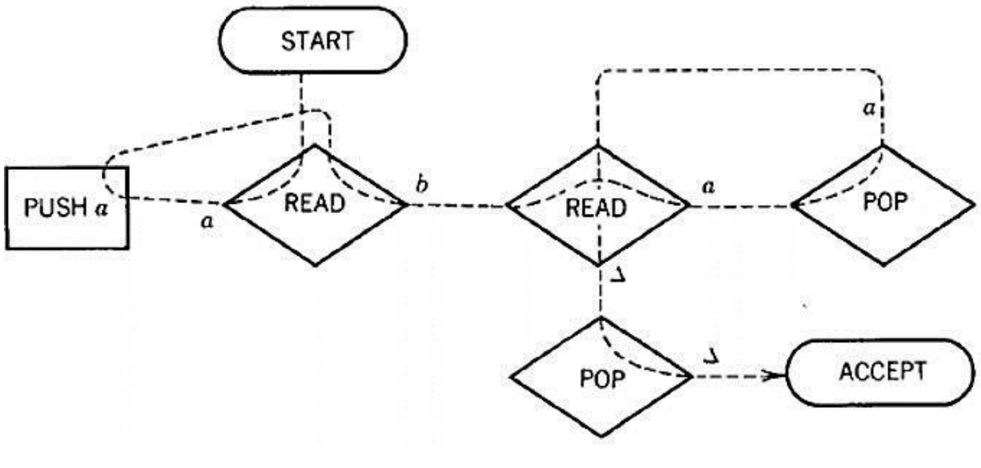


The problem here is that the middle letter does not stand out, so it is harder to recognize where the first half ends and the second half begins. In fact, it's not only harder; it's impossible.

In PALINDROMEX we knew that X marked the spot; now we have lost our treasure map. If we accidentally push into the STACK even one letter too many, the STACK will be larger than what is left on the TAPE and the front and back will not match. The algorithm we used to accept PALINDROMEX cannot be used without modification to accept ODDPALINDROME. We are not completely lost, though. The algorithm can be altered to fit our needs by introducing one nondeterministic jump.

For every word in ODDPALINDROME, if we make the right choices the path does lead to acceptance.

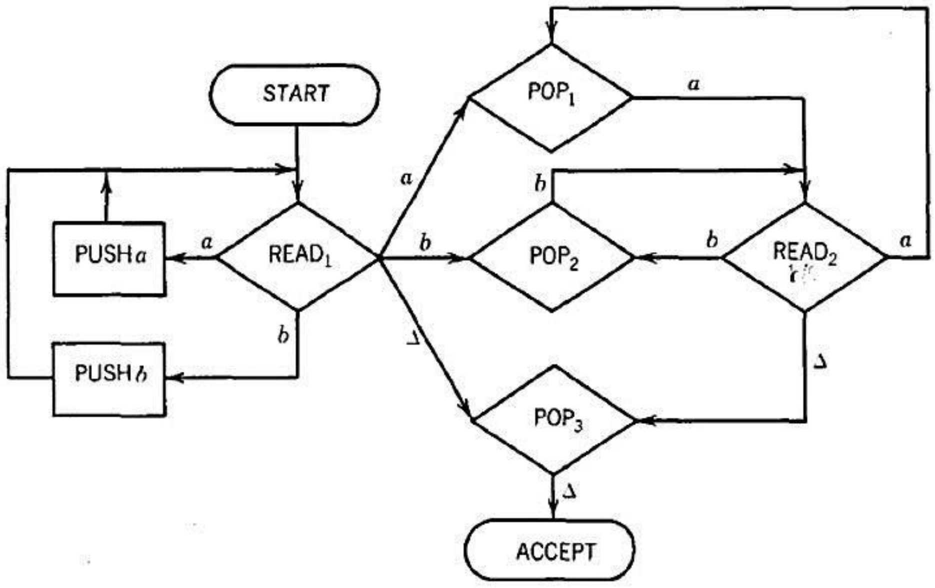
The word "aba" can be accepted by this machine if it follows the dotted path:



State	**Stack**	**Tape**
START	Δ	abaΔ
READ	Δ	**a**baΔ
PUSH a	aΔ	**a**ba Δ
READ	aΔ	**ab**aΔ
READ	aΔ	**aba**Δ
POP	aΔ	**aba**Δ
READ	Δ	**abaΔ**
POP	-	**abaΔ**
ACCEPT		

Example: EVENPALINDROME = {s reverse(s), where s is in (a + b)*}
{Λ, aa, bb, aaaa, abba, baab, babbab, bbbbaaaaaa, ... }

We will check the string “babbab” is ACCEPT or not in the following machine:

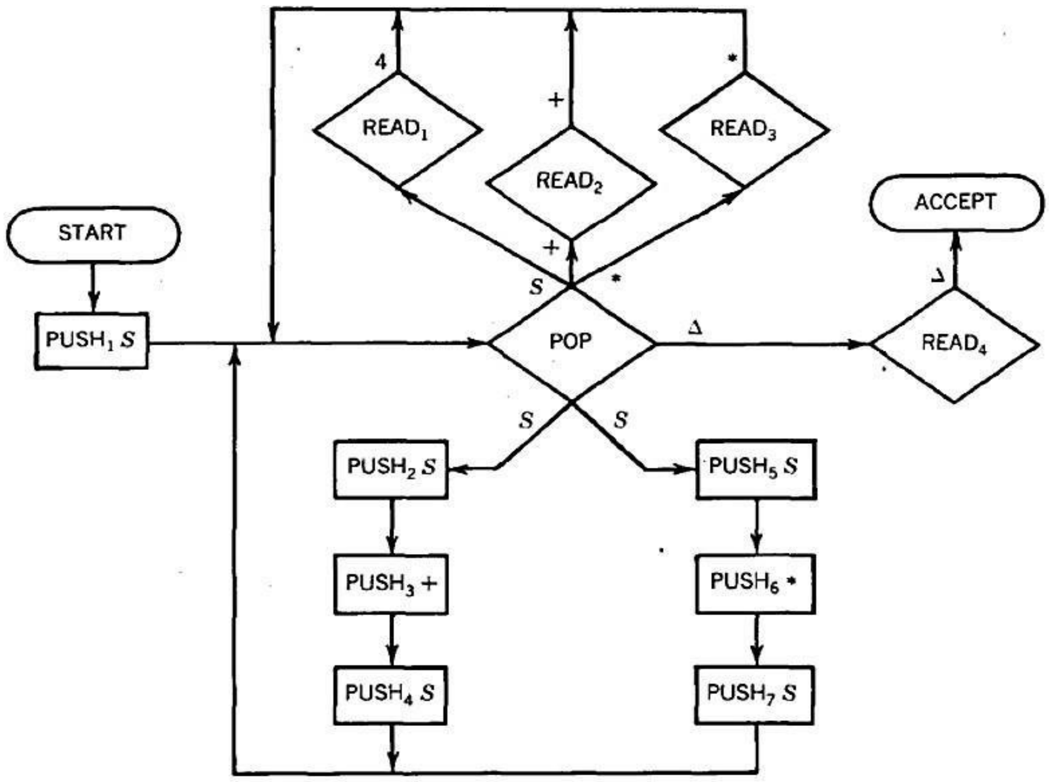


State	Stack	Tape
START	Δ	babbabΔ
READ	Δ	**b**abbabΔ
PUSH b	bΔ	**b**abbabΔ
READ	bΔ	**ba**bbabΔ
PUSH a	baΔ	**ba**bbabΔ
READ	baΔ	**bab**babΔ
PUSH b	babΔ	**bab**babΔ
READ	babΔ	**babb**abΔ
POP	baΔ	**babb**abΔ
READ	baΔ	**babba**bΔ
POP	bΔ	**babba**bΔ
READ	bΔ	**babbab**Δ
POP	Δ	**babbab**Δ
READ	Δ	**babbabΔ**
POP	-	**babbabΔ**
ACCEPT		

✍ **Note:** The type of PDA above is called **nondeterministic PDA** is one for which at certain times we may have to choose among possible paths through the machine.

Homework: Consider the language generated by the CFG:
S → S + S | S ∗ S | 4 The terminals are +, ∗, and 4 and the only nonterminal is S.

- Is the string "4 + 4 ∗ 4" accept in the following PDA or not?


START
PUSH₁ S
POP
READ₁
READ₂
READ₃
READ₄
ACCEPT
PUSH₂ S
PUSH₃ +
PUSH₄ S
PUSH₅ S
PUSH₆ *
PUSH₇ S
4
+
*
S
Δ