CFG = PDA

Shakir Ullah Shah

CFG = PDA

- the set of all languages accepted by PDAs is the same as the set of all languages generated by CFGs.

- We can prove this in two steps.

Theorem 30 and 31

Theorem 30:
Given a CFG that generates the language *L*, there
is a PDA that accepts exactly *L*.

Theorem 31:
Given a PDA that accepts the language *L*, there exists a CFG that generates exactly *L*.

Proof of Theorem 30

- The proof will be by constructive algorithm.
- we can assume that the CFG is in CNF

Example

- Consider the following CFG in CNF:

  $S \rightarrow SB \mid AB$

  $A \rightarrow CC$

  $B \rightarrow b$

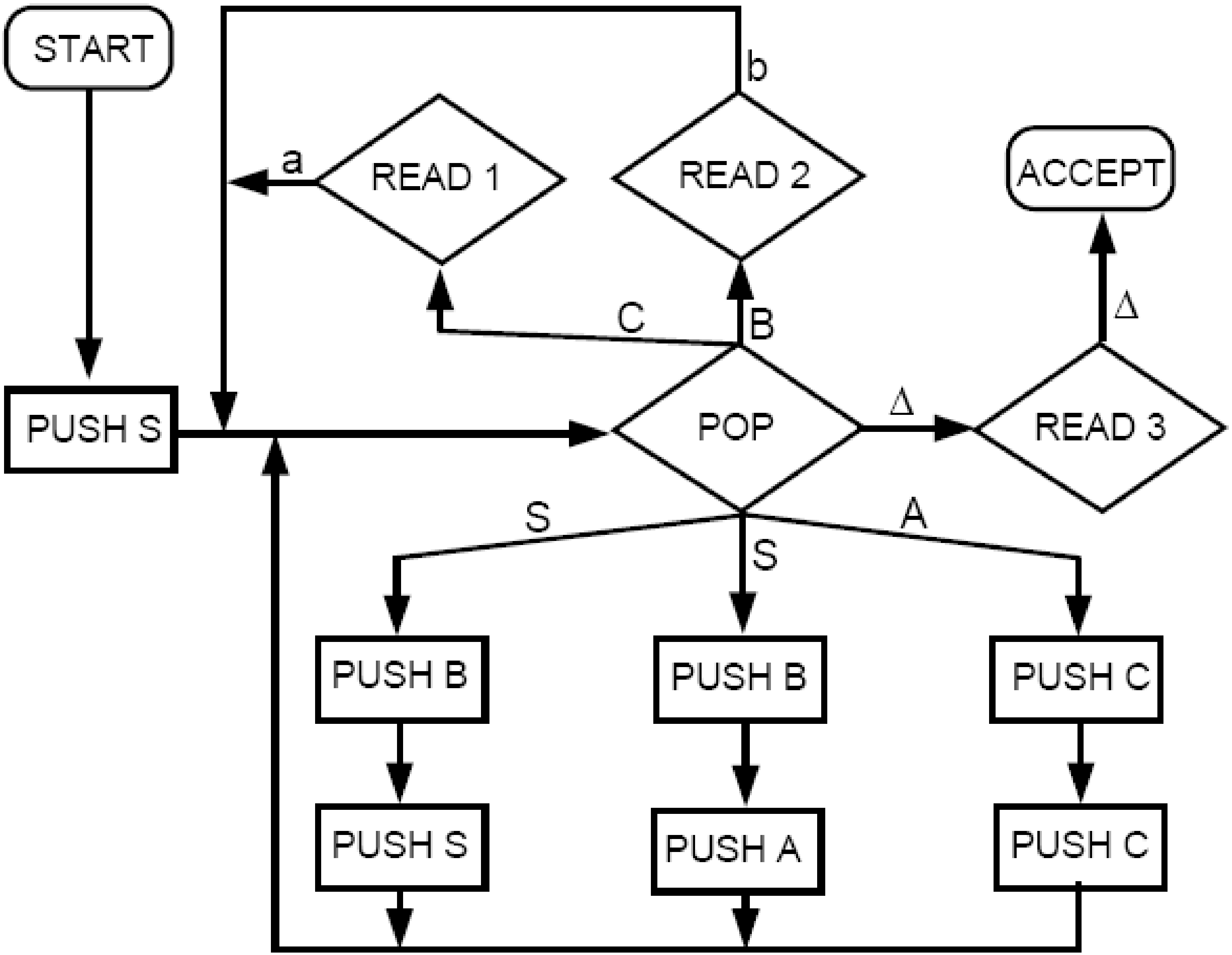
  $C \rightarrow a$

- We now propose the following nondeterministic PDA where the STACK alphabet is

  $\Gamma = \{S, A, B, C\}$

  and the TAPE alphabet is only

  $\Sigma = \{a, b\}$


START
PUSH S
READ 1
a
READ 2
b
C
B
ACCEPT
Δ
POP
Δ
READ 3
S
S
A
PUSH B
PUSH B
PUSH C
PUSH S
PUSH A
PUSH C

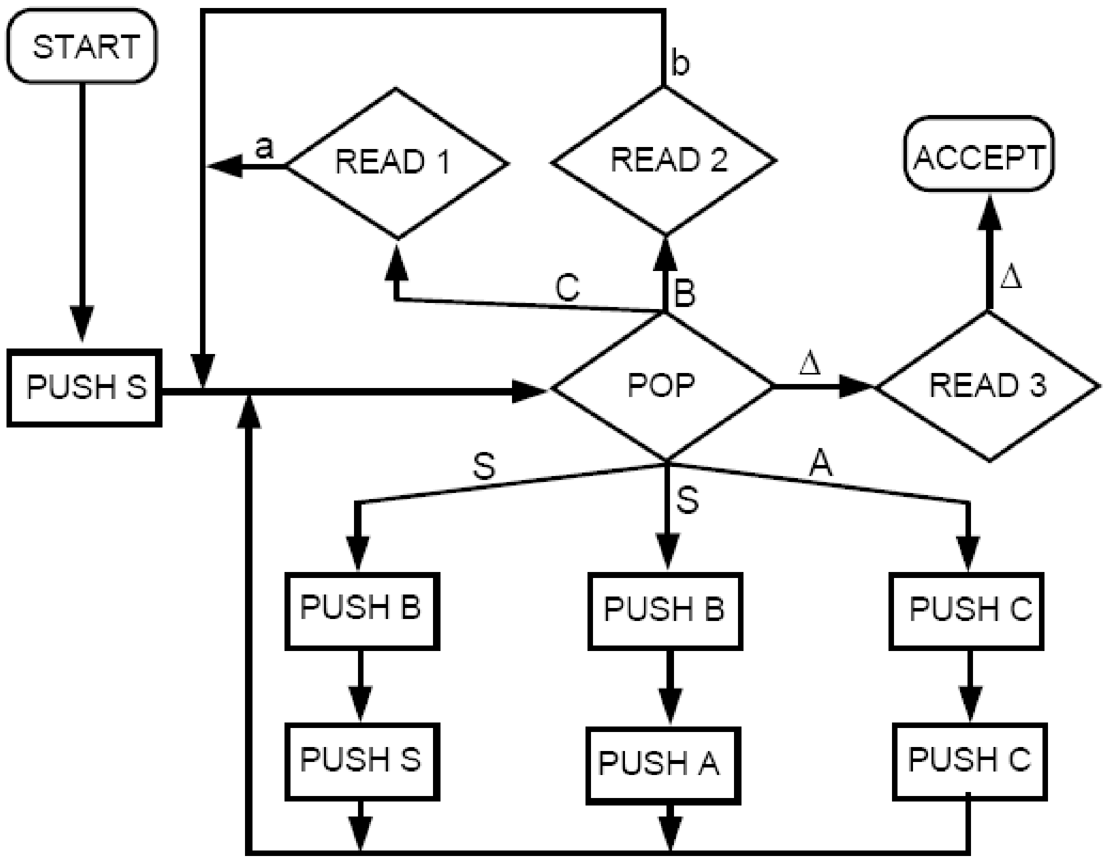
- We begin by pushing S onto the top of the STACK.
- We then enter the central POP state. Two things are possible when we pop the top of the STACK:
  - We either replace the removed non-terminal with two other non-terminals, thereby simulating a production,
  - Or we go to a READ state, which insists that we must read a specific terminal from the TAPE, or else it crashes.
- To get to ACCEPT, we must have encountered the READ states that wanted to read exactly the letters on the INPUT TAPE.
- We now show that doing this is equivalent to simulating a leftmost derivation of the input string in the given CFG.

Example

- Let’s consider a specific example. Let’s generate the word ***aab*** using leftmost derivation in the given CFG:

Working string	**Production used**
S => AB	S ✉ AB
=> CCB	A ✉ CC
=> aCB	C ✉ a
=> aaB	C ✉ a
= aab	B ✉ b

- Let us run this word (***aab***) on the proposed PDA, following the same sequence of productions in the leftmost derivation above.



- We begin at START

STACK	TAPE
Δ	$aab\Delta$

- We push the symbol S on the STACK

STACK	TAPE
S	$aab\Delta$

- We then go to POP state. The first production we must simulate is
  S ✉ AB. So, we POP S and then PUSH B and PUSH A:

STACK	TAPE
AB	$aab\Delta$

- We go back to POP. We now simulate A ✉ CC by popping A and do PUSH C and PUSH C:

STACK	TAPE
CCB	$aab\Delta$

- Again, we go back to POP. This time, we must simulate C ✉ a by poping C and reading a from the TAPE:

STACK	TAPE
CB	$aab\Delta$

- We simulate another C ✉ a:

STACK	TAPE
B	$aab\Delta$

- We now re-enter the POP state and simulate the last production. B → b. We POP B and READ b from the

STACK	TAPE
Δ	aabΔ

- At this point the STACK is empty, and the blank Δ is the only thing we can read next from the TAPE.
- Hence, we follow the path

  POP Δ→ READ3 Δ→ ACCEPT

- So, the word ***aab*** is accepted by the PDA.

- It should also be clear that if any input string reaches the ACCEPT state in the PDA, that string must have got there by having each of its letters read via simulating the Chomsky production of the form

  Nonterminal → terminal

- This means that we have necessarily formed a complete leftmost derivation of this word through CFG productions with no nonterminals left over in the STACK. Therefore, every word accepted by this PDA is in the language generated by the CFG.

- We are now ready to present the algorithm to construct a PDA from a given CFG.

Algorithm

Given a CFG in CNF as follows:

X1 → X2X3

X1 → X3X4

X2 → X2X2

…

X3 → a

X4 → a

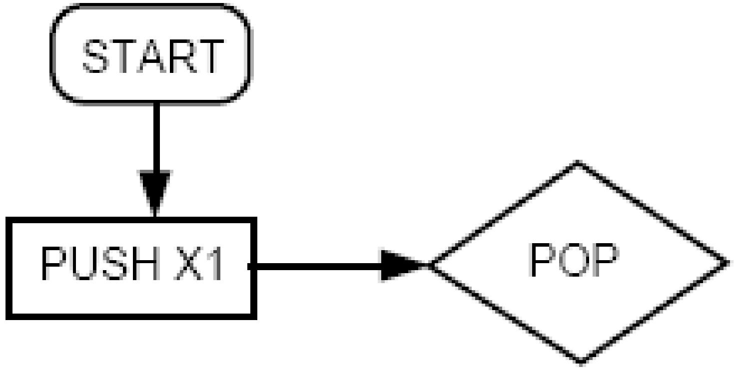
X5 → b

…

where the start symbol S = X1 and the other non-terminals are X2, X3, …

Algorithm (contd.)

- We build the corresponding PDA as follows:
  We begin with

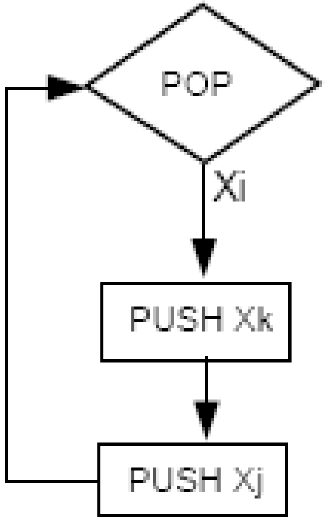


Algorithm (contd.)

- For each production of the form

  Xi ✉ XjXk

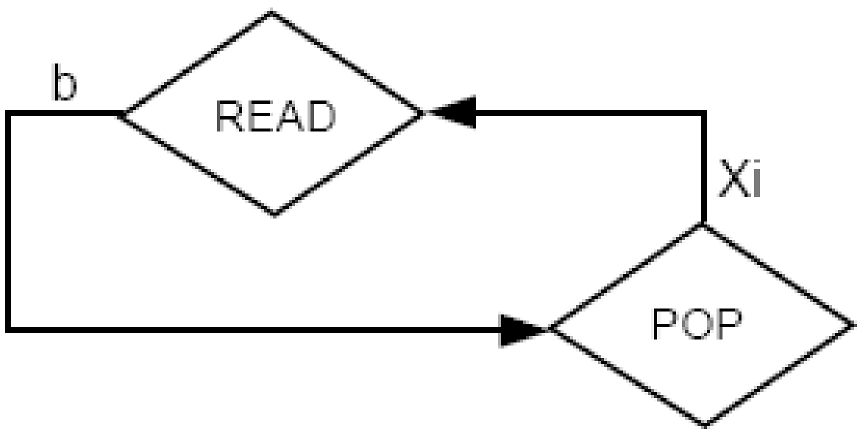
- We include this circuit from the POP state back to itself: PUSHloop fragment



Algorithm (contd.)

- For each production of the form

  Xi 🖂 b
- We include this circuit: READ-loop fragment



Algorithm (contd.)

- When the STACK is empty, which means that we have converted our last non-terminal to a terminal and the terminals have matched the INPUT TAPE, we add this path:

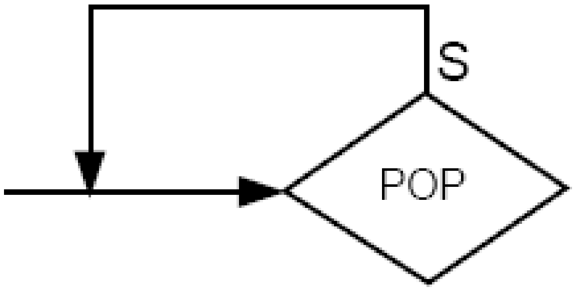
Algorithm contd.

- From the reasons and example above, we know that all words generated by the given CFG will be accepted by the PDA, and all words accepted by this PDA will have leftmost derivations in the given CFG.

- At the beginning we assumed that the CFG was in CNF. But there are some CFLs that cannot be put into CNF. These are the languages that include the word λ.

Algorithm contd.

- In this case, we can convert all productions into CNF and construct the PDA as described above. In addition, we must also include λ. This can be done by adding a simple circuit at the POP:



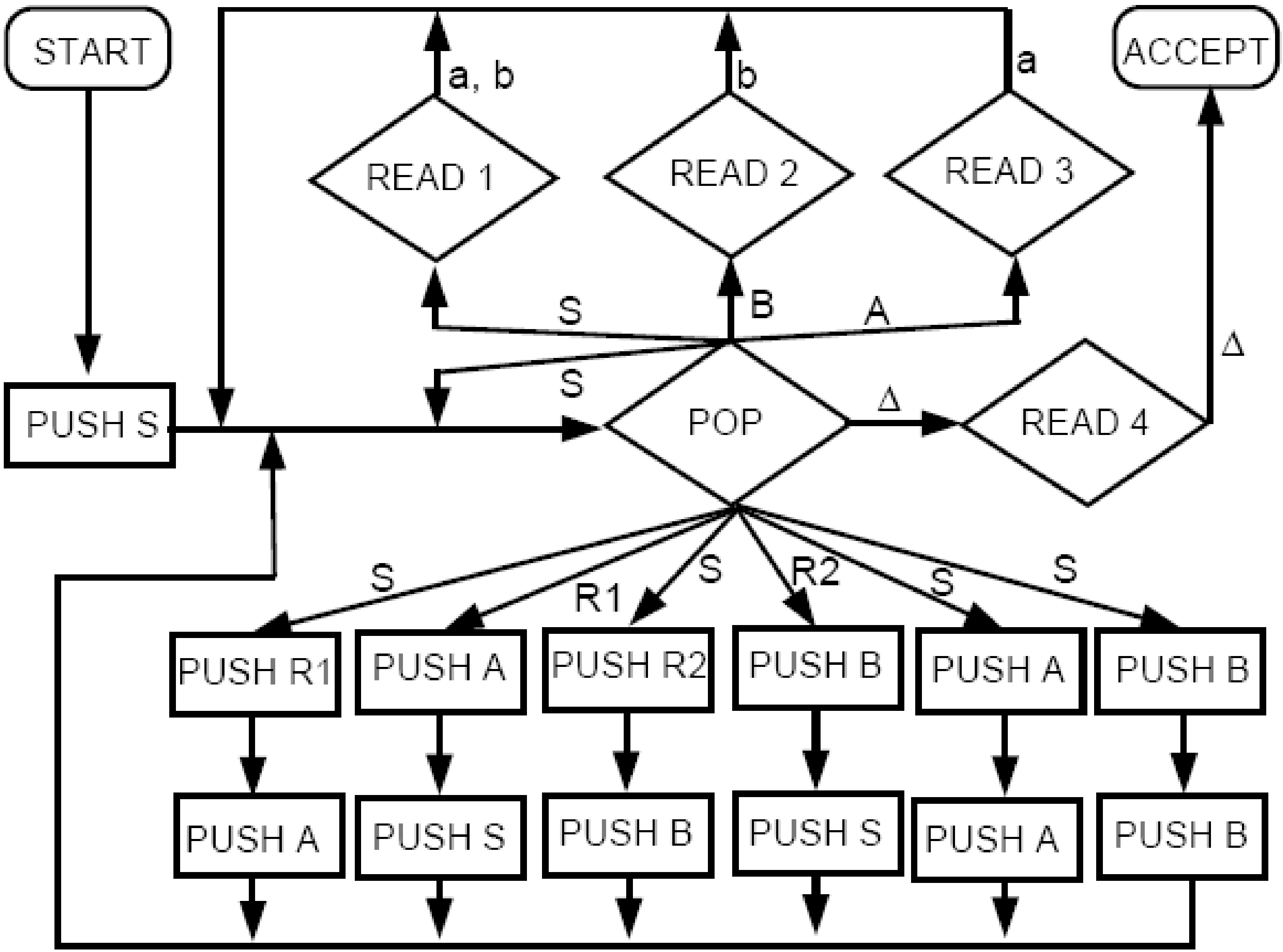
- This kills the non-terminal S without replacing it with anything. So, the next time we enter the POP, we get a blank and can proceed to accept the word.

Example

- The language PALINDROME (including λ) can be generated by the following CFG in CNF (plus one λ-production):

  S → AR1 | BR2 | AA | BB | a | b| λ
  R1 → SA
  R2 → SB
  A → a
  B → b

- Using the algorithm above, we build the following PDA that accepts exactly the same language:


START
ACCEPT
a, b
b
a
READ 1
READ 2
READ 3
S
B
A
Δ
S
PUSH S
POP
Δ
READ 4
S
R1
S
R2
S
S
PUSH R1
PUSH A
PUSH R2
PUSH B
PUSH A
PUSH B
PUSH A
PUSH S
PUSH B
PUSH S
PUSH A
PUSH B

- Theorems 30 and 31 together prove that **the set of all languages accepted by PDAs is the same as the set of all languages generated by CFGs.**