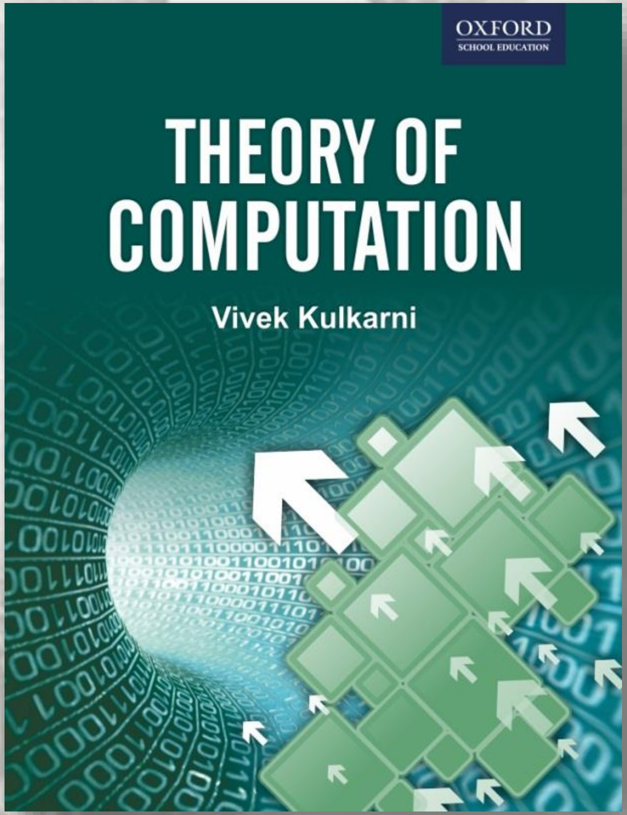
OXFORD
HIGHER EDUCATION

THEORY OF COMPUTATION

Vivek Kulkarni

Slides for Faculty Assistance



Chapter 6

Pushdown Stack Memory Machine

Author: Vivek Kulkarni

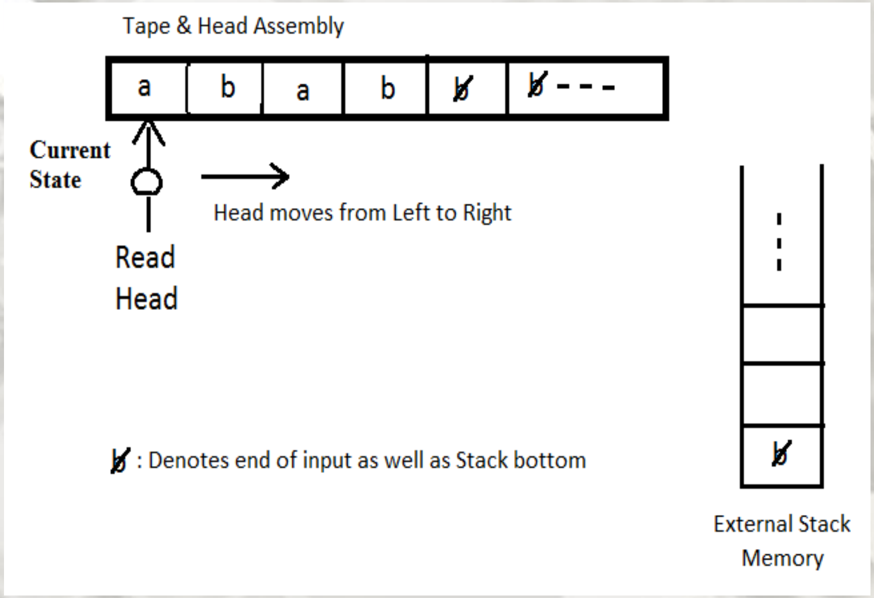
vivek_Kulkarni@yahoo.com

Outline

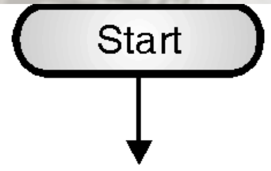
- Following topics are covered in the slides:
  - Formal model of pushdown stack-memory machine, called pushdown automaton (PDA).
  - Difference between deterministic PDA (DPDA) and non-deterministic PDA (NPDA), and their relative computational powers.
  - Equivalence between PDA and context-free grammars (CFGs).
  - Acceptance of a CFL by an empty stack against acceptance by a final state.
  - Application of Chomsky normal form (CNF) for controlling stack growth.
  - Closure properties of context-free languages (CFLs).

Elements of Pushdown Stack Memory Machine

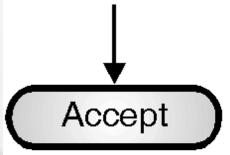
- An input alphabet, Σ, i.e., finite set of input symbols.
- An unbounded input tape, bounded only by the input length .
- An alphabet of stack symbols (Γ).
- A pushdown stack. Initially the stack is empty and is assumed to contain a blank character at the bottom
- Start, accept, and reject indicators.
- Branching state: read.
- Stack operations, namely push and pop.


Tape & Head Assembly
a
b
a
b
ƀ
ƀ - - -
Current State
Head moves from Left to Right
Read Head
ƀ : Denotes end of input as well as Stack bottom
ƀ
External Stack Memory

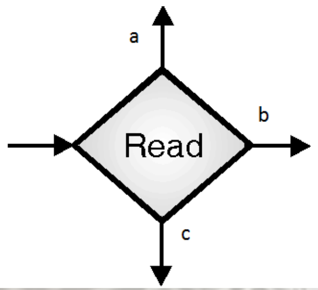
Pictorial Representation of PDM Elements



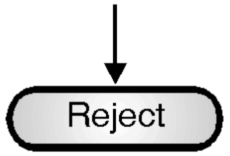
Start the Process



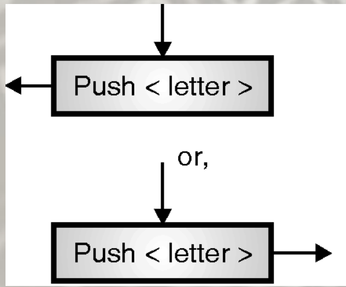
Accept halt



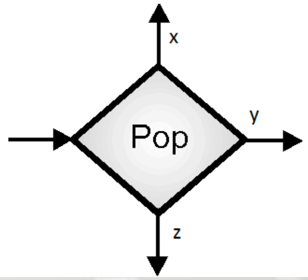
Read the input symbol from tape.
(This is a conditional block and
depending on the symbol read,
it performs different operations)



Reject halt



Push a < letter > i.e. some symbol
on to stack and proceed.



Pop a symbol from the stack.
(It is also conditional like read block)

Pushdown Automata (PDA)

- Pushdown automaton (plural: automata; abbreviated as PDA) is the mathematical model (formalism) of the PDM.
- **Formal definition:** A pushdown automaton *M* is denoted as:

  M = {Q, Σ, Γ, δ , q_0, Z_0, F), where:

  Q: Finite set of states

  Σ : Input alphabet (input strings are composed of symbols from)

  Γ: Stack alphabet

  q_0: Initial state q_0, which is a member of Q

  Z_0: Z_0, which is a member of Γ, is a particular stack symbol called the start symbol. It indicates the bottom of the stack, and is considered as (blank character) .

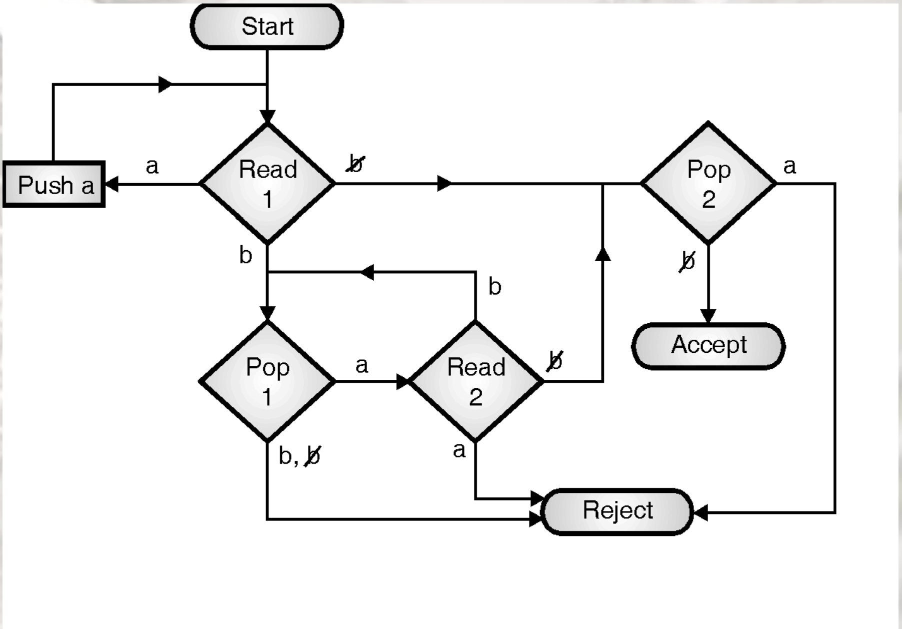
  F: Set of final states, $F \subseteq Q$

  δ: $Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$ (This defines the transition function δ for a *deterministic PDA*, or ***DPDA***. The transition function for a non-deterministic PDA is different, and is stated later)
- For a *non-deterministic PDA* or ***NPDA***, the transition function δ is defined as follows:

  δ: $Q \times \{\Sigma \cup (\in)\} \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma^*$

DPDA Example

- DPDA accepting the language $L = \{a^n b^n \mid n \geq 0\}$
- Algorithm:
  - Keep on pushing all *a*'s onto the stack till the machine reads the first *b*.
  - Each time it reads *b*, it pops one *a* from the stack.
  - If the number of *a*'s are equal to the number of *b*'s, then at the end of the input string, i.e., when blank character is read from the tape, the top of the stack should also contain the blank character , indicating that the stack is empty. The string is accepted only in this case; else it is rejected.



$Read_1$	: Keep on pushing 'a's till you get first 'b'
Pop_1	: Pop an 'a' when you read one 'b'
$Read_2$	: When you get '$\not b$' while reading 'b's go to "Pop_2"
Pop_2	: Input is finished, so check whether stack is empty or not. If empty goto "Accept"

Acceptance of CFL by PDA

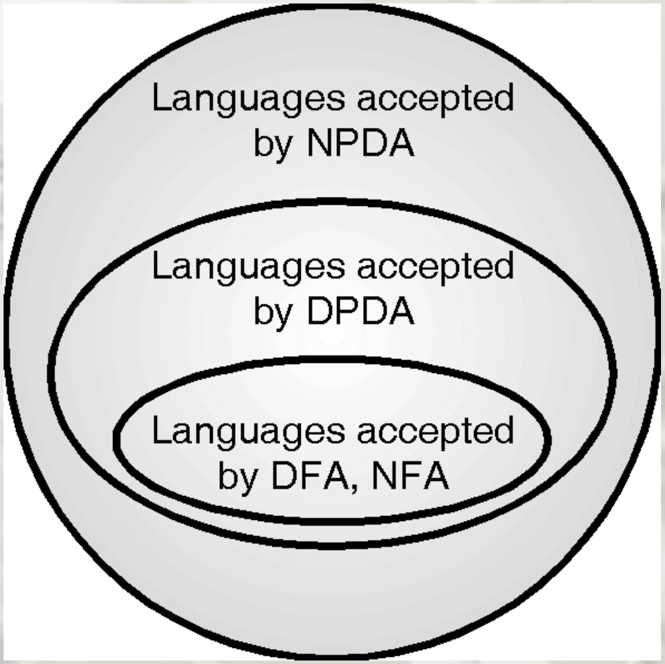
- **Acceptance of CFL by Empty Stack**
  - A string *w* is said to be accepted by a PDA, if:

    $(q_0, w, Z_0) \vdash^* (p, \in, \in)$, where *p* indicates the state after reading the entire input while $(\in, \in)$ indicate respectively that there is no input symbol remaining to be read and stack is empty.
- **Acceptance of CFL by final state**
  - A string w is said to be accepted by a PDA, if:

    (q0, w, Z0) $\vdash^*$ $(p, \in, \gamma)$, where, p is a member of F, which is a designated set of final states and γ is any stack string.

DPDA vs NPDA

- For every NPDA, there may not exist an equivalent DPDA.
- The NPDA can accept any CFL, while DPDA is a special case of NPDA that accepts only a subset of the CFLs accepted by the NPDA. Thus, DPDA is less powerful than NPDA.
- For every NFA there exists an equivalent DFA accepting the same regular language. Hence, their power are same.

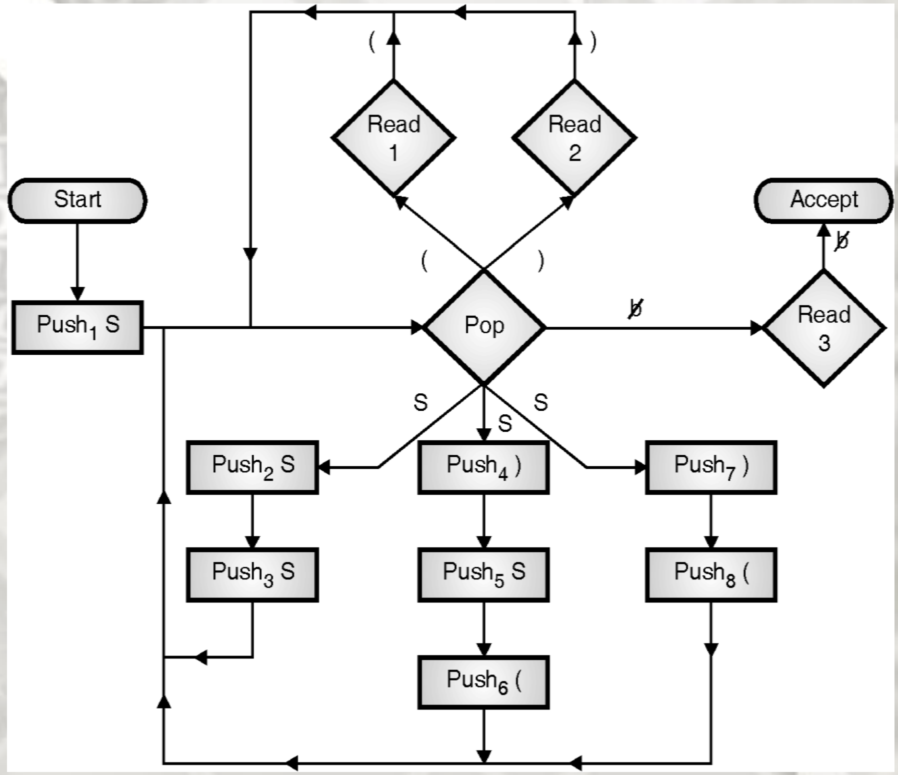

Languages accepted by NPDA
Languages accepted by DPDA
Languages accepted by DFA, NFA

Equivalence of CFG and PDA

- Example PDA that accepts the language generated by the CFG:

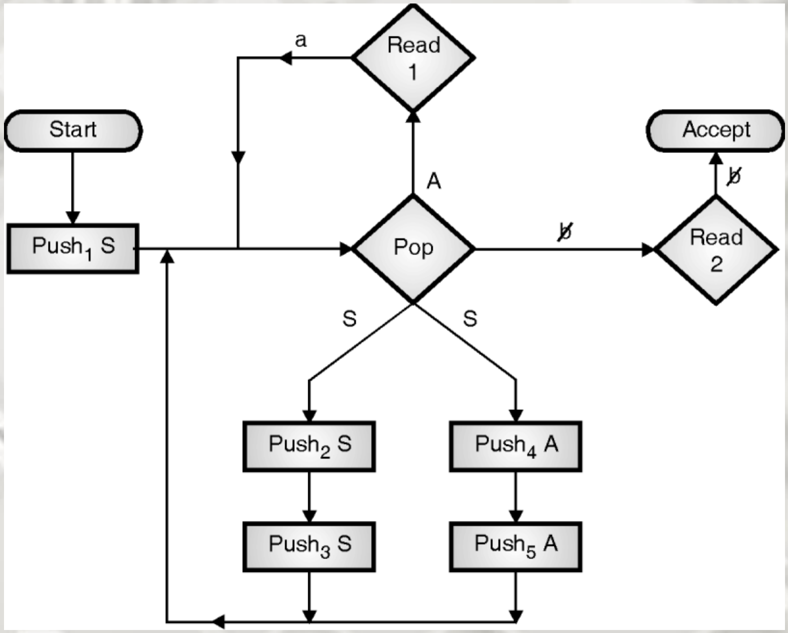
  $S \rightarrow SS \mid (S) \mid (\ )$

- The process of obtaining the PDA is based on the leftmost derivation.
- if the provided grammar rule is of the form '$A \rightarrow \alpha$', then A is replaced (popped) by α (pushed) on the stack.
- Whenever a terminal symbol is generated on the stack, an input symbol is read to match it.



PDA construction and CNF

- CNF has only two types of rules:
  - $S \rightarrow AB$
  - $S \rightarrow a$
- Benefits of using CNF:
  - Terminals are not pushed onto the stack.
  - At most, two non-terminal symbols are pushed onto the stack at every step in the derivation process.
  - This in turn helps control the stack growth
- PDA for example grammar in CNF, $S \rightarrow SS \mid AA$, $A \rightarrow a$



Closure Properties of CFLs

- Context-free languages (CFLs), just like regular languages, are also closed under:
  - **Union**: $S \rightarrow S1 \mid S2$; S1 and S2 are start symbols of the two grammars which are combined into S using union operation.
  - **Concatenation**: $S \rightarrow S1\ S2$; S1 and S2 are start symbols on the two grammars which are combined into S using concatenation operation.
  - **Kleene closure**: $S1 \rightarrow S\ S1 \mid \in$; S1 is repeatedly concatenated to itself to define S.