Formal Language and Automata Theory

Chapter six

Pushdown Automata

Pushdown Automata

- Is there some way to build an automaton that can **recognize** the context-free languages?

Pushdown Automata

- A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar.
- A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

Con’t

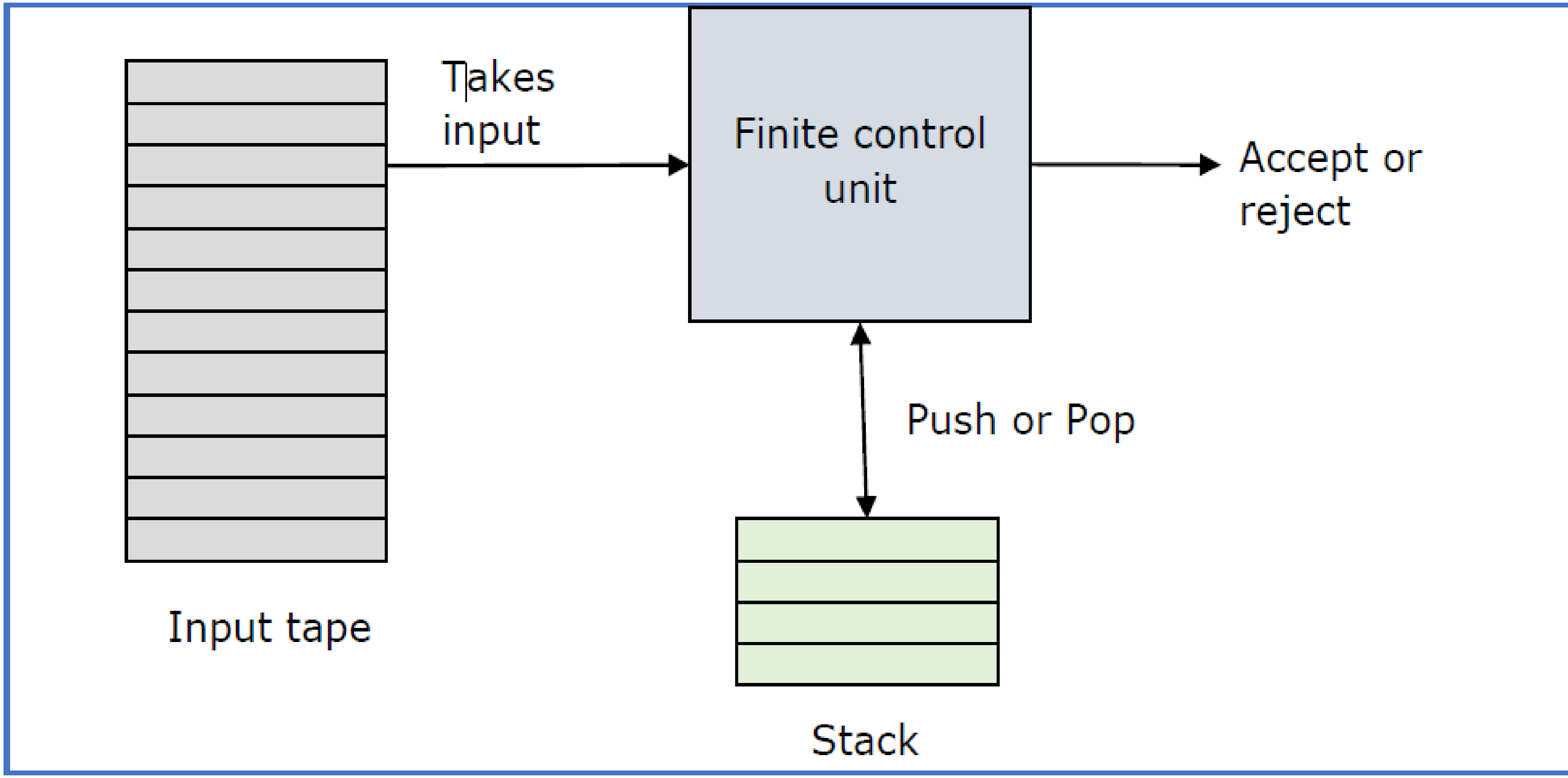
- Basically a pushdown automaton is:

  "Finite state machine" + "a stack“

- A pushdown automaton has three components:
  - an input tape,
  - a control unit, and
  - a stack with infinite size.
- The stack head scans the top symbol of the stack.
- A stack does two operations:
  - **Push**: a new symbol is added at the top.
  - **Pop**: the top symbol is read and removed.

Con’t

- A PDA may or may not read an input symbol, but it has to read the top of the stack in every transition.



Con’t

Informal Definition of Acceptance

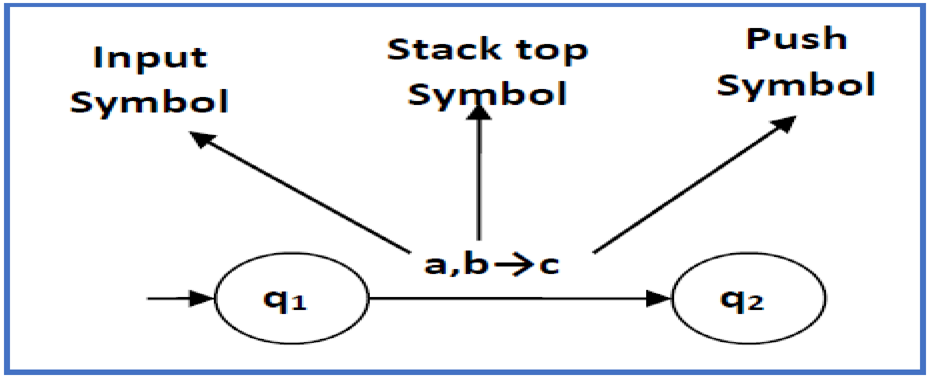
- A pushdown automation accepts if, after reading the entire input, it ends in an accept state
  - Sometimes: (with an empty stack)
- Each move of the control unit is determined by the current input symbol as well as by the symbol currently on top of the stack.
- The result of the move is a new state of the control unit and a change in the top of the stack.

Con’t

- Only the top of the stack is visible at any point in time.
- New symbols may be **pushed** onto the stack, which cover up the old stack top.
- The top symbol of the stack may be **popped**, exposing the symbol below it.

Con't

- The following diagram shows a transition in a PDA from a state q_1 to state q_2, labeled as a,b $\longrightarrow$ c :



- This means at state **q1**, if we encounter an input string '**a**' and top symbol of the stack is '**b**', then we pop '**b**', push '**c**' on top of the stack and move to state $\mathbf{q_2}$.

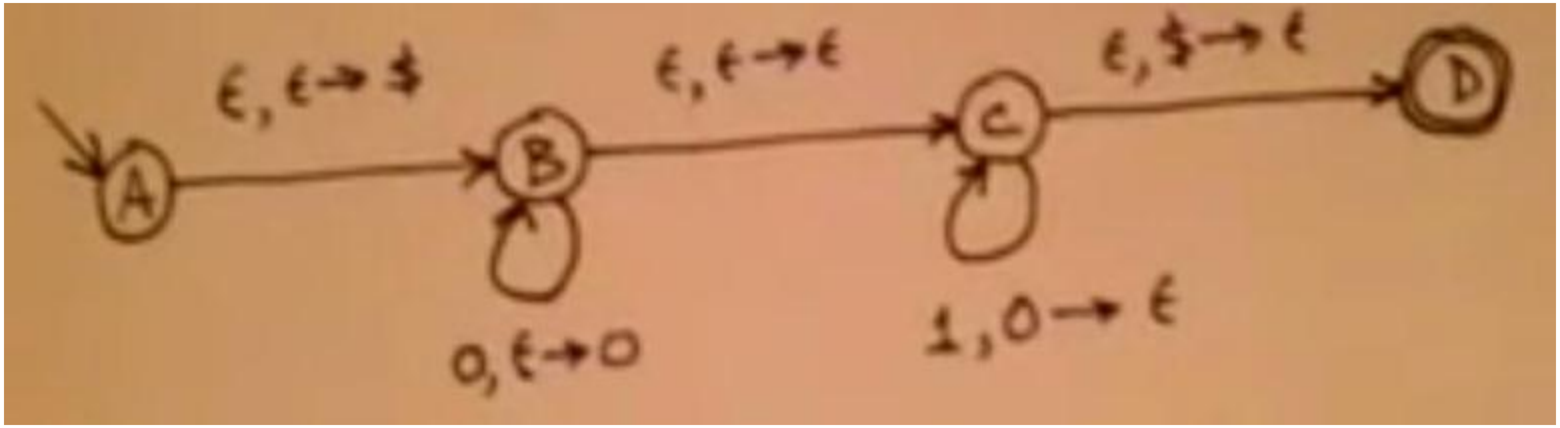
Con’t

- **Example:**

{0^n1^n / $n\geq0$**}**

Σ={0,1} input alphabet

Ѓ={ $, 0} Stack alphabet



Non deterministic Pushdown Automata

- In a PDA, if there are multiple nondeterministic choices, you **cannot** treat the machine as being in multiple states at once.
- Each state might have its own stack associated with it.
- Instead, there are multiple parallel copies of the machine running at once, each of which has its own stack.

Con’t

- A PDA can be formally described as a 7-tuple (Q, Σ, S, δ, q_0, I, F):
  - **Q** is the finite number of states
  - **Σ** is input alphabet
  - **S** is stack symbols
  - **δ** is the transition function: Q × (Σ∪{ε}) × S × Q × S*
  - $\mathbf{q_0}$ is the initial state ($q_0 \in Q$)
  - **Ѓ** is the initial stack top symbol (**Ѓ** ∈ S)
  - **F** is a set of accepting states (F ∈ Q)

Con't

- Example: palindrome
- A **palindrome** is a string that is the same forwards and backwards.

  Was it a cat I saw
- Let $\Sigma = \{\mathbf{0}, \mathbf{1}\}$ and consider the language
- $PALINDROME = \{ w \in \Sigma^* \mid w \text{ is a palindrome } \}$.
- ***Idea***: Push the first half of the symbols on to the stack, then verify that the second half of the symbols match.

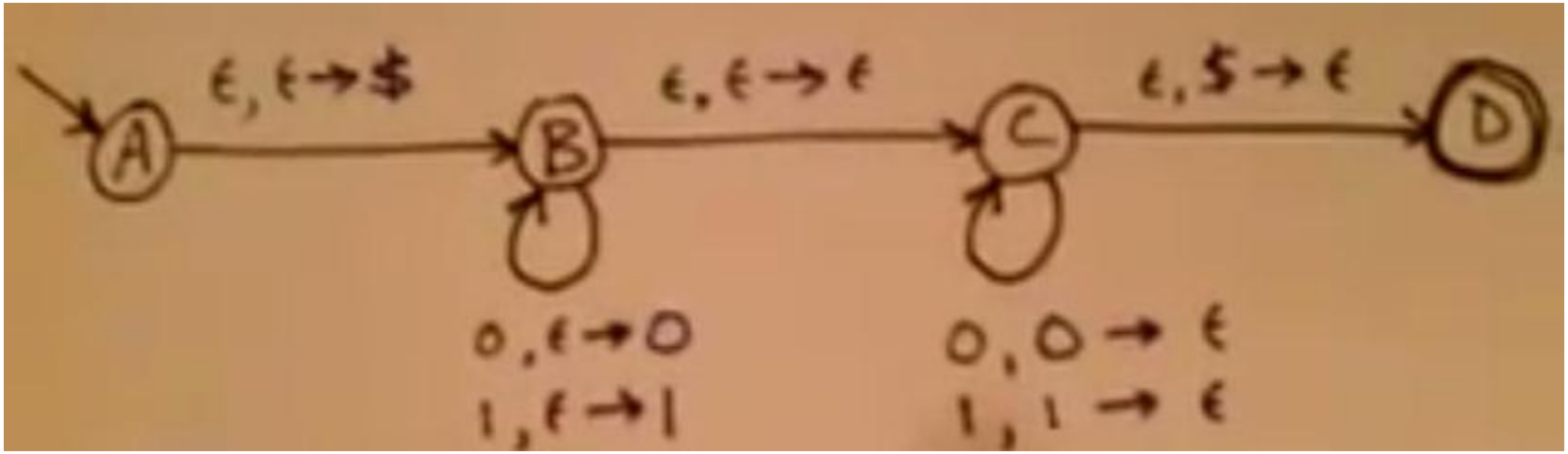
Con’t

Given Grammar

$S \rightarrow 0S0$

$S \rightarrow 1S1$

$S \rightarrow \varepsilon$



Pushdown Automata and Context-Free Languages

From CFGs to PDAs

- ***Theorem:*** If *G* is a CFG for a language *L*, then there exists a PDA for *L* as well.
- **Idea:** Build a PDA that simulates expanding out the CFG from the start symbol to some particular string.
- Stack holds the part of the string we haven't matched yet.

Deterministic push down automata

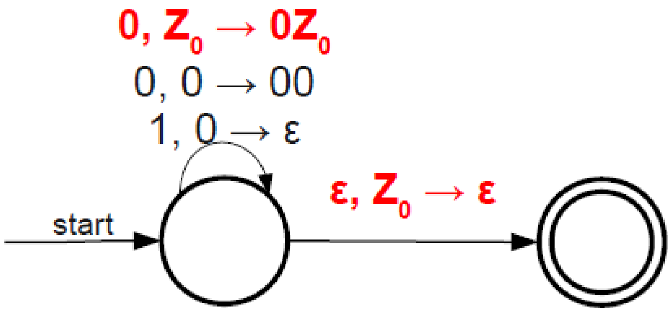
- A **deterministic pushdown automaton** is a PDA with the extra property that For each state in the PDA, and for any combination of a current input symbol and a current stack symbol, there is **at most** one transition defined.
- In other words, there is ***at most*** one legal sequence of transitions that can be followed for any input.

Con’t

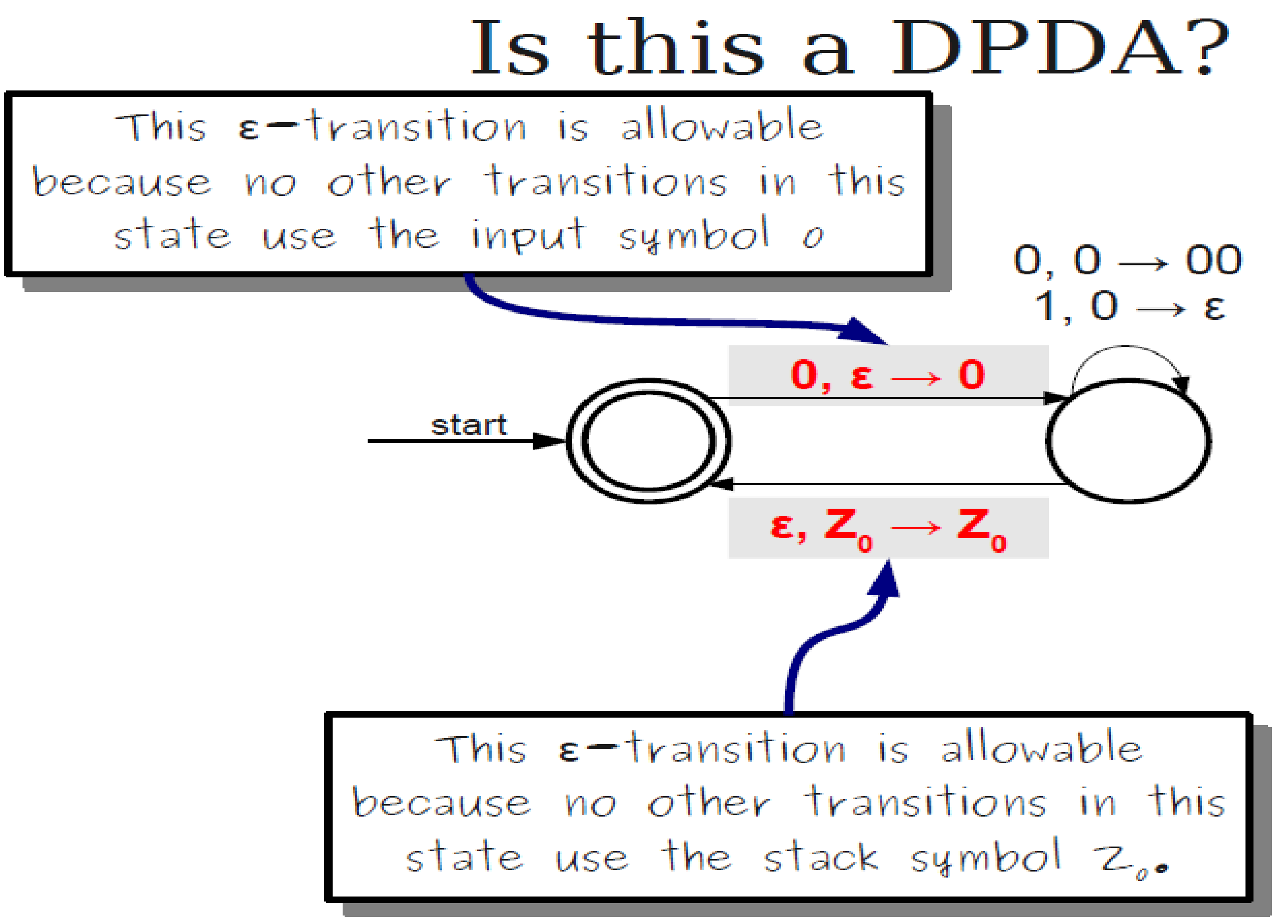
- This does ***not*** preclude ε-transitions, as long as there is never a conflict between following the ε-transition or some other transition.
- However, there can be ***at most*** one ε-transition that could be followed at any one time.
- This does ***not*** preclude the automaton “dying” from having no transitions defined; DPDAs can have undefined transitions.

Con’t

Is this a DPDA?



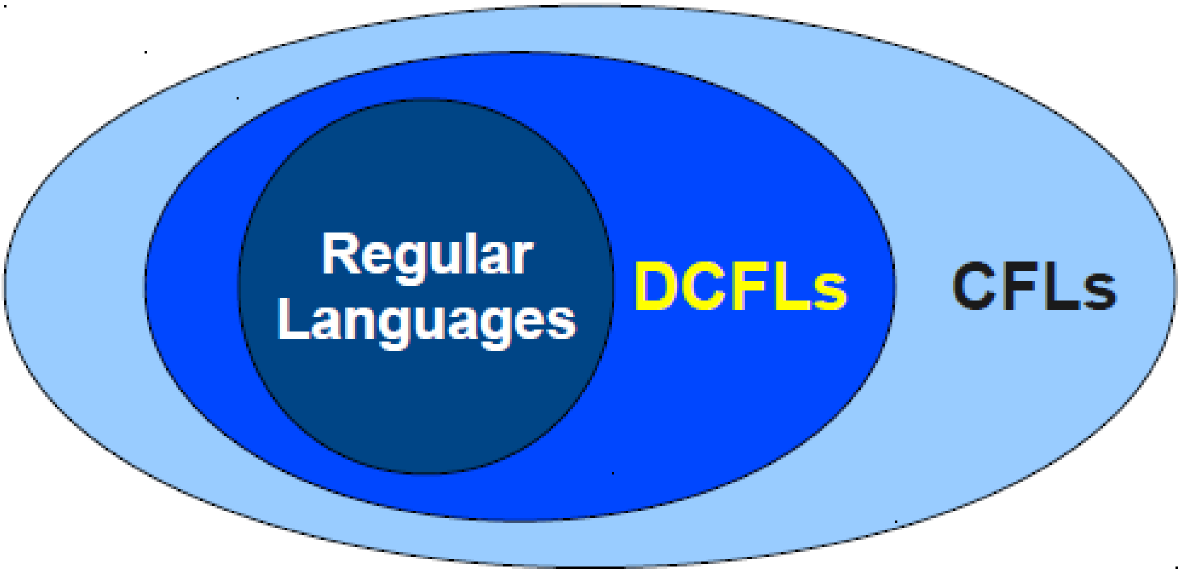
Con’t


Is this a DPDA?
This ε−transition is allowable because no other transitions in this state use the input symbol 0
0, 0 → 00
1, 0 → ε
0, ε → 0
start
ε, Z0 → Z0
This ε−transition is allowable because no other transitions in this state use the stack symbol Z0.

Deterministic CFLs

- A context-free language L is called a **deterministic context-free language** (DCFL) if there is some DPDA that recognizes L.
- Not all CFLs are DCFLs, though many important ones are.
- Balanced parentheses, most programming languages, etc.

Con’t



Con’t

- NPDAs are **more powerful** than DPDAs.
  - when dealing with PDAs, there are CFLs that can be recognized by NPDAs that **cannot** be recognized by DPDAs.

Assignment

- Construct npda's that accept the following regular languages'
  A. $L_1=L(aaa^*b)$
  B. $L_2=L(baa^*ba)$