PUSHDOWN AUTOMATA & TURING MACHINE

Dr. Nadeem Akhtar

Assistant Professor
Department of Computer Science & IT
The Islamia University of Bahawalpur

PhD – Formal methods in Software engineering
IRISA – University of South Brittany – FRANCE.

PUSHDOWN AUTOMATA (1)

- A computational model like nondeterministic finite automata but have an extra component called a **stack**.
- The stack provides additional memory beyond the finite amount available in the control.
- The stack allows pushdown automata to recognize some non-regular languages.

PUSHDOWN AUTOMATA (2)

Pushdown automata are equivalent in power to **Context-Free Grammars**.

- A context-free grammar generates a language whereas a Pushdown automaton recognizes it.

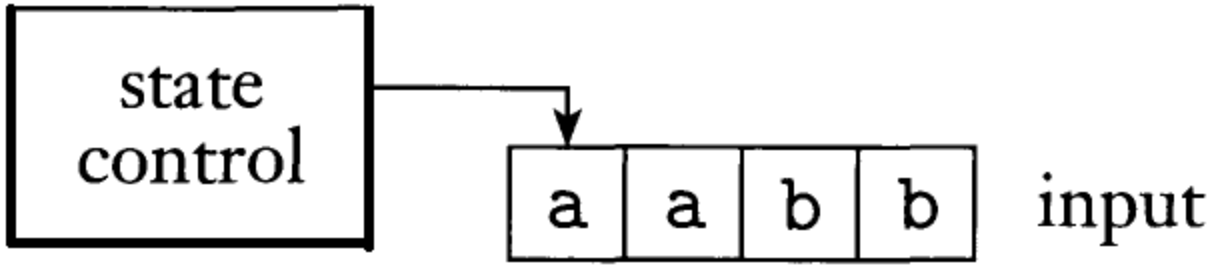
PUSHDOWN AUTOMATA (3)

- Certain languages are more easily described in terms of ***generators***, whereas others are more easily described in terms of ***recognizers***.

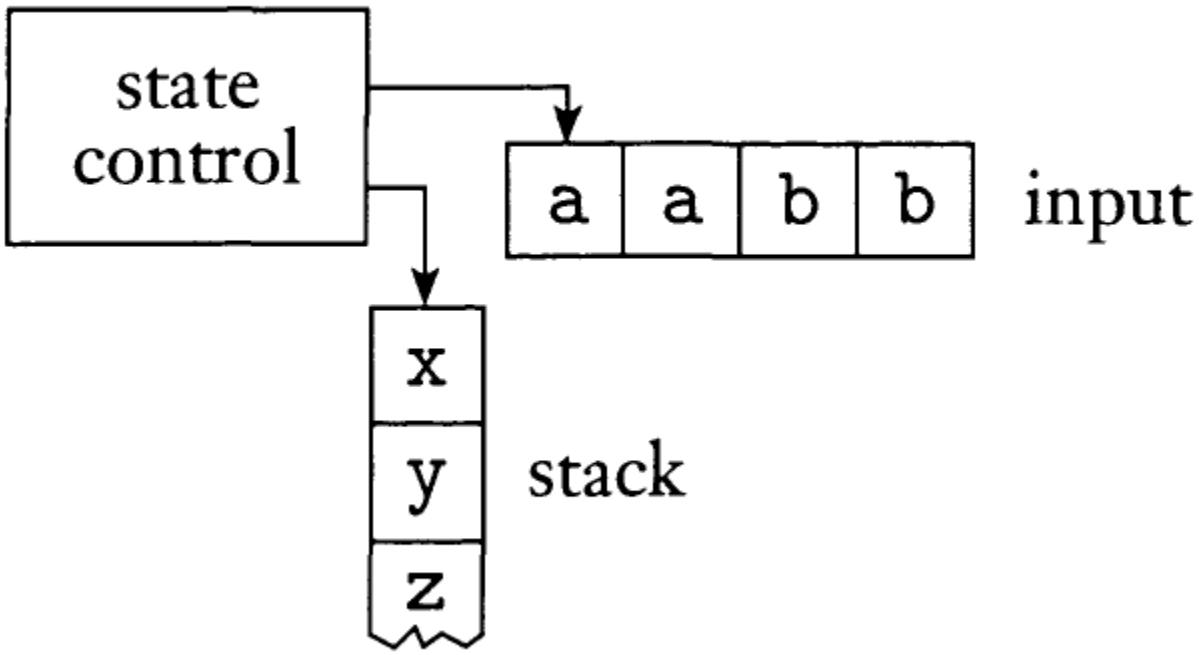
- In order to prove that a language is context free. We can give either a context-free grammar generating it or a push-down automaton recognizing it.

Schematic of a Finite Automaton

Schematic representation of a finite automaton
- The control represents the **States** and **transition function**
- The tape contains the **input string**, and the **arrow represents the input head**, pointing at the next input symbol to be read.



Schematic of a Pushdown Automaton



PUSHDOWN AUTOMATA (4)

A PushDown Automaton (PDA) can write symbols on the stack and read them back later.

Writing a symbol "***Pushes down***" all other symbols on the stack. At any time only the symbol on the ***top*** of the stack can be read and removed.

Writing a symbol on the stack is often referred to as ***pushing the symbol***, and removing a symbol is referred to as ***popping*** it.

Only top access

Stack is a "last in, first out" storage device

PUSHDOWN AUTOMATA (5)

A stack is valuable because it can hold an unlimited amount of information.

A finite automaton is unable to recognize the language $\{\mathbf{0}^{\boldsymbol{n}}\mathbf{1}^{\boldsymbol{n}} \mid \boldsymbol{n} > \mathbf{0}\}$ because it cannot store very large numbers in its finite memory.

A PDA is able to recognize this language because it can use its stack to store the number of 0s it has seen.

PUSHDOWN AUTOMATA (6)

Informal description:

Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read.

If reading the input is finished exactly when the stack becomes empty of 0s, accept the input.

If the stack becomes empty while is remain or if the 1s are finished while the stack still contains 0s or if any 0s appear in the input following 1s, reject the input.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

Input alphabet Σ and a stack alphabet Γ

The transition function, which describes its behavior.

$$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\} \quad \text{and}$$
$$\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}.$$

The domain of the transition function is $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$

The current state, next input symbol read, and top symbol of the stack determine the next move of a pushdown automaton.

Symbol can also be ε, causing the machine to move without reading a symbol from the input or without reading a symbol from the stack.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

A pushdown automaton is a 6-tuple (**Q, Σ, Γ, δ, q_o, F**), where **Q**, **Σ, Γ, and F** are all finite sets, and

1. **Q** is the set of states
2. **Σ** is the input alphabet
3. **Γ** is the stack alphabet
4. **δ:** $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ is the transition function
5. $q_o \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

A pushdown automaton M = $(\mathbf{Q, \Sigma, \Gamma, \delta, q_o, F})$ computes as follows.

- It accepts input *w* if *w* can be written as $\boldsymbol{w = w_1 w_2 \ldots w_m}$, where each $\boldsymbol{w_i \in \Sigma}$ and
- Sequences of states $\boldsymbol{r_o, r_1, \ldots, r_m \in Q}$ and
- Strings $\boldsymbol{s_o, s_1, \ldots, s_m \in \Gamma^*}$ exist that satisfy the following three conditions.

(i.e. the strings $\boldsymbol{s_i}$ represent the sequence of stack contents that ***M*** has on the accepting branch of the computation)

FORMAL DEFINITION OF A PUSHDOWN AUTOMATON

1. $r_o = q_o$ and $s_o = \varepsilon$. This condition signifies that M starts out properly, in the start state and with an empty stack.

2. For $i = 0,\ldots,m - 1$, we have $(r_{i+1},b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+l} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$. This condition states that M moves properly according to the state, stack, and next input symbol.

3. $r_m \in F$. This condition states that an accept state occurs at the input end.

Example

- Formal description of the PDA that recognizes the language $\{0^n1^n \mid n \geq 0\}$.

Let M_1 be $(\boldsymbol{Q, \Sigma, \Gamma, \delta, q_o, F})$, where

$\mathbf{Q} = \{\boldsymbol{q_1}, \boldsymbol{q_2}, \boldsymbol{q_3}, \boldsymbol{q_4}\}$,

$\boldsymbol{\Sigma} = \{0, 1\}$,

$\boldsymbol{\Gamma} = \{0, \$\}$,

$\boldsymbol{F} = \{\boldsymbol{q_1}, \boldsymbol{q_4}\}$, and

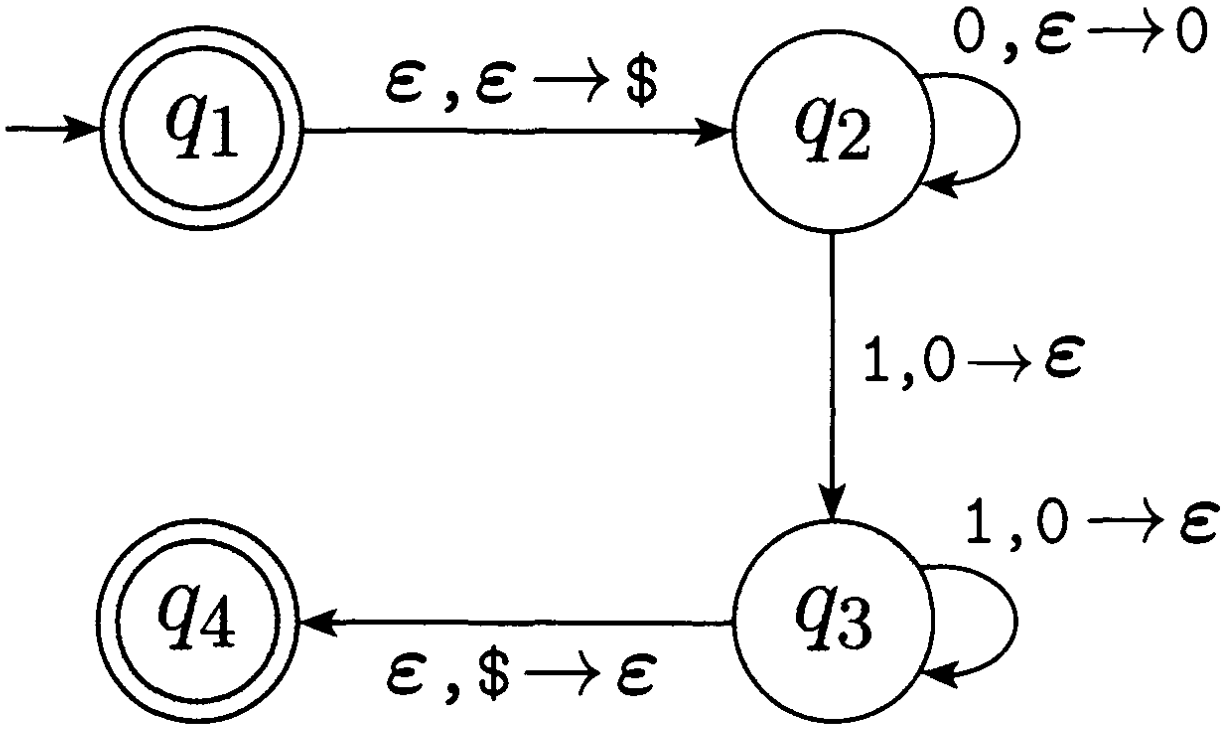
- δ is given by the following table, wherein blank entries signify ϕ.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

- We write "**a, b → c**" to signify that when the machine is reading an a from the input it may replace the symbol b on the top of the stack with a c.

- Any of a, b, and c may be **ε.** If a is ε, the machine may make this transition without reading any symbol from the input.
- If b is ε, the machine may make this transition without reading and popping any symbol from the stack.
- If c is ε, the machine does not write any symbol on the stack when going along this transition.

State diagram for the PDA M_1 that recognizes $\{0^n1^n \mid n \geq 0\}$



TURING MACHINE (1)

- Turing machine is a powerful model, first proposed by Alan Turing in 1936.
- Similar to a finite automaton but with an unlimited and unrestricted memory, a Turing machine is a much more accurate model of a **general purpose computer**.

- A Turing machine can do everything that a real computer can do.

Turing machine cannot solve certain problems, these problems are beyond the theoretical limits of computation.

TURING MACHINE (2)

- The Turing machine model uses an **infinite tape as its unlimited memory**. It has a tape head that can read and write symbols and move around on the tape.

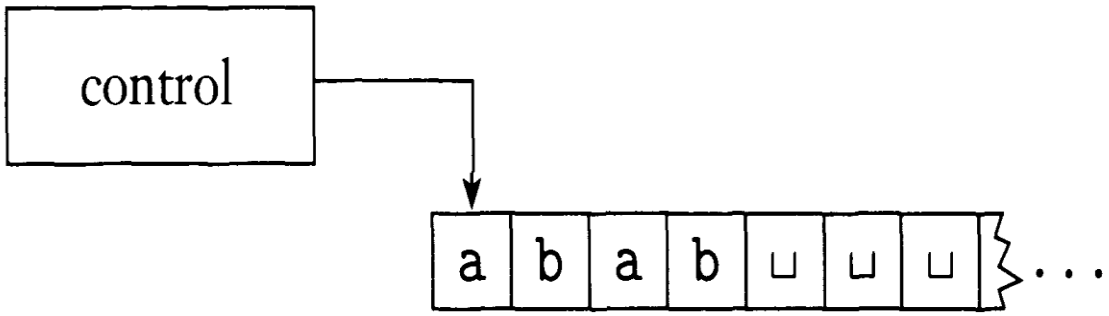
- Initially the tape contains only the input string and is blank everywhere else. If the machine needs to store information, it may write this information on the tape. To read the information that it has written, the machine can move its head back over it.

TURING MACHINE (3)

- The machine continues computing until it decides to produce an output.

The outputs accept and reject are obtained by entering designated accepting and rejecting states. If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting.

SCHEMATIC OF A TURING MACHINE



Differences between Finite Automata and Turing Machines

1. A Turing machine can both write on the tape and read from it.
2. The read-write head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately.

Example

A Turing machine M_1 for testing membership in the language

$$B = \{w\#w \mid w \in \{0,1\}^*\}.$$

We want M_1 to accept if its input is a member of B and to reject otherwise.

The goal is to determine whether the input is a member of B-that is, whether the input comprises two identical strings separated by a # symbol.

The input is too long to remember it all, but we are allowed to move back and forth over the input and make marks on it.

Example

Strategy

To zig-zag to the corresponding places on the two sides of the # and determine whether they match. Place marks on the tape to keep track of which places correspond.

Example

- In summary, M_1's algorithm is as follows:

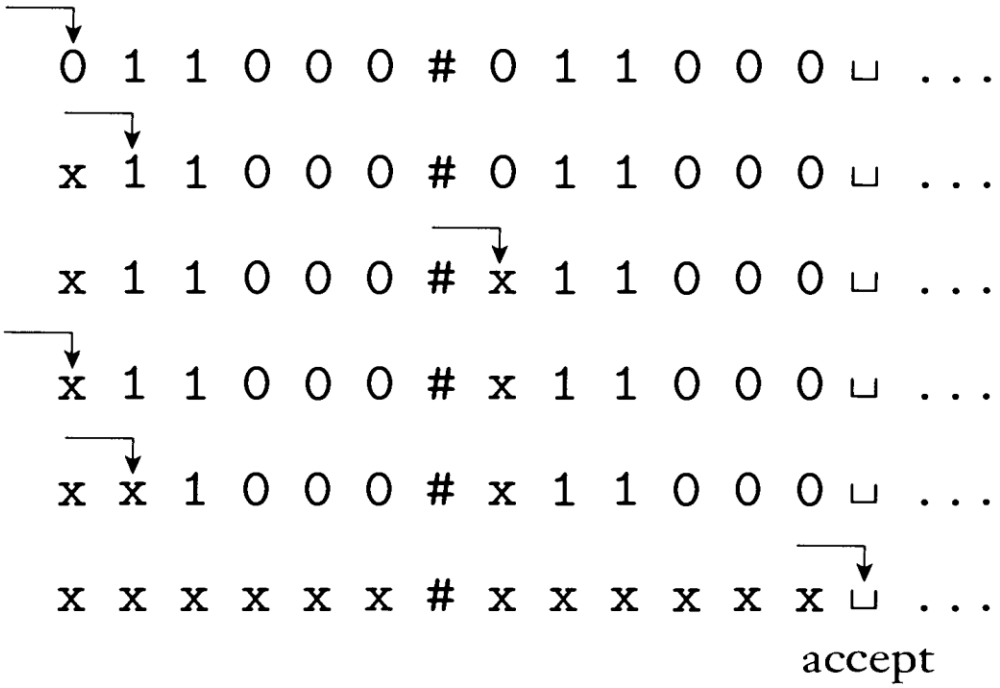
M_1 = "On input string *w*:

1) Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, reject. Cross off symbols as they are checked to keep track of which symbols correspond.

2) When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."

Example

- Snapshots of M_1's tape while it is computing in stages 2 and 3 when started on input 011000#011000.



FORMAL DEFINITION OF A TURING MACHINE

- A Turing machine, δ takes the form:

$$Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}.$$

That is, when the machine is in a certain state q and the head is over a tape square containing a symbol a, and if

$$\delta(q, a) = (r, b, L)$$

the machine writes the symbol b replacing the a, and goes to state r. The third component is either L or R and indicates whether the head moves to the left or right after writing. In this case the L indicates a move to the left.

FORMAL DEFINITION OF A TURING MACHINE

A Turing machine is a 7 -tuple, (**Q, Σ, Γ, δ,** q_0, q_{accept}, q_{reject}), where **Q, Σ, Γ** are all finite sets and

1. **Q** is the set of states,
2. **Σ** is the input alphabet not containing the **blank symbol** □,
3. **Γ** is the tape alphabet, where □ **Є Γ** and **Σ ⊆ Γ**,
4. **δ: Q x Γ → Q x Γ x {L, R}** is the transition function,
5. q_0 Є Q is the start state,
6. q_{accept} Є Q is the accept state, and
7. q_{reject} Є Q is the reject state, where q_{reject} ≠ q_{accept}.

Configuration

- A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, computes as follows.

Initially M receives its input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ on the leftmost n squares of the tape, and the rest of the tape is blank (i.e., filled with blank symbols).

The head starts on the leftmost square of the tape. Note that Σ does not contain the blank symbol, so the first blank appearing on the tape marks the end of the input.

Configuration

Once M has started, the computation proceeds according to the rules described by the transition function.

If M ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates L.

The computation continues until it enters either the accept or reject states at which point it halts. If neither occurs, M goes on forever.

Configuration

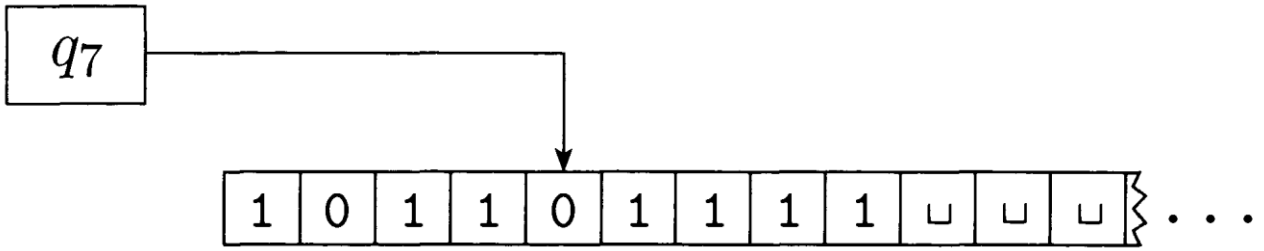
As a Turing machine computes, changes occur in the current state, the current tape contents, and the current head location. A setting of these three items is called a configuration of the Turing machine.

Configurations often are represented in a special way. For a state q and two strings *u* and *v* over the tape alphabet Γ we write *u q v* for the configuration where the current state is *q*, the current tape contents is *uv*, and the current head location is the first symbol of v. The tape contains only blanks following the last symbol of *v*.

A Turing machine with configuration $1011q_701111$

For example,

$1011q_701111$ represents the configuration when the tape is 101101111, the current state is q_7, and the head is currently on the second 0. The following figure depicts a Turing machine with that configuration.



FORMAL DEFINITION OF A TURING MACHINE COMPUTATION

Say that configuration C_1 yields configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single step. We define this notion formally as follows:

Suppose that we have a, b, and c in Γ, as well as u and v in Γ^* and states q_i and q_j. In that case $\boldsymbol{u a\ q_i b v}$ and $\boldsymbol{u\ q_j\ a c v}$ are two configurations. Say that

$$\boldsymbol{u a\ q_i b v} \text{ yields } \boldsymbol{u\ q_j\ a c v}$$

if in the transition function , $\delta\ (q_i, b) = (q_j, c, \mathrm{L})$.

That handles the case where the Turing machine moves leftward.

FORMAL DEFINITION OF A TURING MACHINE COMPUTATION

For a rightward move, say that

$$ua\,q_i\,bv \quad \text{yields} \quad uac\,q_j\,v$$

if

$$\delta\,(q_i,\, b) = (q_j,\, c,\, \mathrm{R})$$

FORMAL DEFINITION OF A TURING MACHINE COMPUTATION

- The start configuration of M on input w is the configuration q_0 w, which indicates that the machine is in the start state q_0 with its head at the leftmost position on the tape.

In an accepting configuration the state of the configuration is q_{accept}

In a rejecting configuration the state of the configuration is q_{reject}

Accepting and rejecting configurations are halting configurations and do not yield further configurations.

FORMAL DEFINITION OF A TURING MACHINE COMPUTATION

The transition function :

$\delta: Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

where Q' is Q without q_{accept} and q_{reject}.

A Turing machine *M accepts* input *w* if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where

1. C_1 is the start configuration of *M* on input *w*,
2. each C_i yields C_{i+1}, and
3. C_k is an accepting configuration.

TURING-RECOGNIZABLE

- The collection of strings that *M* accepts is the ***language of M***, or the ***language recognized by M***, denoted *L(M)*.

- Call a language ***Turing-recognizable*** if some Turing machine recognizes it.

TURING-DECIDABLE

- Turing machines that halt on all inputs; such machines never loop. These machines are called ***deciders*** because they always make a decision to accept or reject.

A decider that recognizes some language also is said to ***decide*** that language.

Call a language Turing-decidable or simply decidable if some Turing machine decides it.

Every decidable language is Turing-recognizable.

Questions and Discussion