Module 03
Models of Computation
Stack Machines
(Push Down Automata)

CS 332 Organization of Programming Languages
Embry-Riddle Aeronautical University
Daytona Beach, FL

M03 Outcomes

At the end of this module you should be able to ...

Given a grammar, justify why it is or is not context free.

State the definition of the Push Down Automata(PDA) and its component parts.

Given a context free language create the PDA for it.

State the acceptance criteria for a PDA.

Given a PDA and a string, demonstrate the state transitions and stack operations for processing the string.

Given a PDA and a string, state whether the PDA accepts or rejects the string.

Provide real world examples of a PDA with justification.

Stack Machines: Introduction (1/2)

- PDA’s have theoretical value.
- We’re still asking “What can a computer compute?”
- The PDA is another mathematical model of computation.
- Pure computation is the mindless manipulation of symbols using formal rules:
    - Mindless – or a computer couldn’t do it!
    - Manipulation of symbols – operators modifying operands
    - Formal rules – require no judgment, opinion, thought. Just do what is says.
- PDAs are a stronger model than the FSM – PDAs have memory.

Stack Machines: Introduction (2/2)

- PDAs have practical value

- All real world programming languages are context free
  - When parsing a statement in a program, the stack records what has been seen so far
- Many systems utilize a stack to schedule tasks
  - Function calls reserve memory on the memory stack, and remove when done
  - Some “intelligent” systems sequence behaviors/actions using a stack of tasks

- CS 332 consider PDAs as <u>language recognizers</u>.
  - PDAs will <u>*accept*</u> anything in the language, and <u>*reject*</u> anything not in language.
  - Each language has it’s own PDA (actually many PDA’s).

The Chomsky Hierarchy (Review)

- For now, define the Chomsky Hierarchy in terms of grammars
- Will describe restrictions on the Left Hand Side (LHS) and Right Hand Side (RHS)
- “Unrestricted” is sometimes called “recursively enumerable” but not covering that in this course – it would take several weeks.

Type	Name	Characteristics of Grammar
Type 3	Regular	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease. Strings derived from right to left, or left to right.
Type 2	Context Free	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease.
Type 1	Context Sensitive	LHS may have terminals and non-terminals. Number of terminals in RHS cannot decrease.
Type 0	Unrestricted	No restrictions

The Chomsky Hierarchy (Review)

- Previously defined the Chomsky Hierarchy in terms of grammars
- Can also define in terms of model of computation
- Will discuss why these models and language match as we go through

Type	Name	Equivalent Model of Computation
Type 3	Regular	Finite State Machine, FSM, also known as Discrete Finite Automata, DFA
Type 2	Context Free	Stack Machine (Push Down Automata, PDA)
Type 1	Context Sensitive	Turing Machine, TM, with finite tape
Type 0	Unrestricted	Turing Machine, TM, with infinite tape

Context Free versus Context Sensitive

- Context free languages allow only a single non-terminal on the LHS
  - A → aA | a | AB

- Context sensitive languages allow groups of terminal and non-terminals on the LHS
  - Rules can only be used in certain contexts
  - A → aA | a
  - aaaA → AB (the 'aaa' is the context, AB can only appear if three a's precede the A)

- The context allows for more "if-then" conditions –
  - For context free, a rule may be used at any time
  - For context sensitive, a rule may be used only in the correct context
  - More "if-then" rules implies a more sophisticated language

PDA: Basic Definitions

- A Push Down Automata (PDA) is a mathematical device to recognize certain languages (context free languages).
  - An PDA will *accept* all strings in the language it is designed for.
  - An PDA will *reject* all strings not in the language it is designed for.
  - Each language may have many PDAs; most are trivial.
  - The class of language PDAs recognize are called the *context free languages*.

- Deterministic: A system is deterministic if there is exactly one unambiguous action defined for every situation
  - You can always determine what to do next
  - You can determine ahead of time what a result will be

- Non-Deterministic FSMs and PDAs exist – we will not focus them

PDA: Informal Definition

- Stack (Review)
  - Data structure holding multiple values, but only the “top” value is visible
  - Push: An item is added to the stack
  - Pop: The top item is removed from the stack
  - LIFO behavior: Last In First Out

- The PDA is just a FSM with a stack added to it
- In addition to the input alphabet, Σ, there is a stack alphabet, Γ
- Every transition is now dependent on current state, input symbol, and top of stack symbol
- Every transition now has a state change and a change to the stack

PDA: Formal Definition

- Formally, a PDA $M = \{Q, \Sigma, \Gamma, q_o, Z, F, \delta\}$, where,
  - Q = a finite set of states
  - Σ = a finite set of symbols for input strings
  - Γ = finite set of symbols for use on stack
  - q_o = a single *start state*, where processing begins ($q_o \in Q$)
  - Z = a single symbol representing an empty stack ($Z \in \Gamma$)
  - F = a set of *final*, or *accepting*, states. (F subset of Q)
  - δ = a transition function

PDA Operation

- A PDA, M, processes a string, u, as follows:
  - M begins in the start state, q_0, with Z (empty stack) on the stack
  - M processes string u one symbol at a time, from left to right.
  - The top of stack is "popped" every time an input symbol is processed.
  - Each symbol that is processed results in a state transition and a "push" on the stack.
  - Accepted strings are in the language, L, and rejected strings are not.
  - Acceptance criteria discussed on next slide.

- Will also allow transition to occur without processing a symbol
  - ε transitions, where ε represents "no input processed."
  - Sometimes useful, will still keep them deterministic

PDA Acceptance Criteria ….

- Given an FSM, M, then M accepts a string u iff $\delta(u, q_o) \in F$

- PDAs may accept strings based on accepting states or empty stack.

- Given a PDA, M, then M accepts a string u iff $\delta(u, Z, q_o) \in F \times \Gamma^*$
  - $F \times \Gamma^*$ is the formal way of saying "You must end up in a final state and we don't care what is on the stack."
- OR given a PDA, M, then M accepts string u iff $\delta(u, Z, q_o) \in Q \times Z$
  - $Q \times Z$ is the formal way of saying "we don't care what state you end up in, you have to have an empty stack."

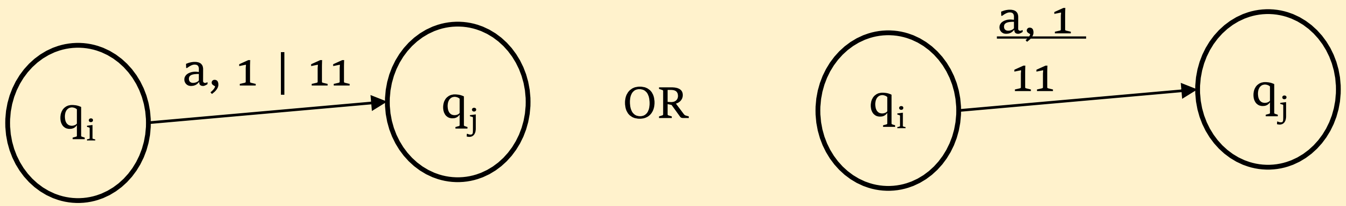
- Design a specific PDA to accept one way or the other, not both.
- We'll use empty stack in the examples.

PDA Transition Function ….

- For the FSM, the transition function, δ, takes as input a symbol and a state
  - This is actually an ordered pair (symbol, state).
- For the FSM, δ outputs a state -- $\delta(a, q_i) = q_j$
- For the PDA, δ takes in a symbol from the string being processed, the current state, and the current symbol at the top of the stack
- For the PDA, δ outputs a state and a "push" on the stack
- Formally, $\delta(a, x, q_i) = q_j, v$
  - 'a' is a symbol from Σ. (The next symbol processed from the input string.)
  - 'x' is a symbol from Γ. <u>It is popped from the stack</u>
  - q_i and q_j are states in the machine
  - v is a <u>string</u>, which may be empty, pushed onto the stack
  - $\delta(\varepsilon, x, q_i) = q_j, v$ is allowed

PDA Graphical Representation

1. State, start state, final state are all the same as for FSM.
2. Transition must now include the input symbol, top of stack, and string written to the stack. Let $\Sigma = \{a, b\}$, $\Gamma = \{0, 1, \$\}$:



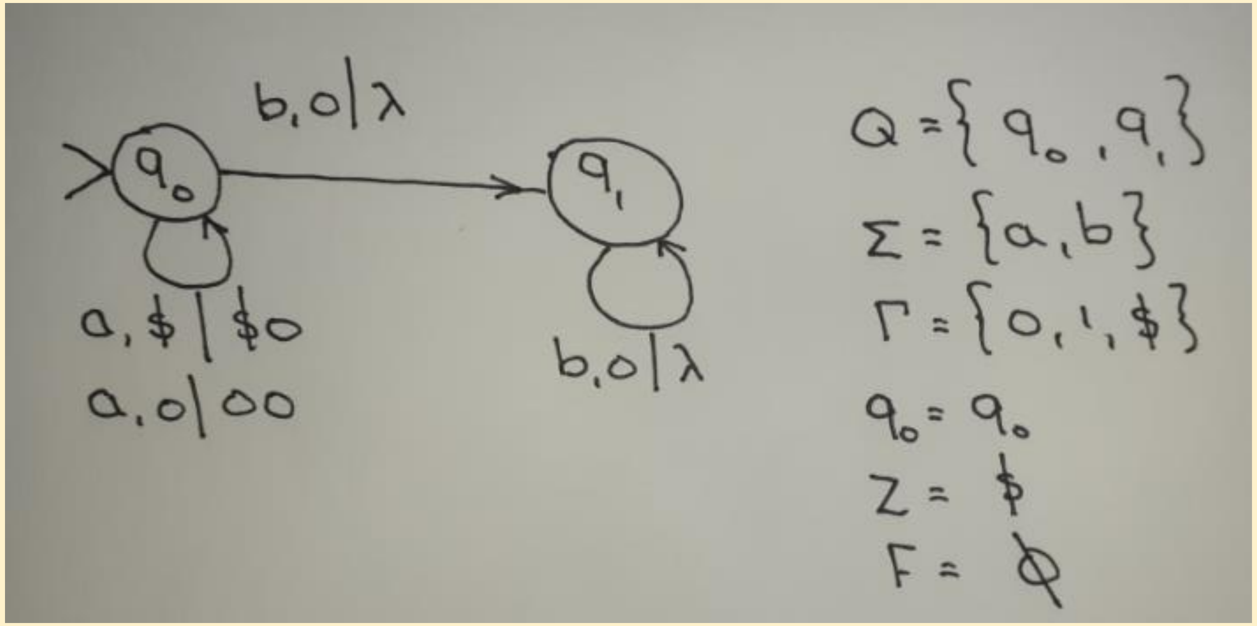
Read this as "If in state q_i, and processing an 'a' from the input string, and a '1' is popped of the stack, then transition to state q_j and push the string '11' on the stack.

3. We're going to cheat – including all transitions leads to clutter.
4. If a transition is not shown, assume immediate rejection of string.

PDA Example #1

- Let $\Sigma = \{a, b\}$ (For all examples unless stated otherwise)
- Let L_1 be the set of strings having some number of 'a's followed by the same number of 'b's.
- This is the canonical non-regular language – it requires memory to keep track of the 'a's.
- $L_1 = a^n b^n$ (Not really a regular expression, borrowing the form)

PDA Example #1, $L_1 = a^n b^n$



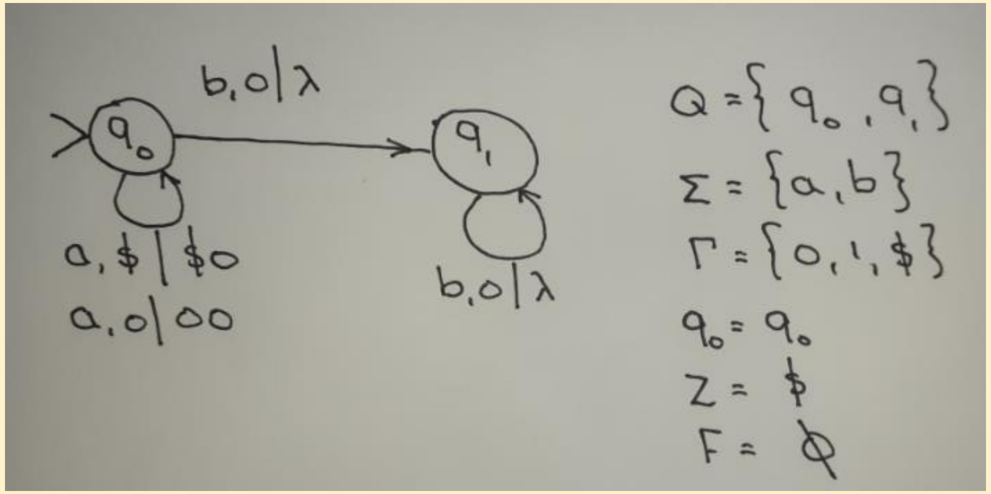
	State Transitions						Pushed on Stack					
	a			b			a			b		
State	$	0	1	$	0	1	$	0	1	$	0	1
0	0	0	R	R	1	R	$0	00	na	na	λ	na
1	R	R	R	R	1	R	na	na	na	na	λ	na

R = Reject string

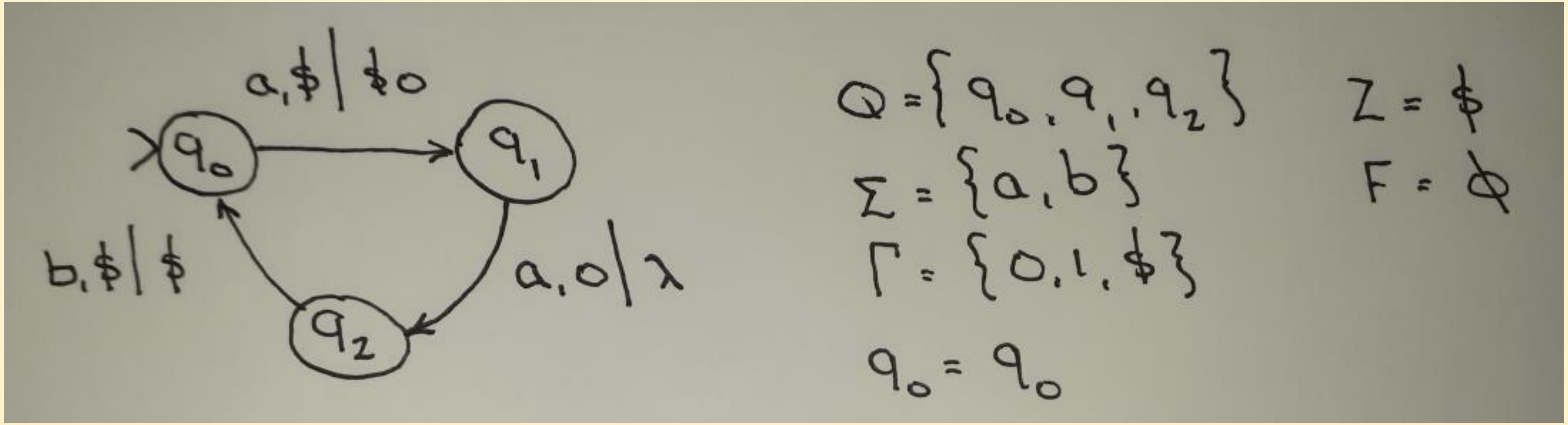
PDA Processing a String Example

$L_1 = a^n b^n$ •u = aaabbb

Beginning State	Existing Stack Contents	Remaining String	Input Symbol	Popped From Top of Stack	New State	Pushed Onto Stack	New Stack Contents
0	$	aaabbb	a	$	0	$0	0$
0	0$	aabbb	a	0	0	00	00$
0	00$	abbb	a	0	0	00	000$
0	000$	bbb	b	0	1	λ	00$
1	00$	bb	b	0	1	λ	0$
1	0$	b	b	0	1	λ	$
1	$ = ACCEPT	λ					



PDA Example #2, $L_2 = (aab)^*$



	State Transitions						Pushed on Stack					
	a			b			a			b		
State	$	0	1	$	0	1	$	0	1	$	0	1
0	1	R	R	R	R	R	$0	na	na	na	na	na
1	R	2	R	R	R	R	na	λ	na	na	na	na
2	R	R	R	0	R	R	na	na	na	$	na	na

R = Reject string

PDA Example #2, $L_2 = (aab)^*$, u = aabaab

$L_1 = (aab)^*$		•u = aabaab					
Beginning State	Existing Stack Contents	Remaining String	Input Symbol	Popped From Top of Stack	New State	Pushed Onto Stack	New Stack Contents
0	$	aabaab	a	$	1	$0	0$
1	0$	abaab	a	0	2	λ	$
2	$	baab	b	$	0	$	$
0	$	aab	a	$	1	$0	0$
1	0$	ab	a	0	2	λ	$
2	$	b	b	$	0	$	$
0	$ = ACCEPT	λ					

PDA Example #2, $L_2 = (aab)^*$, u = aaba (not in L)

$L_1 = (aab)^*$		u = aaba					
Beginning State	Existing Stack Contents	Remaining String	Input Symbol	Popped From Top of Stack	New State	Pushed Onto Stack	New Stack Contents
0	\$	aaba	a	\$	1	\$0	0\$
1	0\$	aba	a	0	2	λ	\$
2	\$	ba	b	\$	0	\$	\$
0	\$	a	a	\$	1	\$0	0\$
1	0\$ = REJECT	λ	a	0	2	λ	\$