

CS3063 Theory of Computing

Semester 4 (20 Intake), Feb – Jun 2023

Lecture 7

Context-Free Languages: Session 2

Sanath Jayasena

Announcement

- **Mid-semester Test** next week
 - Date: **4th May**, at **8.15am**
 - Venue: **Exam Hall 2**
 - 1 hour duration (approx.)
 - Based on lecture 1-7 (today's lecture included)
- Quiz 6 (on L7, today's lecture) will be on 8th May
 - No quiz for next week

Previous Lecture

- Context-free Languages
 - Context-free Grammars (CFGs)
 - Derivations
 - Properties of CFLs
 - CFG for a Regular Language
 - CFG from an FA
 - Regular Grammars
 - Derivation Trees
 - Ambiguous CFGs

Today's Outline:

Lecture 7

Context-free Languages (CFLs) - 2

- **Ambiguous CFGs (continued)**
- **Simplified Forms and Normal Forms**
- **Pushdown Automata**
 - Definition
 - Acceptance
- **Examples**

PART 1

Today's Outline:

Lecture 7

Context-free Languages (CFLs) - 2

- **Ambiguous CFGs (continued)**
- Simplified Forms and Normal Forms
- Pushdown Automata
 - Definition
 - Acceptance
- Examples

Ambiguous CFGs: Review

- Ambiguous CFG means the grammar can produce a string that has more than 1 derivation tree
- For some ambiguous CFGs, it is possible to find an equivalent unambiguous CFG

Example (from last time)

- Suppose we have the ambiguous CFG (for algebraic-expressions), G:

$$S \rightarrow S + S \mid S * S \mid (S) \mid a$$

- Is there an equivalent unambiguous CFG?
 - Avoid $S \rightarrow S + S$ and $S \rightarrow S * S$ because these produce ambiguity
 - Can impose
 - rules of order; e.g., $a+a+a$ means $(a+a)+a$
 - operator precedence; e.g., $*$ has higher precedence than $+$

Solution

- An **expression** is a **sum of terms**
 - Replace $S \rightarrow S + S$ by $S \rightarrow S + T \mid T$ where T stands for **term**
- A **term** is a **product of factors**
 - $T \rightarrow T * F \mid F$ where F stands for **factor**
- Precedence of $*$ over $+$ incorporated
- Association from left-to-right incorporated

Solution ...contd

- How about parenthesized expressions?
 - Most appropriate to consider as a factor
 - That is, it should get high precedence
- So we have the CFG, G' :

$$S \rightarrow S + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (S) \mid a$$


Can prove G' is
equivalent to the
original G and that G'
is unambiguous

Associativity of Operators

- Associativity
 - Tells us how to group operators with same precedence when parentheses are not used

- **Left associativity**

- Examples


$$5 + 3 + 2 = (5 + 3) + 2 = 5 + (3 + 2) = 10$$

$$9 - 3 - 2 = (9 - 3) - 2 = 4 \text{ but } 9 - (3 - 2) = 8$$

$$5 - 3 + 2 = (5 - 3) + 2 = 4 \text{ but } 5 - (3 + 2) = 0$$

$$16 \div 4 * 2 = (16 \div 4) * 2 = 8 \text{ but } 16 \div (4 * 2) = 2$$

Associativity ...

- Subtraction and division: we normally consider as inherently *left associative*
- Addition, multiplication
 - No inherent associativity, but usually considered as left associative
- **Right associativity**
 - Examples: exponentiation, assignment
 $4^3^2 = 4^{(3^2)}=4^9=2^{18}$ but $(4^3)^2=2^{12}$
 $a = b = c$ means $a = (b = c)$

PART 2

Today's Outline:

Lecture 7

Context-free Languages (CFLs) - 2

- Ambiguous CFGs (continued)
- **Simplified Forms and Normal Forms**
- Pushdown Automata
 - Definition
 - Acceptance
- Examples

Simplified & Normal Forms

- Possible improvements to grammars (without changing the language)
 - Eliminate Λ -productions
 - Productions of the form $A \rightarrow \Lambda$
 - Eliminate unit productions
 - Productions in which one non-terminal is replaced simply by another
 - Standardize productions for a “normal form”

Λ -Productions

- Example (E.g., 6.14, p. 233 in text book)
- Let G be a CFG such that:

$$S \rightarrow ABCBCDA$$

$$A \rightarrow CD$$

$$C \rightarrow a \mid \Lambda$$

$$B \rightarrow Cb$$

$$D \rightarrow bD \mid \Lambda$$

- Cannot simply remove the Λ -productions
- Rewrite the 1st one as $S \rightarrow A_1BC_1BC_2DA_2$
 - A_1, A_2, C_1, C_2 and D have Λ -productions
 - These can also derive non-null strings

Λ -Productions ...contd

- Without Λ -productions, we need to allow all possibilities by adding productions of the form $S \rightarrow \alpha$
 - α is a string obtained from ABCBCDA by deleting some subset of $\{A_1, A_2, C_1, C_2, D\}$
 - There are $2^5=32$ subsets
 - From the original $S \rightarrow ABCBCDA$, obtain 31 others and add to G

Λ -Productions ...contd

- Do similarly with other original productions
- The final CFG has 40 productions, including the above 32 and:

$$A \rightarrow CD \mid C \mid D$$

$$B \rightarrow Cb \mid b$$

$$C \rightarrow a$$

$$D \rightarrow bD \mid b$$

Λ -Productions ...contd

- **Definition:** A *nullable* non-terminal in a CFG, $G=(V, \Sigma, S, P)$, is defined as:
 - Any non-terminal A for which P contains the production $A \rightarrow \Lambda$ is *nullable*
 - If P contains the production $A \rightarrow B_1 B_2 \dots B_n$ and B_1, B_2, \dots, B_n are *nullable*, then A is *nullable*
 - No other non-terminal in V is *nullable*
- Nullable non-terminals are those A for which $A \Rightarrow^* \Lambda$

Algorithm *FindNull*

- Finding the nullable non-terminals in a given CFG, $G=(V, \Sigma, S, P)$
[p. 234 of text book]

FindNull

$N_0 = \{ A \in V \mid \text{production } A \rightarrow \Lambda \text{ exists in } P \}$

$i = 0$

do {

$i = i + 1$

$N_i = N_{i-1} \cup \{ A \mid P \text{ has } A \rightarrow \alpha \text{ for some } \alpha \in N_{i-1}^* \}$

} while $N_i \neq N_{i-1}$

N_i is the set of nullable non-terminals

Λ -Productions ...contd

Given a CFG, $G=(V, \Sigma, S, P)$, algorithm to get an equivalent CFG, $G1=(V, \Sigma, S, P1)$ where $L(G1)=L(G) - \{\Lambda\}$, $G1$ has no Λ -productions [pp. 234-235]

1. Initialize $P1$ to P
2. Find all nullable non-terminals in V using *FindNull*
3. For every $A \rightarrow \alpha$ in P , add to $P1$ every production that can be obtained from this one by deleting from α one or more occurrences of nullable non-terminals in α
4. Delete all Λ -productions from $P1$, any duplicates and productions of the form $A \rightarrow A$

Λ -Productions ...contd

- Eliminating Λ -productions will likely increase the number of productions substantially
 - Example on slides 14-16 (E.g., 6.14, p.233)
- If G is an ambiguous CFG, then the grammar produced by eliminating Λ -productions in G is also ambiguous

Eliminating Unit Productions

- Similar to eliminating Λ -productions
 - Need to consider all pairs of non-terminals A, B for which $A \Rightarrow^* B$ (and also if $A \rightarrow B$ exists)
- Must ensure we do not eliminate legitimate strings in the language
 - Whenever $B \rightarrow \alpha$ is a non-unit production and $A \Rightarrow^* B$, add the production $A \rightarrow \alpha$

Unit Productions ...contd

Given a CFG, $G=(V, \Sigma, S, P)$, algorithm to get an equivalent CFG, $G1=(V, \Sigma, S, P1)$ having no unit productions [pp. 236-237]

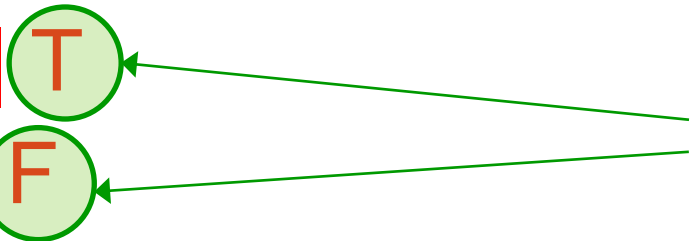
1. Initialize $P1$ to P
2. For each A in V , find the set of *A-derivable* non-terminals (those that can be derived from A)
3. For every pair (A,B) such that B is *A-derivable*, and every non-unit production $B \rightarrow \alpha$ in P , add $A \rightarrow \alpha$ to $P1$ if it is not already in $P1$
4. Delete all unit productions from $P1$

Example

- Consider a previous example we studied

$$\begin{aligned} S &\rightarrow S + T \mid \textcircled{T} \\ T &\rightarrow T * F \mid \textcircled{F} \\ F &\rightarrow (S) \mid a \end{aligned}$$

Unit productions



- After deleting unit productions we have:

$$\begin{aligned} S &\rightarrow S + T \mid T * F \mid (S) \mid a \\ T &\rightarrow T * F \mid (S) \mid a \\ F &\rightarrow (S) \mid a \end{aligned}$$

Normal Forms

- Normal forms impose further restrictions upon the forms of productions in a CFG
- Examples
 - Chomsky Normal Form (CNF)
 - Greibach Normal Form (GNF)

Chomsky Normal Form (CNF)

- A CFG is in CNF if every production is one of the two types:

$$A \rightarrow BC$$

$$A \rightarrow a$$

where A , B and C are non-terminals and a is a terminal

Converting a CFG to CNF

- Follow 4 steps
 1. Eliminate Λ -productions
 2. Eliminate unit productions
 3. Restrict the RHS of productions to single terminals or strings of ≥ 2 non-terminals
 4. Replace each production having > 2 non-terminals on RHS by an equivalent set of productions each having exactly 2 non-terminals on the RHS

Example

- If $A \rightarrow aAb$ and $B \rightarrow ab$
then the CNF will be:

$$A \rightarrow XZ$$

$$Z \rightarrow AY$$

$$B \rightarrow XY$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

PART 3

Today's Outline:

Lecture 7

Context-free Languages (CFLs) - 2

- Ambiguous CFGs (continued)
- Simplified Forms and Normal Forms
- **Pushdown Automata**
 - Definition
 - Acceptance
- Examples

Pushdown Automata (PDA)

- **Example:** Consider the following CFG

$$S \rightarrow aSa \mid bSb \mid c$$

- Generates odd-length palindromes of $\{a, b\}$ with c being the middle symbol
- How to recognize this language?
 - Scan a string L-to-R, pushing symbols in the 1st half onto a stack as we read each one
 - After reaching the middle “ c ”, begin matching next ones with those on stack
 - Can do this with an automaton of 3 states: start state, state after seeing “ c ” and accepting state

Pushdown Automata (PDA)

- A PDA is a 7-tuple $M=(Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$, where:
 - Q is a finite set of states
 - Σ and Γ are finite sets (the input and stack alphabets, respectively)
 - q_0 , the initial state, is an element of Q
 - Z_0 , the initial stack symbol, is an element of Γ
 - A , the set of accepting states, is a subset of Q
 - δ is the transition function

Chapter 7 in textbook

Pushdown Automata (PDA)

- Transition δ maps $Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$
- Current state is not enough to specify the status of the machine
- A **move** depends on
 - The current state
 - The next input
 - The symbol currently on top of the stack

Pushdown Automata (PDA)

- A **move** consists of
 - Changing states (can stay in current state)
 - Replacing the symbol on top of the stack by a string of zero or more symbols
- Popping the top symbol off the stack
 - Means replacing it by Λ
- Pushing **Y** onto the stack
 - Means replacing the top symbol **X** by **YX** (assume left end of string is on top of stack)

Pushdown Automata (PDA)

- We allow the possibility that stack alphabet is different from input alphabet
- For a state q , an input a and a stack symbol X ,

$$\delta(q, a, X) = (p, \beta)$$

means in state q , with X on top of the stack, we read the symbol a , move to state p , and replace X on the stack by string β

Pushdown Automata (PDA)

- Initially special start symbol Z_0 on the stack
 - Z_0 never removed from stack
 - No additional copies of Z_0 pushed onto stack
 - Z_0 on top means the stack is empty
 - No move when stack is empty
- We allow moves when Λ is input

Terminology / Notations

- A **configuration** of the PDA $M=(Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$, is a triple

$$(q, x, \alpha)$$

where q is in Q , x is in Σ^* and α is in Γ^*

- “ (q, x, α) is the **current configuration**” \rightarrow
 - q is the current state
 - x is the string of remaining unread input
 - α is the current stack contents (leftmost is top)

Terminology / Notations ...contd

$$(p, x, \alpha) \vdash_M (q, y, \beta)$$

means, one of the possible moves in the first configuration takes M to the second

- This can happen in 2 ways
 - Either with an input symbol
 - Or, a Λ -transition

Notations ...contd

- In both cases, $x=ay$ for some a in $\Sigma \cup \{\Lambda\}$
 - β is obtained from α by replacing the 1st symbol X by some string μ
 - That is (q, μ) is in $\delta(p, a, X)$

- Then,

$$(p, x, \alpha) \vdash_M^* (q, y, \beta)$$

means, there is a sequence of moves that takes M from the first to the second configuration

Acceptance by a PDA

- **Definition:** If $M=(Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is a PDA and x is in Σ^* , x is accepted by M if
$$(q_0, x, Z_0) \vdash_M^* (q, \Lambda, \alpha)$$
for some α in Γ^* and some q in A
 - There *exists* a sequence of moves
 - The stack may or may not be empty because $\alpha=\Lambda$ or $\alpha\neq\Lambda$; this is *acceptance by final state*
 - Possible to have *acceptance by empty stack*
 - Are the two forms equivalent?

PART 4

Today's Outline:

Lecture 7

Context-free Languages (CFLs) - 2

- Ambiguous CFGs (continued)
- Simplified Forms and Normal Forms
- Pushdown Automata
 - Definition
 - Acceptance
- **Examples**

Example 1

- Consider the previous CFG again

$$S \rightarrow aSa \mid bSb \mid c$$

- Generates odd-length palindromes of $\{a, b\}$ with c being the middle symbol (let's call this language *SimplePal*)
- What is the PDA to accept *SimplePal*?
 - Discussion from E.g. 7.1 (p. 251)

Example 1 ...contd

- Simple palindrome (*SimplePal*) recognizer
 - 3 states $Q = \{q_0, q_1, q_2\}$
 - q_0 : initial state
 - Processes first $\frac{1}{2}$ of string, each input symbol pushed to stack
 - Go to q_1 upon receiving c , the middle symbol
 - q_1 : state for processing second $\frac{1}{2}$ of string
 - Each input symbol compared to top stack symbol; if they match pop and discard, else crash (reject)
 - Go to q_2 when stack is empty
 - q_2 : accepting state

Example 1 ...contd

- Simple palindrome (*SimplePal*) recognizer
 - Input alphabet $\Sigma = \{a, b, c\}$
 - Stack alphabet $\Gamma = \{a, b, Z_0\}$
 - Transition function δ : Table 7.1 (p. 254), next slide
 - Machine will crash when no move specified
 - Trace the moves for sample inputs *abcba*, *ab*

Transition Function/Table

Move #	State	Input	Stack Symbol	Move(s)
1	q_0	a	Z_0	(q_0 , aZ_0)
2	q_0	b	Z_0	(q_0 , bZ_0)
3	q_0	a	a	(q_0 , aa)
4	q_0	b	a	(q_0 , ba)
5	q_0	a	b	(q_0 , ab)
6	q_0	b	b	(q_0 , bb)
7	q_0	c	Z_0	(q_1 , Z_0)
8	q_0	c	a	(q_1 , a)
9	q_0	c	b	(q_1 , b)
10	q_1	a	a	(q_1 , Λ)
11	q_1	b	b	(q_1 , Λ)
12	q_1	Λ	Z_0	(q_2 , Z_0)
	All other combinations			none

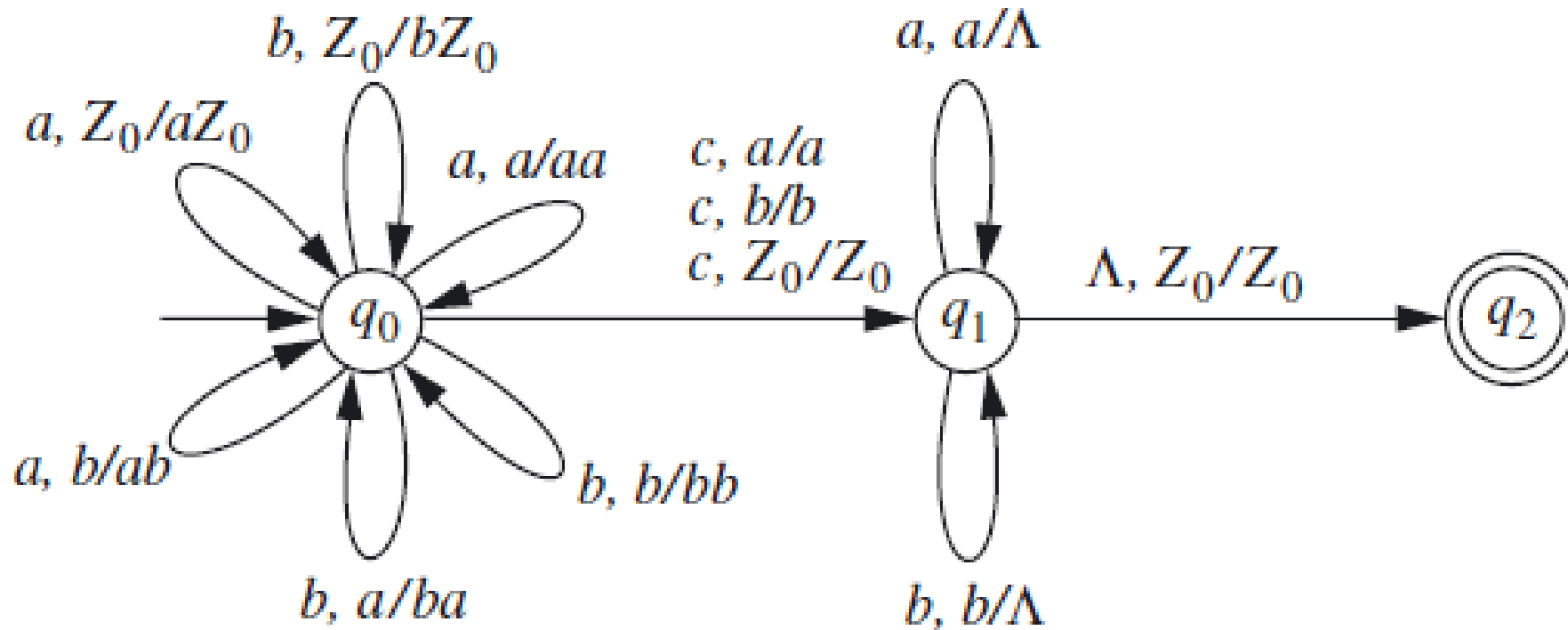
Tracing Moves; “abcba”, “ab”

Move #	Resulting state	Unread input	Stack
(initially)	q_0	abcba	Z_0
1	q_0	bcba	aZ_0
4	q_0	cba	baZ_0
9	q_1	ba	baZ_0
11	q_1	a	aZ_0
10	q_1	-	Z_0
12	q_2	-	Z_0
(accept)			
(initially)	q_0	ab	Z_0
1	q_0	b	aZ_0
4	q_0	-	baZ_0
(crash)			

Example 1 ...contd

- Simple palindrome (*SimplePal*) recognizer
 - Consider an input string like *acaa*
 - Portion of the string “*aca*” can be considered as accepted, but not the whole string
 - “**Transition diagram**” in Fig. 7.1 (p. 255)
 - Cannot be seen in the same way as for an FA
 - Special notations used for transitions
 - *Input symbol, stack symbol / string for top of stack*
 - Stack contents not seen; full status not shown
 - Form of this discussed again later

Transition Diagram



Example 2

- Example 7.2 in text book p. 257
- A PDA accepting *the language of all palindromes* over $\{a, b\}$ even or odd-length (let's call this *Pal*)
 - When we reach the mid-point, if odd-length, discard mid symbol
 - Push all symbols in first half to stack
 - Match second half symbols to symbols on stack

Example 2 ...contd

- How does PDA know mid-point reached?
 - PDA has to **guess**
 - Guessing OK if non-palindromes not accepted
 - First a sequence of “**not yet**” guesses
 - Push each symbol to stack (first half)
 - Then a “**yes**” guess stops the “not yet” sequence
 - Odd-length (xsx^r): discard next, match after that
 - Even-length (xx^r): : match from next
 - After that no more guesses

Example 2 ...contd

- Consequences of above guessing
 - Non-palindromes will not be accepted
 - Can accept all palindromes
 - There is a sequence of choices involving making the correct “yes” guess at the right time
 - PDA may also guess at the wrong time for a given palindrome
 - May not accept or accept a different palindrome

Example 2 ...contd

- PDA for ***Pal*** (similar to that of Example 1)
 - 3 states $Q = \{q_0, q_1, q_2\}$
 - q_0 = initial (guessing) state, q_1 = comparison-making state, q_2 = accepting state
 - Input alphabet $\Sigma = \{a, b\}$
 - Stack alphabet $\Gamma = \{a, b, Z_0\}$
 - Transition function δ : Table 7.2 (p. 258), next slide

Example 2: Transition Table

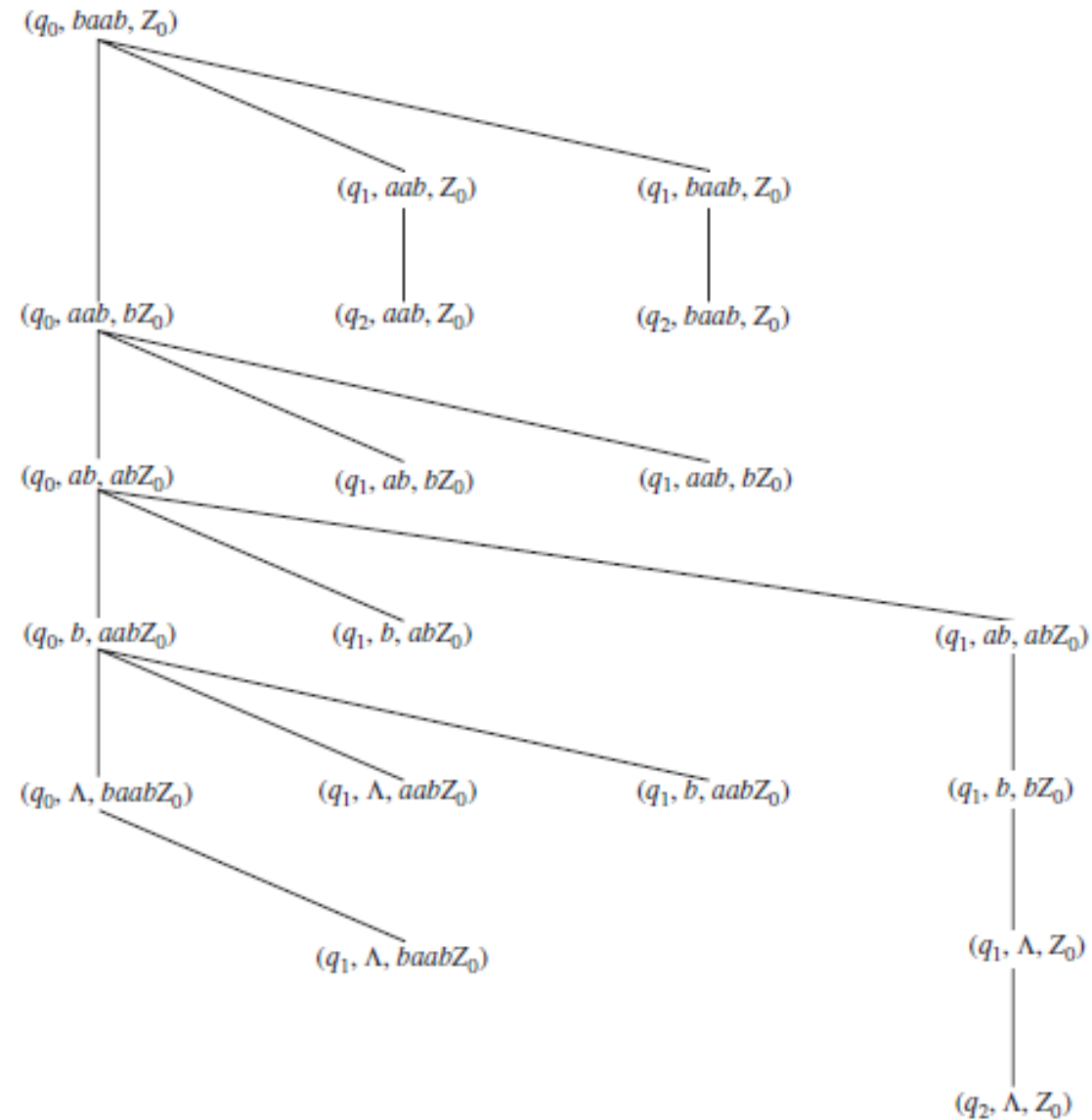
- Table 7.2 Note the non-determinisms

Move Number	State	Input	Stack Symbol	Move(s)
1	q_0	a	Z_0	$(q_0, aZ_0), (q_1, Z_0)$
2	q_0	a	a	$(q_0, aa), (q_1, a)$
3	q_0	a	b	$(q_0, ab), (q_1, b)$
4	q_0	b	Z_0	$(q_0, bZ_0), (q_1, Z_0)$
5	q_0	b	a	$(q_0, ba), (q_1, a)$
6	q_0	b	b	$(q_0, bb), (q_1, b)$
7	q_0	Λ	Z_0	(q_1, Z_0)
8	q_0	Λ	a	(q_1, a)
9	q_0	Λ	b	(q_1, b)
10	q_1	a	a	(q_1, Λ)
11	q_1	b	b	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
(all other combinations)				none

Example 2: Computation Tree

- Computation tree shows the configuration at each step and possible choices of moves at each step
- Fig. 7.2 (p.259, next slide) shows the *computation tree* that traces the moves for input *baab*

Computation Tree



Example 2 ...contd

- Sequence of moves leading to acceptance of input string *baab*

$$\begin{array}{lcl} (q_0, baab, Z_0) & \vdash & (q_0, aab, bZ_0) \\ & \vdash & (q_0, ab, abZ_0) \\ & \vdash & (q_1, ab, abZ_0) \\ & \vdash & (q_1, b, bZ_0) \\ & \vdash & (q_1, \Lambda, Z_0) \\ & \vdash & (q_2, \Lambda, Z_0) \quad (\text{accept}) \end{array}$$

Conclusion

- Our discussion today
 - Review on Ambiguity
 - Simplified Forms
 - Normal Forms
 - Pushdown Automata
 - Definition, Acceptance
 - Examples