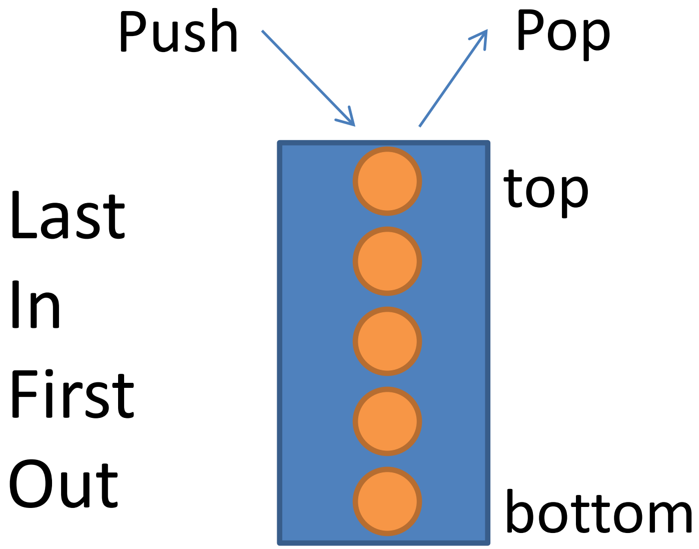
Chapter 5

Pushdown Automata

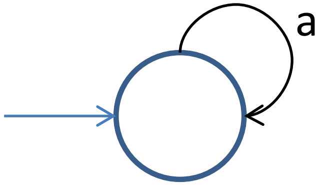
Definitions and Examples

- A language can be generated by a CFG if and only if it can be accepted by a ***pushdown automaton***
- A pushdown automaton is similar to an FA but has an auxiliary memory in the form of a stack

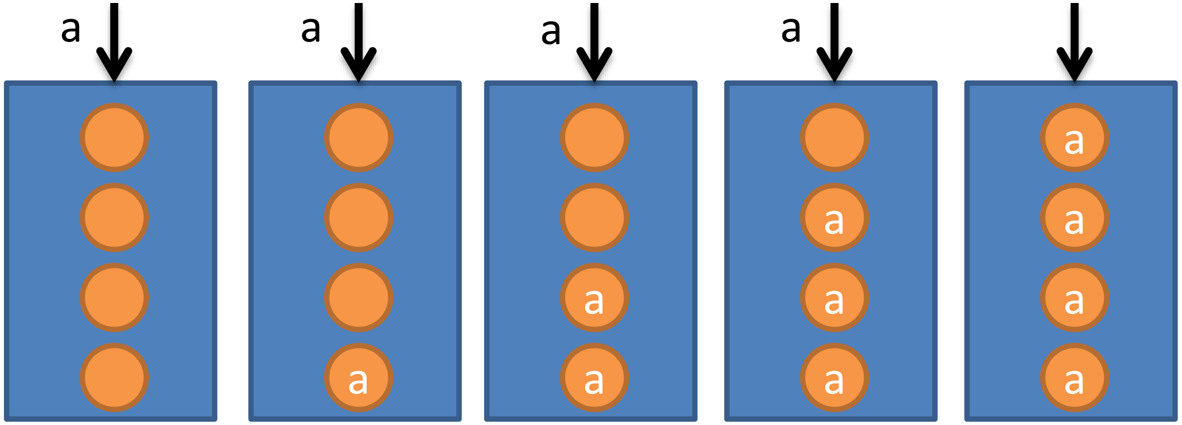


- Pushdown automata are, by default, nondeterministic. Unlike FA/NFA’s, the nondeterminism cannot always be removed

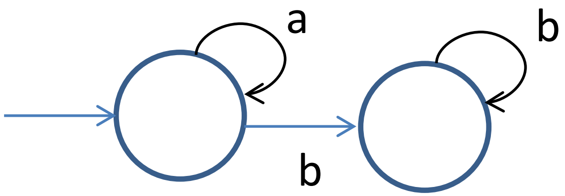
- We’ll start with a simple example, the language *AnBn* = $\{a^n b^n \mid n \geq 0\}$
  - This is not a regular language (there is no FA for it), but it is context-free (there is a CFG)
- In processing the first part of an input string that might be in *AnBn,* all we need to remember is the number of *a*’s
  - Saving the actual *a*’s is a simple way to do this
  - So, the PDA will start by reading *a*’s and pushing them onto the stack



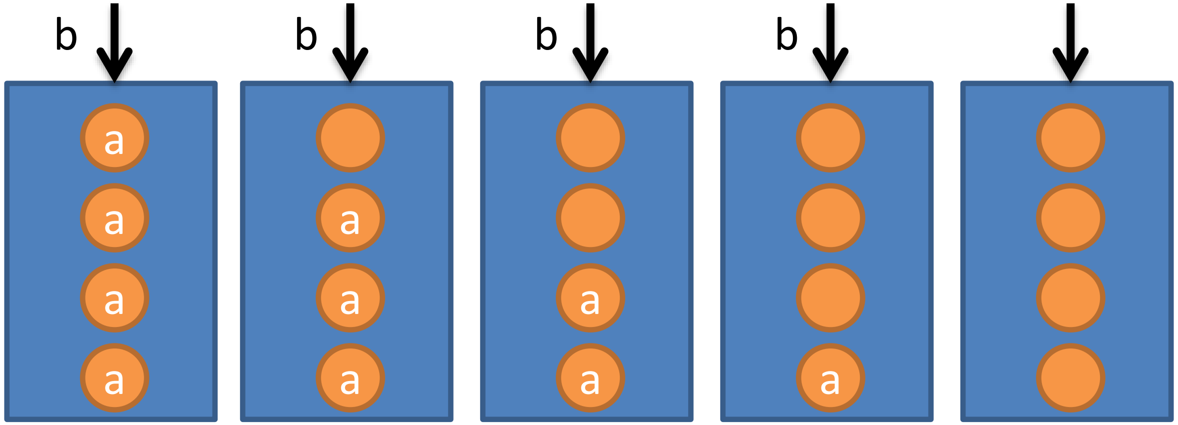
This pair (state,transition) saves *a* in the stack



- As soon as the PDA reads *b*, two things should happen
  - It enters a new state in which only *b*’s are legal inputs
  - It pops one *a* off the stack to cancel this *b*

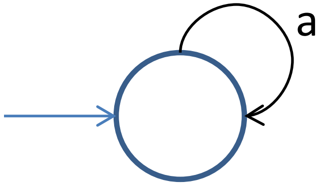


This pair (state,transition) removes *b* from the stack

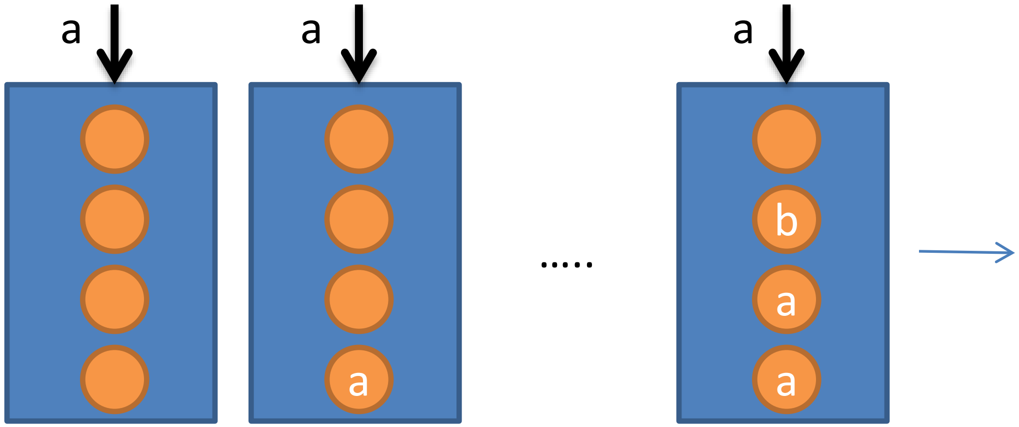


- In the new state, the correct move on *b* is to pop *a* off the stack to cancel it. Once enough *b*’s have been read to cancel the *a*’s on the stack, the string read so far is accepted
- We assume that the stack has no limit to its size, so the PDA can handle anything in *AnBn*

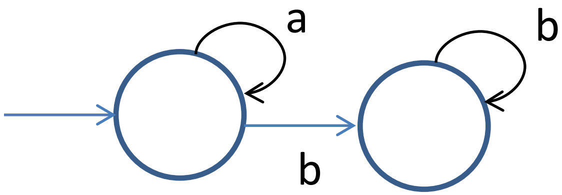
- Problem: in the transition function we need to consider what we have in the stack



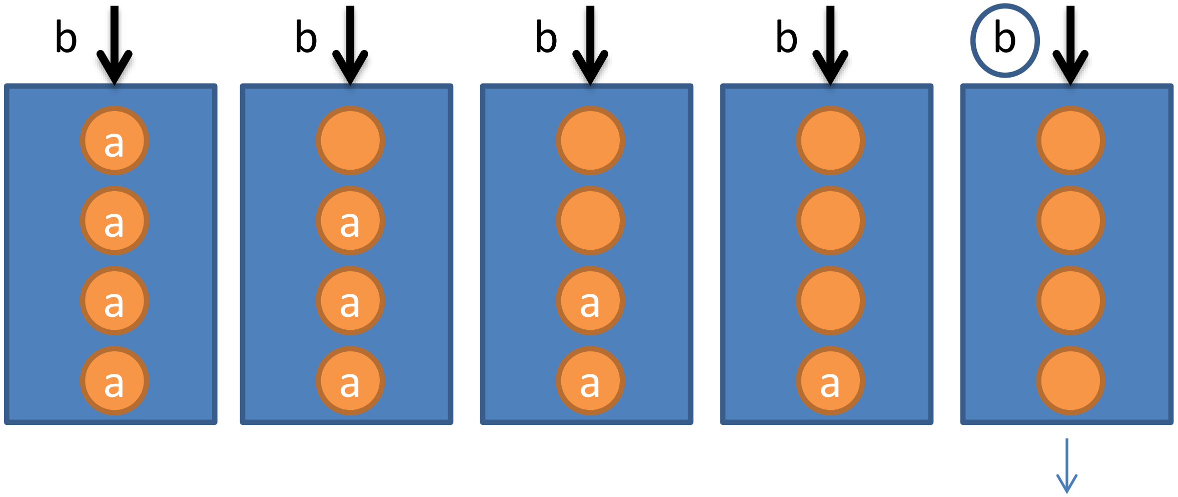
What if there is *b* on top?



Not accepted

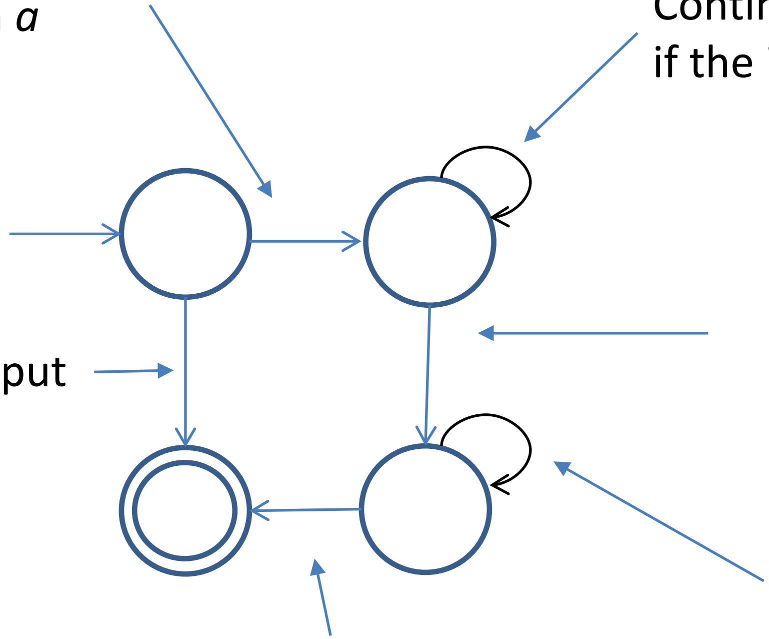


Or what if one more *b* is in the string?



Not accepted

- A single move of a PDA depends on the current state, the next input, and the symbol currently on top of the stack (the only one the PDA can see)
- In the move, the PDA is allowed to change states and to modify the top of the stack. We will allow the PDA to replace the top symbol X by a string α of stack symbols
- Example: push a symbol Y means "replace X by YX"; and pop X *means* "replace X by Λ"



If the stack is empty and the input is *a* then push *a*

Continue to push *a's* into stack if the input symbols are *a*'s

First *b* in the input erases the top *a* in the stack

Nothing in the input

Continue to erase *a's* if the input symbols are *b*'s

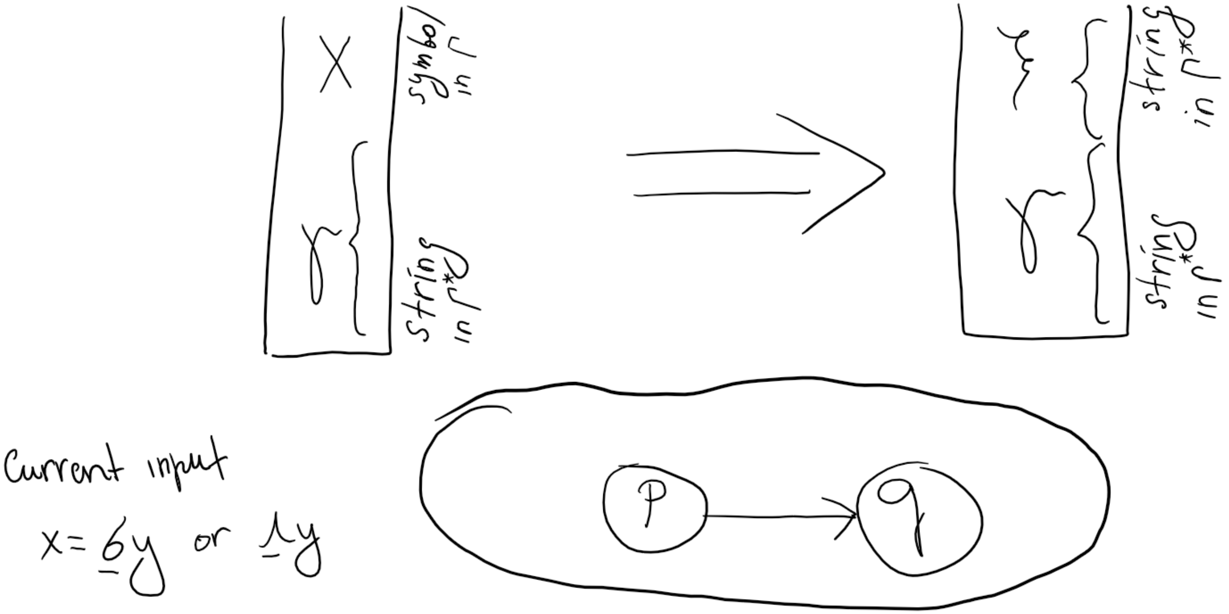
Input and stack are empty

Definition: PDA

- **Definition:** A *pushdown automaton* is a 7-tuple $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$, where:
  - Q is a finite set of states
  - The *input* and *stack* alphabets Σ and Γ are finite sets
  - $q_0 \in Q$ is the initial state
  - $Z_0 \in \Gamma$ is the initial stack symbol
  - $A \subseteq Q$ is the set of accepting states
  - The transition function is $\delta : Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma \rightarrow$ a finite **subset** of $Q \times \Gamma^*$
- Because values of δ are sets, M may be **nondeterministic**
- A move requires that there be at least one symbol on the stack. Z_0 is the one on the stack initially

- A *configuration* of a PDA is a triple (q, x, α)
  - $q \in Q$ is the current state
  - $x \in \Sigma^*$ is the portion of the input string that has not yet been read
  - The complete contents of the stack is $\alpha \in \Gamma^*$ (the convention will be that the top corresponds to the leftmost symbol). The last symbol of α is often Z_0.

- We write $(p, x, \alpha) \vdash_M (q, y, \beta)$ to mean that one of the possible moves in the first configuration takes M to the second.
  - This happens if $x=\sigma y$ or $x= \Lambda y=y$, i.e., where $\sigma\in \Sigma \cup \{\Lambda\}$
  - If $\alpha=X\gamma$, $X\in\Gamma$ and $\gamma\in\Gamma^*$ then $\beta=\xi\gamma$ for some string ξ, for which $(q, \xi) \in \delta(p, \sigma ,X)$



- $\vdash_M^n$ and $\vdash_M^*$ refer to n moves and zero or more moves

Definition: If $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ and $x \in \Sigma^*$, the string x is *accepted* by M if

$$(q_0, x, Z_0) \vdash_M^* (q, \Lambda, \alpha)$$

There is nothing left in the input string

for some $\alpha \in \Gamma^*$ and some $q \in A$

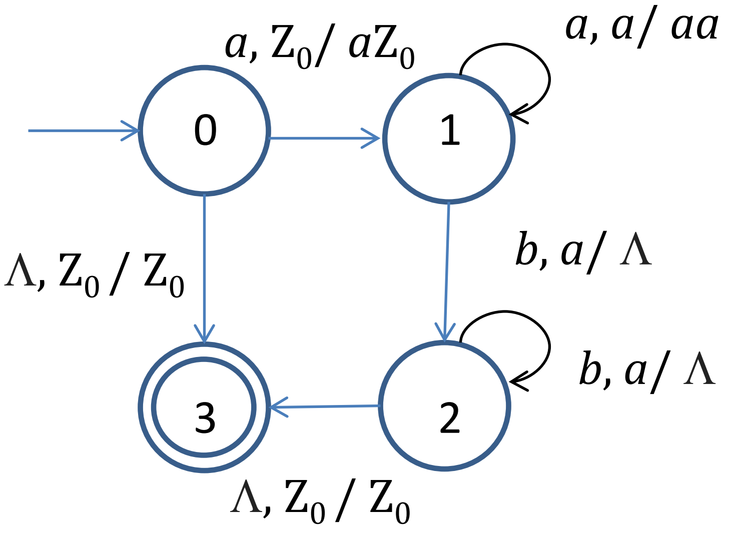
- A language L is said to be accepted by M if L is precisely the set of strings accepted by M

- Sometimes a string accepted by M is said to be accepted *by final state*, because acceptance does not depend on the final stack contents at all

PDA for AnBn is $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ where $Q = \{q_0, q_1, q_2, q_3\}$, $A = \{q_3\}$, and the transitions are shown in this table:

Move #	State	Input	Stack top	Move(s)
1	q_0	a	Z_0	(q_1, aZ_0)
2	q_1	a	a	(q_1, aa)
3	q_1	b	a	(q_2, Λ)
4	q_2	b	a	(q_2, Λ)
5	q_2	Λ	Z_0	(q_3, Z_0)
all other combinations				none

Stack top *a* is replaced with *aa*

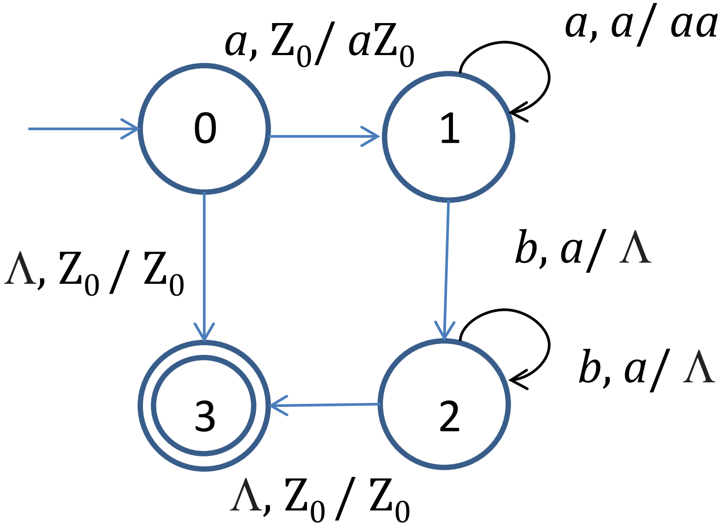


- The moves that M makes as it processes the string $aabb$ are shown below:

Rest of the input string

$$\begin{aligned}
(q_0, aabb, Z_0) &\vdash (q_1, abb, aZ_0) \\
&\vdash (q_1, bb, aaZ_0) \\
&\vdash (q_2, b, aZ_0) \\
&\vdash (q_2, \Lambda, Z_0) \\
&\vdash (q_3, \Lambda, Z_0)
\end{aligned}$$

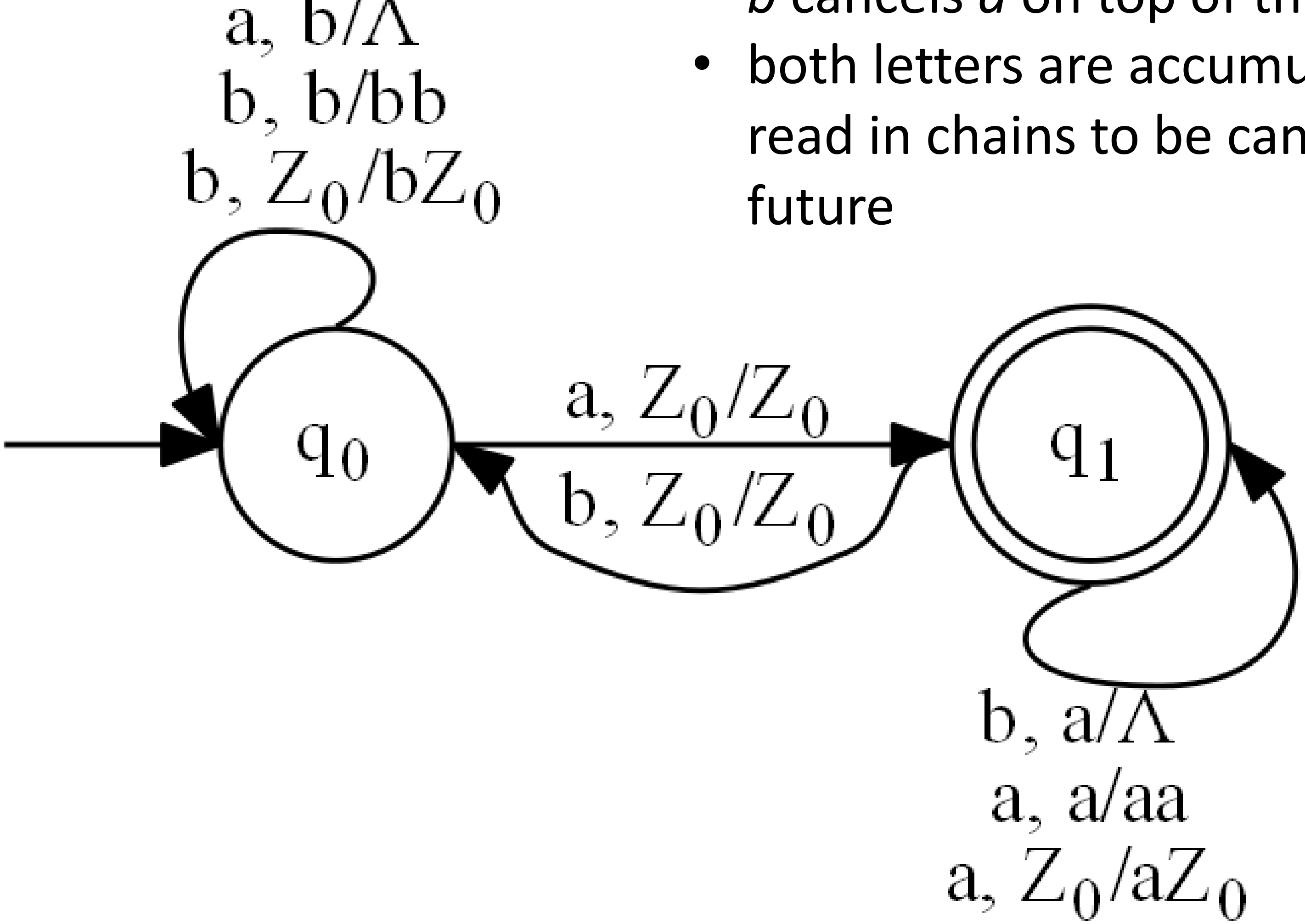
- M is deterministic. It never has a choice of moves



Example: Language $L = \{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$

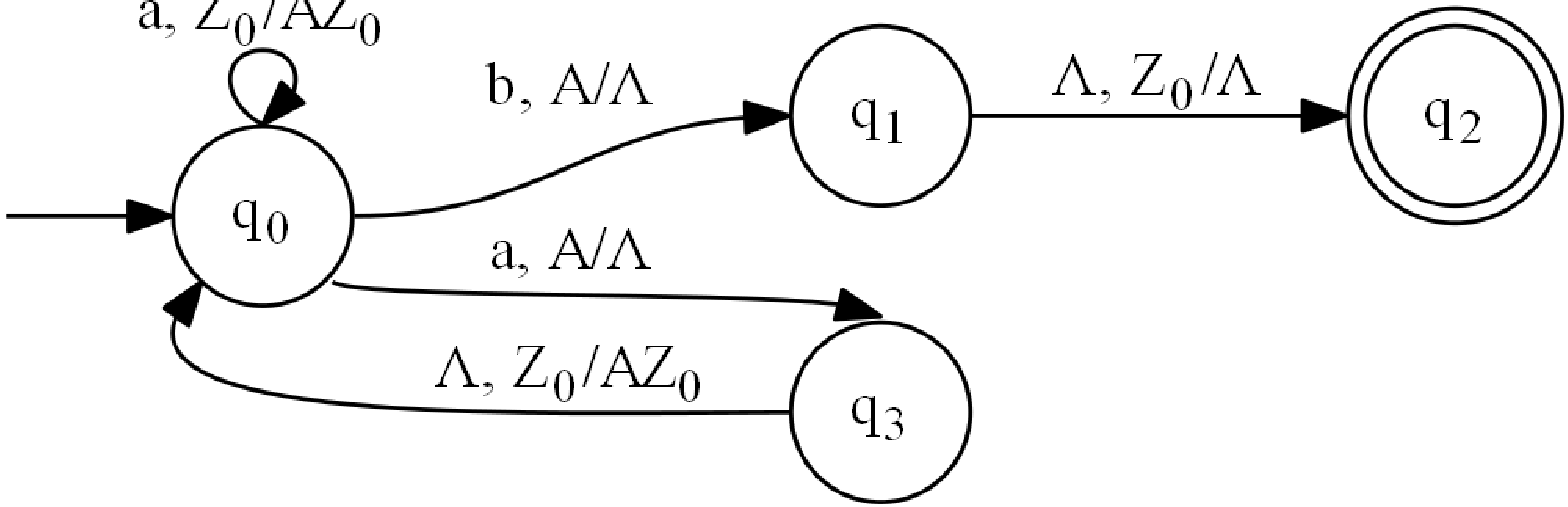
Example: Language $L = \{x \in \{a,b\}^* \mid n_a(x) > n_b(x)\}$

- *a* cancels *b* on top of the stack
- *b* cancels *a* on top of the stack
- both letters are accumulated if read in chains to be canceled in future



Example: Language L for regular expression aa^*b. Try to minimize the size of stack.

Example: Language L for regular expression aa^*b. Try to minimize the size of stack.
$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, Z_0\}$



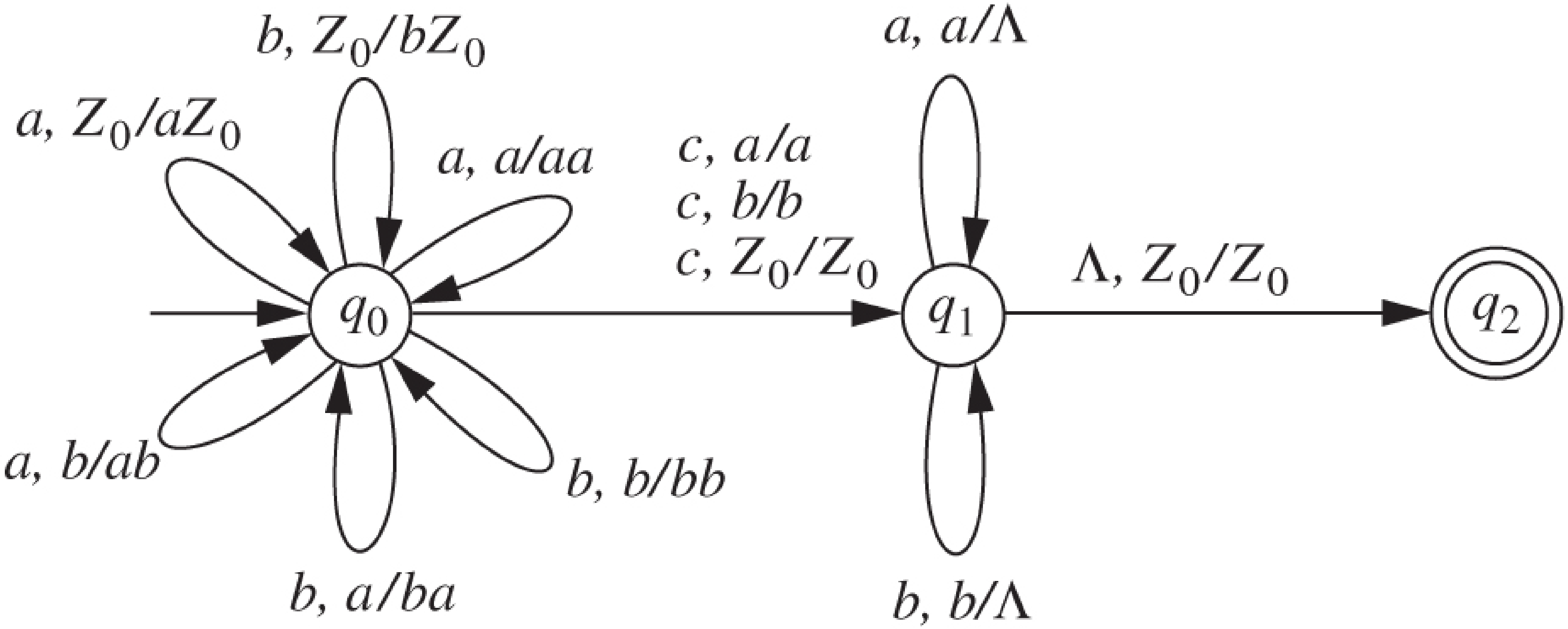
$$(q_0, aaab, Z_0) \vdash (q_0, aab, AZ_0) \vdash (q_3, ab, Z_0) \vdash (q_0, ab, AZ_0) \vdash (q_3, b, Z_0) \vdash (q_0, b, AZ_0) \vdash (q_1, \Lambda, Z_0) \vdash (q_2, \Lambda, \Lambda)$$

Example: Language $SimplePal = \{xcr(x) \mid x \in \{a, b\}^*\}$

Example: Language $SimplePal = \{xcr(x) \mid x \in \{a, b\}^*\}$

The moves by which the string *abcba* is accepted are shown below:

$$(q_0, abcba, Z_0) \vdash (q_0, bcba, aZ_0) \vdash (q_0, cba, baZ_0) \vdash (q_1, ba, baZ_0)$$
$$\vdash (q_1, a, aZ_0) \vdash (q_1, \Lambda, Z_0) \vdash (q_2, \Lambda, Z_0)$$



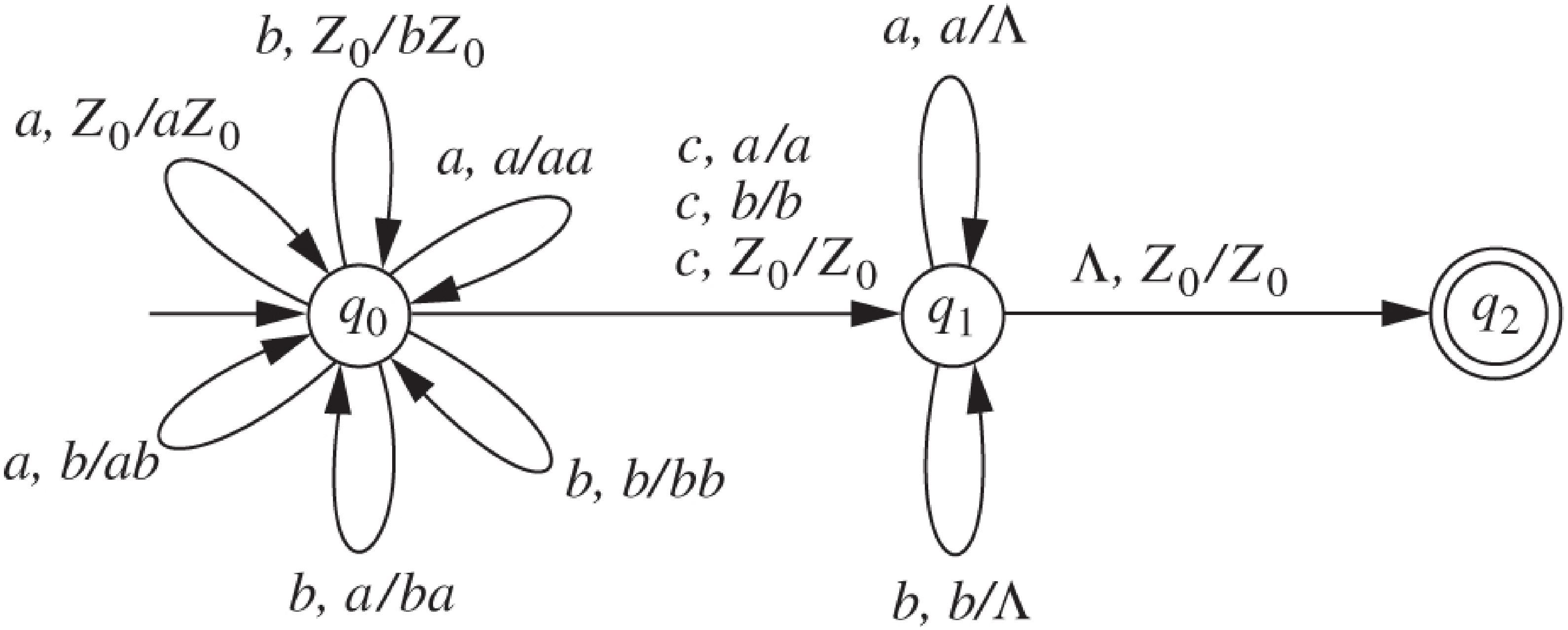
Example: Language $SimplePal = \{xcr(x) \mid x \in \{a, b\}^*\}$

A string can fail to be accepted either because it is never processed completely $\boxed{\text{or}}$ because the current state at the end of the processing is not an accepting state. These two scenarios are illustrated by the strings *acab* and *abc*, respectively:

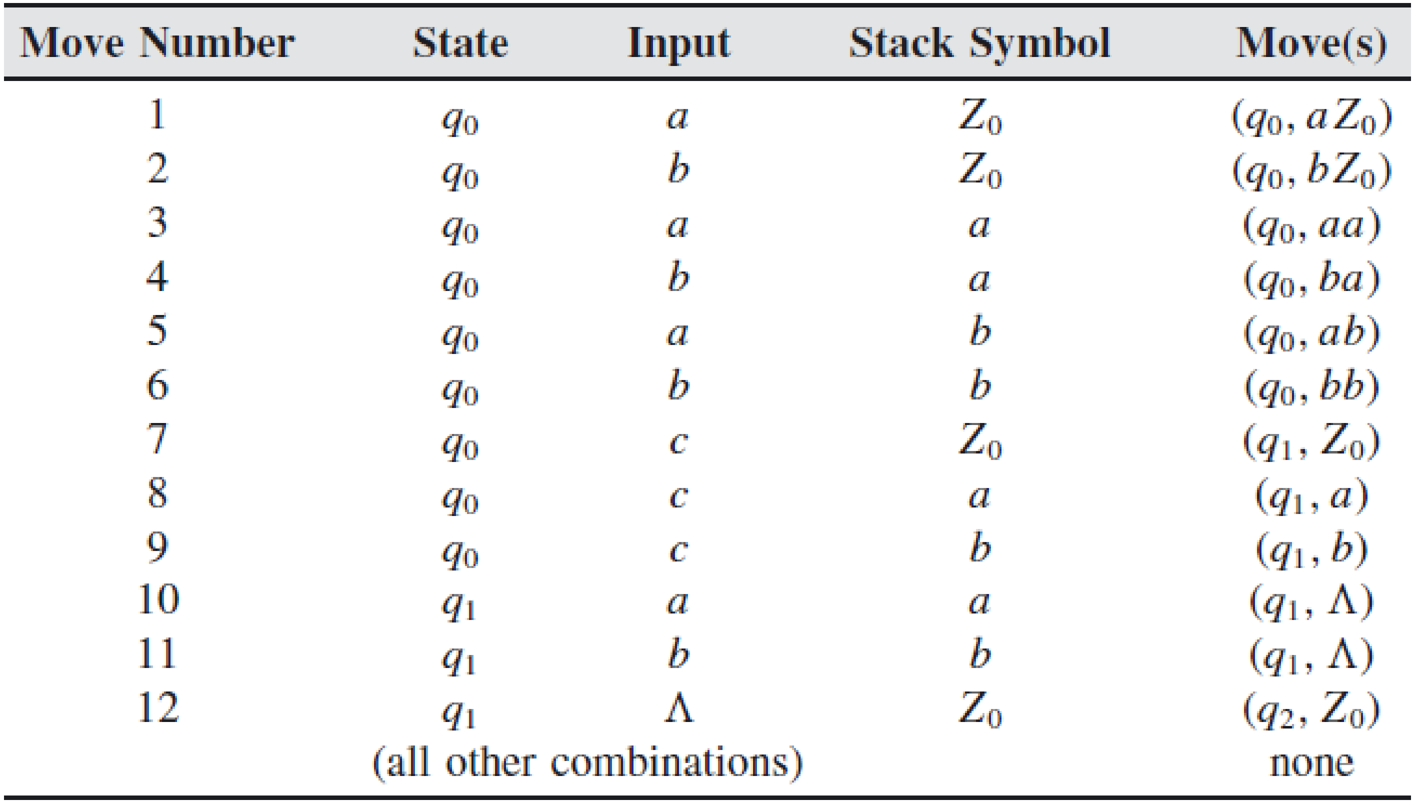
$$(q_0, acab, Z_0) \vdash (q_0, cab, aZ_0) \vdash (q_1, ab, aZ_0) \vdash (q_1, b, Z_0) \vdash (q_2, b, Z_0)$$

$$(q_0, abc, Z_0) \vdash (q_0, bc, aZ_0) \vdash (q_0, c, baZ_0) \vdash (q_1, \Lambda, baZ_0)$$

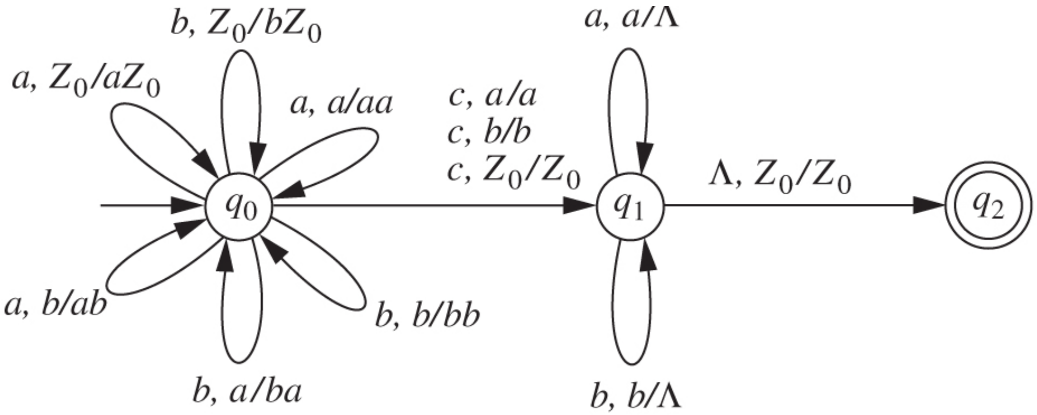
In the first case, although the sequence of moves ends in the accepting state, it is not *acab* that has been accepted, but only the prefix *aca*.



Example: Language $SimplePal = \{xcr(x) \mid x \in \{a, b\}^*\}$

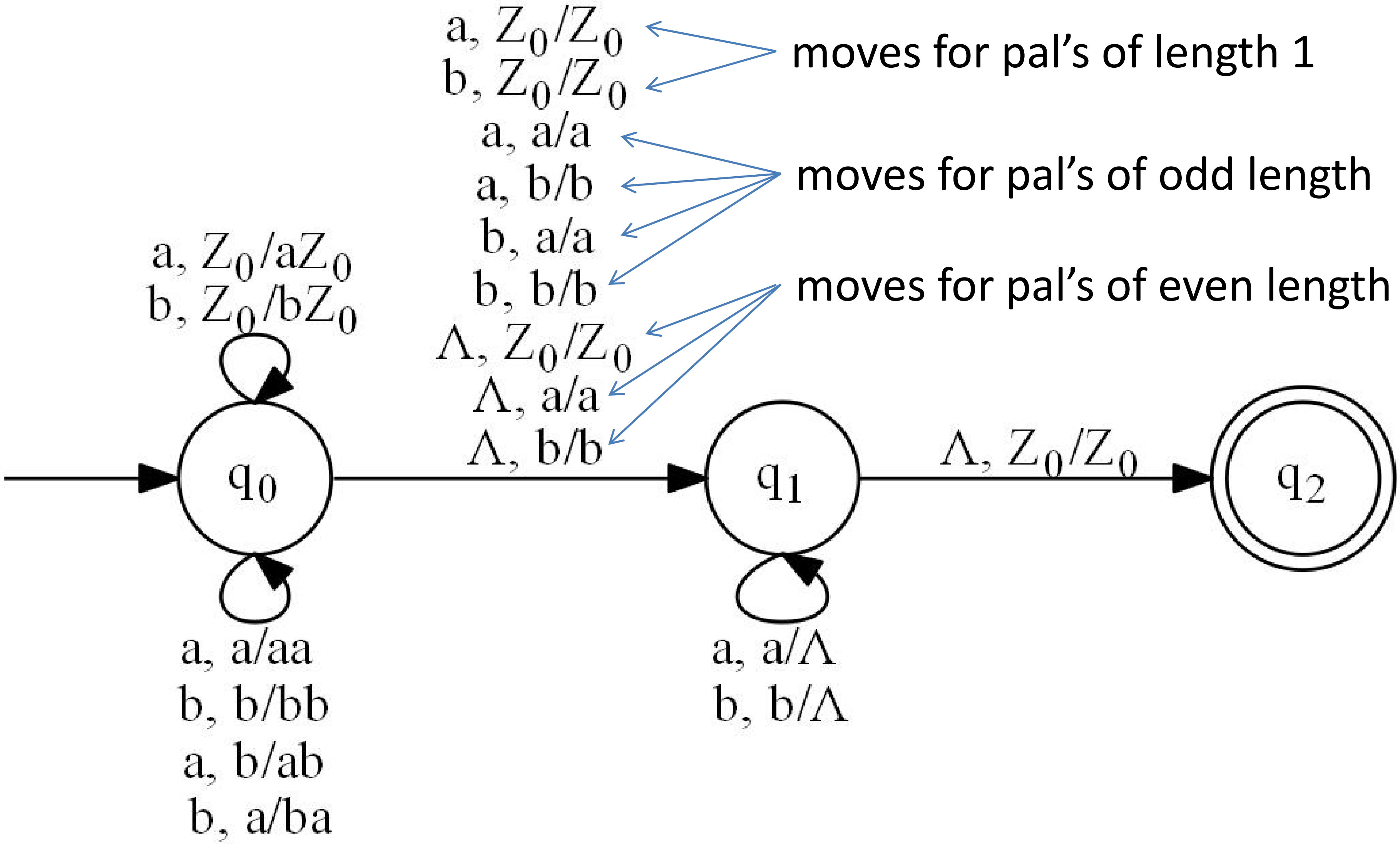


Move Number	State	Input	Stack Symbol	Move(s)
1	q_0	a	Z_0	(q_0, aZ_0)
2	q_0	b	Z_0	(q_0, bZ_0)
3	q_0	a	a	(q_0, aa)
4	q_0	b	a	(q_0, ba)
5	q_0	a	b	(q_0, ab)
6	q_0	b	b	(q_0, bb)
7	q_0	c	Z_0	(q_1, Z_0)
8	q_0	c	a	(q_1, a)
9	q_0	c	b	(q_1, b)
10	q_1	a	a	(q_1, Λ)
11	q_1	b	b	(q_1, Λ)
12	q_1	Λ	Z_0	(q_2, Z_0)
	(all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



This PDA is also deterministic

Example: Language $Pal = \{xr(x) \mid x \in \{a,b\}^*\}$



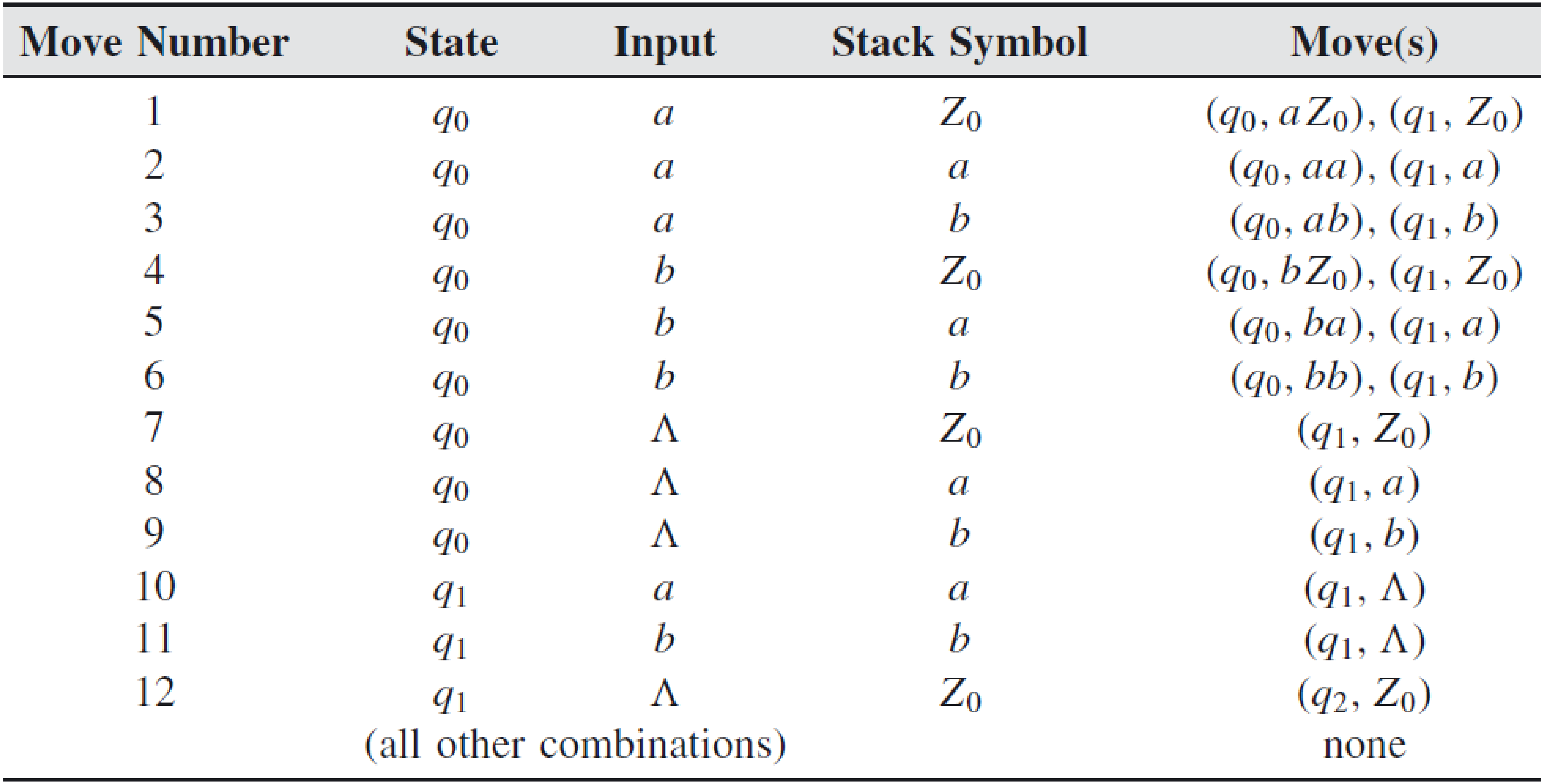
moves for pal’s of length 1

moves for pal’s of odd length

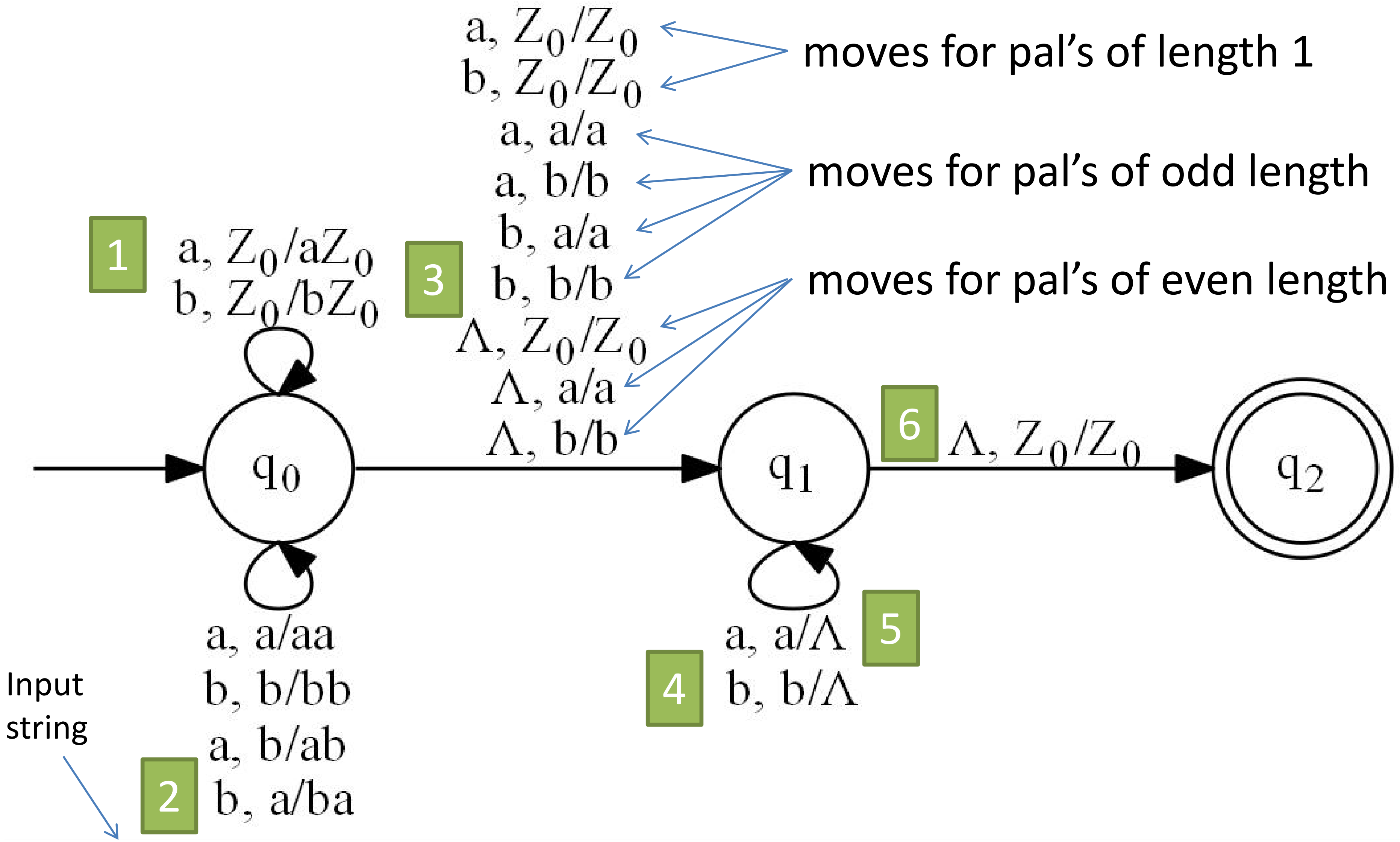
moves for pal’s of even length

This PDA is not deterministic

Example: Language $Pal = \{xr(x) \mid x \in \{a, b\}^*\}$



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



moves for pal's of length 1

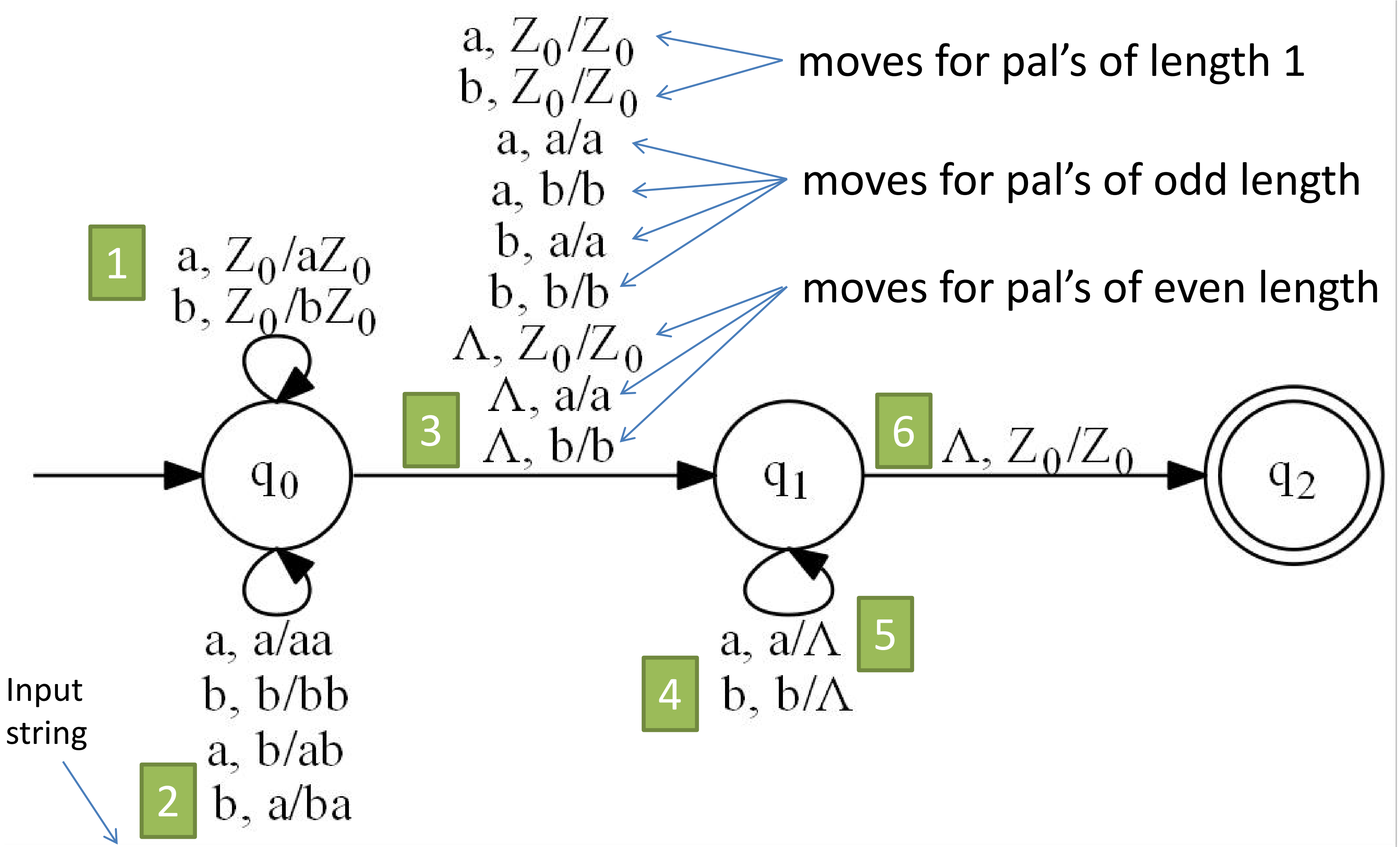
moves for pal's of odd length

moves for pal's of even length

Input string

$$(q_0, abbba, Z_0) \vdash (q_0, bbba, aZ_0) \vdash (q_0, bba, baZ_0)$$

$$\vdash (q_1, ba, baZ_0) \vdash (q_1, a, aZ_0) \vdash (q_1, \Lambda, Z_0) \vdash (q_2, \Lambda, Z_0)$$



moves for pal's of length 1

moves for pal's of odd length

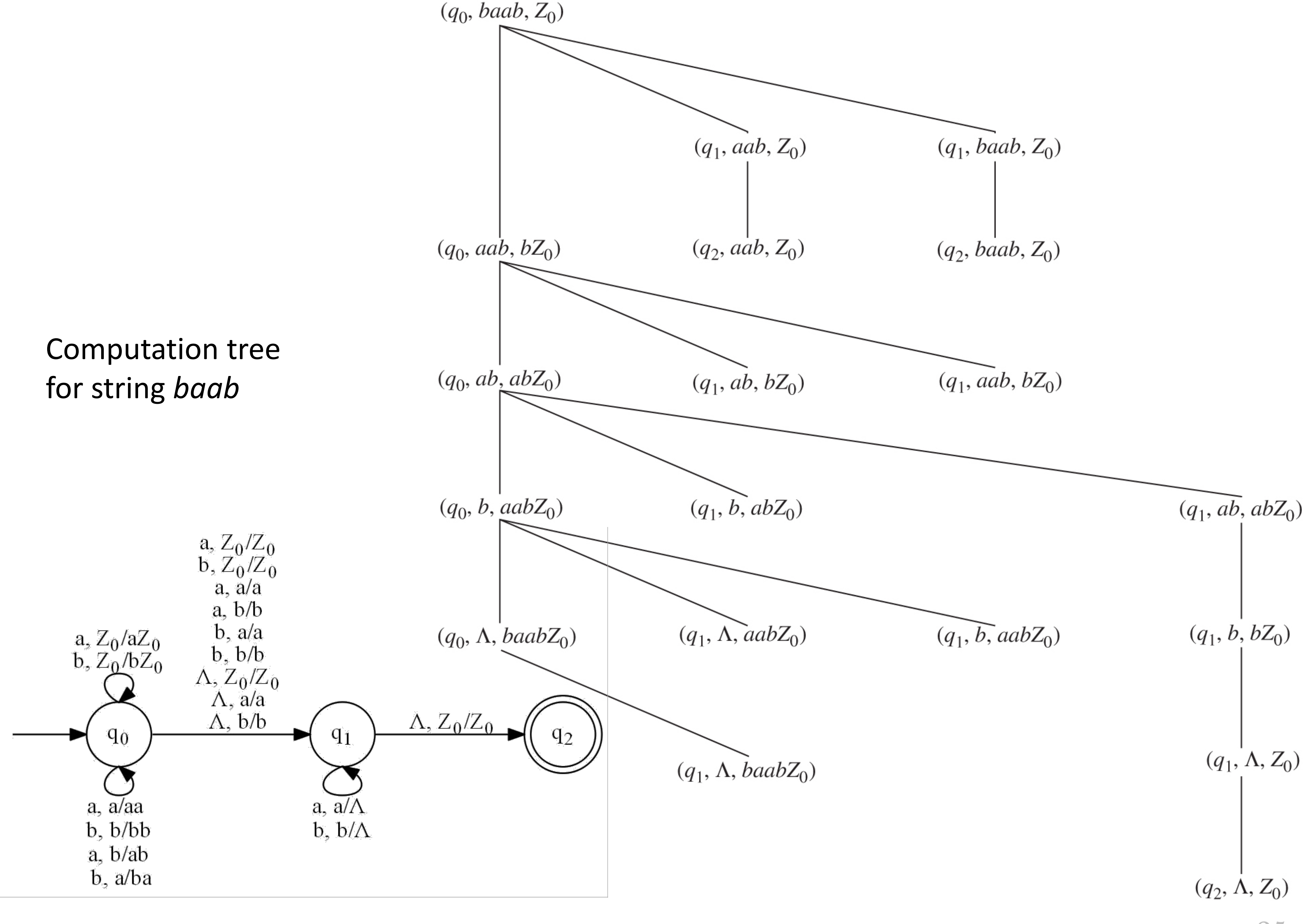
moves for pal's of even length

Input string

$$(q_0, abba, Z_0) \vdash (q_0, bba, aZ_0) \vdash (q_0, ba, baZ_0)$$

$$\vdash (q_1, ba, baZ_0) \vdash (q_1, a, aZ_0) \vdash (q_1, \Lambda, Z_0) \vdash (q_2, \Lambda, Z_0)$$

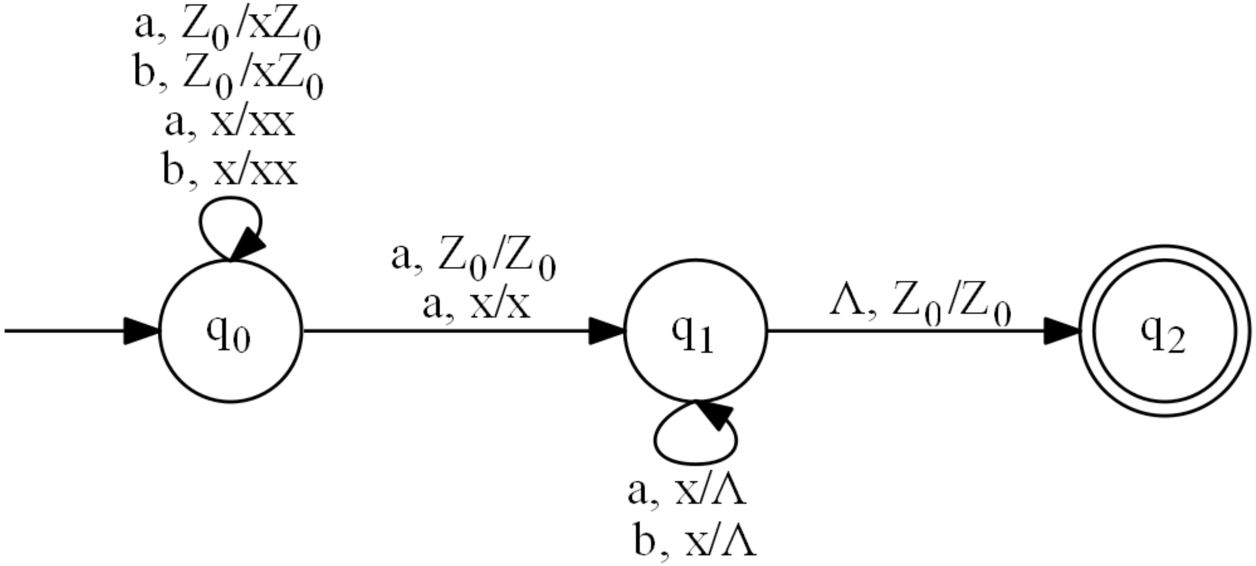
Computation tree for string *baab*



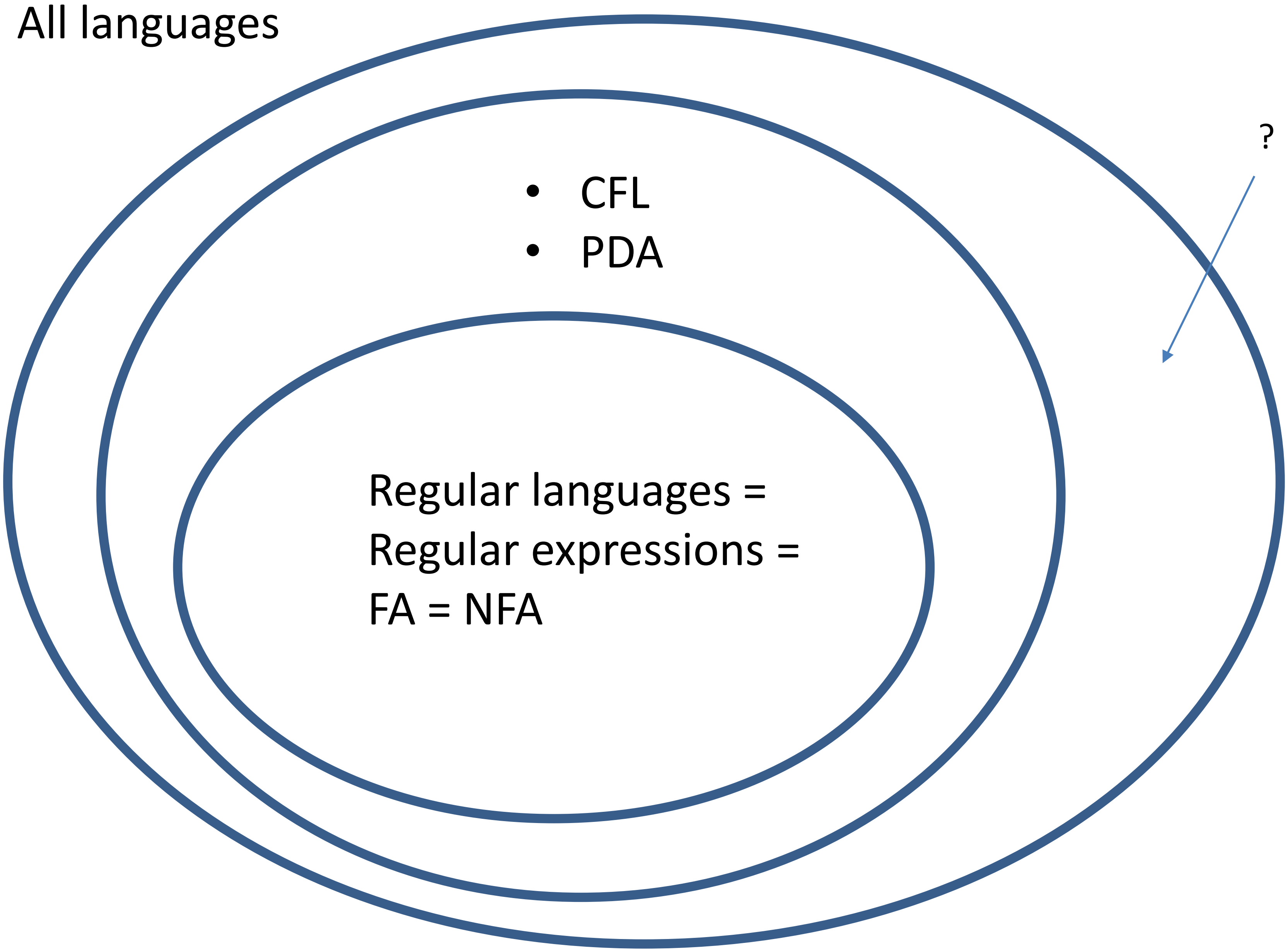
Example: Draw PDA for the language of all odd-length strings with middle a.

State	Input	Stack symbol	Move(s)
q_0	a	Z_0	$(q_0, xZ_0), (q_1, Z_0)$
q_0	b	Z_0	(q_0, xZ_0)
q_0	a	x	$(q_0, xx), (q_1, x)$
q_0	b	x	(q_0, xx)
q_1	a	x	(q_1, Λ)
q_1	b	x	(q_1, Λ)
q_1	Λ	Z_0	(q_2, Z_0)
(all other combinations)			none

Stack symbol that is not an input symbol



This is what we know about languages …



All languages

?

- CFL
- PDA

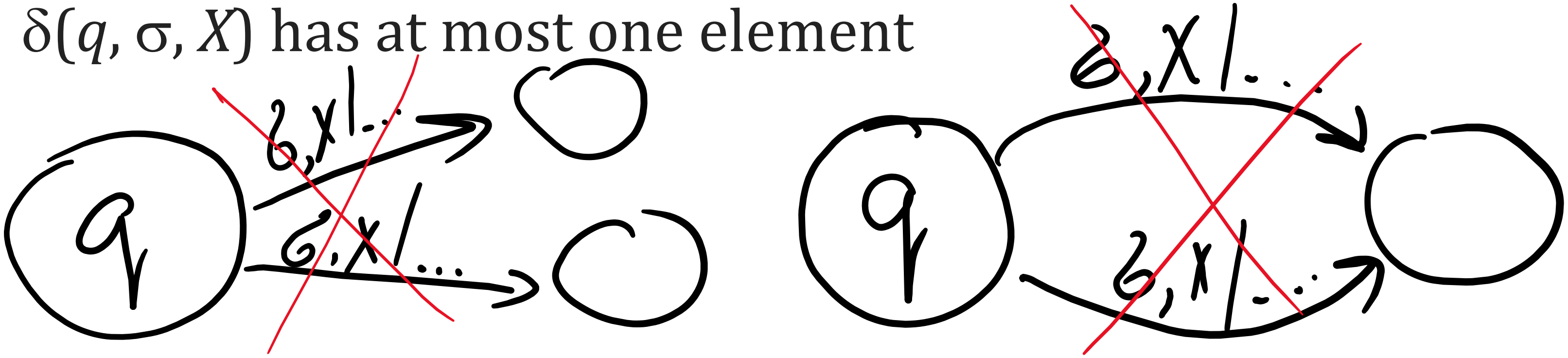
Regular languages =
Regular expressions =
FA = NFA

Deterministic Pushdown Automata

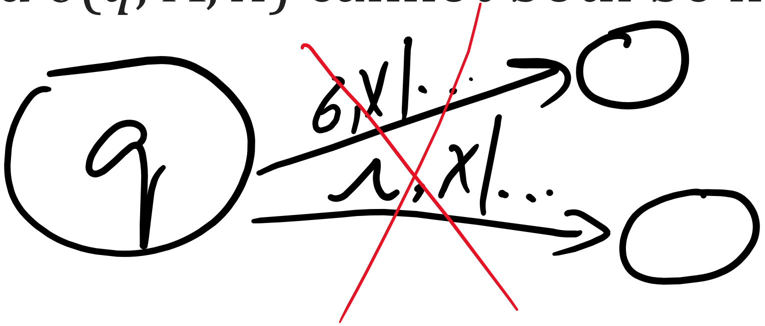
- Definition 5.10: A pushdown automaton $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is *deterministic* if it satisfies both of the following conditions:
  - For every $q \in Q$, every σ in $\Sigma \cup \{\Lambda\}$, and every $X \in \Gamma$, the set $\delta(q, \sigma, X)$ has at most one element
  - For every $q \in Q$, every $\sigma \in \Sigma$, and every $X \in \Gamma$, the two sets $\delta(q, \sigma, X)$ and $\delta(q, \Lambda, X)$ cannot both be nonempty
- A language L is a *deterministic context-free language* (DCFL) if there is a deterministic PDA (DPDA) accepting L

Deterministic Pushdown Automata

- Definition: A PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is *deterministic* if it satisfies both of the following conditions:
  - For every $q \in Q$, every σ in $\Sigma \cup \{\Lambda\}$, and every $X \in \Gamma$, the set $\delta(q, \sigma, X)$ has at most one element



  - For every $q \in Q$, every $\sigma \in \Sigma$, and every $X \in \Gamma$, the two sets $\delta(q, \sigma, X)$ and $\delta(q, \Lambda, X)$ cannot both be nonempty



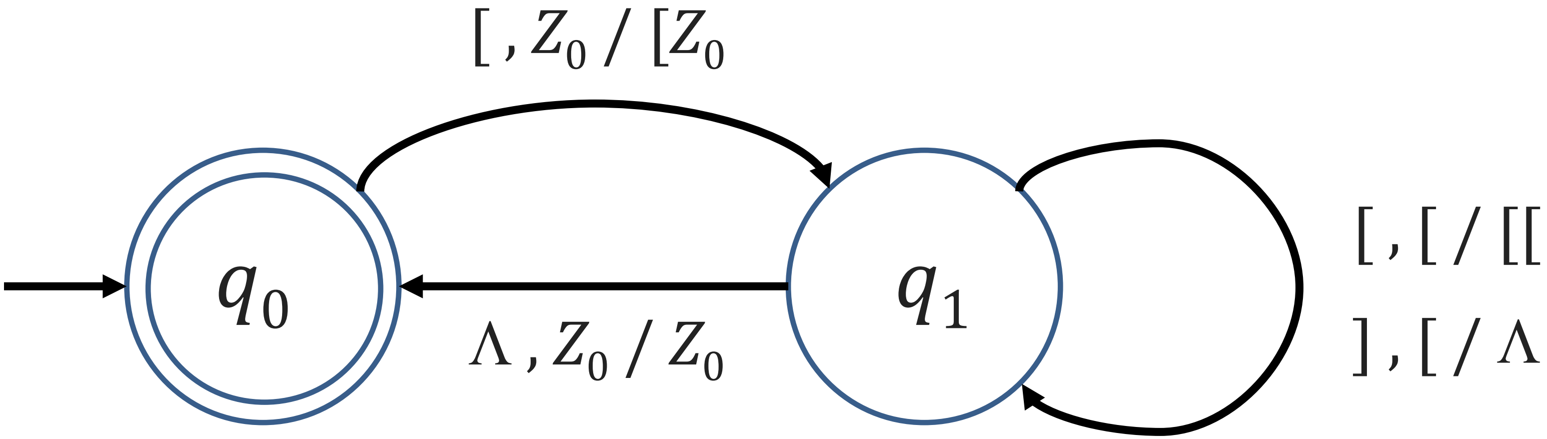
- A language L is a *deterministic context-free language* (DCFL) if there is a deterministic PDA (DPDA) accepting L

Deterministic Pushdown Automata

- One example is the PDA accepting $AnBn$
- Another example: the language of balanced strings of brackets that is “a string of left and right brackets is **balanced** if no prefix has more right than left and there are equal numbers of left and right”
  - Two states q_0 and q_1, where q_0 is the accepting state
  - Input symbols are ‘[‘ and ‘]’
  - Stack symbols are the input symbols and Z_0

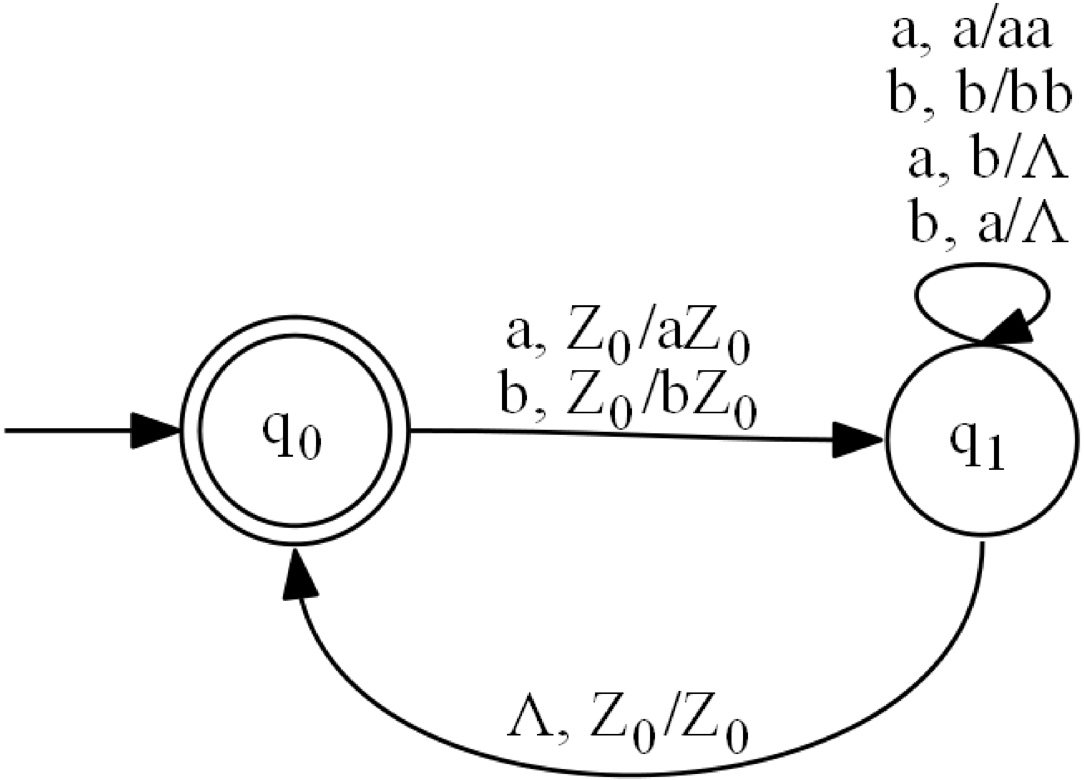
Deterministic PDA: The transition table for a PDA accepting the language of balanced strings of brackets

Move #	State	Input	Stack top	Move
1	q_0	[	Z_0	$(q_1, [Z_0)$
2	q_1	[	[	$(q_1, [[)$
3	q_1	]	[	(q_1, Λ)
4	q_1	Λ	Z_0	(q_0, Z_0)
(all other combinations)				none



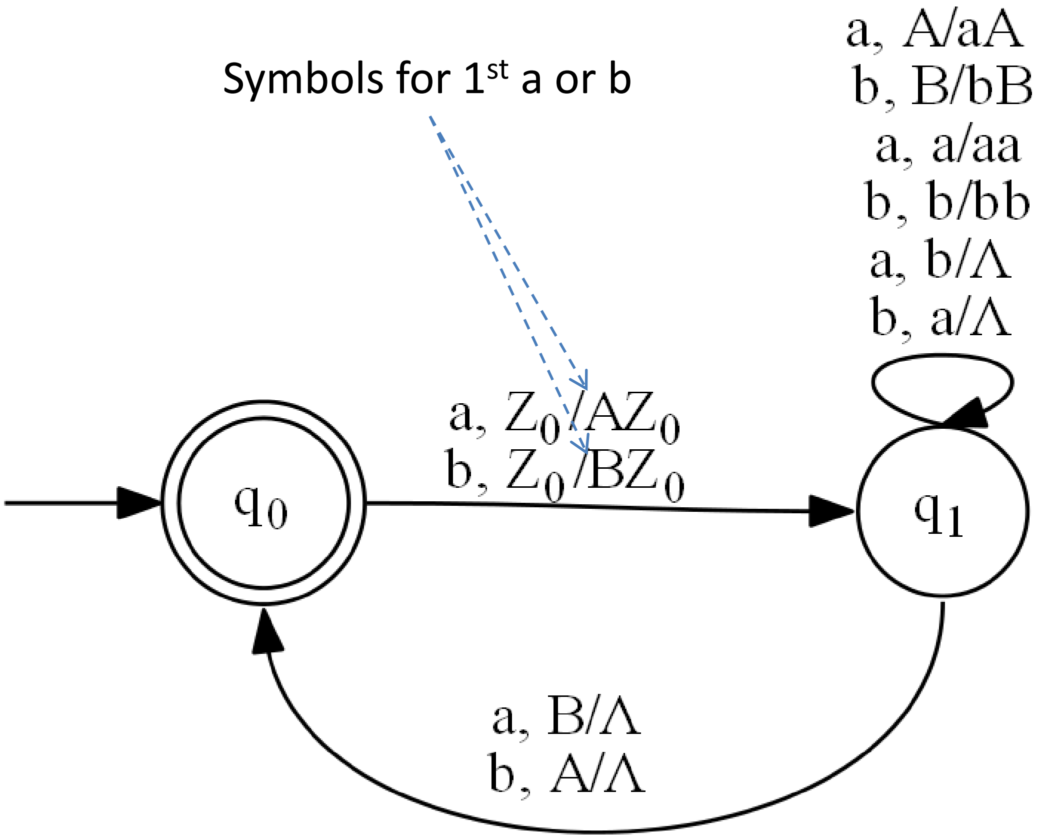
It is easy to prove by induction the equivalence of two languages (see page 173).

Example: Language $AEqB = \{x \in \{a,b\}^* \mid n_a(x) = n_b(x)\}$



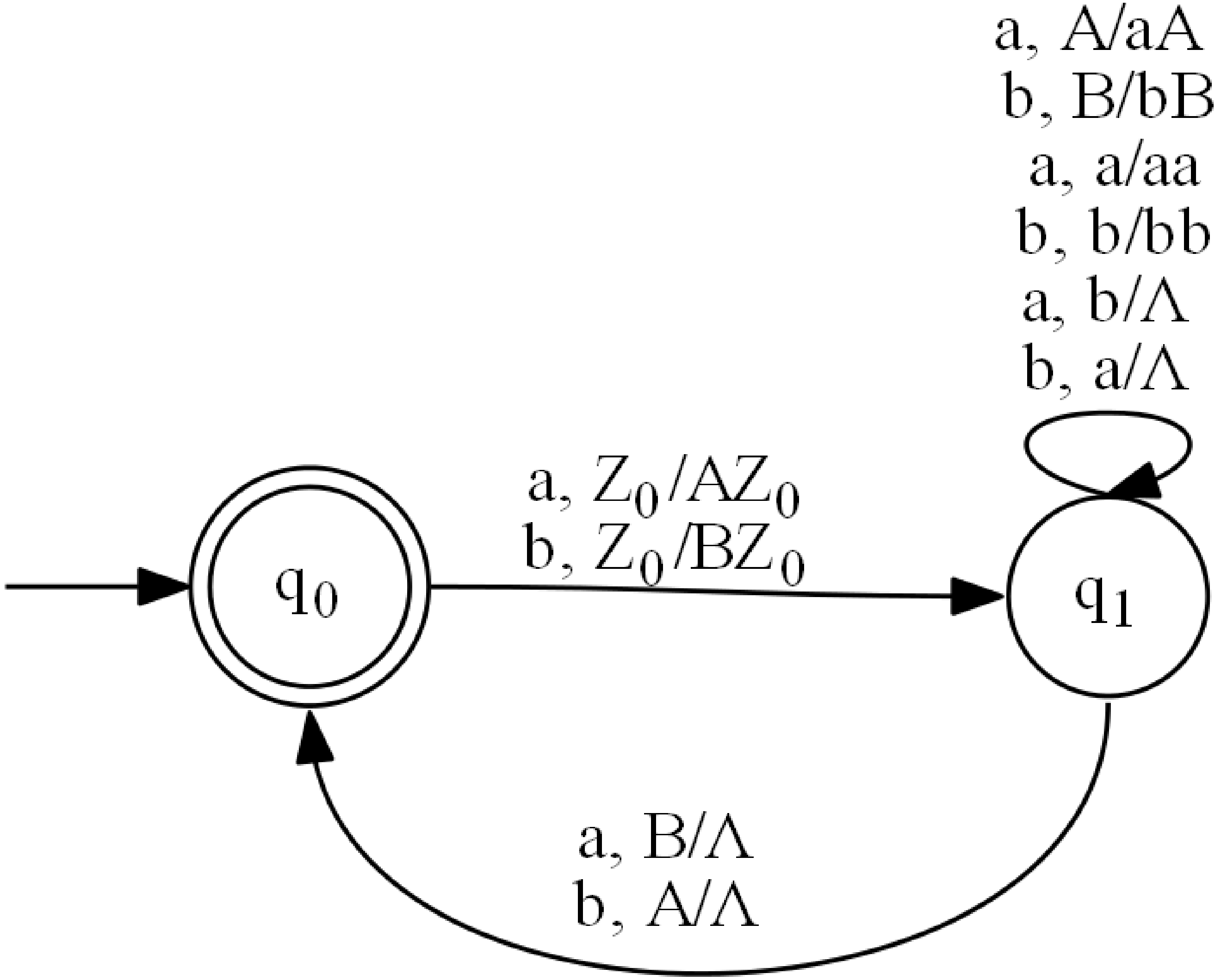
According to the definition of DPDA, Λ-transitions are permissible, if they do not allow the device a choice between reading a symbol and making a Λ-transition. This PDA is deterministic, because the only Λ-transition is in q_1 with top stack symbol Z_0, and with that combination of state and stack symbol there are no other moves.

Example: Language $AEqB = \{x \in \{a, b\}^* \mid n_a(x) = n_b(x)\}$



We can modify the DPDA such that no Λ-transition is used. We must provide a way for the device to determine whether the symbol currently on the stack is the only one other than Z_0; an easy way to do this to use special symbols, say A and B, to represent the first extra a or extra b, respectively.

Example: Language $AEqB = \{x \in \{a, b\}^* \mid n_a(x) = n_b(x)\}$



$$(q_0, abbaaabb, Z_0) \vdash (q_1, bbaaabb, AZ_0) \vdash (q_0, baaabb, Z_0) \vdash (q_1, aaabb, BZ_0)$$

$$\vdash (q_0, aabb, Z_0) \vdash (q_1, abb, AZ_0) \vdash (q_1, bb, aAZ_0) \vdash (q_1, b, AZ_0) \vdash (q_0, \Lambda, Z_0)$$

Constructing a PDA from a Given CFG

- We’ll examine two ways of constructing a PDA from an arbitrary CFG
  - In both cases the PDA will be nondeterministic
- In both approaches, the PDA attempts to simulate a derivation in the grammar, using the stack to hold portions of the current string
  - The PDA terminates the computation if it finds that the derivation-in-progress is not consistent with the input string
  - It attempts to construct a derivation tree from the input string

A PDA from a Given CFG

- The two approaches are *top-down* and *bottom-up,* which refer to the way the tree is constructed
  - We'll describe for each one the PDA obtained from the grammar and indicate why it accepts the language
- The top-down PDA:
  - Begins by placing the start symbol of the grammar on the stack
  - From this point, each step in the construction of the derivation tree consists of replacing a variable A that is currently on top of the stack by the right side of a grammar production $A \to \alpha$

A PDA from a Given CFG (cont’d.)

- The top-down PDA: (cont’d.)
  - This step corresponds to building the portion of the tree containing that variable-node and its children
  - The intermediate moves of the PDA are to remove terminal symbols from the stack as they are produced and match them with input symbols
  - To the extent that they continue to match, the derivation being simulated is consistent with the input string
  - Because the variable replaced in each step is the leftmost one, *we are simulating a leftmost derivation*

A PDA from a Given CFG (cont'd.)

Let $G = (V, \Sigma, S, P)$ be a CFG. The nondeterministic top-down PDA corresponding to G is $NT(G)=(Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$, defined as follows:

- $Q = \{q_0, q_1, q_2\}$, $A = \{q_2\}$, $\Gamma = V \cup \Sigma \cup \{Z_0\}$
- The initial move is "push S to stack", i.e.,

$$\delta(q_0, \Lambda, Z_0) = \{(q_1, SZ_0)\}$$

- The only move to the accepting state is $\delta(q_1, \Lambda, Z_0) = \{(q_2, Z_0)\}$
- The moves from q_1 are the following:
  - For every $X \in V$, $\delta(q_1, \Lambda, X) = \{(q_1, \alpha) \mid X \to \alpha \in P\}$
  - For every $\sigma \in \Sigma$, $\delta(q_1, \sigma, \sigma) = \{(q_1, \Lambda)\}$

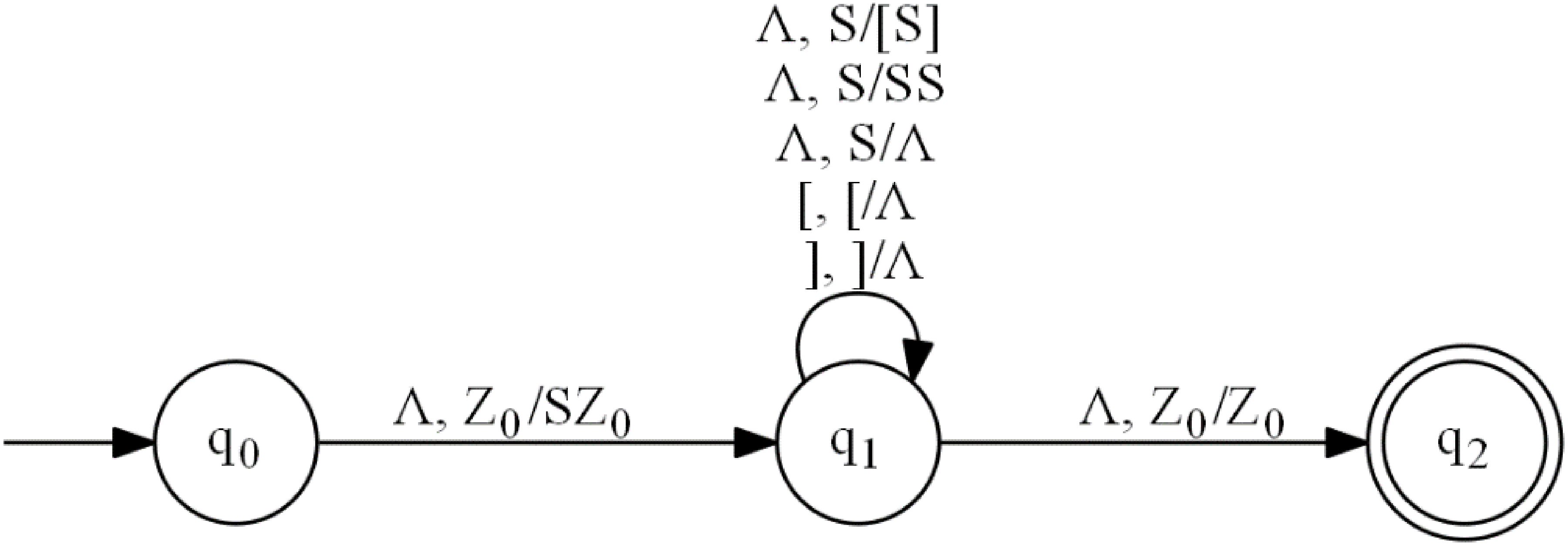
A PDA from a Given CFG (cont’d.)

- After the initial move, before the final move, the PDA remains in state q_1
  - The only moves are to replace a variable with the right side of a production and to match a terminal symbol on the stack with an input symbol and discard both
- Theorem 5.18: If G is a CFG, then the nondeterministic top-down PDA $NT(G)$ accepts the language $L(G)$
  - Proof: see book

Example: CFG→PDA accepting *Balanced*

- The language of balanced strings of brackets that is “a string of left and right brackets is **balanced** if no prefix has more right than left and there are equal numbers of left and right”

$$S \rightarrow [S] \mid SS \mid \Lambda$$



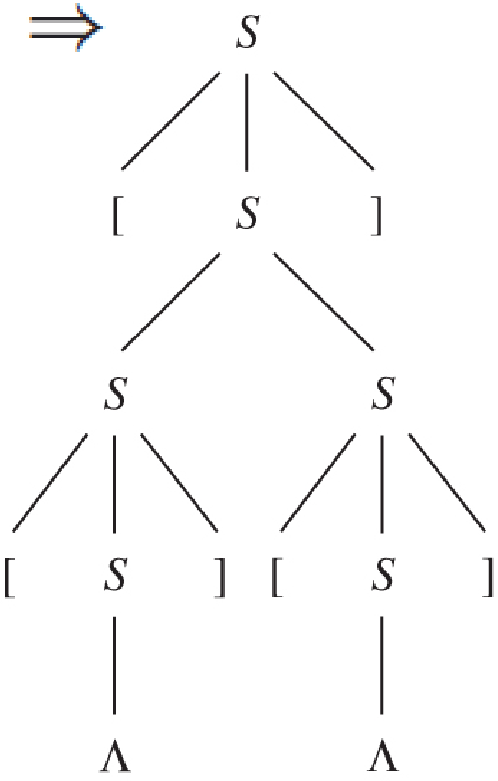
Example: CFG→PDA accepting *Balanced*

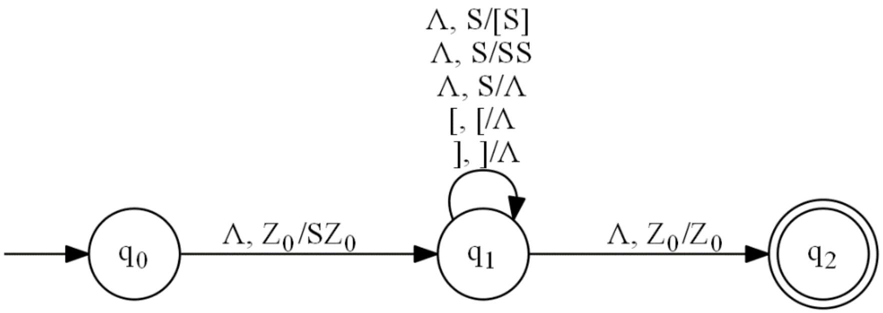
- The language of balanced strings of brackets that is “a string of left and right brackets is **balanced** if no prefix has more right than left and there are equal numbers of left and right”

$$S \rightarrow [S] \mid SS \mid \Lambda$$

$$S \Rightarrow [S] \Rightarrow [SS] \Rightarrow [[S]S] \Rightarrow [[]S] \Rightarrow [[][S]] \Rightarrow [[][]]$$

State	Input	Stack Symbol	Move
q_0	Λ	Z_0	(q_1, SZ_0)
q_1	Λ	S	$(q_1, [S]), (q_1, SS), (q_1, \Lambda)$
q_1	[	[	(q_1, Λ)
q_1	]	]	(q_1, Λ)
q_1	Λ	Z_0	(q_2, Z_0)
(all other combinations)			none





$$
\begin{array}{lllll}
(q_0,\ [\,[\,]\,[\,]\,],\ Z_0) & & & & \\
 & \vdash & (q_1,\ [\,[\,]\,[\,]\,],\ SZ_0) & & S \\
 & \vdash & (q_1,\ [\,[\,]\,[\,]\,],\ [S]Z_0) & \Rightarrow & [S] \\
 & \vdash & (q_1,\ [\,]\,[\,]\,],\ S]Z_0) & & \\
 & \vdash & (q_1,\ [\,]\,[\,]\,],\ SS]Z_0) & \Rightarrow & [SS] \\
 & \vdash & (q_1,\ [\,]\,[\,]\,],\ [S]S]Z_0) & \Rightarrow & [[S]S] \\
 & \vdash & (q_1,\ ]\,[\,]\,],\ S]S]Z_0) & & \\
 & \vdash & (q_1,\ ]\,[\,]\,],\ ]S]Z_0) & \Rightarrow & [[\,]S] \\
 & \vdash & (q_1,\ [\,]\,],\ S]Z_0) & & \\
 & \vdash & (q_1,\ [\,]\,],\ [S]]Z_0) & \Rightarrow & [[\,][S]] \\
 & \vdash & (q_1,\ ]\,],\ S]]Z_0) & & \\
 & \vdash & (q_1,\ ]\,],\ ]]Z_0) & \Rightarrow & [[\,][\,]] \\
 & \vdash & (q_1,\ ],\ ]Z_0) & & \\
 & \vdash & (q_1,\ \Lambda,\ Z_0) & & \\
 & \vdash & (q_2,\ \Lambda,\ Z_0) & &
\end{array}
$$

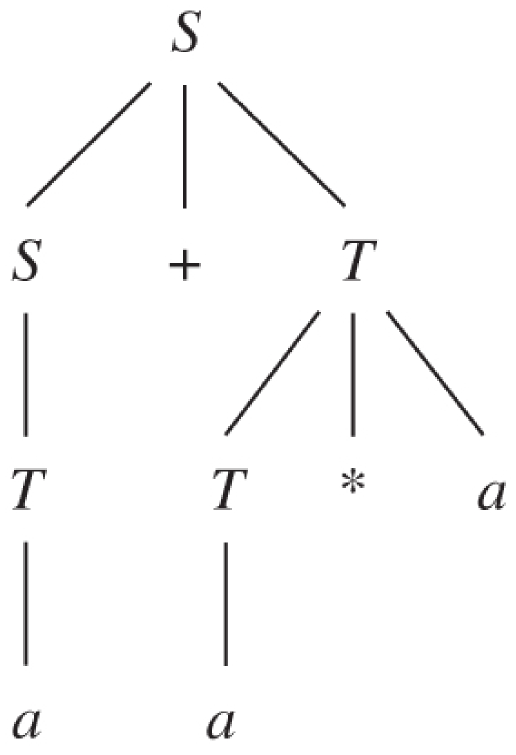
A PDA from a Given CFG (cont'd.)

- The bottom-up PDA:
  - The tree is constructed from the bottom up
  - The PDA simulates the reverse of a derivation starting with the last move and ending with the first
  - Another thing that's reversed is the order in which we read the symbols on the stack when doing a "reduction": the reduction corresponding to $A \to \beta$ is performed when the string r(β) appears on the top of the stack; it is replaced by *A*.
  - *The simulated derivation is a rightmost derivation*

Example: Bottom-Up PDA for simplified Algebraic Expressions

$$S \rightarrow S + T \mid T \qquad T \rightarrow T * a \mid a$$

Reduction	Stack (reversed)	Unread Input	Derivation Step
	Z_0	$a + a * a$	
	$Z_0\ \underline{a}$	$+ a * a$	
(1)	$Z_0\ \underline{T}$	$+ a * a$	$\Rightarrow\ a + a * a$
(2)	$Z_0\ S$	$+ a * a$	$\Rightarrow\ T + a * a$
	$Z_0\ S +$	$a * a$	
	$Z_0\ S + \underline{a}$	$* a$	
(3)	$Z_0\ S + T$	$* a$	$\Rightarrow\ S + a * a$
	$Z_0\ S + T\ *$	a	
	$Z_0\ S + \underline{T * a}$		
(4)	$Z_0\ \underline{S + T}$		$\Rightarrow\ S + T * a$
(5)	$Z_0\ S$		$\Rightarrow\ S + T$
	(accept)		S



Tree for a+a*a

A PDA from a Given CFG (cont'd.)

Let G=(V, Σ, S, P) be a CFG. The nondeterministic bottom-up PDA corresponding to G is NB(G)=(Q, Σ, Γ, q_0, Z_0, A, δ), defined as follows:

- For every $\sigma \in \Sigma$ and every $X \in \Gamma$, $\delta(q_0, \sigma, X)=\{(q_0, \sigma X)\}$, the *shift* moves
- For every production $B \rightarrow \alpha$, and every nonnull string $\beta \in \Gamma^*$, $(q_0, \Lambda, \mathrm{r}(\alpha)\,\beta) \vdash^* (q_0, B\,\beta)$, where this *reduction* is a sequence of one or more moves in which, if there is more than one, the intermediate configurations involve other states that are specific to this sequence and appear in no other moves of NB(G)
- $(q_1, \Lambda) \in \delta(q_0, \Lambda, S)$ and $\delta(q_1, \Lambda, Z_0)=\{(q_2, Z_0)\}$

A PDA from a Given CFG (cont’d.)

- Theorem 5.23: If G is a CFG, then the non-deterministic bottom-up PDA $NB(G)$ accepts $L(G)$
  - Proof: see book

The Pumping Lemma for Context-Free Languages

- It's easy to find a language that cannot be accepted by a finite automaton, even if proving it is a little harder
  - For example, *AnBn* cannot be accepted by a FA, because with only a finite number of states, we can't keep track of how many *a*'s we've seen
  - It might be argued, in a similar way, that neither $AnBnCn = \{a^n b^n c^n \mid n \geq 0\}$ nor $XX = \{xx \mid x \in \{a,b\}^*\}$ can be accepted by a PDA

Even if the PDA is nondeterministic, and we know when do we need to switch from the first *x* to the second *x*, we cannot access the first symbol of *x* to match it because it is at the bottom of the stack!

The Pumping Lemma for Context-Free Languages (cont'd.)

- The way a PDA processes $a^i b^j c^k$ allows it to confirm that $i = j$ but not to remember that number long enough to compare it to k
- One way to prove *AnBn* is not regular is to use the pumping lemma for regular languages
- Now we'll establish a result for CFLs that is similar to the pumping lemma, but a little more complicated
- The basic idea is that …

A sufficiently long derivation in a grammar *G* will have to contain a *self-embedded* variable

The Pumping Lemma for Context-Free Languages (cont'd.)

- For instance, in $S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz$, the string derived from the first occurrence of A also includes an occurrence of A
  - $S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vw^kAy^kz \Rightarrow^* vw^kxy^kz$ must also be a valid derivation, for every $k \geq 1$, and

    $S \Rightarrow^* vAz \Rightarrow^* vxz = vw^0xy^0z$ is also valid
  - This observation will be useful if we can guarantee that the strings w and y are not both null, and even more useful if we can impose some other restrictions on the five strings v, w, x, y, and z of terminals

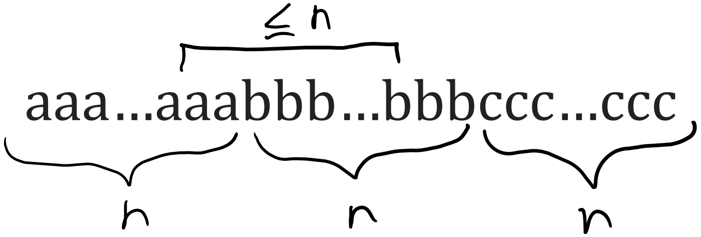
The Pumping Lemma for Context-Free Languages (cont'd.)

- **Theorem**: Suppose L is a CFL
  - Then there is an integer n so that for every $u \in L$ with $|u| \geq n$, u can be written as $u = vwxyz$ so that:
    - (c1) $|wy| > 0$
    - (c2) $|wxy| \leq n$
    - (c3) For every $m \geq 0$, $vw^mxy^mz \in L$

Applying the pumping lemma to *AnBnCn*

Suppose, for the sake of contradiction, that *AnBnCn* is a context-free language, and let n be the integer in the pumping lemma

- Let u (= $vwxyz$) be the string $a^n b^n c^n$
  - Then $u \in AnBnCn$ and $|u| \geq n$
  - Therefore, according to the pumping lemma, $u=vwxyz$ for some strings satisfying the three conditions
  - (c1) ➔ wy contains at least one symbol, $|wy| > 0$
  - (c2) ➔ wxy ($|wxy| \leq n$) contains no more than two distinct symbols.


aaa…aaabbb…bbbccc…ccc

  - If σ_1 is one of the three symbols that occurs in wy and σ_2 is one that doesn't, then vw^0xy^0z contains fewer than n occurrences of σ_1 and exactly n occurrences of σ_2
  - This is a contradiction because the third condition implies that vw^0xy^0z is in *AnBnCn* (not equal numbers of all three symbols)

Example: Prove that the following language

$$L = \{a^i b^j c^k \mid i \leq j \leq k\}$$

is not context-free.

- Let's take n from the PL, and initial string (of the form vwxyz) from L, $a^n b^n c^n$
- $|wxy| \leq n$. Notice that w cannot be simultaneously in a^n and b^n, since if we repeat w (we take w^2xy^2) then a's are mixed with b's and the resulting string is not in L. Similarly, w cannot be simultaneously in b^n and c^n. A similar observation holds for y.
- What if wxy is completely within a^n? If we pump up the string wxy and we take w^2xy^2, then the resulting string has the form $a^{n+k}b^nc^n$, for some $k > 0$ (since $|wy| > 0$), which is not in L. For similar reasons wxy cannot be completely within b^n, and c^n (in this case take $i = 0$).

Example (cont): Prove that the following language $L = \{a^i b^j c^k \mid i \le j \le k\}$ is not context-free.

- What if wxy is such that w is completely within a^n , and y is completely within b^n. If pump up the string wxy once and we obtain the string w^2xy^2, then the resulting string has the form $a^{n+k_1}b^{n+k_2}c^n$, with $k_1 + k_2 \ge 1$ (since $|wy| \ge 1$), which is not in L. We treat similarly the case where wxy is such that w is completely within b^n , and y is completely within c^n.

- These 3 cases complete the proof.