Pushdown Automata

Robert M. Keller

Harvey Mudd College

28 April 2005

Wanted:

- Similar to the DFA characterization of type 3 languages, we'd like a **machine characterization** of type 2 languages.
- We know that finite-state machines are inadequate.
- We need to add an extra memory component, one that permits **unbounded storage**.

A Proper Context-Free Language

- Example: $\{xcx^R \mid x \in \Sigma^*\}$ where $c \notin \Sigma$.
- Supposing $\Sigma = \{a, b\}$, give a context-free grammar for this language.
- Is this language regular?

Stacks to the Rescue

- By adding a stack to a DFA, we can accept non-regular languages.
- Example: $\{xcx^R \mid x \in \Sigma^*\}$ where $c \notin \Sigma$.
  - Begin reading symbols.
  - Until we encounter a 'c', **push** the symbols read onto a stack.
  - After 'c' is encountered, **pop** the symbols from the stack, comparing with the next symbol read, if any.
  - Accept when the stack is **empty**.
  - Have to check for no c's, more than one c, etc. and reject these cases.

PDA's

- The type of behavior described on the preceding slide is that of a (deterministic) PDA (DPDA).
- In general, PDA's are **non-deterministic**.
- The situation for PDA's is different from DFA's: there is **no subset construction**.

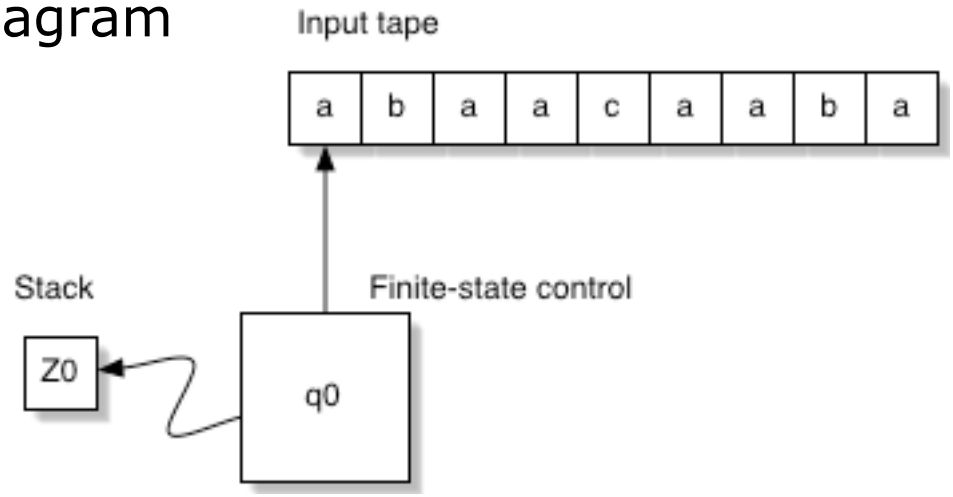
PDA Defined: $(Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$

- Q is a finite set of **control states**
- Σ is the **input** alphabet
- Γ is the **stack** alphabet
- q_0 is the **initial** control state
- Z_0 is the **initial** stack symbol
- $A \subseteq Q$ is the set of **accepting** control states
- δ is the **state transition relation** (or, in the case of a DPDA, **function**)

  δ: Q x $(\Sigma \cup \{\Lambda\})$ x $\Gamma \rightarrow$ finite subsets of Q x Γ^*

Say What?

PDA Diagram



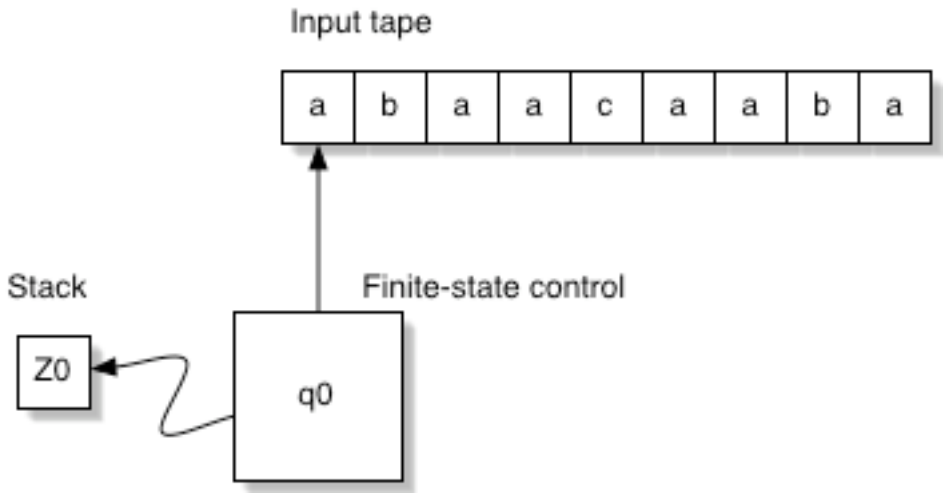
The **state** is generally of the form $(q, \gamma) \in Q \times \Gamma^*$,
i.e. (control-state, stack-contents).
Sometimes this is called a **configuration** or **instantaneous description (ID)**, to distinguish it from the control state. I prefer to call it the state, which is what it is.

The initial state is (q_0, Z_0). We will describe δ presently.

PDA Acceptance

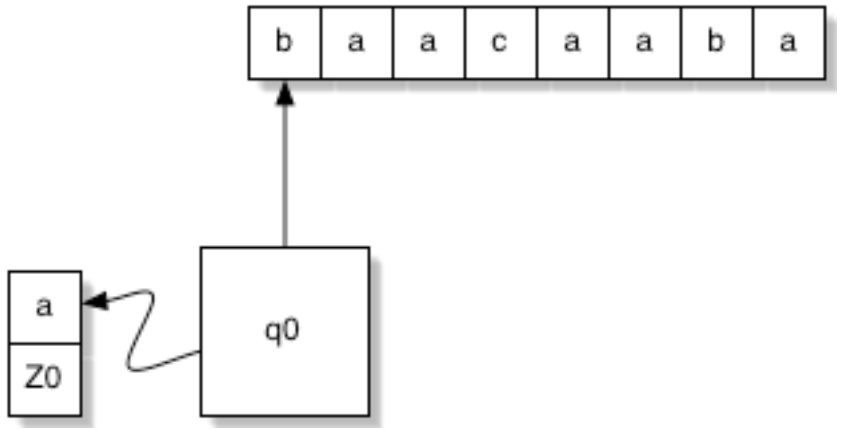
- There are different models for acceptance.
- **Final-state acceptance** means that the PDA accepts when it is in a **designated control state** *after the input has been entirely read*.
- **Empty-stack acceptance** means that the PDA accepts when its **stack is empty** *after the input has been entirely read*.
- The choice is a matter of convenience; the two can be shown **inter-convertible**.

Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

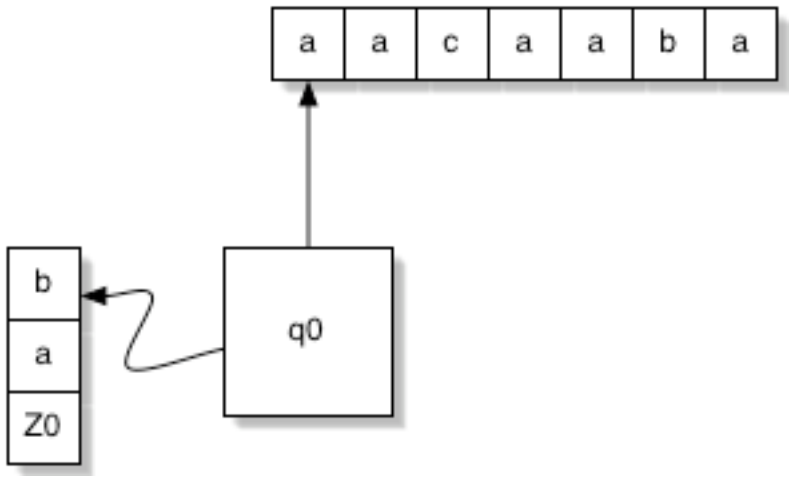


Note that control-state is in q_0 and will stay there until c is read.

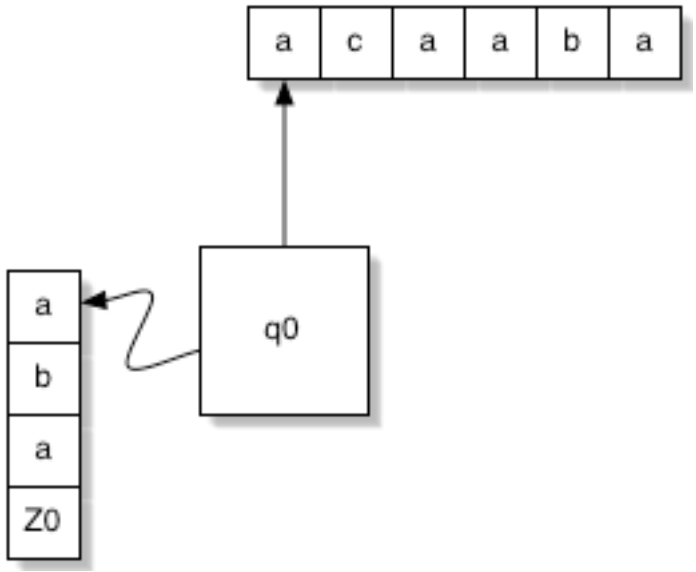
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



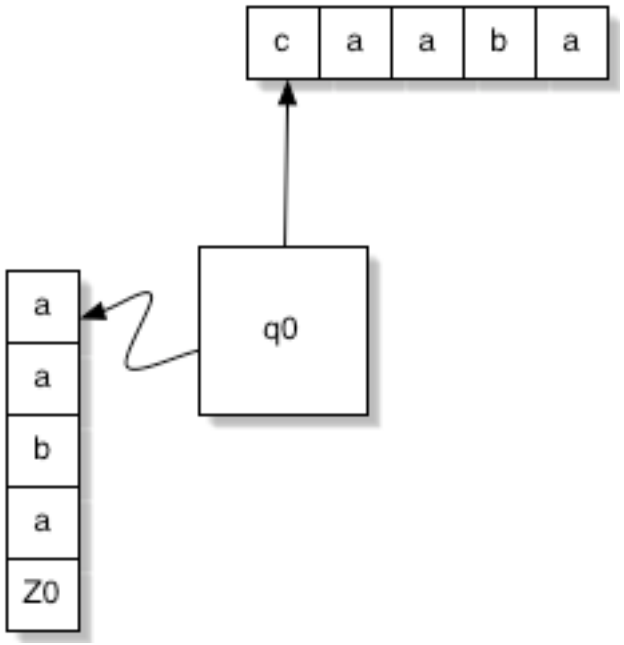
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



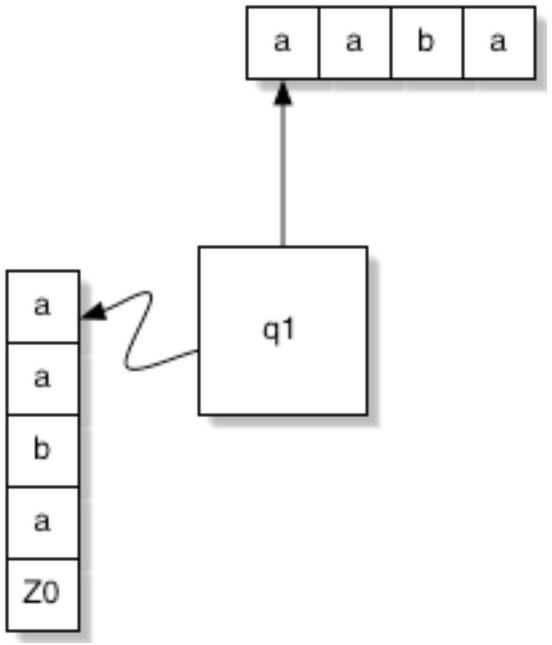
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

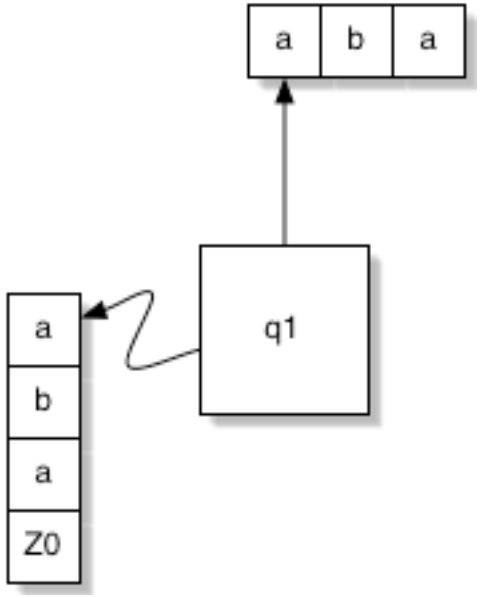


Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

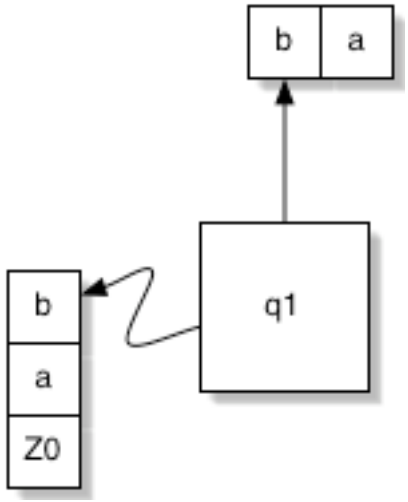


Note that control-state has changed to q_1 after reading c.

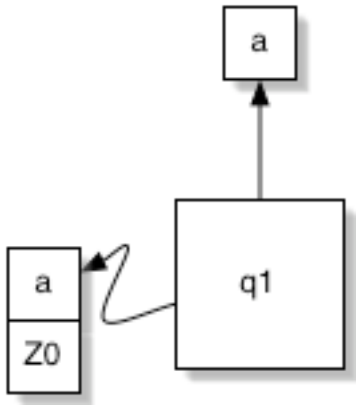
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



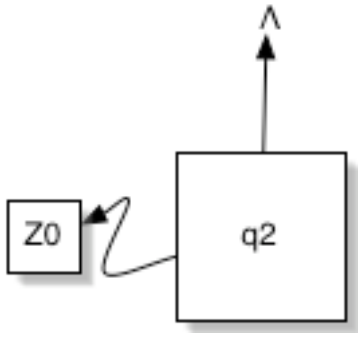
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



q_2 is the accepting control state.
The input is now empty.

So the PDA **accepts** the original input: abaacaaba.

Design of the PDA for the xcx^R language

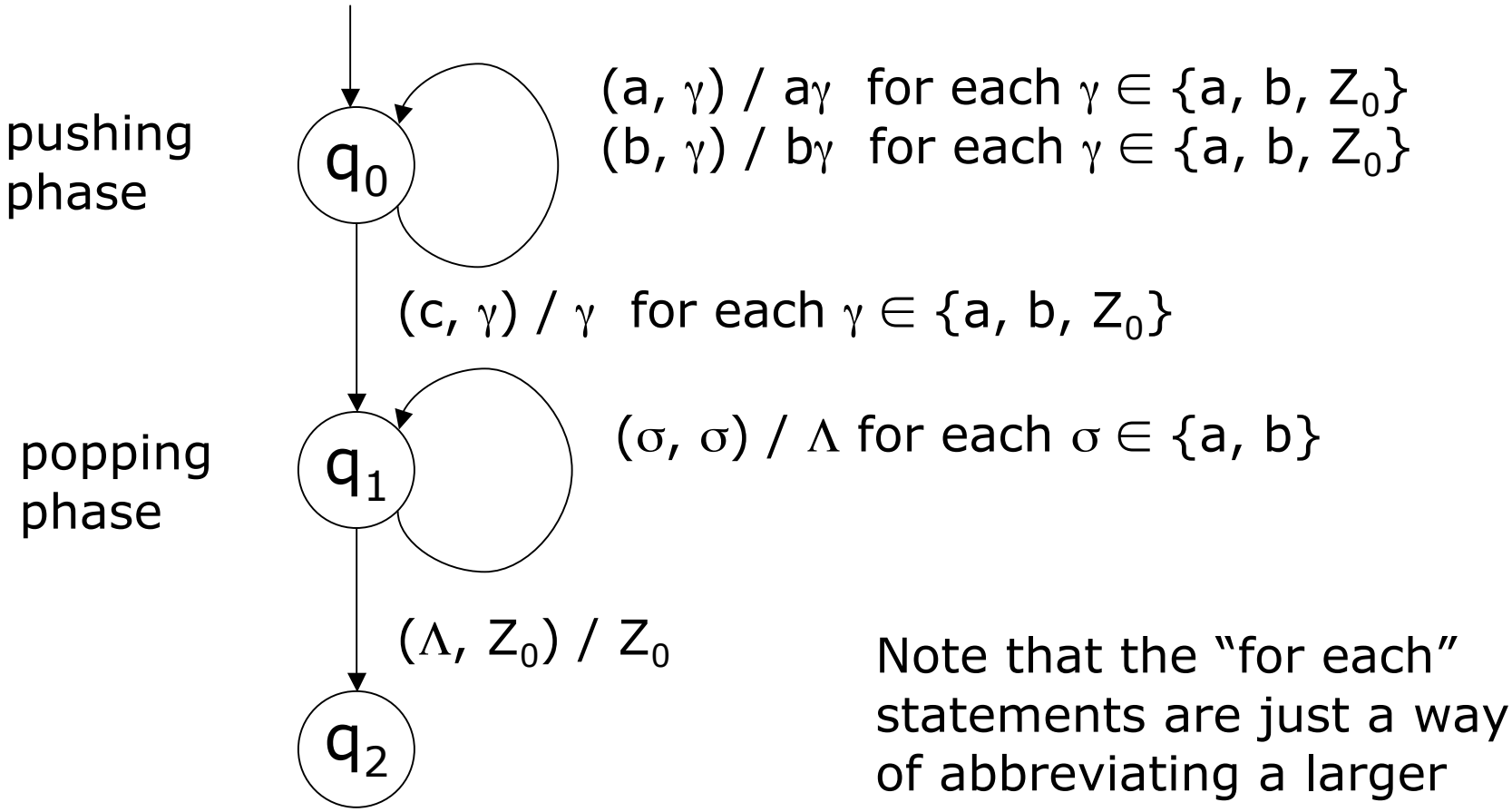
- δ: Q x ($\Sigma \cup \{\Lambda\}$) x $\Gamma \rightarrow$ finite subsets of Q x Γ^*
  - $\delta(q_0, \sigma, \gamma) = \{(q_0, \sigma\,\gamma)\}$, for each $\sigma \in \{a, b\}$, $\gamma \in \{a, b, Z_0\}$
  - $\delta(q_0, c, \gamma) = \{(q_1, \gamma)\}$, for each $\gamma \in \{a, b, Z_0\}$
  - $\delta(q_1, \sigma, \sigma) = \{(q_1, \Lambda)\}$, for each $\sigma \in \{a, b\}$
  - $\delta(q_1, \Lambda, Z_0) = \{(q_2, Z_0)\}$

- The top of the stack is the leftmost symbol in the string.

- The "for each" statements are just a way of abbreviating a larger number of separate transitions.

- For all other combinations, the RHS is empty, meaning that there is no transition defined. Unless the control state is accepting in this case, the input is **rejected** at this point.

δ can also be described as a graph



Note that the "for each" statements are just a way of abbreviating a larger number of transitions.

δ can also be described by "transition rules"

$q_0, a, \gamma \rightarrow q_0, a\gamma$ for each $\gamma \in \{a, b, Z_0\}$
$q_0, b, \gamma \rightarrow q_0, b\gamma$ for each $\gamma \in \{a, b, Z_0\}$
$q_0, c, \gamma \rightarrow q_1, \gamma$ for each $\gamma \in \{a, b, Z_0\}$
$q_1, \sigma, \sigma\gamma \rightarrow q_1, \gamma$ for each $\gamma \in \{a, b, Z_0\}$, for each $\sigma \in \{a, b\}$
$q_1, \Lambda, Z_0 \rightarrow q_2, Z_0$

The rules with "for each" are actually abbreviations for other rules with the γ and σ variables by literal symbols.

The important thing is that **the total number of rules is finite.**

The |— (turnstyle) and |—* notation.

- Let $q, q' \in Q$; $x, x' \in \Sigma^*$; $\gamma, \gamma' \in \Gamma^*$

  $(q, x, \gamma) \vdash (q', x', \gamma')$

  means that there is a 1-step transition of the PDA from "state" (q, x, γ) to (q', x', γ').

- |—* is the **transitive closure** of |—, meaning:
  - $(q, x, \gamma) \vdash^* (q, x, \gamma)$
  - If $(q, x, \gamma) \vdash^* (q', x', \gamma')$ and $(q', x', \gamma') \vdash (q'', x'', \gamma'')$,

    then $(q, x, \gamma) \vdash^* (q'', x'', \gamma'')$.

The key property ("stack property") of PDA's:

- If $(q, x, \gamma) \vdash^* (q', x', \gamma')$

  where $x, x' \in \Sigma^*$; $\gamma , \gamma' \in \Gamma^*$

  then for any $y \in \Sigma^*$ and $\zeta \in \Gamma^*$

  also $(q, xy, \gamma\zeta) \vdash^* (q', x'y, \gamma'\zeta)$.

- In other words, steps that can take place with a given input and stack contents can also take place with additional input and lower stack contents, without depending upon or using the additional input or contents.

Use of Non-Determinism

- PDA's are non-deterministic in the general case: Some of the range values of δ can have more than one element.
- Unlike the situation with NFA's, this non-determinism is more of a mathematical artifice.
- It can also be used to understand various parsing algorithms.

Guessing Metaphor

- Think back to NFA's. One way to describe what they do in a given state with a given input is to "guess" which of several possible next states will ultimately be the "right" one (take the machine to an accepting state).

- This metaphor is extra helpful for PDA's.

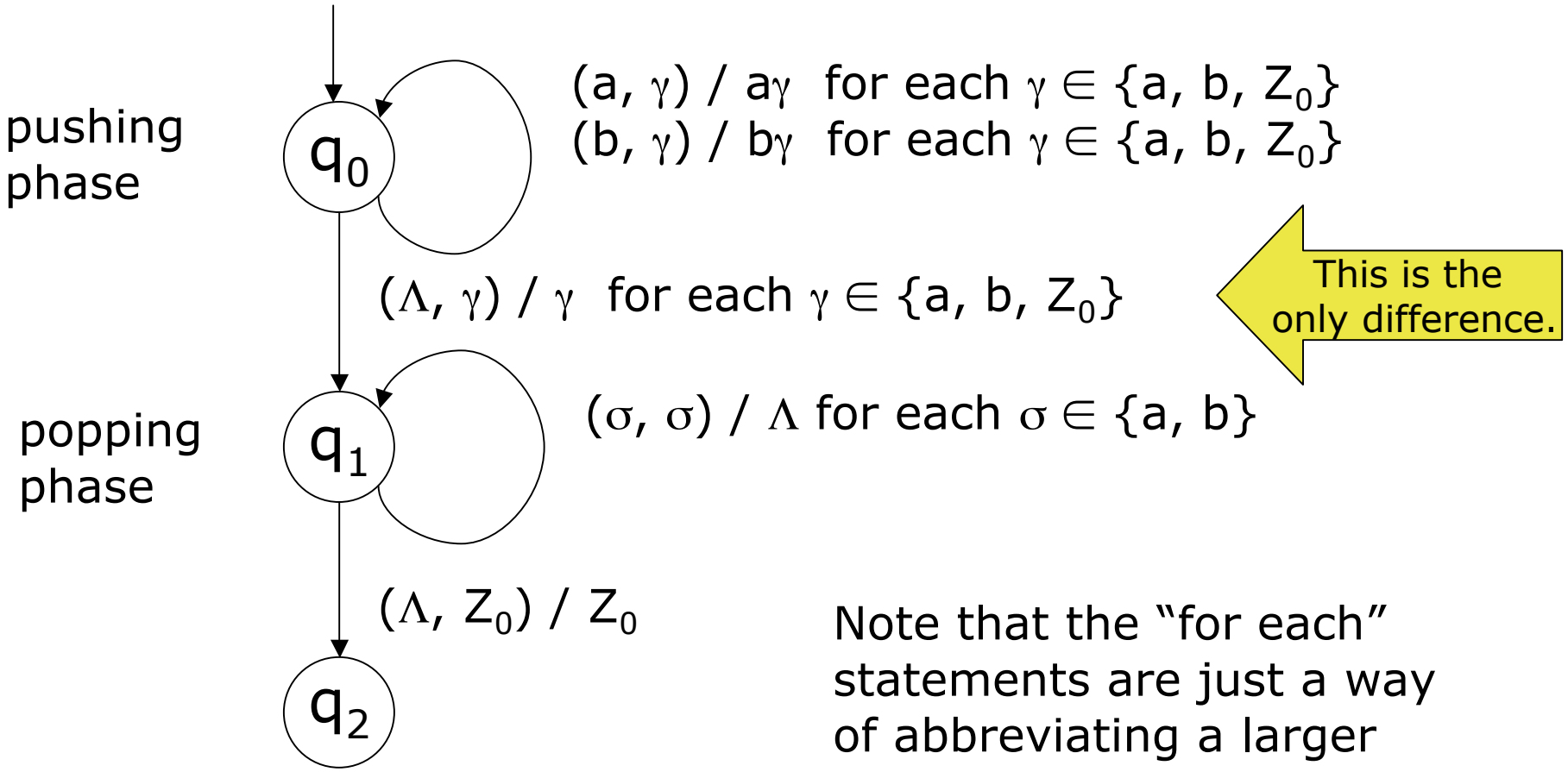
PDA accepting $\{xx^R \mid x \in \{a, b\}^*\}$

- While it appears similar to $\{xcx^R \mid x \in \{a, b\}^*\}$, this set is, in some sense, more difficult to recognize.
- The reason is that for a given input, we don't have an indicator of when x ends and x^R starts.
- Here a PDA can use its non-determinism: It can **guess** at the point it thinks x has ended.
  - If it guessed right, it will get to an accepting state.
  - If it guessed wrong, nothing lost.
  - An important thing is that guessing never leads to an accepting state when the input is not in the language.

PDA accepting $\{xx^R \mid x \in \{a, b\}^*\}$

- Here's how a PDA would work for this language:
  - It begins reading the input symbols.
  - For each symbol read, it would push the symbol onto its stack,
  - until it guesses it has reached the end of the x part of the input,
  - at which time it would transition to a new control state.
  - In the new control state, it would read input symbols, and pop the top of the stack, continuing as long as the symbol read matched the symbol on the stack.
  - Eventually it can guess has seen all of the input, and go to the accepting state, provided the top of the stack is Z_0.

δ for $\{xx^R \mid x \in \{a, b\}^*\}$



Note that the "for each" statements are just a way of abbreviating a larger number of transitions.

Interconvertibility of Acceptance Modes

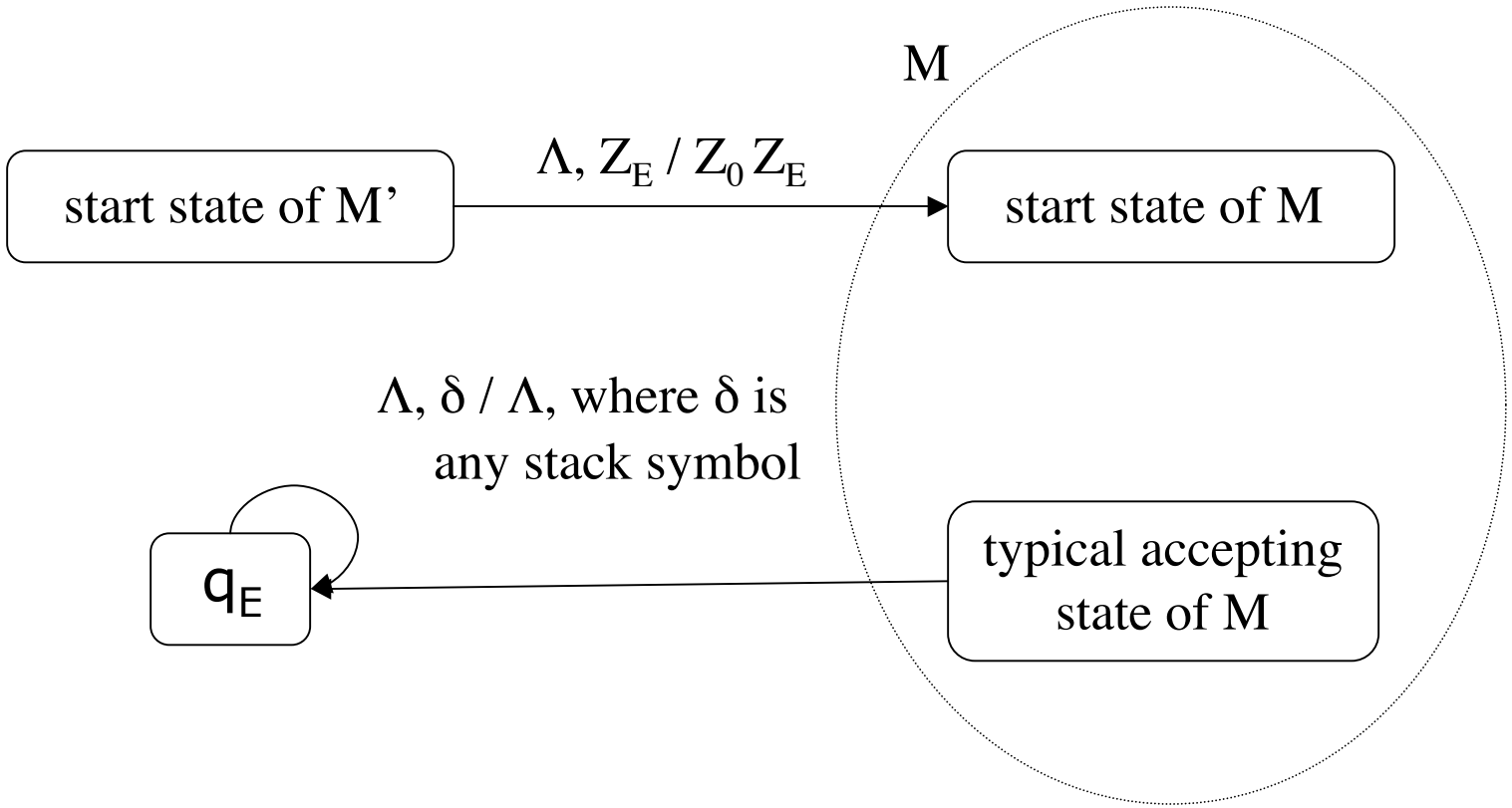
- **For every PDA M accepting by final state, there is a PDA M' accepting the same language by empty stack.** Proof:
- Create M' from M as follows: Add a new state q_E and a transition from each accepting state of M to q_E.
- Add transitions from q_E to itself which do nothing but pop symbols from the stack. This ensures that M' can empty its stack whenever M would have accepted.

- **However**, we must also ensure that M' empties its stack **only** in this case; M could have emptied its stack, so M' might do the same.

(Proving: For every PDA M accepting by accepting state, there is a PDA M' accepting the same language by empty stack.)

- Make the initial stack symbol of M' a new symbol Z_E not used in M.
- Transitions of M can never remove Z_E.
- Where M might have emptied its stack, M' will now have Z_E on the stack.
- By design, M' cannot remove Z_E unless doing so from q_E.

Accepting State ⇒ Empty Stack

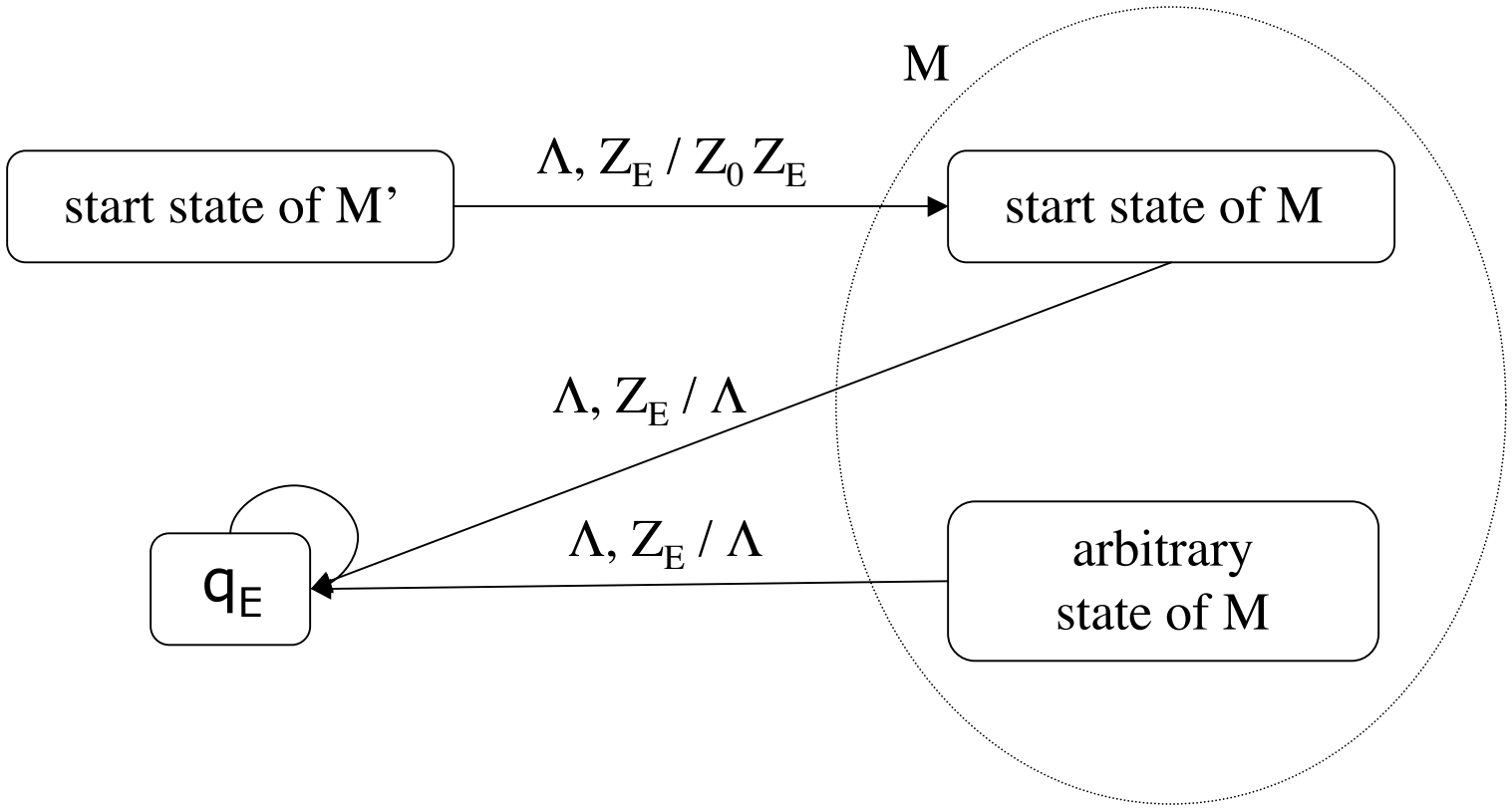


legend: *input*, *top-of-stack* / *push-on-stack (top at left)*

Interconvertibility of Acceptance Modes (2)

- **For every PDA M accepting by empty stack, there is a PDA M' accepting the same language by accepting state.** Proof:
- Create M' from M as follows: The empty stack symbol Z_E of M' is a new stack symbol not used in M.
- M' begins from a new initial state q_I, with a Λ transition to the initial state of M, with which M' pushes $Z_0 Z_E$ where Z_0 is the initial stack symbol of M.
- Introduce a new accepting state q_F as the only accepting state of M'.
- Add new Λ transitions from each state of M to q_F conditioned on being Z_E on top of the stack.
- Whenever M would have emptied its stack, M' will see Z_E and can make a transition to the accepting state.

Empty Stack ⇒ Accepting State



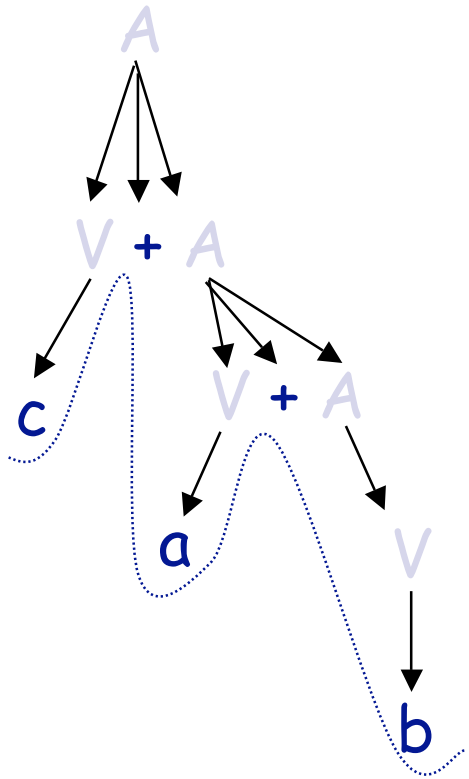
legend: *input*, *top-of-stack* / *push-on-stack (top at left)*

Leftmost/Rightmost Derivation Nomenclature

- A derivation in a context-free grammar is called leftmost if it is always the leftmost auxiliary that is replaced.
- A derivation in a context-free grammar is called rightmost if it is always the rightmost auxiliary that is replaced.
- Observation: For each derivation tree there is exactly one leftmost and one rightmost derivation.

Left/Rightmost Derivations

$A \rightarrow V \mid V + A$
$V \rightarrow a \mid b \mid c$



leftmost: $\underline{A} \Rightarrow \underline{V}+A \Rightarrow c+\underline{A} \Rightarrow c+\underline{V}+A \Rightarrow c+a+\underline{A} \Rightarrow c+a+\underline{V} \Rightarrow c+a+b$

rightmost: $\underline{A} \Rightarrow V+\underline{A} \Rightarrow V+V+\underline{A} \Rightarrow V+V+\underline{V} \Rightarrow V+\underline{V}+b \Rightarrow \underline{V}+a+b \Rightarrow c+a+b$

CFL Characterization Theorem

- A language is context-free iff there is a pushdown acceptor that recognizes it.

Parallel Between CFG and PDA

- Each leftmost derivation

  $S \Rightarrow^* x$

  in a CFG

  corresponds to a series of moves

  $(q, x, S) \vdash^* (q', \Lambda, \Lambda)$

  of some PDA accepting by empty stack.

CFL $\Rightarrow$ PDA Lemma

- Every context free language is accepted by some PDA.

1st Proof of the CFL $\Rightarrow$ PDA Lemma: **top-down** or **produce-match** technique

- Assume L is a context-free language. Then L has a context-free grammar G in Greibach normal form.
- (It is easy to dispose of the Greibach normal form assumption, but we keep it initially for simplicity).
- Construct a PDA M with one control state q.
- This machine will accept by **empty-stack**.
- Each production, which has the form:
  $A \rightarrow \sigma B_1 B_2 B_3 \ldots B_n$,
  where is σ terminal and each B_i is auxiliary, add to M the transition:
  $(q, \sigma, A) \rightarrow (q, B_1 B_2 B_3 \ldots B_n)$
- The initial stack symbol is the start symbol S of G.
- We claim that for any $x \in \Sigma^*$
  $(q, x, S) \vdash^* (q, \Lambda, \Lambda)$ iff $S \Rightarrow^* x$
  This requires an inductive proof (given later).

Example of CFG to Machine

Production	Transition
$S \rightarrow (T$	$q, `(', S \rightarrow q, T$
$S \rightarrow (ST$	$q, `(', S \rightarrow q, ST$
$S \rightarrow (TS$	$q, `(', S \rightarrow q, TS$
$S \rightarrow (STS$	$q, `(', S \rightarrow q, STS$
$T \rightarrow)$	$q, `)', T \rightarrow q, \Lambda$

Example accepting sequence

- Input: (()())
- Rules are:

  q, '(', S → q, T
  q, '(', S → q, ST
  q, '(', S → q, TS
  q, '(', S → q, STS
  q, ')', T → q, Λ

- State sequence:
  - q, (()()), S use rule: q, '(', S → q, ST to get
  - q, ()()), ST use rule: q, '(', S → q, TS to get
  - q,)()), TST use rule: q, ')', T → q, Λ to get
  - q, ()), ST use rule: q, '(', S → q, T to get
  - q,)), TT use rule: q, ')', T → q, Λ to get
  - q,), T use rule: q, ')', T → q, Λ to get
  - q, Λ, Λ accept by empty stack

What's going on here?

- The pda is simulating a **leftmost derivation** of a string.
- In a leftmost derivation, the leftmost auxiliary is always rewritten.
  - The symbols to the left of that auxiliary in the derived string are the ones that have been read.
  - The symbols to the right are the stack contents.

Simulating Leftmost Derivation

$S \rightarrow (T$	$q, `(`, S \rightarrow q, T$
$S \rightarrow (ST$	$q, `(`, S \rightarrow q, ST$
$S \rightarrow (TS$	$q, `(`, S \rightarrow q, TS$
$S \rightarrow (STS$	$q, `(`, S \rightarrow q, STS$
$T \rightarrow)$	$q, `)`, T \rightarrow q, \Lambda$

- (()()) | $\underline{S}$
- (()()) | ($\underline{S}$T
- (()()) | (($\underline{T}$ST
- (()()) | (()$\underline{S}$T
- (()()) | (()($\underline{T}$T
- (()()) | (()())$\underline{T}$
- (()()) | (()())

Red shows matched portions of input not remaining in input and not actually on the stack.

Underscore shows the LHS symbol being rewritten.

Exercise

- Show that Greibach normal form is not essential for the lemma and proof technique.
- (Allow terminals as well as auxiliaries on the stack.
  Allow σ to be either Λ or an element of Σ.)

PDA $\Rightarrow$ CFL Lemma

- For every PDA M, there is a context-free grammar G such that L(G) = L(M).
- Proof outline:
  - Assume that M has just **one control state** q and acceptance is by empty stack.
    Reverse the construction of the CFL $\Rightarrow$ PDA Lemma; for each transition create a corresponding grammar rule:
    - The stack symbols will become auxiliaries in the grammar. The initial stack symbol is S, the start symbol.
    - For each transition rule of the form

      $q, \sigma, Z \rightarrow q, \gamma \qquad (\sigma \in \Sigma \cup \{\Lambda\})$

      introduce a production

      $Z \rightarrow \sigma\, \gamma$
- We claim that for any $x \in \Sigma^*$

  $(q, x, S) \vdash^* (q, \Lambda, \Lambda)$ iff $S \Rightarrow^* x$

  This requires an inductive proof (given later).

Proof that One State Suffices

- Assume WLOG that $M = (Q, \Sigma, \Gamma, q_0, Z_0, \{q_F\}, \delta)$ is a pda accepting by empty stack, where q_F is a single final state, from which M empties its stack.
  Construct a new pda
  $$M' = (\{\kappa\}, \Sigma, \Gamma', q, Z'_0, A', \delta')$$
  that accepts the same language:
  - $\Gamma' = Q \times \Gamma \times Q$
  - Each element of Γ' is written $<q\ Z\ q'>$ but is really just a single symbol.
  - The initial stack symbol of M′ is $Z'_0 = <q_0\ Z_0\ q_F>$.
  - For each transition of M: $(q', B_1B_2B_3\ldots B_n) \in \delta(q, \sigma, Z)$ include as transitions of M′:
    $(\kappa, <q\ B_1\ q_1> <q_1\ B_2\ q_2> \ldots <q_{n-1}\ B_n\ q_n>) \in \delta'(\kappa, \sigma, <q\ Z\ q_n>)$
    for every choice of $q_1, q_2, \ldots, q_n \in Q$ (choices not necessarily distinct).
    [If n = 0, then $(\kappa, \Lambda) \in \delta'(\kappa, \sigma, <q\ Z\ q'>)$ **for every choice** of $q' \in Q$.
  - That M′ accepts the same language as M needs to be proved by induction.

2nd Proof of the CFL $\Rightarrow$ PDA Lemma: **bottom-up** or **shift-reduce** technique

- Assume L is a context-free language. Then L has a context-free grammar G.
- Create a pda that accepts by final state.
- For each terminal symbol σ, create a transition:

  $q_0, \sigma, \Lambda \rightarrow q_0, \sigma$

  These have the effect of **shifting** the input string onto the stack (and reversing it in the process).
- For each production $A \rightarrow x_1x_2x_3\ldots x_n$ create transitions:

  $q_0, \Lambda, x_n \rightarrow q_1, \Lambda$
  $q_1, \Lambda, x_{n-1} \rightarrow q_2, \Lambda$
  $q_2, \Lambda, x_{n-2} \rightarrow q_3, \Lambda$
  . . .
  $q_n, \Lambda, x_1 \rightarrow q_0, A$

  where the q_i other q_0 than are distinct for each production.
- This pda simulates a **rightmost** derivation of an accepted input string.

Example of Bottom-Up

Production	Transitions
$S \rightarrow (T$	$q_0, \Lambda, T \rightarrow q_1, \Lambda$ $q_1, \Lambda, (\rightarrow q_0, S$
$T \rightarrow S)$	$q_0, \Lambda,) \rightarrow q_2, \Lambda$ $q_2, \Lambda, S \rightarrow q_0, T$
$S \rightarrow ()$	$q_0, \Lambda,) \rightarrow q_3, \Lambda$ $q_3, \Lambda, (\rightarrow q_0, S$
$S \rightarrow SS$	$q_0, \Lambda, S \rightarrow q_4, \Lambda$ $q_4, \Lambda, S \rightarrow q_0, S$
shift transitions for each δ	$q_0, (, \delta \rightarrow q_0, (\delta$ $q_0,), \delta \rightarrow q_0,)\delta$
accept transitions	$q_0, \Lambda, S \rightarrow q_5, \Lambda$ $q_5, \Lambda, Z_0 \rightarrow q_a, Z_0$

Derivation: $S \Rightarrow (T \Rightarrow (S) \Rightarrow (SS) \Rightarrow (S()) \Rightarrow (()())$

State Sequence		
$(()())$	q_0	Z_0
$(()())$	q_0	$(Z_0$
$(()())$	q_0	$((Z_0$
$(()())$	q_0	$)((Z_0$
$(()())$	q_3	$((Z_0$
$(()())$	q_0	$S(Z_0$
$(()())$	q_0	$(S(Z_0$
$(()())$	q_0	$)(S(Z_0$
$(()())$	q_3	$(S(Z_0$
$(()())$	q_0	$SS(Z_0$
$(()())$	q_4	$S(Z_0$
$(()())$	q_0	$S(Z_0$
$(()())$	q_0	$)S(Z_0$
$(()())$	q_2	$S(Z_0$
$(()())$	q_0	$T(Z_0$
$(()())$	q_1	$(Z_0$
$(()())$	q_0	SZ_0
$(()())$	q_5	Z_0
$(()())$	q_a	Z_0