Pushdown Automata

Robert M. Keller
Harvey Mudd College
April 2013

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 FSA, 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**.
  - Must also be checking for no c's, more than one c, etc. and reject these cases.

PDA's

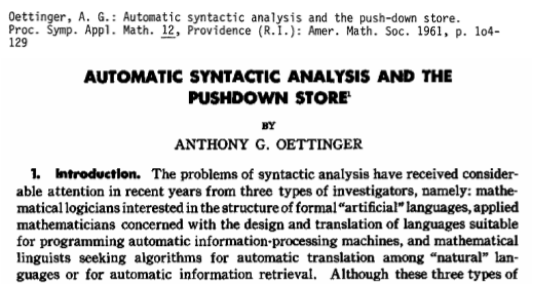
- The type of behavior described on the preceding slide is that of a (deterministic) PDA (DPDA).
- In general, by convention, PDA's are **non-deterministic**.
- The situation for PDA's is different from DFA's: there is **no subset construction**, because the set of all subsets is generally uncountably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

PDA's in practice

- PDA's are a mathematical model designed to capture context-free language recognition.
- They are also model how actual parsers work in a skeletal way.
- In practice, one wants restrictions on the language so that parsing can be done deterministically (without backtracking) using a little bit of "look-ahead".

PDA Historical

- The PDA was invented by Anthony Oettinger at Harvard, my academic grandfather.



Oettinger, A. G.: Automatic syntactic analysis and the push-down store. Proc. Symp. Appl. Math. 12, Providence (R.I.): Amer. Math. Soc. 1961, p. 104-129

AUTOMATIC SYNTACTIC ANALYSIS AND THE PUSHDOWN STORE[1]

BY

ANTHONY G. OETTINGER

1. Introduction. The problems of syntactic analysis have received considerable attention in recent years from three types of investigators, namely: mathematical logicians interested in the structure of formal "artificial" languages, applied mathematicians concerned with the design and translation of languages suitable for programming automatic information-processing machines, and mathematical linguists seeking algorithms for automatic translation among "natural" languages or for automatic information retrieval. Although these three types of

Howard Aiken

|

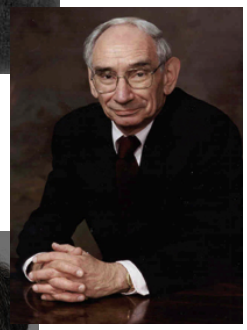
Anthony Oettinger

|

Richard Karp

|

Robert Keller



http://genealogy.math.ndsu.nodak.edu/id.php?id=13305

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

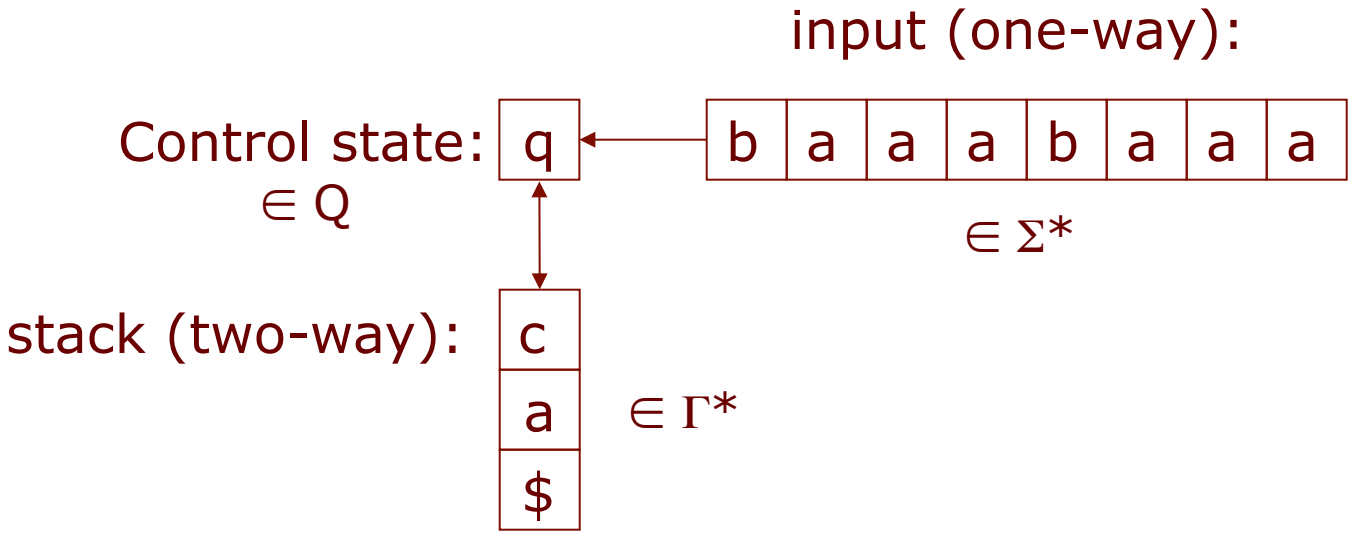
- Q is a finite set of **control states**
- Σ is the **input** alphabet
- Γ is the **stack** alphabet
- q_0 is the **initial** control state
- $F \subseteq Q$ is the set of **accepting** control states
- δ is the **state transition relation** (or, in the case of a DPDA, **function**):

  δ: $Q \times \Sigma\varepsilon \times \Gamma\varepsilon \rightarrow$ subsets of $Q \times \Gamma\varepsilon$

  (The non-determinism is in choosing a member of $\delta(q, \sigma, \gamma)$.)

Notation: $\Sigma\varepsilon = \Sigma \cup \{\varepsilon\}$, $\Gamma\varepsilon = \Gamma \cup \{\varepsilon\}$.

PDA Diagram



The **state** is generally of the form $(q, x, \gamma) \in Q \times \Sigma^* \times \Gamma^*$, i.e. (control-state, remaining input, 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, x, ε). We will describe δ presently.

PDA Acceptance

- The Sipser model says that a PDA **accepts** the **input seen so far** provided that it is in an **accepting control state**.
- Due to non-determinism, the machine could be in *various* control states after seeing a given input.
- It is only required that there be **some** accepting control state for acceptance.

PDA Rejection

- If there is no way for the PDA to be in an accepting state having read the input, the input is implicitly **rejected**.

- Rejection could be insured by making a transition to a state from which no escape to an accepting state is possible.

Empty-Stack Acceptance

- This is another common model.
- **Empty-stack acceptance** means that the PDA accepts when its **stack is empty** after the input has been read. The control state does not matter in this case.
- The two versions of acceptance can be shown **inter-convertible**.

Initial Stack Symbol

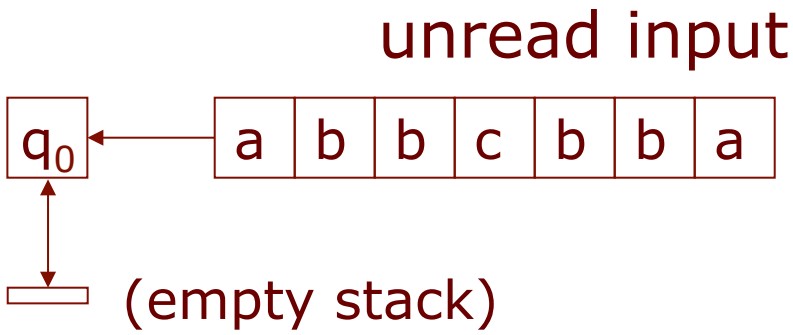
- Some authors specify an **initial stack symbol** as part of the model. This symbol is always on the stack at the start of a run.
- Sipser does not, although many of his examples begin by placing a special symbol, such as $, on the stack before anything else happens.
- This symbol is commonly used to test whether the stack would otherwise be empty and react to this condition.

Other PDA Alternatives

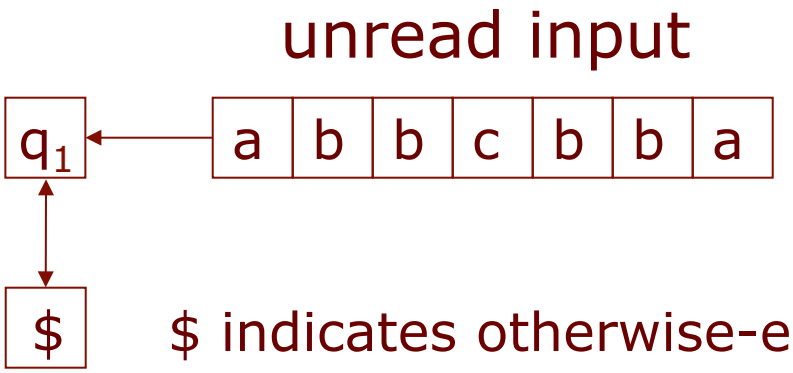
- Most authors allow a **string**, rather than just a letter, to be written to the stack in one step.

- The same effect can be achieved in Sipser's model, which only allows one letter at a time to be added, by introducing additional control states that add the string one symbol at a time.

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

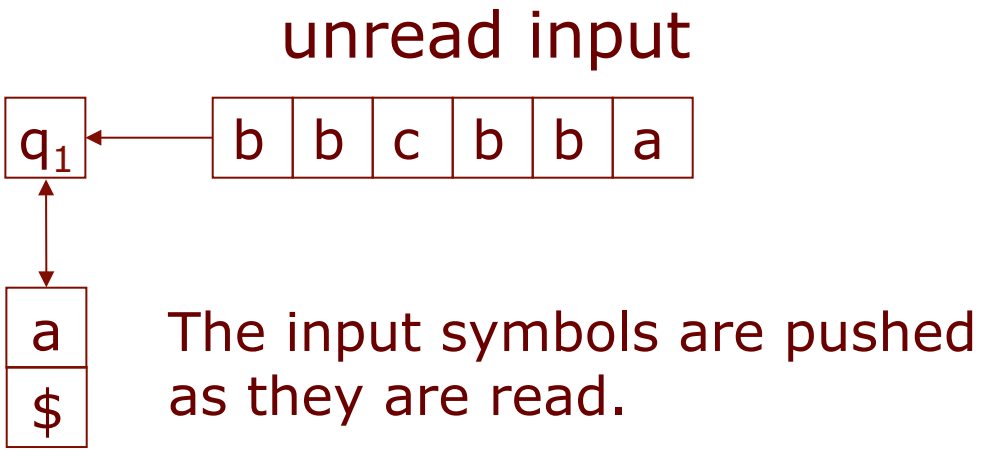


Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

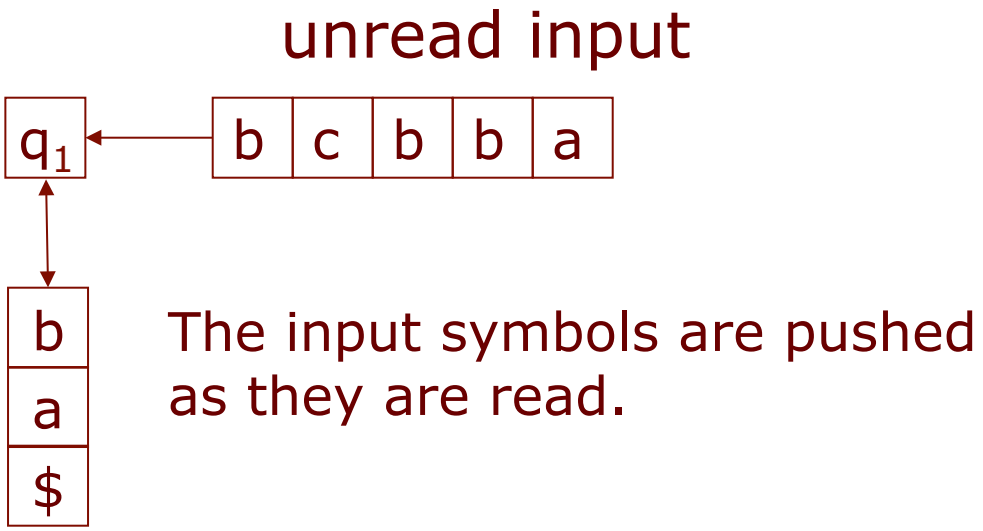


$ indicates otherwise-empty stack, so the machine can test for that when necessary.

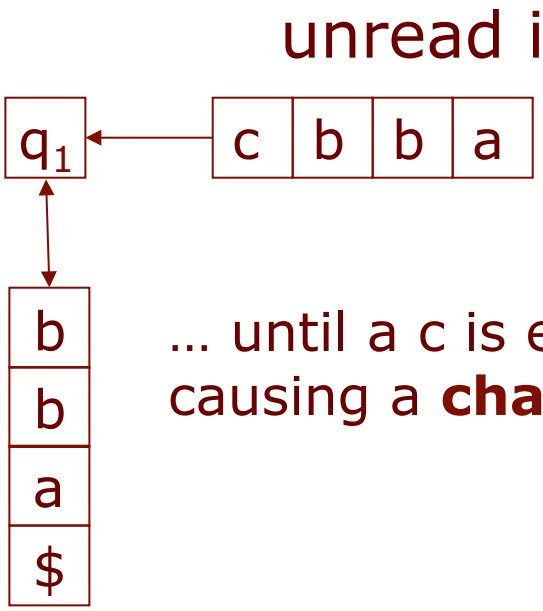
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



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

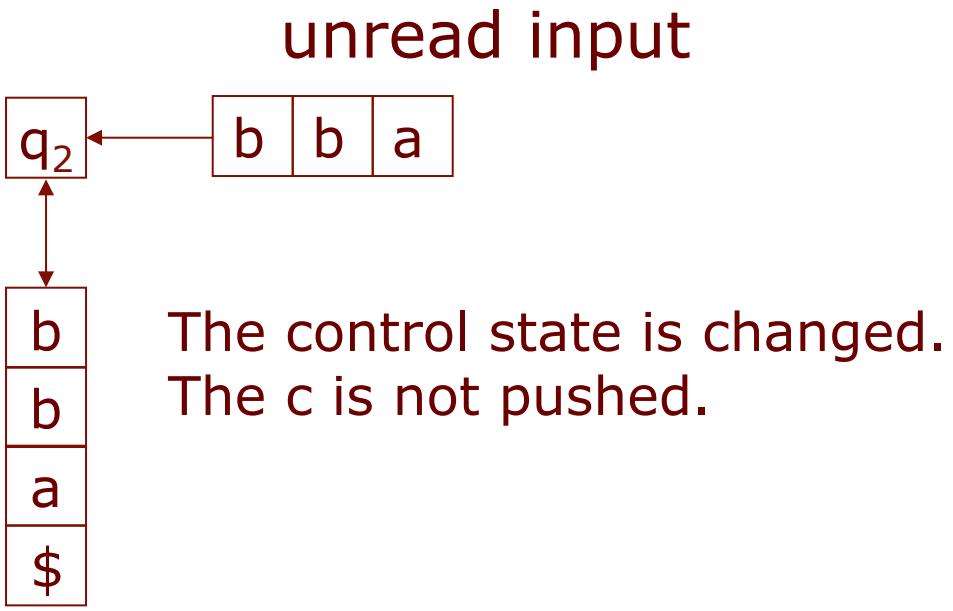


Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$

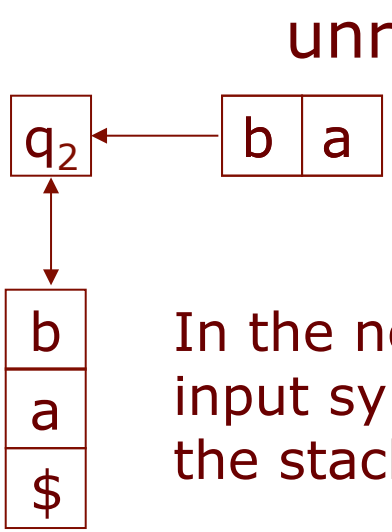


… until a c is encountered in the input, causing a **change** of control state

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



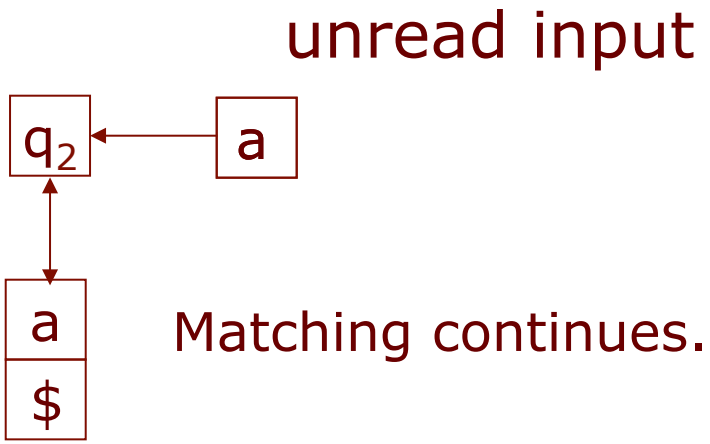
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



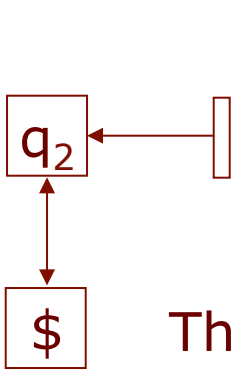
In the new control state,
input symbols are **matched** against
the stack symbols, which are discarded.

A mismatch stops the process, as
no successor is specified.

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



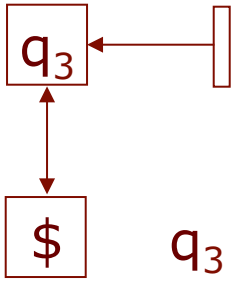
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



The input is empty. However, there is no way to react to input emptiness as such.

The machine can go to an accepting state whenever the current control state is q_2 and \$ is on top of the stack (according to the way we have defined δ in this case).

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



q_3 is the accepting state, so the original input abbcbba is accepted.

Had the input been abbcbb only, the input would not be accepted, because $ would not have been on top of the stack when the input became empty, and q_3 would not have been entered.

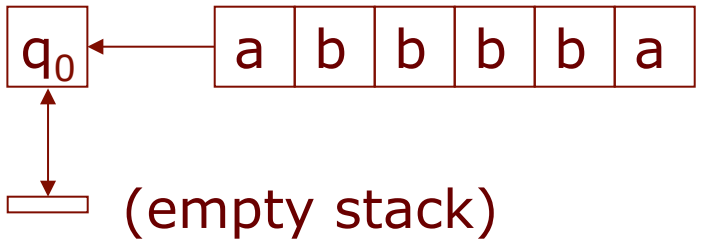
Exercise

- Construct the function δ for the previous machine:
  - $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$ (set of one state)
  - $\delta(q_1, a, \varepsilon) = \{(q_1, a)\}$

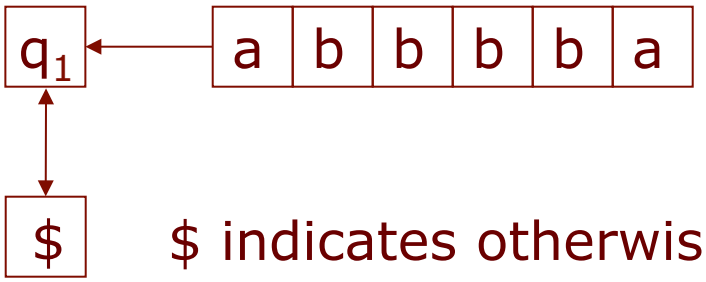
A slightly different language:

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

This requires more non-determinism.

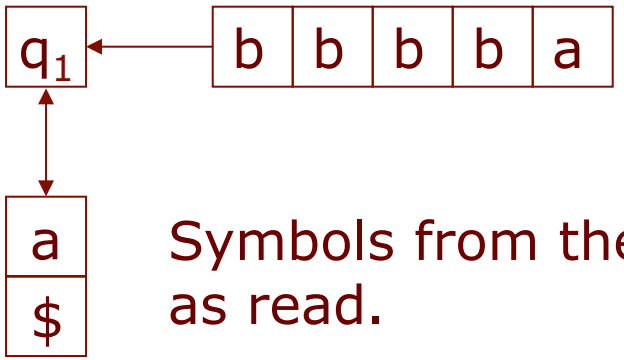


Movie of the PDA accepting a string in $\{xx^R \mid x \in \{a, b\}^*\}$



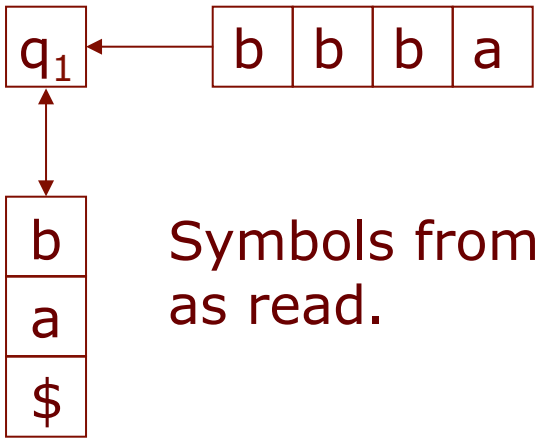
$ indicates otherwise “empty” stack, so the machine can test for that when necessary.

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



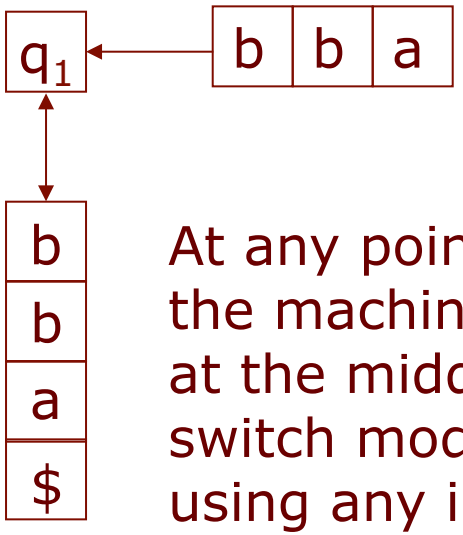
Symbols from the input are pushed as read.

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



Symbols from the input are pushed as read.

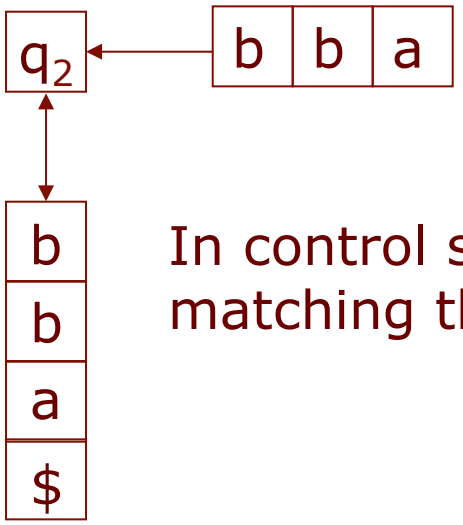
Movie of the PDA accepting a string in $\{xx^R \mid x \in \{a, b\}^*\}$



At any point, such as this one, the machine can "**guess**" that it's at the middle of the string and switch modes (without necessarily using any input).

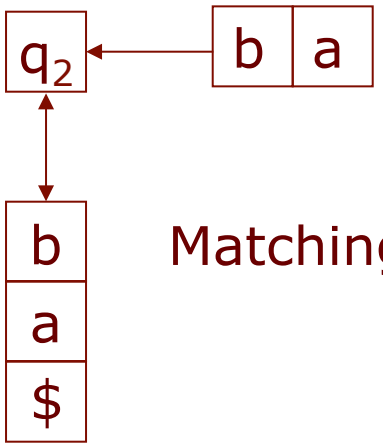
If it makes the "wrong" guess, that particular sequence will end up not accepting. There only need to be at least **one** correct guess.

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



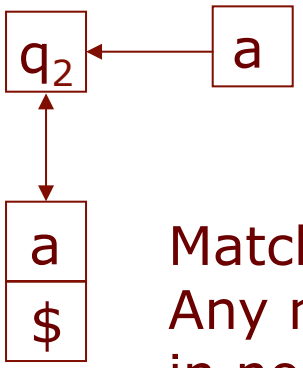
In control state q_2, the machine begins matching the input against the stack.

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



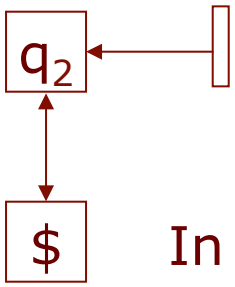
Matching continues.

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



Matching continues.
Any mis-match would result in no further transitions.

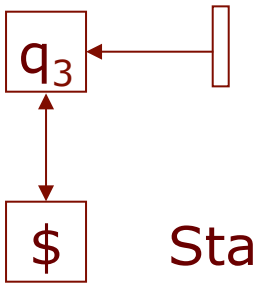
Movie of the PDA accepting a string in $\{xx^R \mid x \in \{a, b\}^*\}$



In state with e on top of stack, the machine can decide to accept what it has seen so far.

That doesn't mean that it accepts an *extension* of that input, however.

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



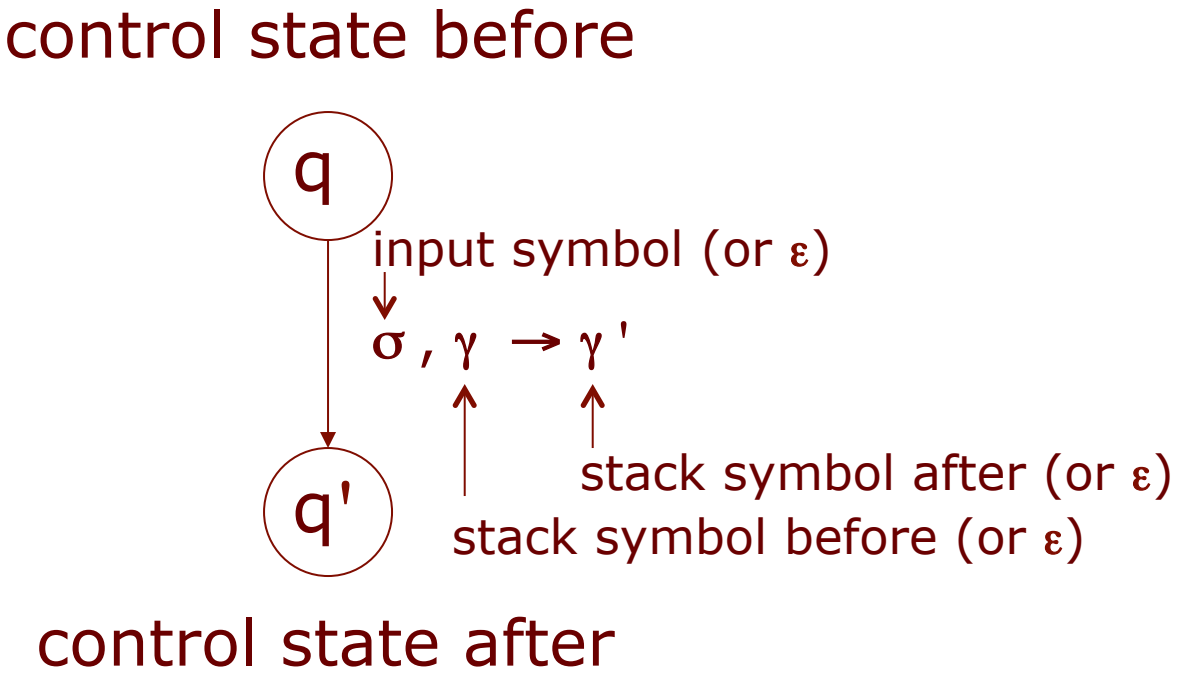
State q_3 is accepting.
The original input abbbba is accepted.

Exercise

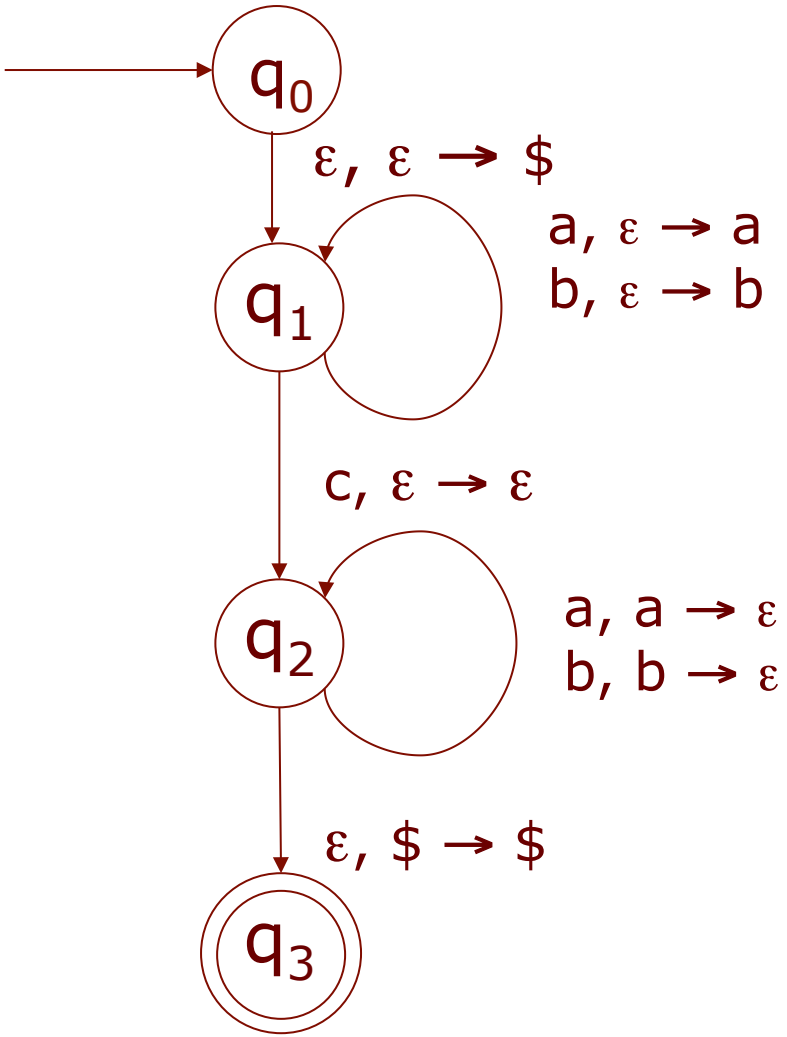
- Define the transition function for the previous PDA.

δ can also be described by a graph

In lieu of $(q', \gamma') \in \delta(q, \sigma, \gamma)$



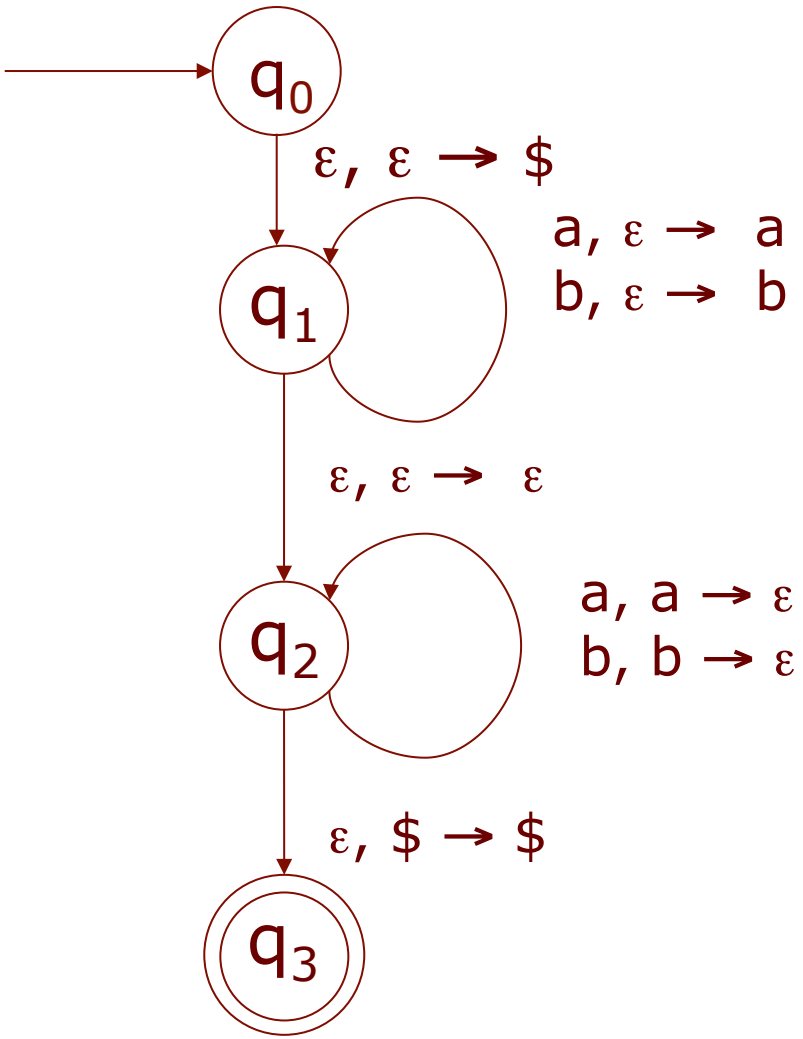
PDA described by a graph



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

- While it appears similar to $\{xcx^R \mid x \in \{a, b\}^*\}$, the set $\{xx^R \mid x \in \{a, b\}^*\}$ 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**.

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



δ can also be described by "transition rules"

$q_1, a, \varepsilon \rightarrow q_1, a$
$q_1, b, \varepsilon \rightarrow q_1, b$
$q_1, \varepsilon, \varepsilon \rightarrow q_2, \varepsilon$
$q_2, a, a \rightarrow q_2, \varepsilon$
$q_2, b, b \rightarrow q_2, \varepsilon$
$q_2, \varepsilon, \$ \rightarrow q_3, \$$

Each rule is essentially a 5-tuple:

current-state, input-read, stack-popped, next-state, stack-pushed

JFLAP Version of PDA

- Graphical specification
- λ rather than ε is used for empty string.
- Initial stack symbol Z is present at start
- String (rather than just symbol) can be pushed: Lowest symbol is to the right.
- Acceptance is by accepting state.

http://www.jflap.org/tutorial/pda/construct/

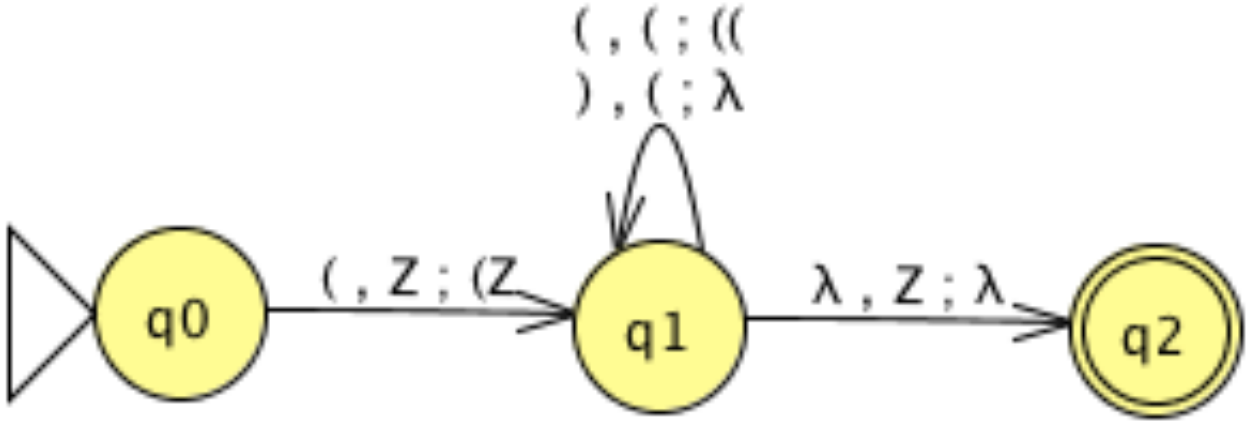
JFLAP defines a nondeterministic pushdown automaton (NPDA) M as the septuple $M = (Q, \Sigma, \Gamma, \delta, q_s, Z, F)$ where

- Q is a finite set of **states** $\{q_i \mid i \text{ is a nonnegative integer}\}$
- Σ is the finite **input alphabet**
- Γ is the finite **stack alphabet**
- δ is the **transition function**,

  $\delta : Q \times \Sigma^* \times \Gamma^* \rightarrow$ finite subsets of $Q \times \Gamma^*$
- q_s (is member of Q) is the **initial state**
- Z is the **start stack symbol** (must be a capital Z)
- F (a subset of Q) is the set of **final states**

Example: Well-balanced Paren Strings

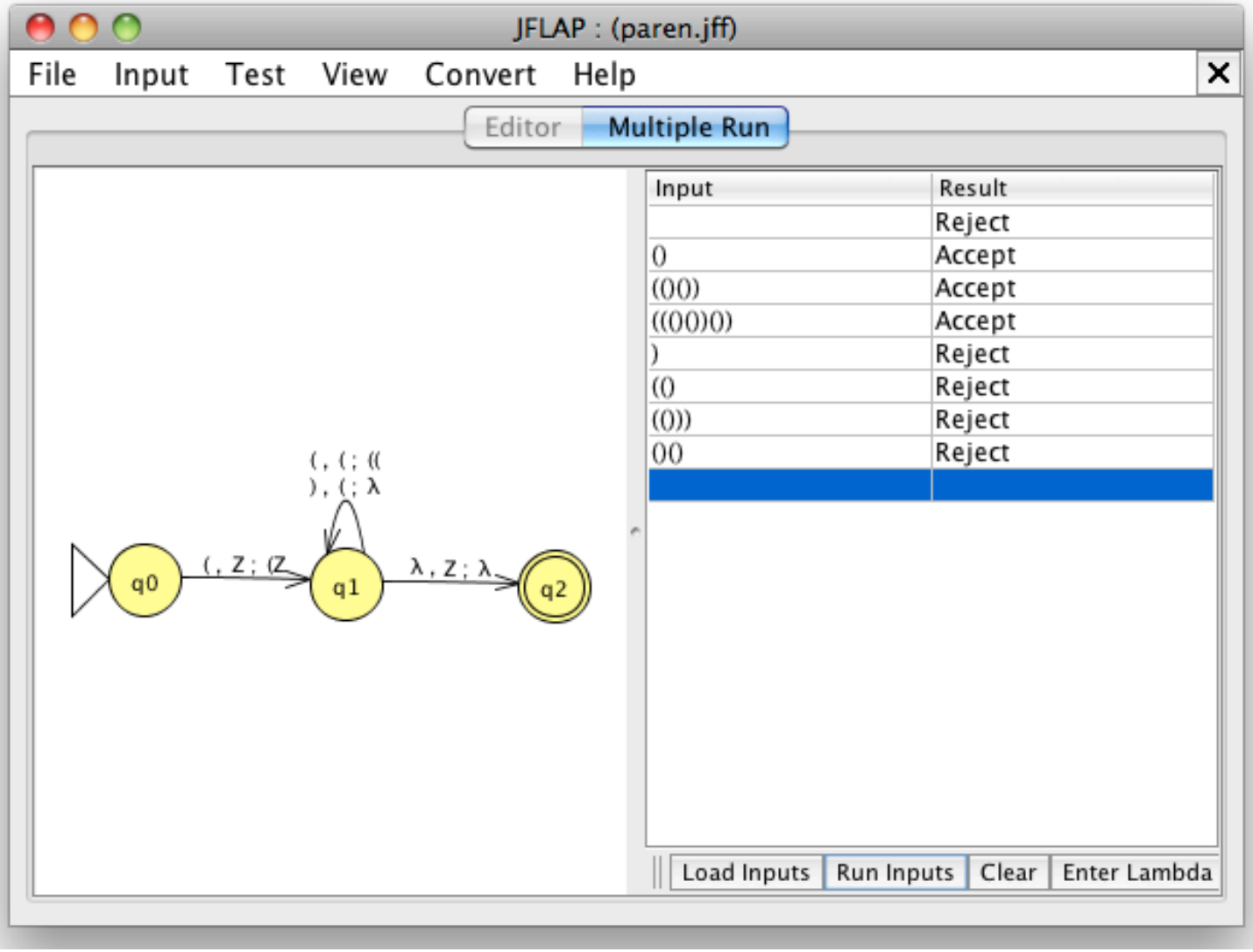
Input, Pop; Push



(Z is atop stack at start) (Z is popped on acceptance)

Intentionally not accepted: ε, ()()

Running Examples



The $\Rightarrow$ and $\Rightarrow^*$ notation.

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

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

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

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

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

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

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

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

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

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

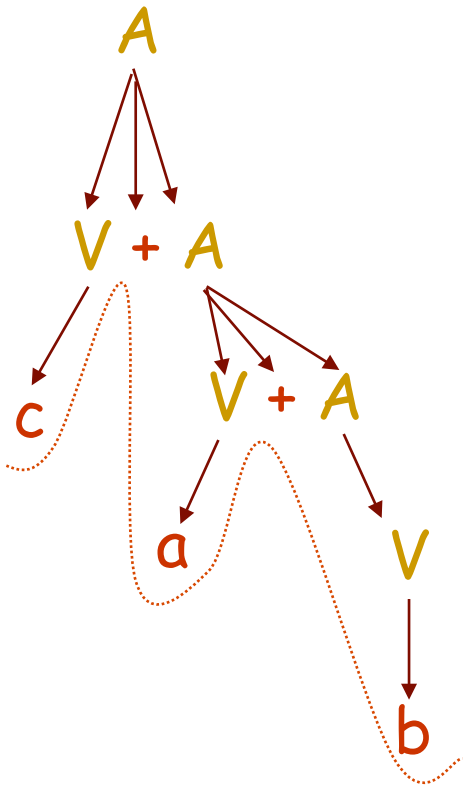
- In other words, steps that can take place with a given input and stack contents can also take place with additional input and lower-down stack contents, without depending upon or using the additional input or stack symbols.

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.

- (In general, the PDA will be a non-deterministic machine.)

Parallel Between CFG and PDA

- Each derivation

  $S \Rightarrow^* x$

  in a CFG

  corresponds to a series of moves

  $(q, x, S) \Rightarrow^* (q', \varepsilon, \varepsilon)$

  of a PDA accepting by empty stack.

CFL → PDA Lemma

- Every context-free language is accepted by some PDA.

- This discussion is simplified if we allow a transition to push a **sequence** of symbols in one step. (No generality is lost.)

Proof of the CFL → PDA Lemma using
top-down, LL, or **produce-match** technique

- Let G be a context-free grammar for the CFL.
- Construct from G a PDA M that simulates an arbitrary **leftmost** derivation in G. Initially M pushes onto the stack an empty-stack marker $, followed by the start symbol of G.
- In a leftmost derivation, the **leftmost auxiliary** in a string is replaced. The stack holds a **suffix of the current working string**, with the **top of stack** corresponding to the leftmost symbol in the string.

Proof of the CFL → PDA Lemma using
top-down, LL, or **produce-match** technique

- If the top of stack is a **terminal**, it is **matched** against the corresponding symbol in the input and removed.

- If the top of stack is an **auxiliary** A, then a production with A as LHS is chosen and the RHS is pushed onto the stack, in reverse order.

- When $ is on top of the stack, and only then, the machine can enter an accepting state.

Example of CFG to PDA Transition Rules

Production	Transition (top of stack at left)	
	$q_0, \varepsilon, \varepsilon \rightarrow q_1, S\$$	initial
$S \rightarrow (T$	$q_1, \varepsilon, S \rightarrow q_1, (T$	produce
$S \rightarrow (ST$	$q_1, \varepsilon, S \rightarrow q_1, (ST$	
$S \rightarrow (TS$	$q_1, \varepsilon, S \rightarrow q_1, (TS$	
$S \rightarrow (STS$	$q_1, \varepsilon, S \rightarrow q_1, (STS$	
$T \rightarrow)$	$q_1, \varepsilon, T \rightarrow q_1,)$	
Extra rules:	$q_1, (, (\rightarrow q_1, \varepsilon$	match
	$q_1,),) \rightarrow q_1, \varepsilon$	
	$q_1, \$, \varepsilon \rightarrow q_2, \varepsilon$	accept

q_0 is initial, q_2 is accepting

Derivation vs. Transition Sequence

$S \to (T$
$S \to (ST$
$S \to (TS$
$S \to (STS$
$T \to)$

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

Derivation	State, Input, Stack	Transitions
$S \Rightarrow$	q_0, ()(), ε	init
$(TS \Rightarrow$	q_1, ()(), S$	produce $S \to (TS$
	q_1, ()(), (TS$	match (
$()S \Rightarrow$	q_1,)(), TS$	produce $T \to)$
	q_1,)(),)S$	match)
$()(T \Rightarrow$	q_1,(), S$	produce $S \to (T$
	q_1,(), (T$	match (
$()()$	q_1,), T$	produce $T \to)$
	q_1,),)$	match)
	q_1, ε, $	
	q_2, ε, $	accept

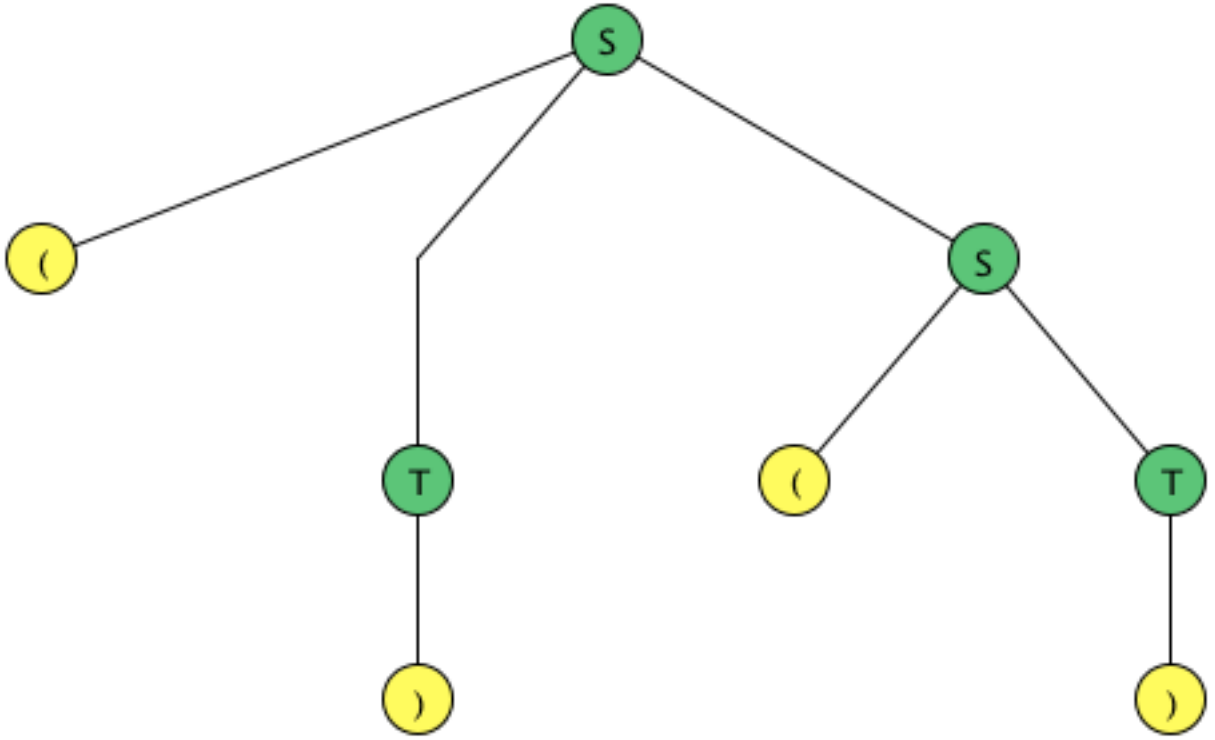
Another Way of Showing

- Productions:
  S → (T
  S → (ST
  S → (TS
  S → (STS
  T →)
- Input | Stack
- (()()) | S$
- (()()) | (ST$
- (()()) | ((TST$
- (()()) | (()ST$
- (()()) | (()(TT$
- (()()) | (()())T$
- (()()) | (()())$

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

Underscore shows the LHS symbol being rewritten.

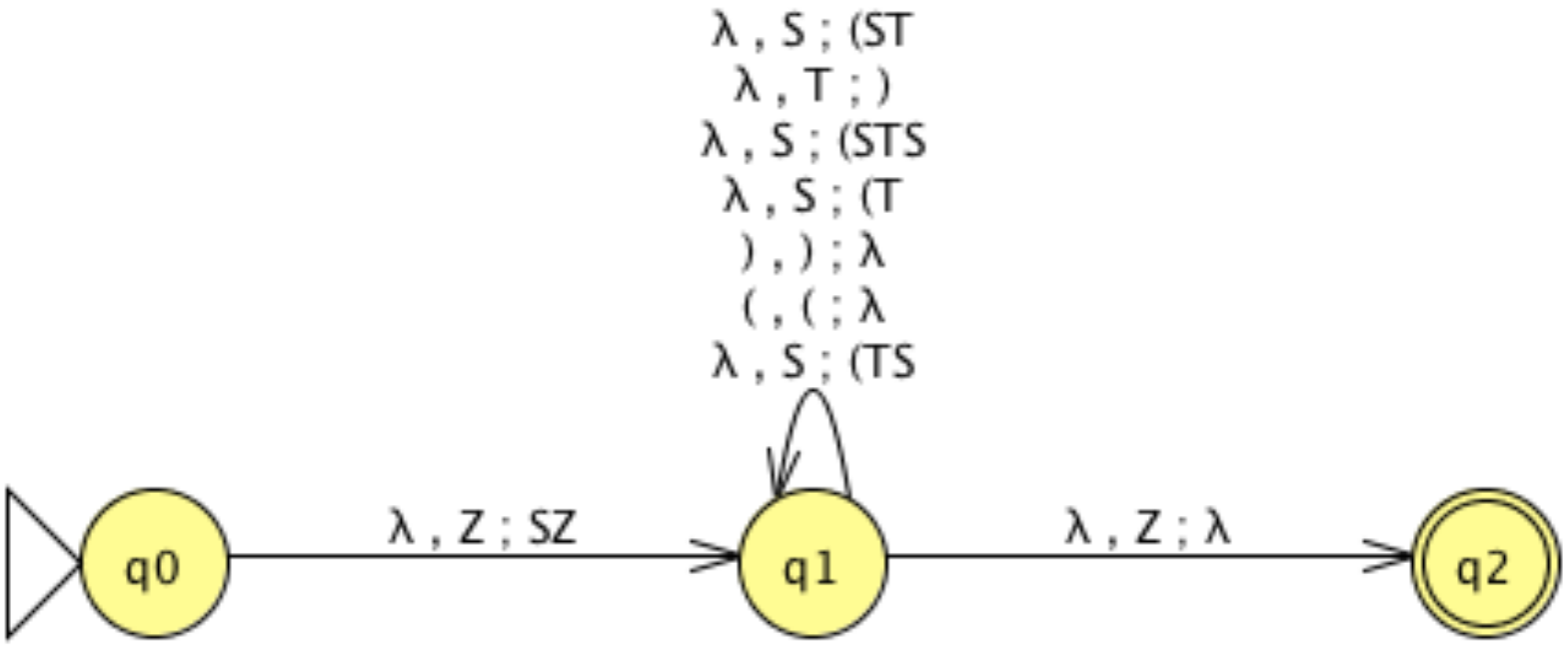
JFLAP-generated Tree for ()()



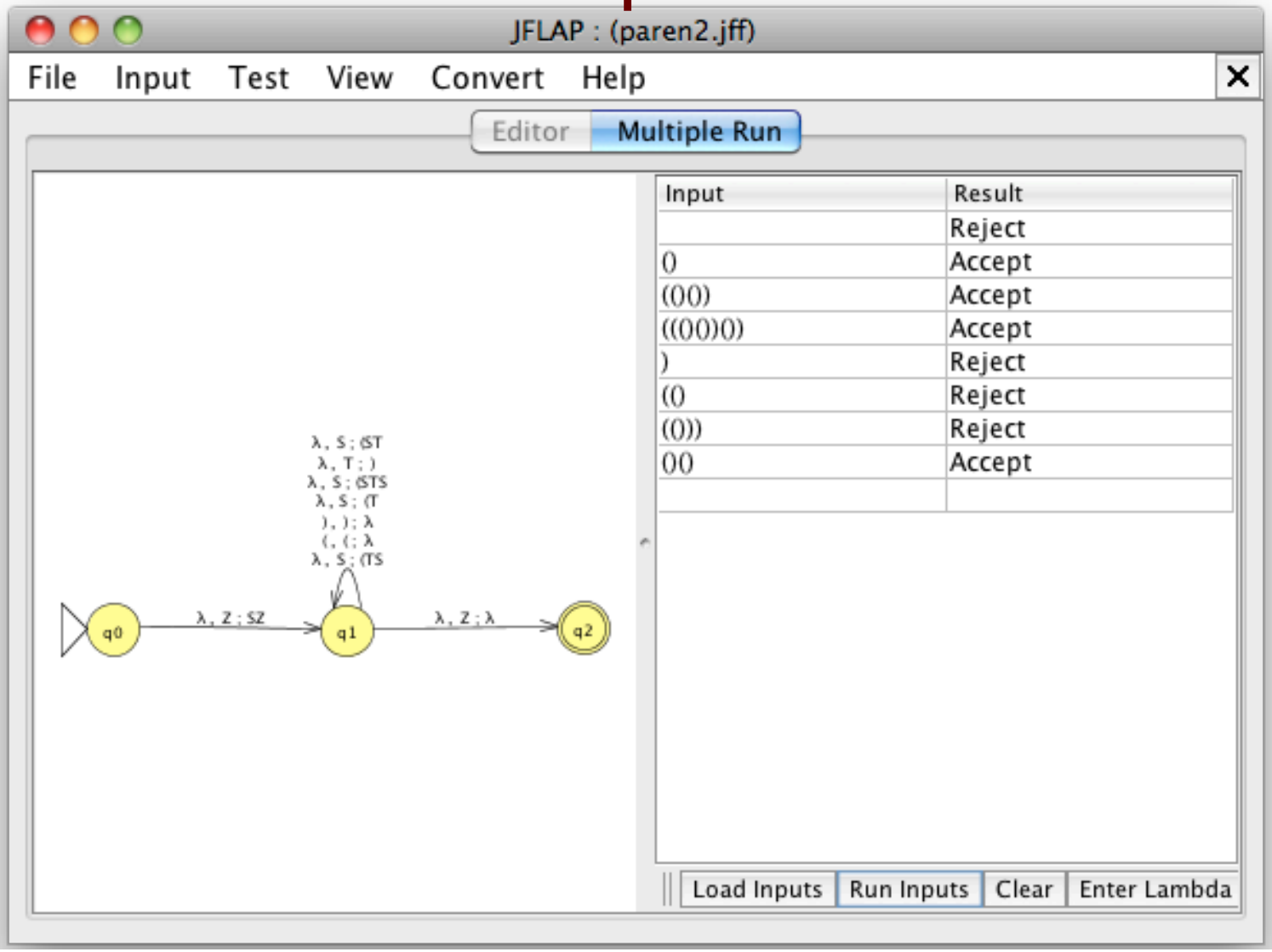
S	⟶	(T
S	⟶	(ST
S	⟶	(TS
S	⟶	(STS
T	⟶	)

Production	Derivation
	S
S→(TS	(TS
S→(T	(T(T
T→)	()(T
T→)	()()

Bottom-Up Version PDA in JFLAP



JFLAP Tests of Top-Down Version



A 2nd Proof of the CFG ⇒ PDA Lemma: **bottom-up, LR,** 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:

  $q_0, \sigma, \varepsilon \rightarrow q_0, \sigma$

  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:

  $q_0, \varepsilon, x_n \rightarrow q_1, \varepsilon$
  $q_1, \varepsilon, x_{n-1} \rightarrow q_2, \varepsilon$ (Symbols pushed in reverse order.)
  $q_2, \varepsilon, x_{n-2} \rightarrow q_3, \varepsilon$
  . . .
  $q_n, \varepsilon, x_1 \rightarrow q_0, A$

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

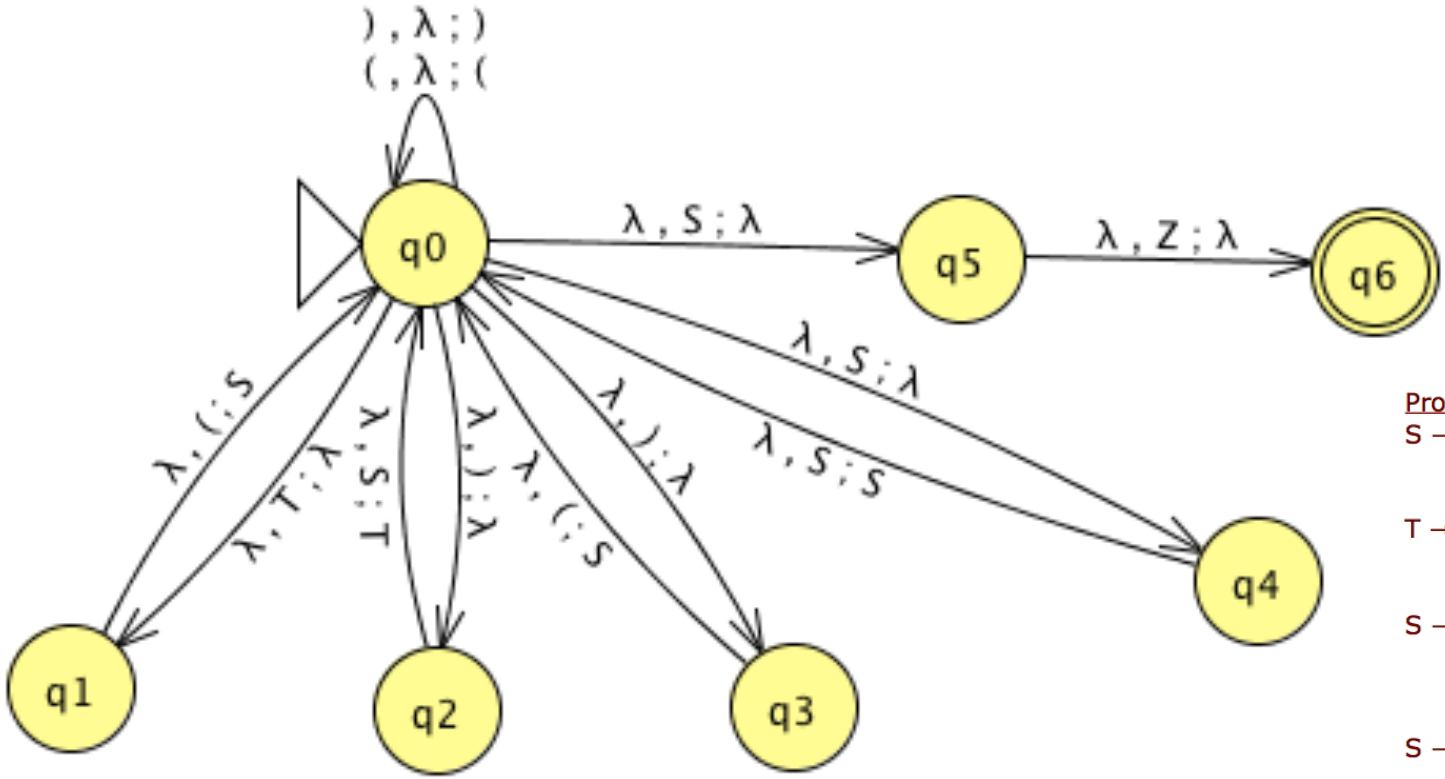
Example of Bottom-Up

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

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

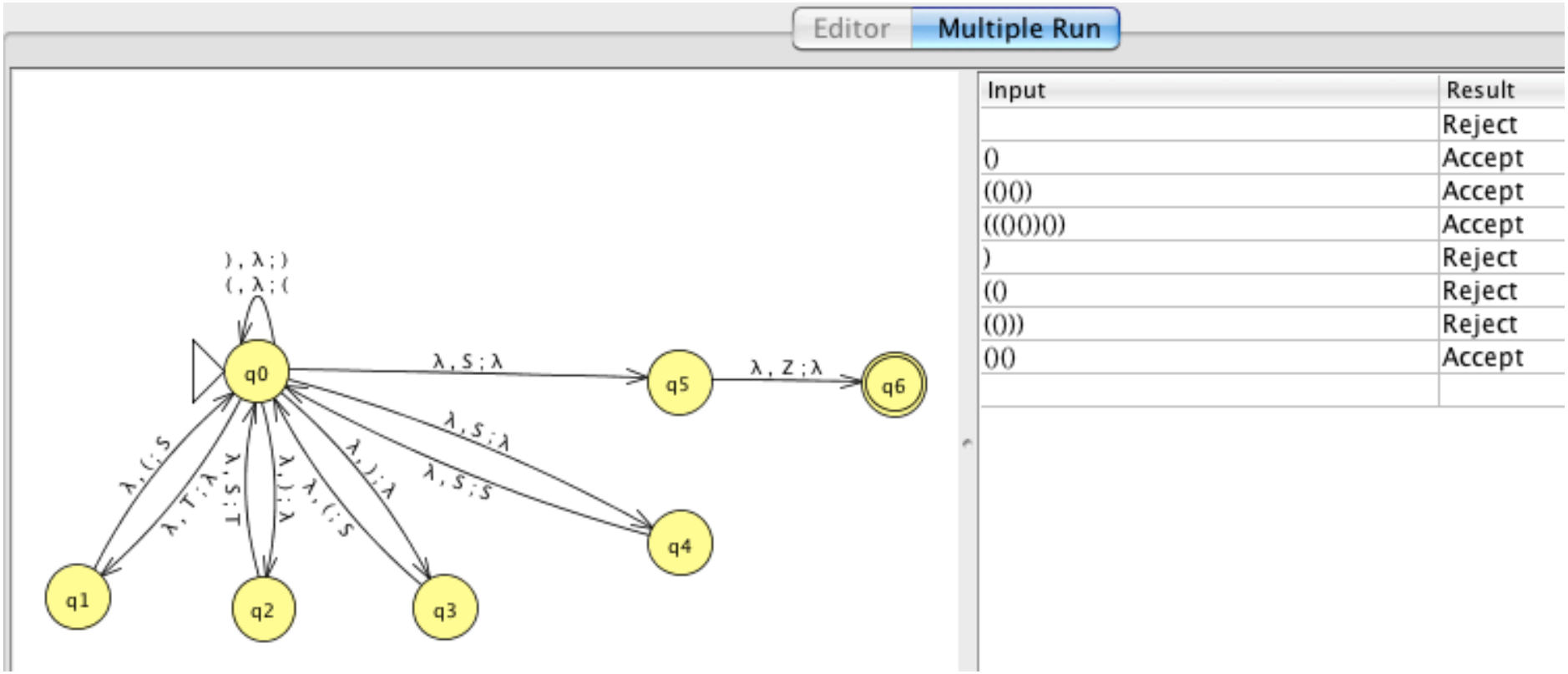
State	Sequence
(()())	q_0 | $
(()())	q_0 | ($
(()())	q_0 | (($
(()())	q_0 |)(($
(()())	q_3 | (($
(()())	q_0 | S($
(()())	q_0 | (S($
(()())	q_0 |)(S($
(()())	q_3 | (S($
(()())	q_0 | SS($
(()())	q_4 | S($
(()())	q_0 | S($
(()())	q_0 |)S($
(()())	q_2 | S($
(()())	q_0 | T($
(()())	q_1 | ($
(()())	q_0 | S$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

Bottom-Up Version in JFLAP



Production	Transitions
S → (T	q_0, ε, T → q_1, ε
	q_1, ε, (→ q_0, S
T → S)	q_0, ε,) → q_2, ε
	q_2, ε, S → q_0, T
S → ()	q_0, ε,) → q_3, ε
	q_3, ε, (→ q_0, S
S → SS	q_0, ε, S → q_4, ε
	q_4, ε, S → q_0, S
shift transitions for each δ	q_0, (, δ → q_0, (δ
	q_0,), δ → q_0,)δ
accept transitions	q_0, ε, S → q_5, ε
	q_5, ε, $ → q_a, $

JFLAP Bottom-Up Acceptance



Input	Result
	Reject
()	Accept
(())	Accept
((())())	Accept
)	Reject
(()	Reject
(())	Reject
()()	Accept

PDA → CFG

- For every PDA M, there is a CFG G that generates the language accepted by M.
- Read Sipser's elegant proof: Lemma 2.27.

PDA → CFG Lemma

Additional **constraints** are imposed on M, without loss of generality:

- On each transition, M either
  a) pushes a symbol, or
  b) pops a symbol,

  but not both.

  [If a transition does **both**, add an intermediate state and have it pop, then push.

  If a transition does **neither**, have it push an arbitrary symbol, then immediately pop it.]

- M has only one accept state q_{accept} and empties its stack before entering it.

Construction of G from M

We want

G to generate a terminal string x

iff

M goes from its initial state to accepting state with x as input, with the stack empty before and after.

Construction (which is slightly more general than the main goal):

For each **pair** of states p, q in M,
there is an auxiliary A_{pq} in G.

We want:

$\forall x \in \Sigma^*$ (**if** $(p, x, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$ in M **then** $A_{pq} \Rightarrow^* x$ in G).

How to achieve this?

Construction continued

- Suppose $(p, x, \varepsilon) \Rightarrow^{+} (q, \varepsilon, \varepsilon)$
  (non-empty transition sequence)
- Since the stack starts empty,
  the first transition must be a push.
- Similarly, the stack ends empty,
  so the last transition must be a pop.

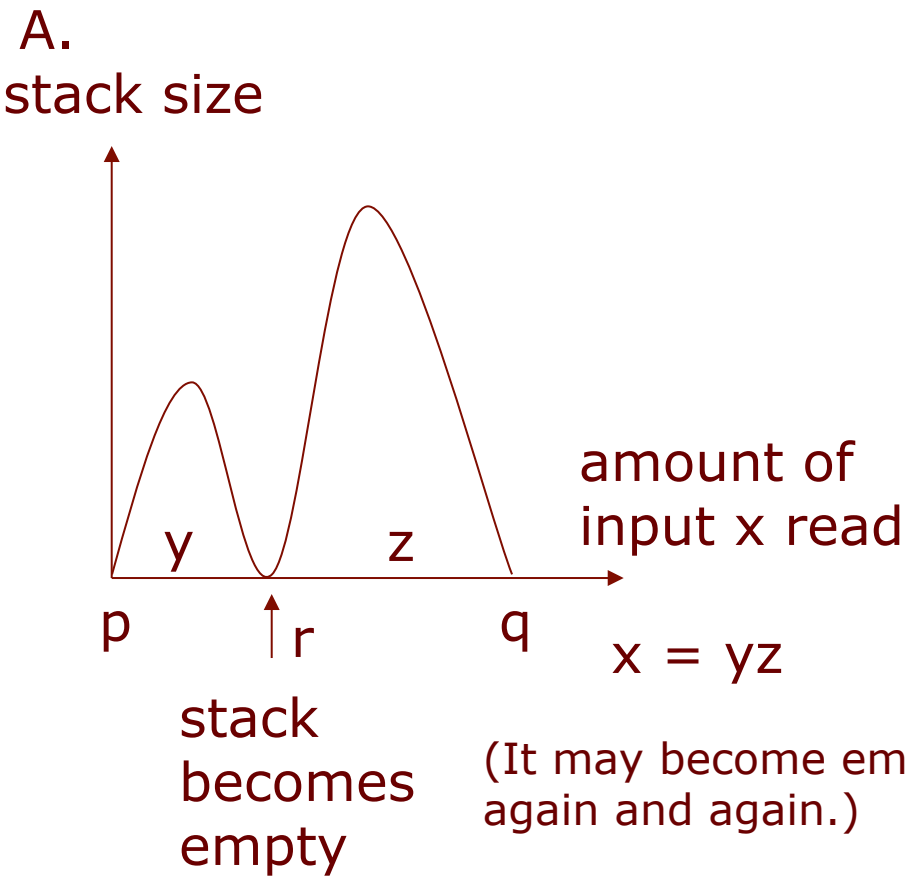
- There are two possibilities:

A. The stack **becomes empty** one or more times between first and last transitions, or

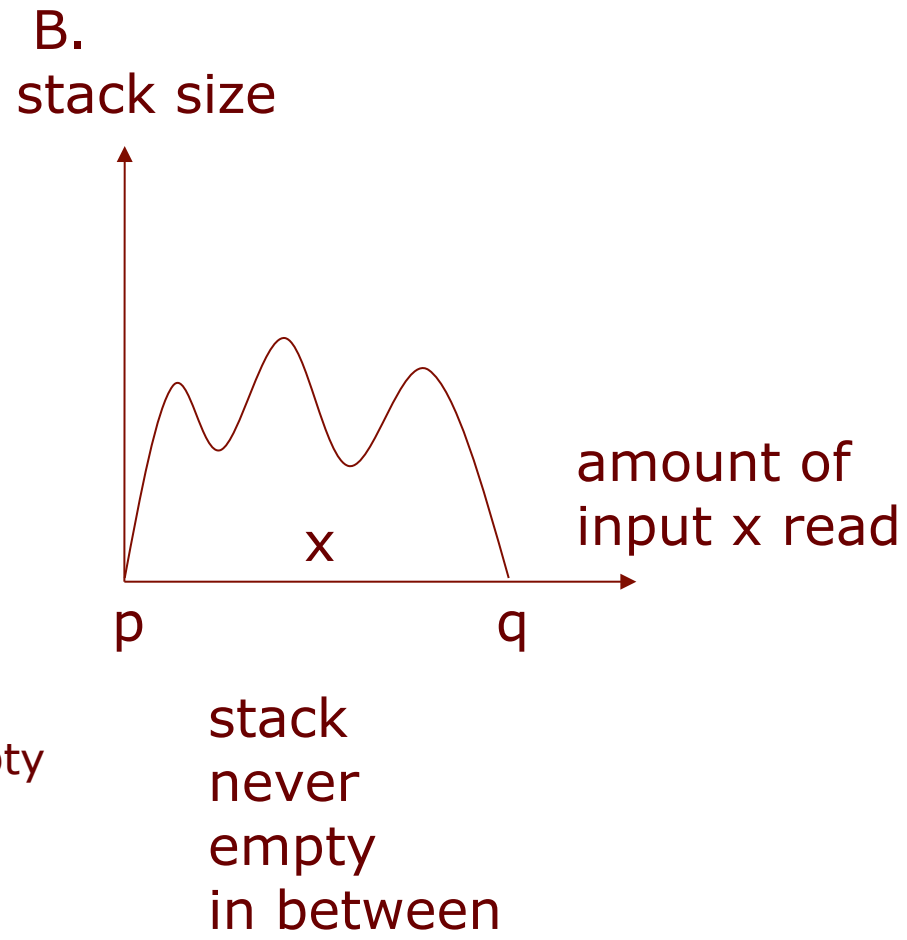
B. The stack **never becomes empty** in between first and last transitions.

Two Possibilities

A.



B.

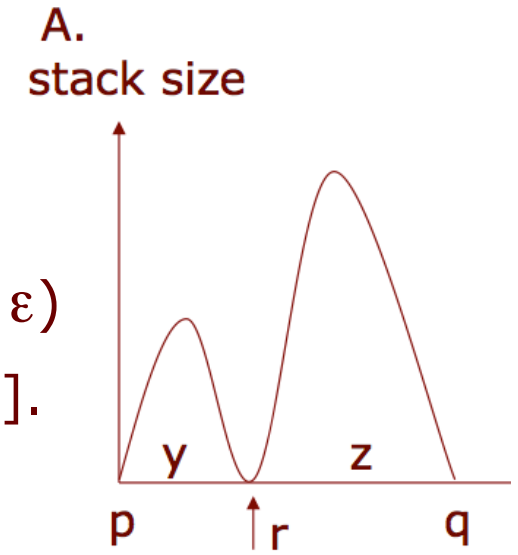


Simulating Transition Sequences with G

- In case A., suppose that r is a state in which the stack becomes empty in between.
  This is achievable by a rule in G of the form:

$$A_{pq} \rightarrow A_{pr} A_{rq}$$

because x can be written yz, where
$(p, yz, \varepsilon) \Rightarrow^* (r, z, \varepsilon)$ and $(r, z, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$
[which combine to $(p, yz, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$].



Simulating Transition Sequences with G

- In case B., let a be the first input symbol read and b be the last. Let u be the first stack symbol pushed, which is the last popped.
  Then in M we have, for one or more r, s, u:

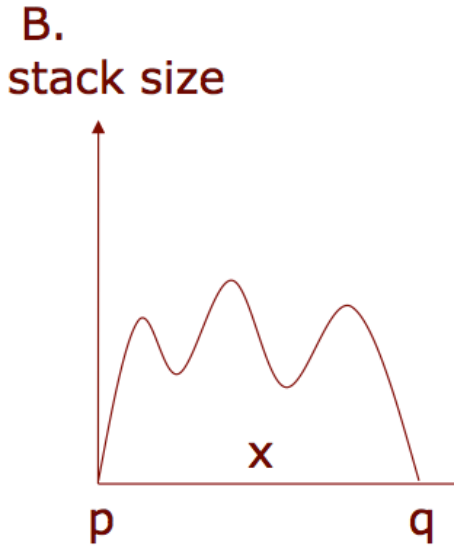
  $(r, u) \in \delta(q, a, \varepsilon)$

  $(q, \varepsilon) \in \delta(s, b, u)$

  so include in G a production

  $A_{pq} \rightarrow aA_{rs}b$

  Note that x can be written ayb, where
  $(p, ayb, \varepsilon) \Rightarrow^* (r, yb, \varepsilon)$ and $(r, z, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$
  [which combine to $(p, yz, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$].



Summary Construction of G

- For each 4-tuple of states p, q, r, s and pair of input symbols a, b, and pair of stack symbols α, β there are rules of G:
  - $A_{pp} \rightarrow \varepsilon$
  - $A_{pq} \rightarrow A_{pr} A_{rq}$
  - $A_{pq} \rightarrow aA_{rs}b$
    provided $(r, u) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, u)$ for some u
- The start symbol of G is: $A_{q0\ qaccept}$
- Note: Not every production will necessarily get used. The productions are enabling, rather than coercing.

Proof that L(G) = L(M)

- Having constructed G from M, it must be shown that the G generates the same language that M accepts.

- This is done in two parts:
  - $L(G) \subseteq L(M)$: More generally, $A_{pq} \Rightarrow^*$ x in G implies $(p, x, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$ in M (claim 2.30).
  - $L(M) \subseteq L(G)$: More generally, $(p, x, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$ in M implies $A_{pq} \Rightarrow^*$ x in G (claim 2.31).

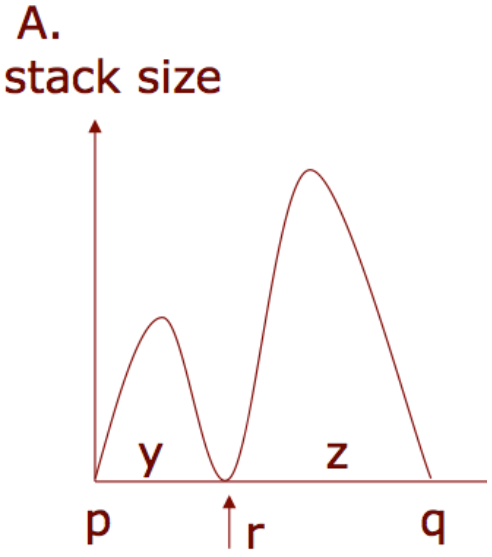
Claim 2.30 (Sipser):L(G) $\subseteq$ L(M)

- For every p, q: If $A_{pq} \Rightarrow^* x$ in G, then $(p, x, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$ in M.
- Proof by induction on the **number n of steps** of the derivation of x in G.
  - **Basis**: n = 1. The only rules in G that yield a terminal string in one step are of the form $A_{pp} \rightarrow \varepsilon$. But in M, $(p, \varepsilon, \varepsilon) \Rightarrow^* (p, \varepsilon, \varepsilon)$ follows from definition of $\Rightarrow^*$.
  - **Induction Step**: n = k+1 > 1. If $A_{pq} \Rightarrow^* x$ in k+1 steps, then we consider the **two possible cases A vs. B** for the first step in the derivation.

Claim 2.30 Induction, case A.

- In case A, the first step is $A_{pq} \rightarrow A_{pr}\, A_{rq}$ and x = yz for some z such that $A_{pr} \Rightarrow^* x$ and $A_{rq} \Rightarrow^* y$.

- By the induction hypothesis, $(p, yz, \varepsilon) \Rightarrow^* (r, z, \varepsilon)$ and $(r, z, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$ in M, so by transitivity of $\Rightarrow^*$, $(p, yz, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$.



Claim 2.30 Induction, case B.

- In case B, the first step is $A_{pq} \rightarrow aA_{rs}b$. Also $x = ayb$ for some y such that $A_{rs} \Rightarrow^* y$.
- $A_{pq} \rightarrow aA_{rs}b$ is a rule of G only due to $(r, u) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, u)$ in M
- By the induction hypothesis, $(r, y, \varepsilon) \Rightarrow^* (s, \varepsilon, \varepsilon)$ in M, so using the transitions above, we have $(p, ayb, \varepsilon) \Rightarrow (r, yb, u) \Rightarrow^*$ [by IH] $(s, b, u) \Rightarrow (q, \varepsilon, \varepsilon)$.

Claim 2.31 (Sipser) $L(M) \subseteq L(G)$

- If $(p, x, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$ in M, then $A_{pq} \Rightarrow^* x$ in G.
- Proof is by induction on the number of steps in the transition sequence.
- The induction step decomposes into the two cases A and B per the construction.
- Please see text for details.

Sipser Proof vs. JFLAP

- Generally, the proof will generate a very large set of productions.
- JFLAP has automation for conversion PDA to Grammar.
- PDA requirements: Every transition pops 1 symbol and pushes 0 or 2 symbols.

PDA to CFG as a way to get grammar $L = \{x \mid \#_1(x) = \#_0(x)\}$

- Think of L as M* where

  $M = \{x \mid \#_1(x) = \#_0(x) \wedge$

  $\forall y < x \ (\#_1(x) \neq \#_0(x))\}$

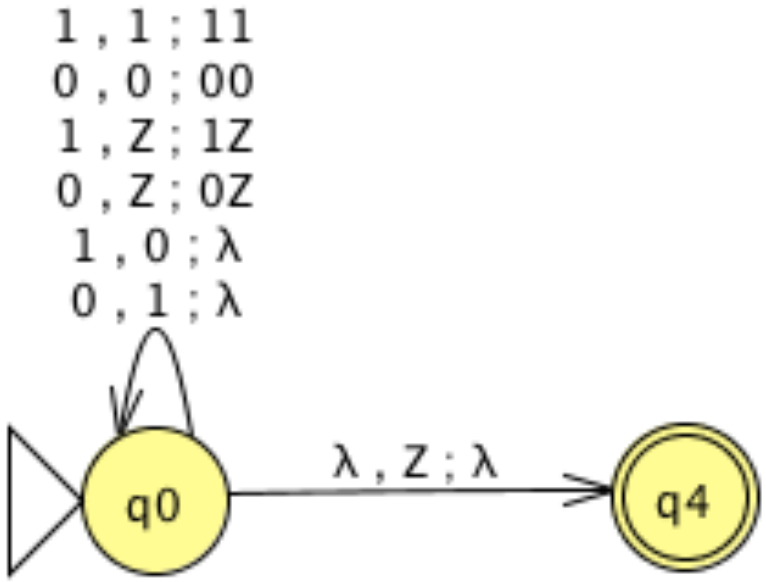
- where $y < x$ means y is a proper prefix of x
- Then a grammar for M is

  $T \rightarrow 0T1 \mid 1T0 \mid 01 \mid 10$

- Thus a grammar for L = M* is

  $S \rightarrow TS \mid \varepsilon$

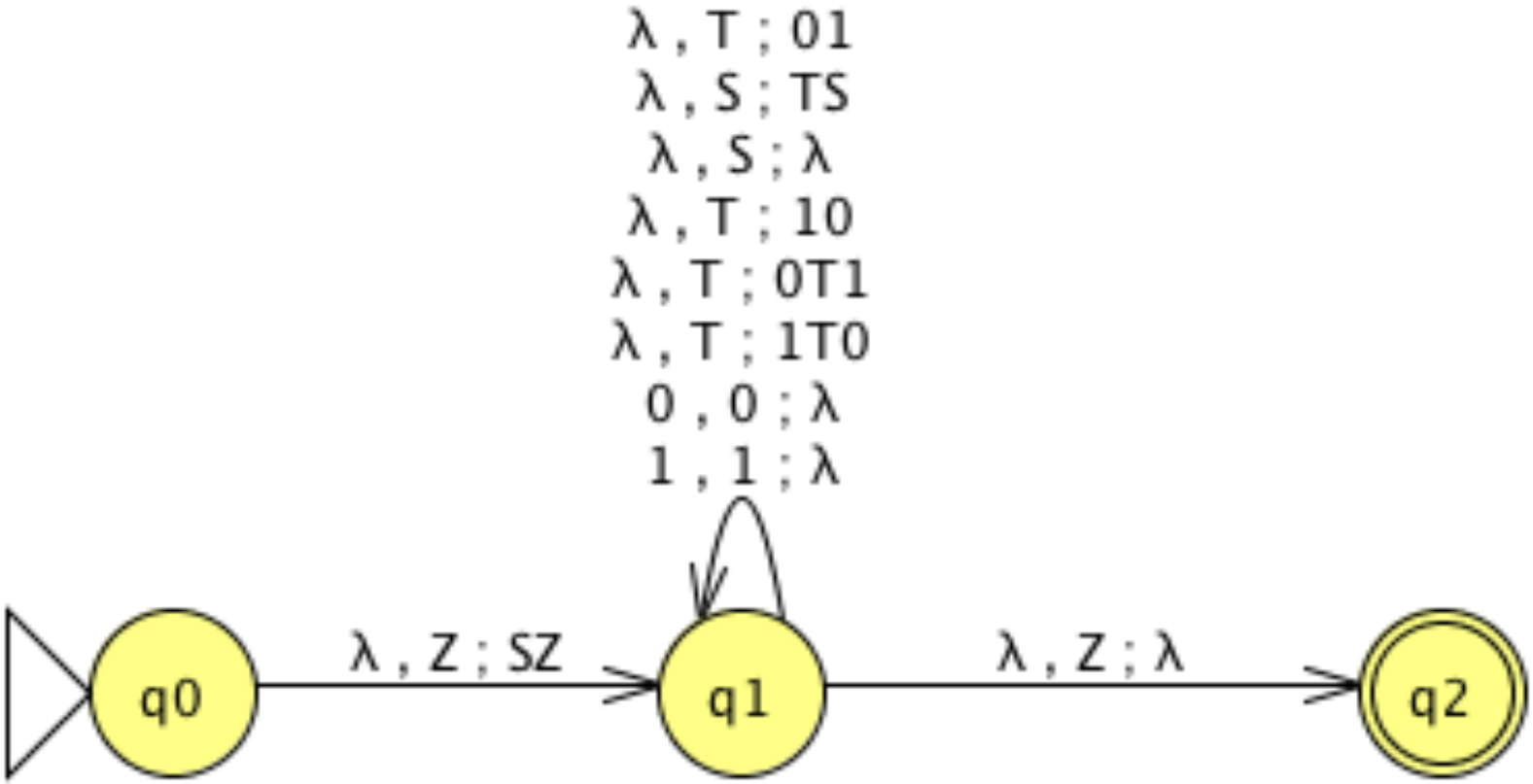
PDA for L



JFLAP-Generated Grammar for L

S	→	λ
S	→	0AS
S	→	1BS
B	→	1BB
A	→	0AA
B	→	0
A	→	1

PDA Using LL (Produce-Match)



PDA Using LR (Shift-Reduce)