Pushdown Automata

Robert M. Keller
Harvey Mudd College
April 2010

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

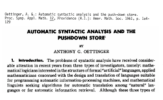
- The type of behavior described on the preceding slide is that of a (deterministic) PDA (DPDA).
- In general, by convention, PDA's are **non-deterministic**.
- The situation for PDA's is different from DFA's: there is **no subset construction**, because the set of all subsets is generally uncountably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


AUTOMATIC SYNTACTIC ANALYSIS AND THE PUSHDOWN STORE

ANTHONY G. OETTINGER

Howard Aiken

Anthony Oettinger

Richard Karp

Robert Keller

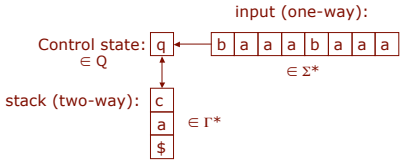
http://genealogy.math.ndsu.nodak.edu/id.php?id=13305

PDA Defined: (Q, Σ, Γ , δ, 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 x Σε x Γε → finite subsets of Q x Γ*
  (The non-determinism is in choosing a member of $\delta(q, \sigma, \gamma)$.)

Notation: $\Sigma\varepsilon = \Sigma \cup \{\varepsilon\}$, $\Gamma\varepsilon = \Gamma \cup \{\varepsilon\}$.
Note: Sipser uses Q x Γε instead of Q x Γ*

PDA Diagram



The **state** is generally of the form (q, x, γ) ∈ Q x Σ* x Γ*, 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.

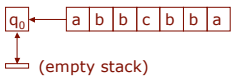
PDA Acceptance

- There are two different **modes** of acceptance.
- **Final-state acceptance** means that the PDA accepts when it is in a **control state** in F after the input has been read. (Sipser uses this.)
- **Empty-stack acceptance** means that the PDA accepts when its **stack is empty** after the input has been read.
- The choice is a matter of convenience; the two can be shown **inter-convertible**.

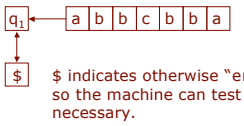
Different authors define PDA's differently.

- Most require that there be a single symbol on the stack to start with, adding one more component to the 6-tuple. (Sipser's machines usually adds this symbol in the first transition they make.)
- Most allow a **string**, rather than just a letter, to be written to the stack in one step. It is simply a matter of adding more control states to achieve the same effect in Sipser's model, which only allows one letter at a time to be added.

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

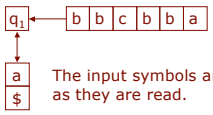


Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



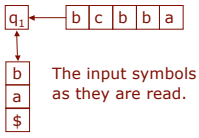
$ indicates otherwise "empty" stack, so the machine can test for that when necessary.

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



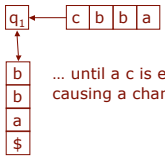
The input symbols are pushed as they are read.

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



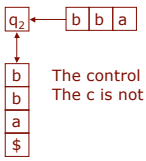
The input symbols are pushed as they are read.

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



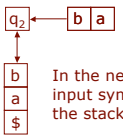
… until a c is encountered in the input, causing a change of control state

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



The control state is changed. The c is not pushed.

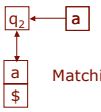
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



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

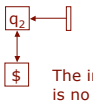
A mismatch stops the process, as no successor is specified.

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



Matching continues.

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

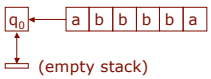
q_3 is the accepting state, so the original input abbcbba is accepted.

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

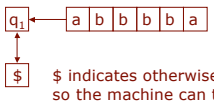
Exercise

- 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.

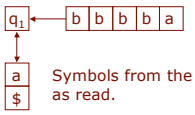


Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



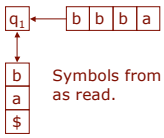
$ indicates otherwise "empty" stack, so the machine can test for that when necessary.

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



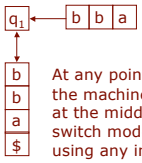
Symbols from the input are pushed as read.

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



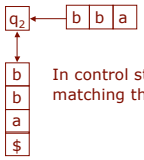
Symbols from the input are pushed as read.

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



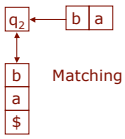
At any point, such as this one, the machine can "guess" that it's at the middle of the string and switch modes (without necessarily using any input).

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



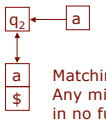
In control state q_2, the machine begins matching the input against the stack.

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



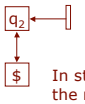
Matching continues.

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



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

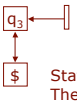
Movie of the PDA accepting a string in $\{xcx^R \mid x \in \{a, b\}^*\}$



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



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

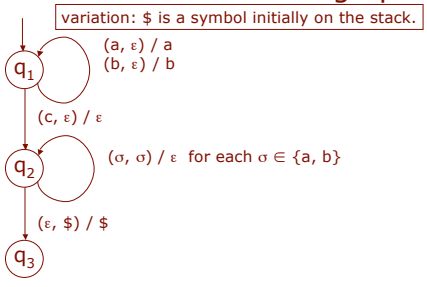
Absolute Rejection

- If we want to reject an input string and all of its extensions, it is only necessary to go to a new state from which no accepting state can ever be reached.

Exercise

- Define the transition function for the previous PDA.

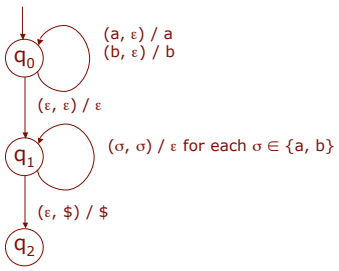
δ can also be described as a graph



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**.

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



δ can also be described by "transition rules"

$q_1, a, \varepsilon \rightarrow q_1, a$
$q_1, b, \varepsilon \rightarrow q_1, b$
$q_1, c, \varepsilon \rightarrow q_2, \varepsilon$
$q_2, \sigma, \sigma \rightarrow q_2, \varepsilon$ for each $\sigma \in \{a, b\}$
$q_2, \varepsilon, \$ \rightarrow q_3, \$$

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

The ⇒ and ⇒* 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 $(q, x, \gamma) \Rightarrow^* (q'', x'', \gamma'')$.

The 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.

Note on Acceptance Modes

- The following discussion is based on the PDA model wherein the stack always starts with a specified initial symbol (unlike Sipser's model).
- No generality is lost.

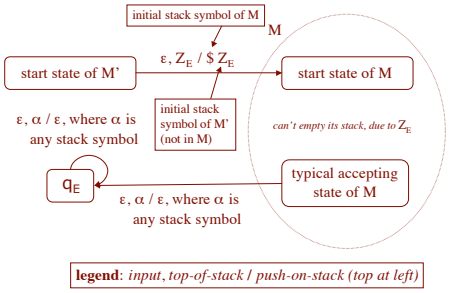
Interconvertibility of Acceptance Modes

- **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

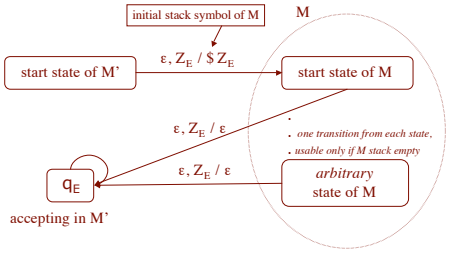


legend: *input*, *top-of-stack* / *push-on-stack* (*top at left*)

Interconvertibility of Acceptance Modes (2)

- **For every PDA M accepting by empty stack, there is a PDA M′ accepting the same language by final 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 an ε transition to the initial state of M, with which M′ pushes $ Z_E where $ is the initial stack symbol of M.
- Introduce a new accepting state q_F as the only accepting state of M′.
- Add new ε 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, but only then.

Empty Stack → Accepting State



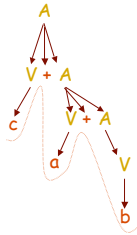
legend: *input*, *top-of-stack* / *push-on-stack* (*top at left*)

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 → V | V + A
V → a | b | 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 some PDA accepting by empty stack.

CFL → PDA Lemma

- Every context free language is accepted by some PDA.

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

- Let G be a context-free grammar for the CFL.
- Construct 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.
- If the top of stack is a terminal, it is **matched** against the corresponding symbol in the input and removed.
- If it 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, the machine can enter an accepting state.

Example of CFG to Machine

Production	Transition (top of stack at left)	
	$q_0, \varepsilon, \varepsilon \to q_1, S\$$	init
$S \to (T$	$q_1, \varepsilon, S \to q_1, (T$	produce
$S \to (ST$	$q_1, \varepsilon, S \to q_1, (ST$	
$S \to (TS$	$q_1, \varepsilon, S \to q_1, (TS$	
$S \to (STS$	$q_1, \varepsilon, S \to q_1, (STS$	
$T \to)$	$q_1, \varepsilon, T \to q_1, ($	
	$q_1, (, (\to q_1, \varepsilon$	match
	$q_1,),) \to q_1, \varepsilon$	
	$q_1, \$, \varepsilon \to q_2, \varepsilon$	accept

q_0 is initial, q_2 is accepting

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

Derivation vs. Transition Sequence

Derivation	Transitions	
	$q_0, ()(), \varepsilon$	init
$S \Rightarrow$	$q_1, ()(), S\$$	produce S → (TS
$(TS \Rightarrow$	$q_1, ()(), (TS\$$	match (
	$q_1,)(), TS\$$	produce T →)
$()S \Rightarrow$	$q_1,)(),)S\$$	match)
	$q_1,(), S\$$	produce S → (T
$()(T \Rightarrow$	$q_1,(), (T\$$	match (
	$q_1,), T\$$	produce T →)
$()()$	$q_1,),)\$$	match)
	$q_1, \varepsilon, \$$	
	$q_2, \varepsilon, \$$	accept

Another Way of Showing

- S → (T
  S → (ST
  S → (TS
  S → (STS
  T →)
- (()()) | 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.

PDA → CFG Lemma

- For every PDA M, there is a CFG G that generates the language accepted by M.
- We will follow Sipser's elegant proof.
- M is assumed to only push one stack symbol at a time.

PDA → CFG Lemma

Additional constraints imposed on M, without loss of generality:

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

  but not both (and not neither).

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

  If M does neither, have it push something, then immediately pop it.]
- M has only one accept state
- M empties its stack before entering the accept state.

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

We want:

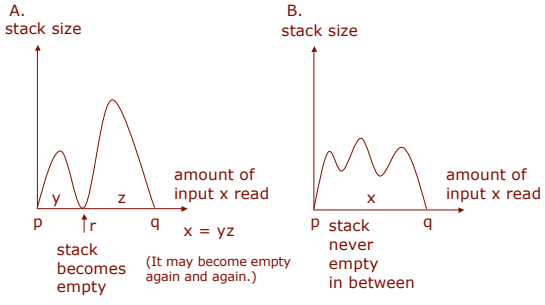
$\forall x \in \Sigma^*$ (**if** $(p, x, \varepsilon) \Rightarrow^* (q, \varepsilon, \varepsilon)$ in M **then** $A_{pq} \Rightarrow^* x$ in G).

How 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 **stays non-empty** in between first and last transitions.

Two Possibilities



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. Then in M we have
  $(r, \alpha) \in \delta(q, a, \varepsilon)$ for some r,α and
  $(q, \varepsilon) \in \delta(s, b, \beta)$ for some s, β

  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

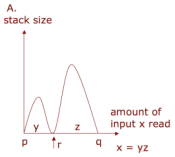
- For each 4-tuple of states p, q, r, s and pair of input symbols a, b, and pair of stack symbols α, β there are rules of G:
  - $A_{pp} \rightarrow \varepsilon$
  - $A_{pq} \rightarrow A_{pr} A_{rq}$
  - $A_{pq} \rightarrow aA_{rs}b$
    provided $(r, \alpha) \in \delta(q, a, \varepsilon)$ and $(q,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

Claim 2.30 (Sipser):

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

Claim 2.31 (Sipser)

- 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.

2nd Proof of the CFL ⇒ 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:
  $q_0, \sigma, \varepsilon \rightarrow q_0, \sigma$
  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:
  $q_0, \varepsilon, x_n \rightarrow q_1, \varepsilon$
  $q_1, \varepsilon, x_{n-1} \rightarrow q_2, \varepsilon$
  $q_2, \varepsilon, x_{n-2} \rightarrow q_3, \varepsilon$
  . . .
  $q_n, \varepsilon, x_1 \rightarrow q_0, A$
  where the q_i other than q_0 are distinct for each production.
- This pda simulates a **rightmost** derivation of an accepted input string.

Example of Bottom-Up

Production	Transitions
S → (T	$q_0, \varepsilon, T \rightarrow q_1, \varepsilon$ $q_1, \varepsilon, (\rightarrow q_0, S$
T → S)	$q_0, \varepsilon,) \rightarrow q_2, \varepsilon$ $q_2, \varepsilon, S \rightarrow q_0, T$
S → ()	$q_0, \varepsilon,) \rightarrow q_3, \varepsilon$ $q_3, \varepsilon, (\rightarrow q_0, S$
S → 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 ⇒ (T ⇒ (S) ⇒ (SS) ⇒ (S()) ⇒ (()())

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 | $