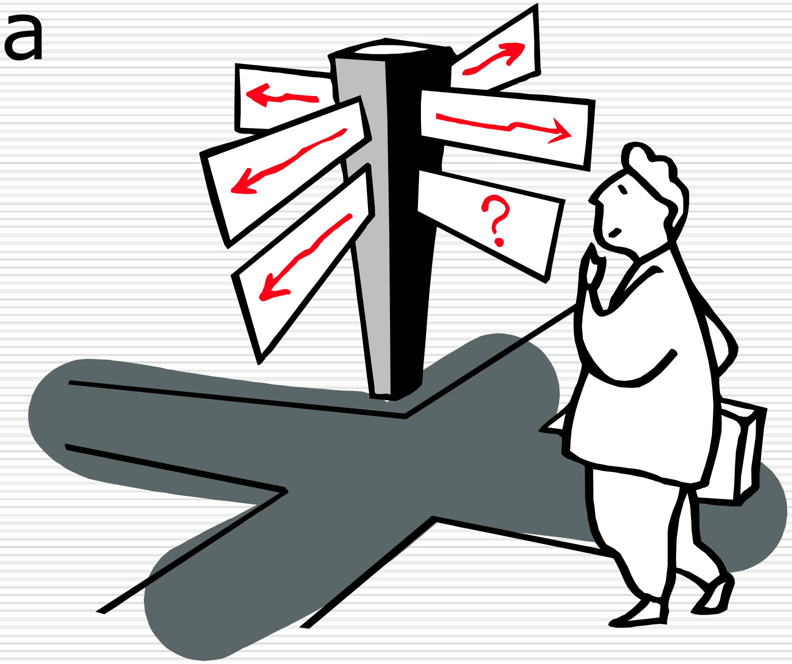
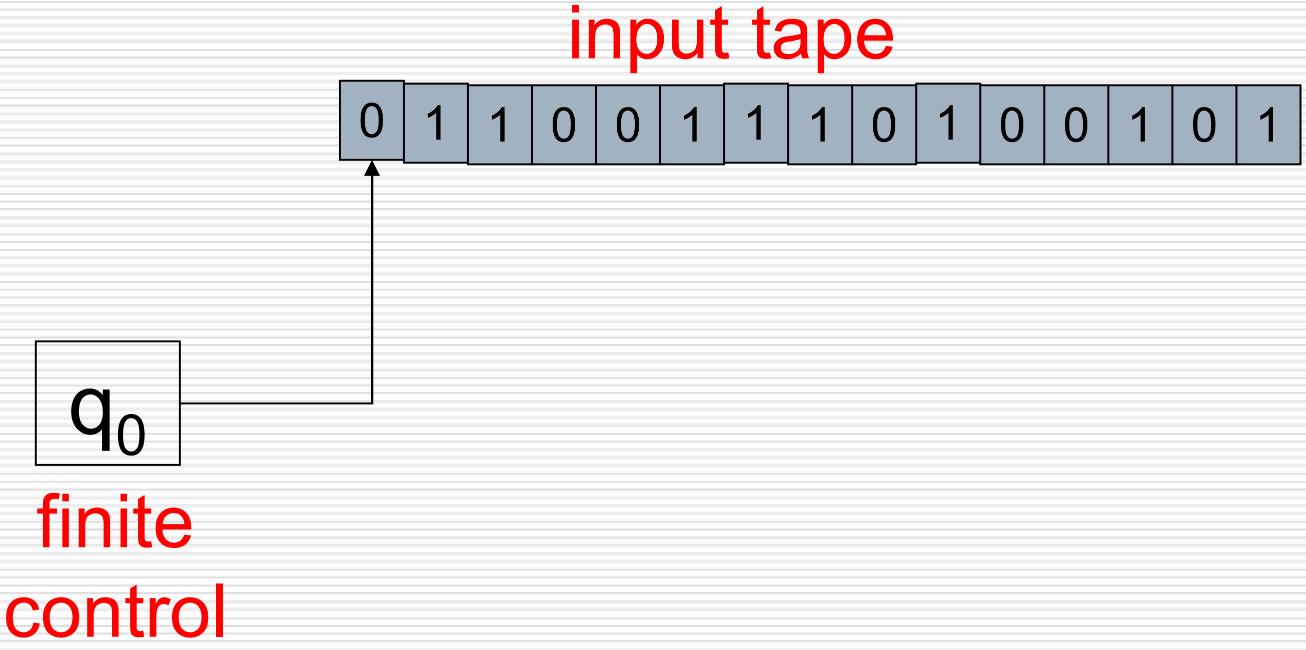
2.2 Pushdown Automata

- ☐ Pushdown Automata
- ☐ CFG = PDA
- ☐ Deterministic PDA



Machine View of FA



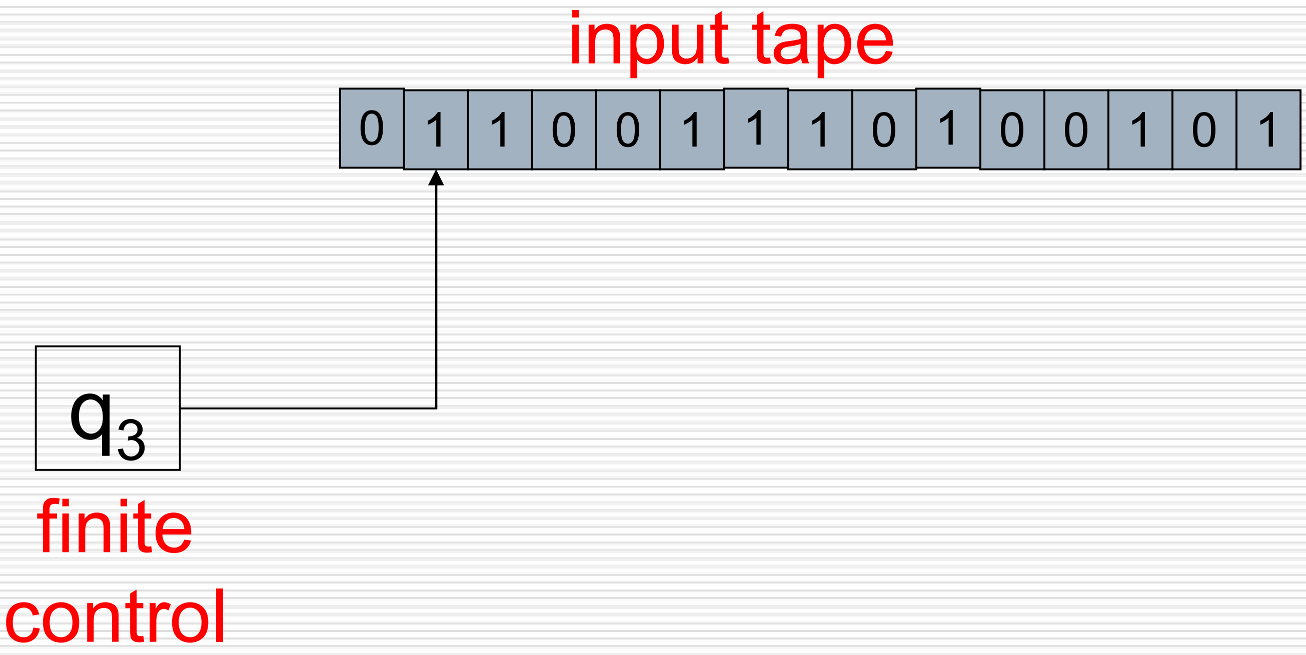
input tape

0 1 1 0 0 1 1 1 0 1 0 0 1 0 1

q_0

finite control

Machine View of FA



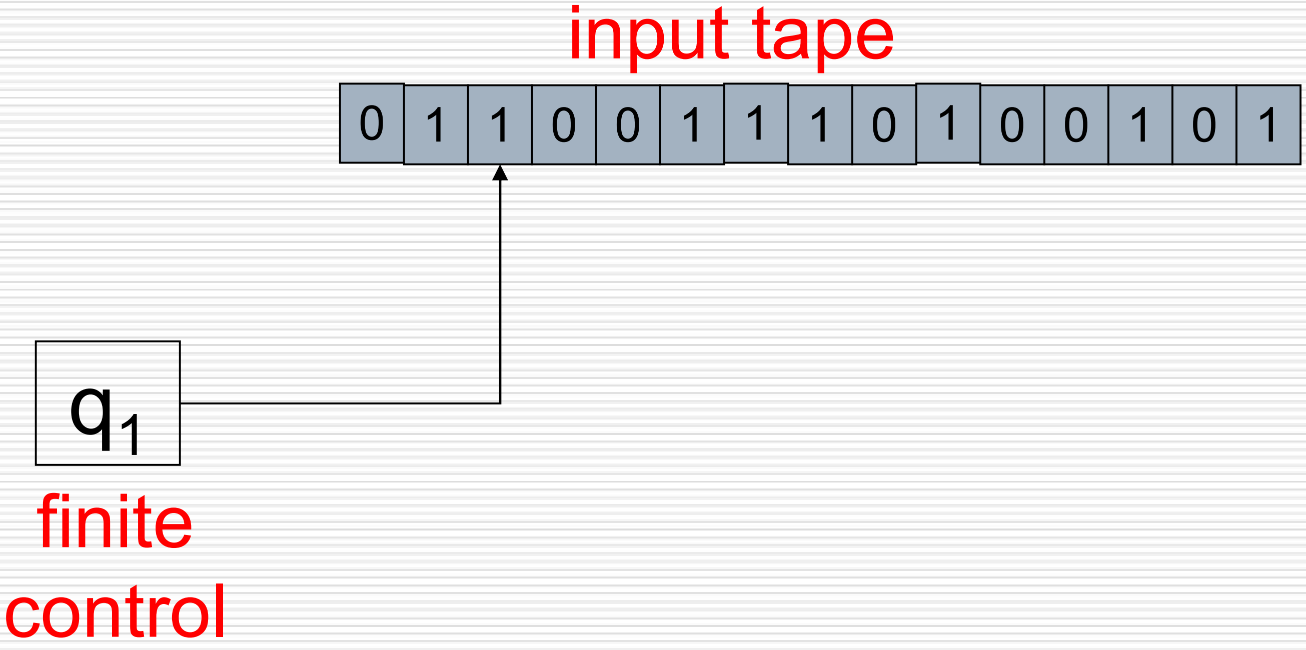
input tape

0 1 1 0 0 1 1 1 0 1 0 0 1 0 1

q_3

finite control

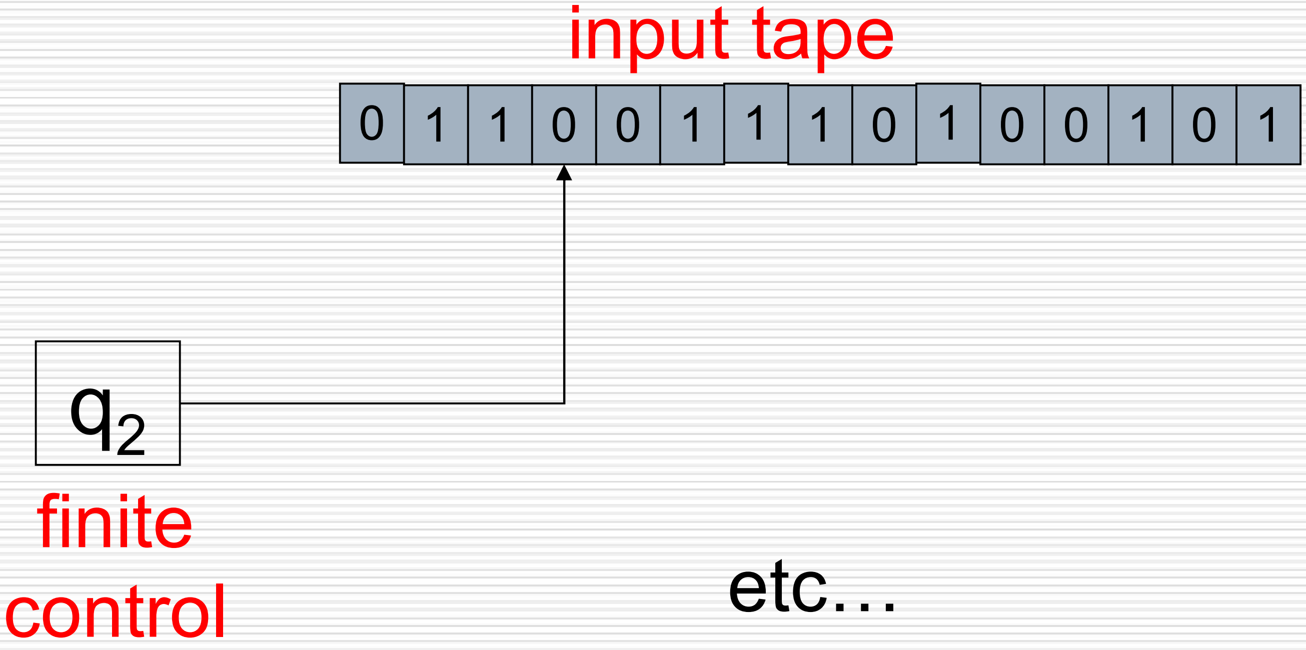
Machine View of FA



input tape

finite control

Machine View of FA



input tape

0	1	1	0	0	1	1	1	0	1	0	0	1	0	1

q_2

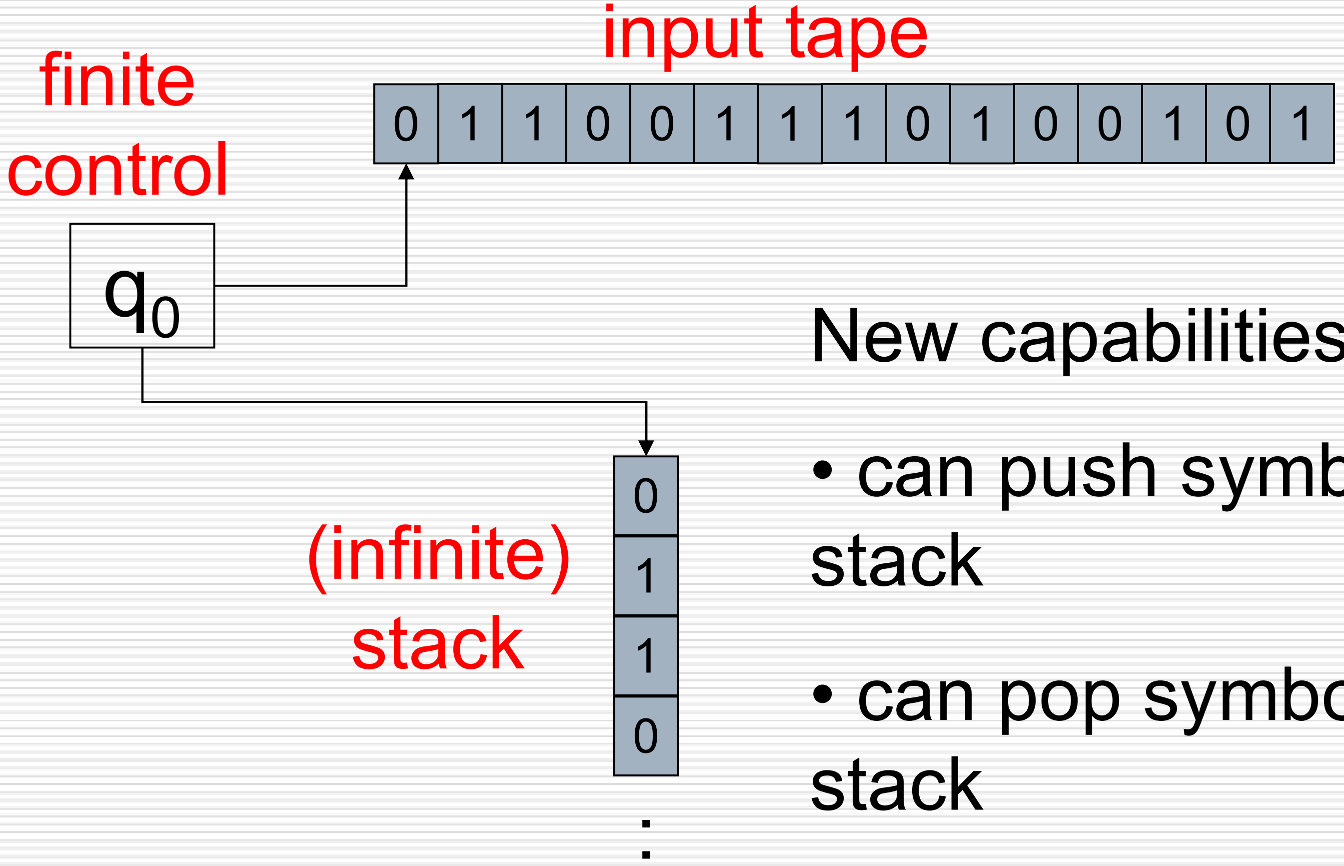
finite control

etc…

A More Powerful Machine

- limitation of FA related to fact that they can only “remember” a bounded amount of information
- What is the simplest alteration that adds unbounded “memory” to our machine?
- Should be able to recognize, e.g., $\{0^n1^n: n \geq 0\}$

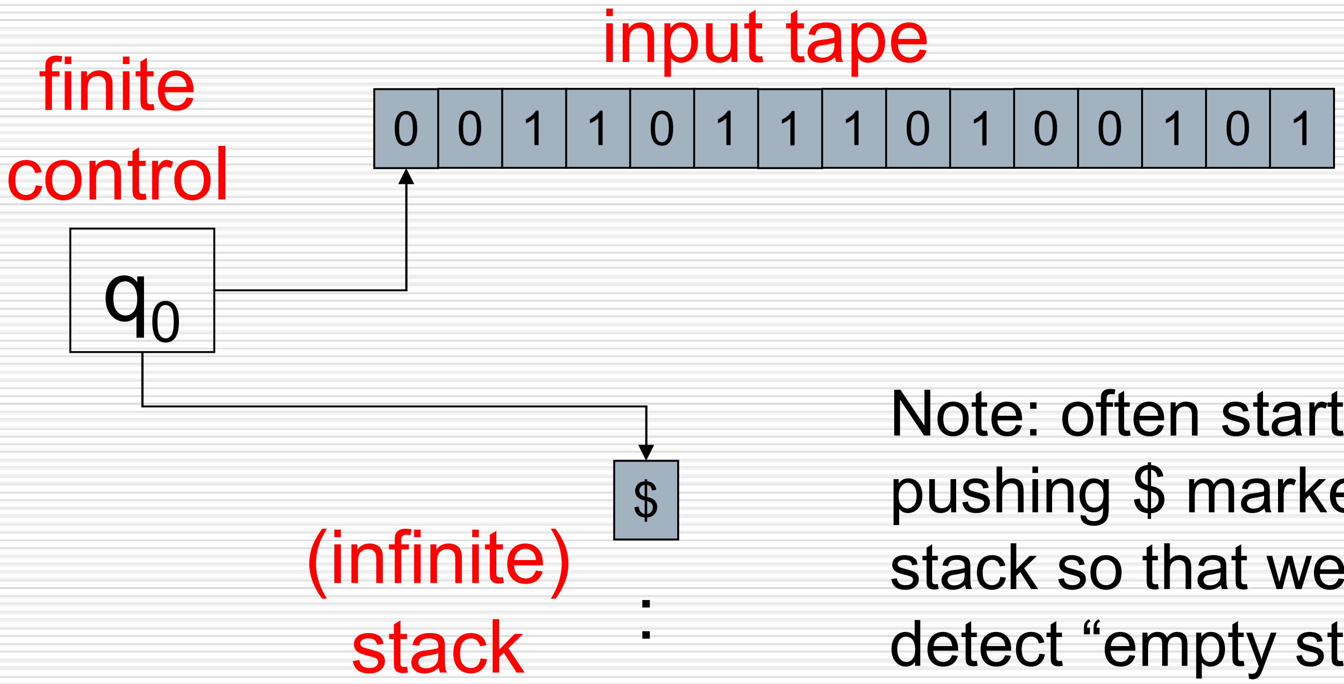
Pushdown Automata



New capabilities:

- can push symbol onto stack
- can pop symbol off stack

Pushdown Automata



input tape

0 0 1 1 0 1 1 1 0 1 0 0 1 0 1

finite control

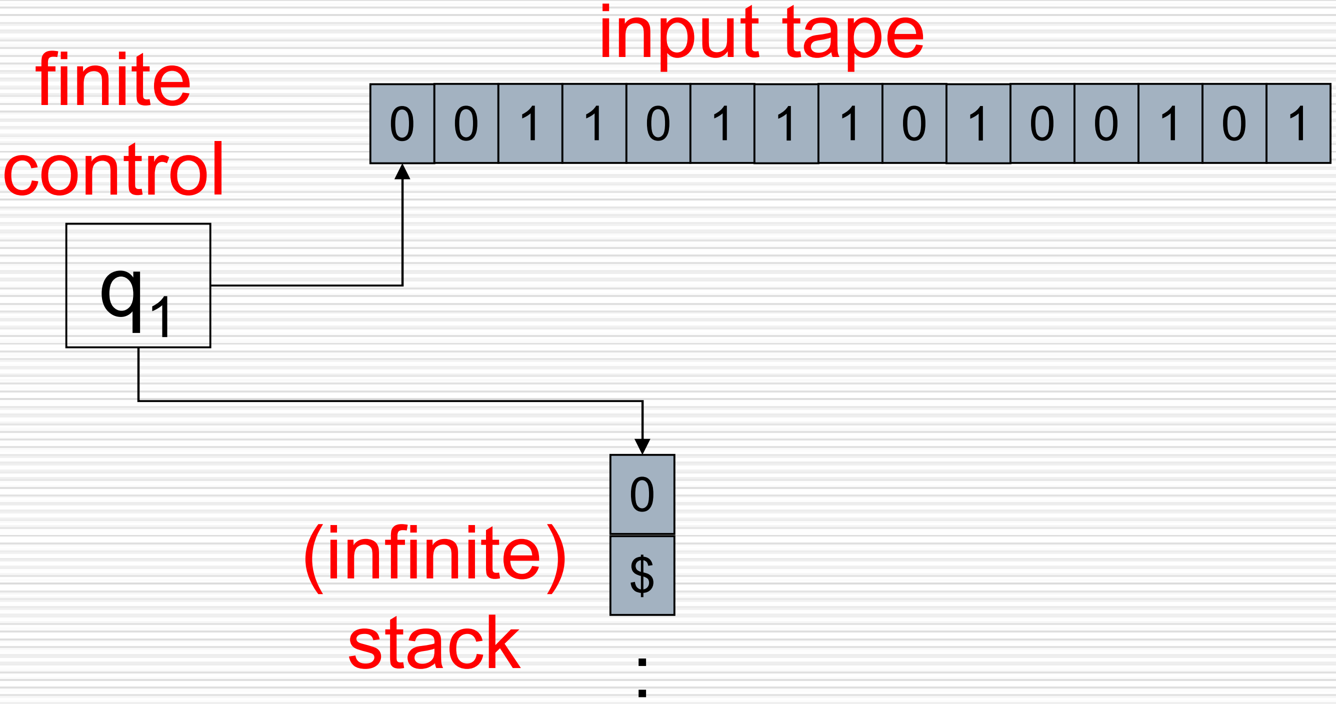
q_0

$

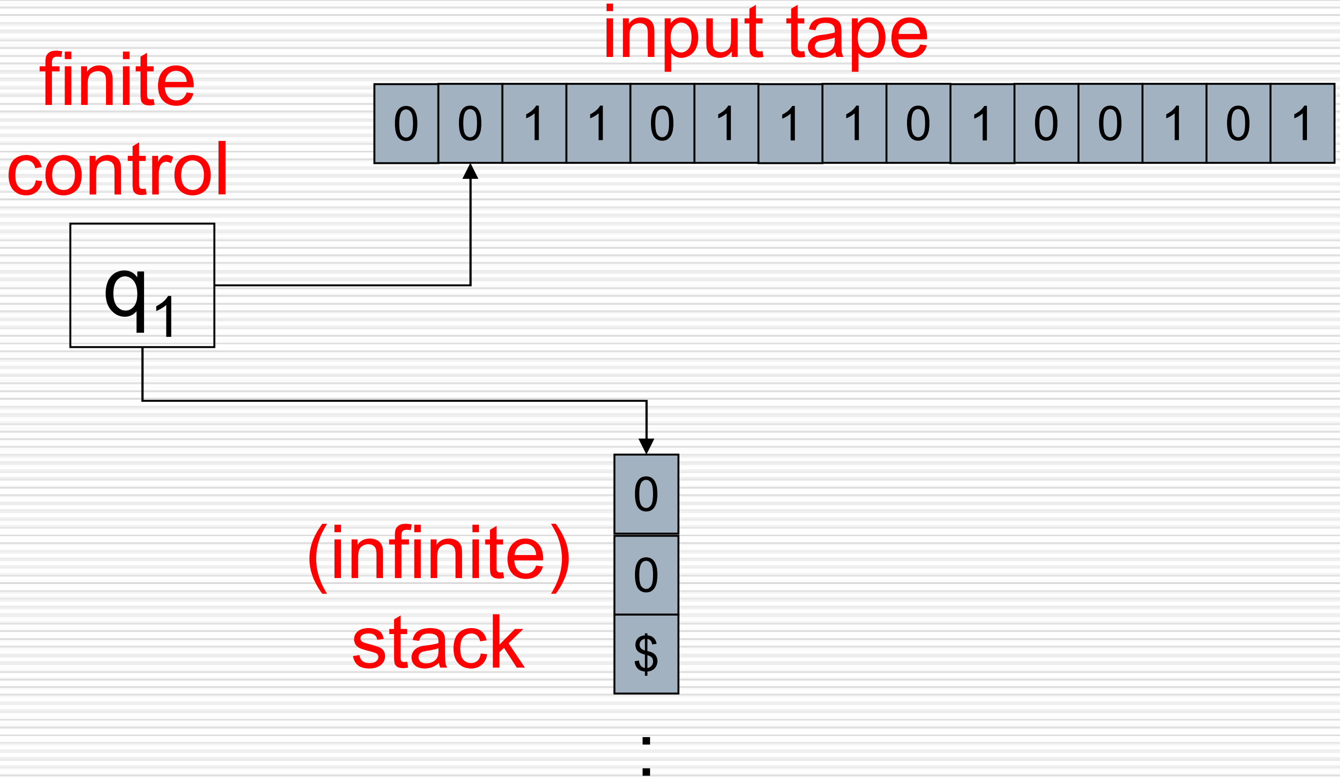
(infinite) stack

Note: often start by pushing $ marker onto stack so that we can detect “empty stack”

Pushdown Automata



Pushdown Automata

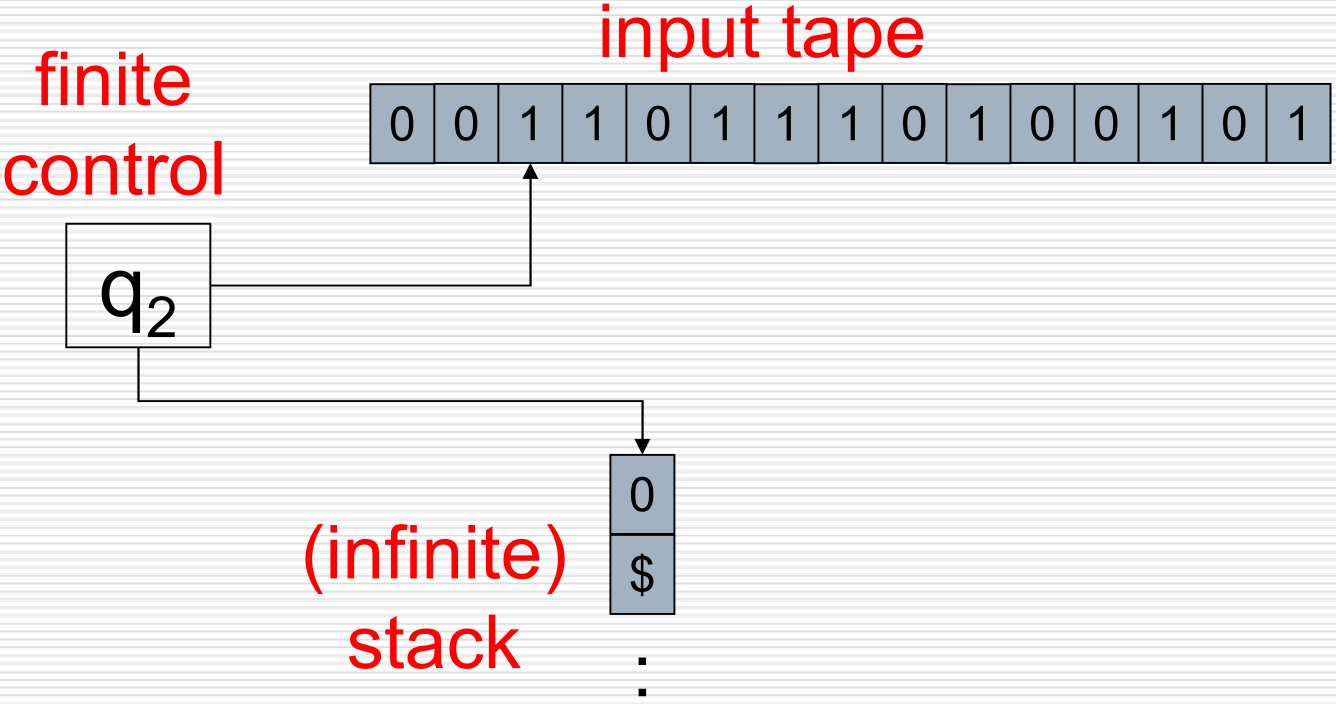


finite control

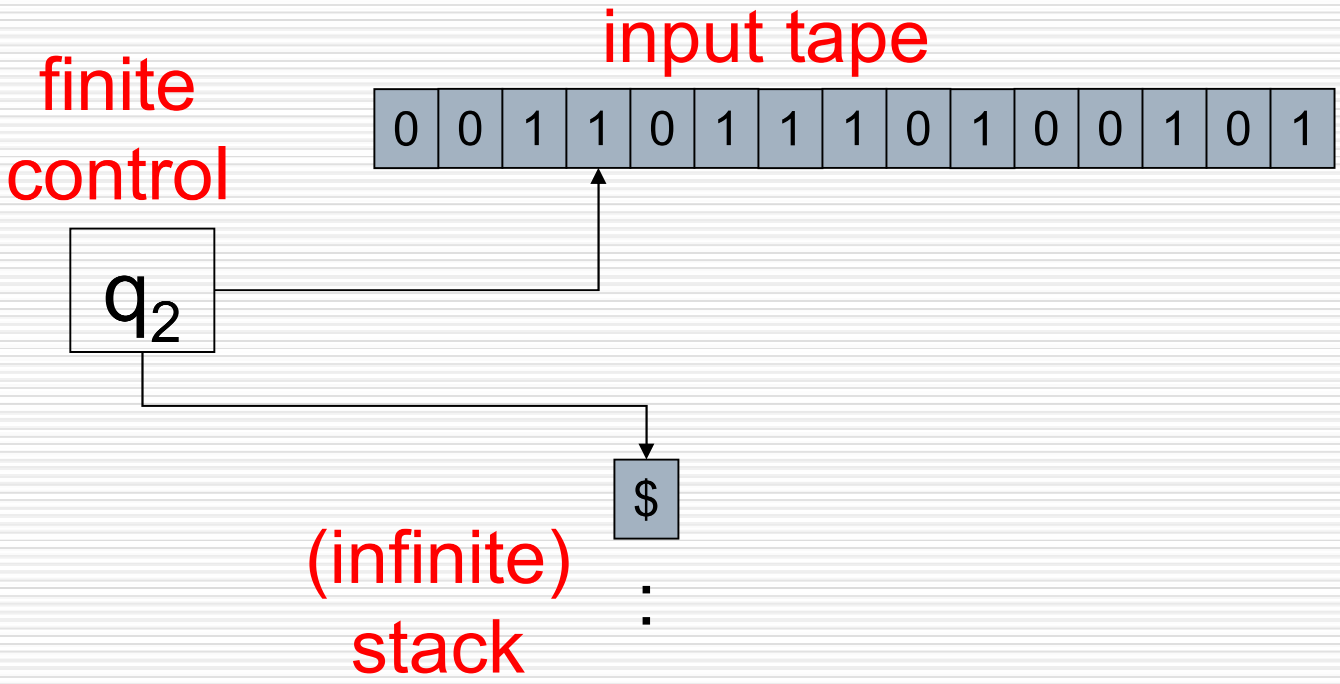
input tape

(infinite) stack

Pushdown Automata



Pushdown Automata

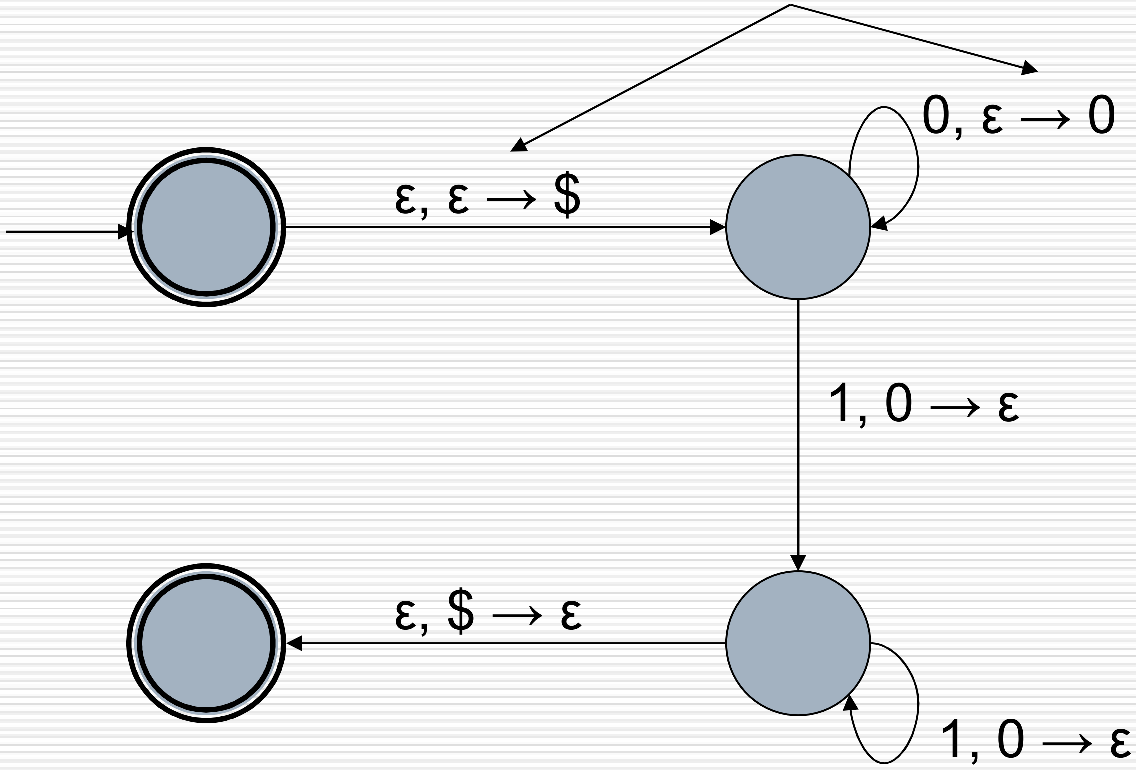


Pushdown Automata (PDA)

- We will define nondeterministic PDA immediately
  - potentially several choices of “next step”
  - essentially an ε-NFA with a stack
- Deterministic PDA defined later
  - weaker than NPDA
- Two ways to describe NPDA
  - diagram
  - formal definition

NPDA Diagram

transition: input symbol read, stack symbol popped → stack symbol pushed



NPDA Operation

- Taking a transition labeled:

  $a, b \rightarrow c$

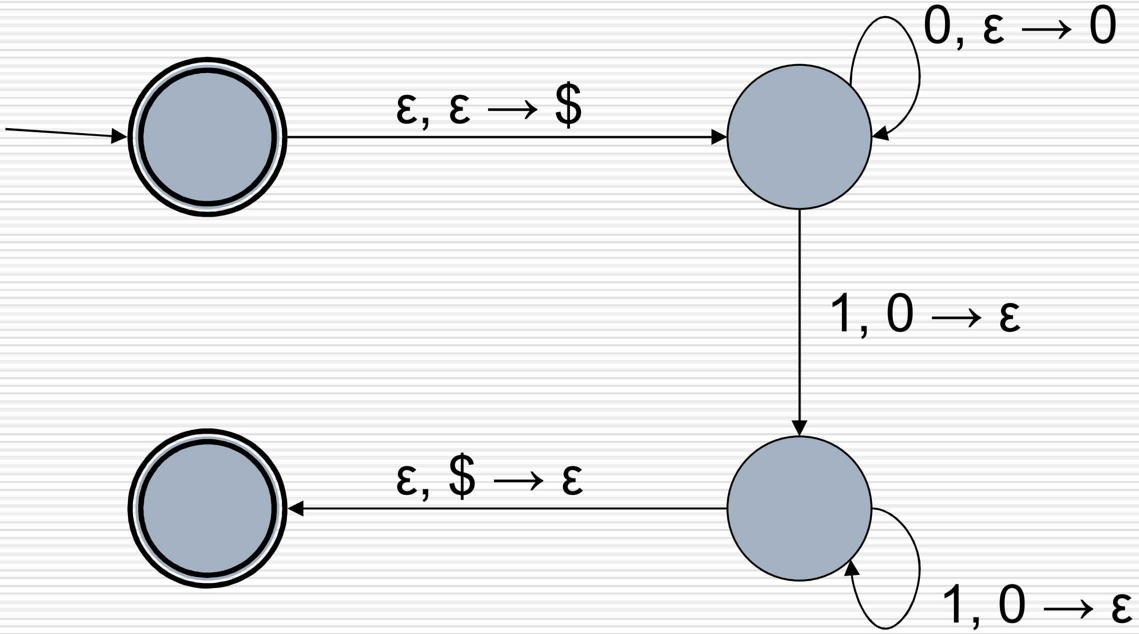
  - $a \in (\Sigma \cup \{\varepsilon\})$ Σ: input alphabet
  - $b, c \in (\Gamma \cup \{\varepsilon\})$ Γ: stack alphabet

  - read a from input, or don’t read from input if $a = \varepsilon$
  - pop b from stack, or don’t pop from stack if $b = \varepsilon$
  - push c onto stack, or don’t push onto stack if $c = \varepsilon$

Example NPDA

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, \$\}$



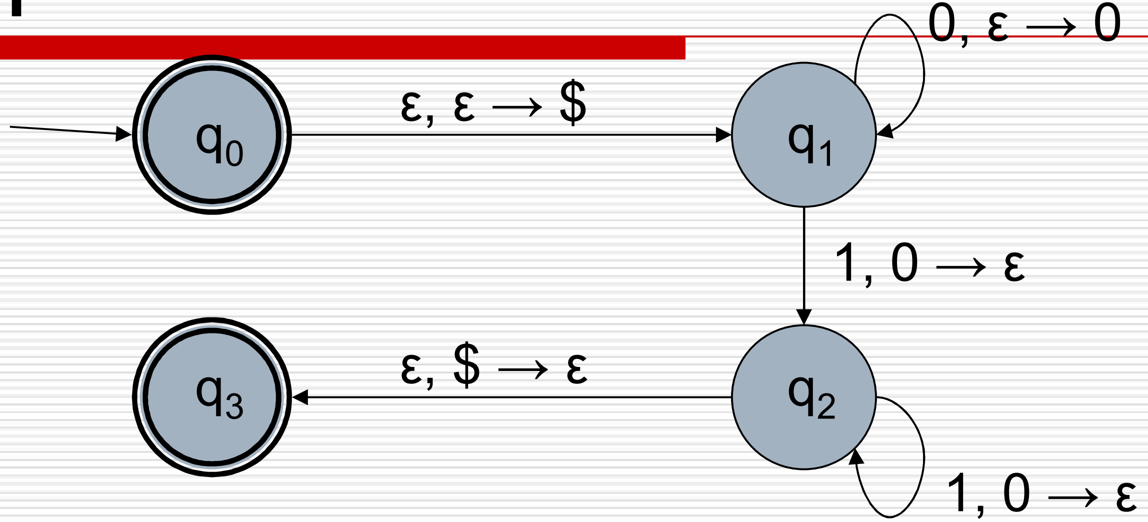
Input: 0011
001
0101

What language does this NPDA accept?

Formal Definition of NPDA

- A NPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:
  - Q is a finite set of states
  - Σ is a finite input alphabet
  - Γ is a finite stack alphabet
  - $\delta: Q \text{ x } (\Sigma \cup \{\varepsilon\}) \text{ x } (\Gamma \cup \{\varepsilon\}) \rightarrow \wp(Q \text{ x } (\Gamma \cup \{\varepsilon\}))$ is the transition function
  - q_0 is an element of Q called the start state
  - F is a subset of Q called the accepting states

Example of Formal Definition



- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \$\}$
- $F = \{q_0, q_3\}$

$\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$

$\delta(q_1, 0, \varepsilon) = \{(q_1, 0)\}$

$\delta(q_1, 1, 0) = \{(q_2, \varepsilon)\}$

$\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$

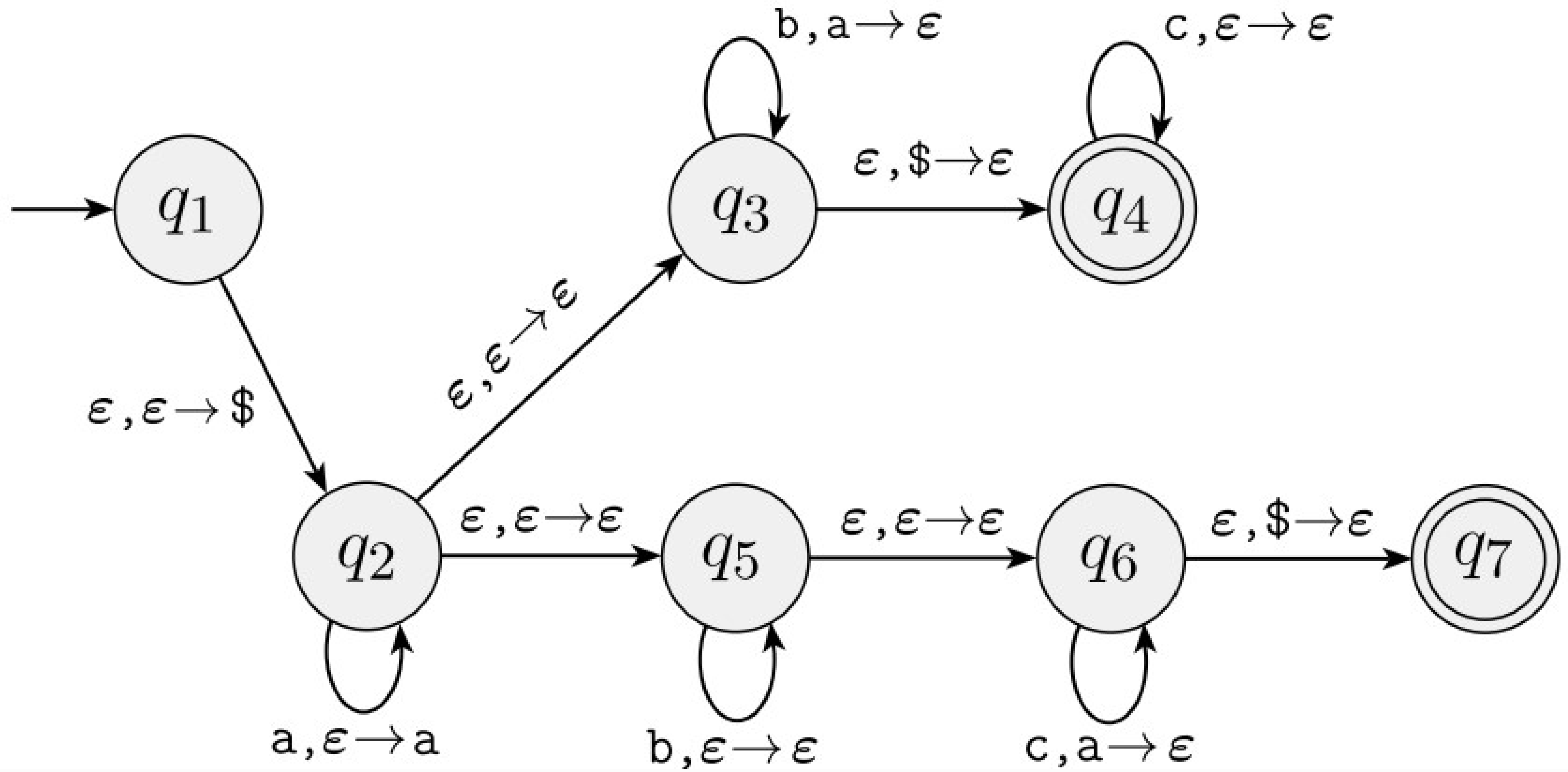
$\delta(q_2, \varepsilon, \$) = \{(q_3, \varepsilon)\}$

Example 2.16

$\{a^ib^jc^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

Example 2.16

$\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

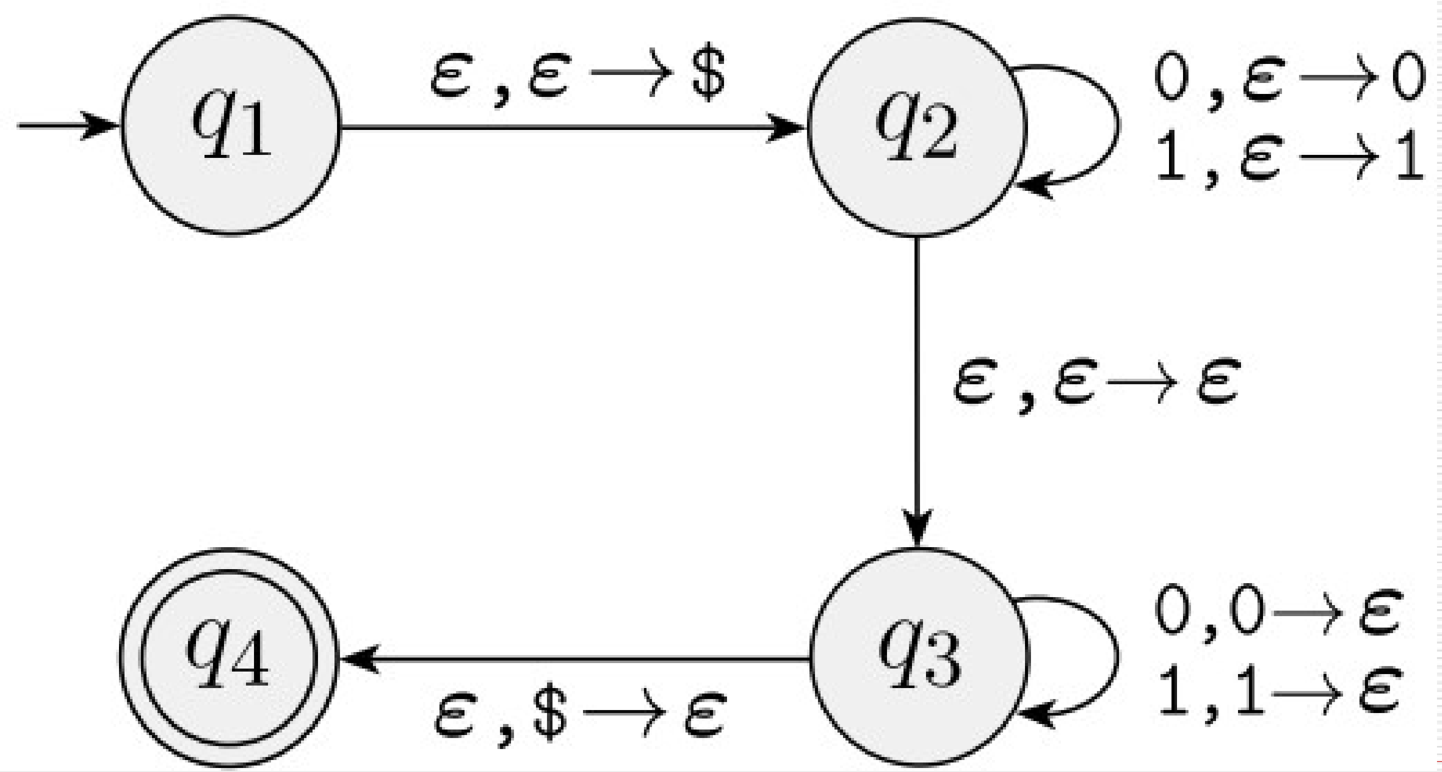


Example 2.18

$\{ww^R \mid w \in \{0, 1\}^*\}$

Example 2.18

$\{ww^R \mid w \in \{0, 1\}^*\}$



Instantaneous Descriptions

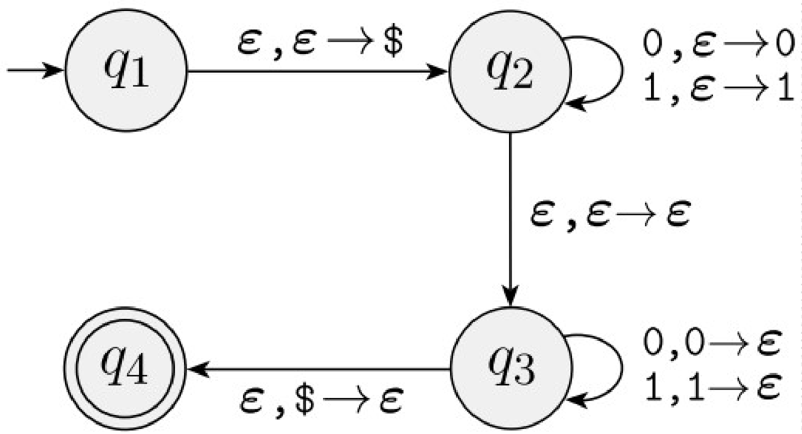
- To reason about PDA computation, we use Instantaneous Descriptions (ID) of the PDA. An ID is a triple:

  $$(q, w, \gamma)$$

  where q is the state, w the remaining input, and γ the stack content.
- If $(p, Y) \in \delta(q, a, X)$, then we denote

  $$(q, aw, X\beta) \vdash (p, w, Y\beta)$$
- We define $\vdash^*$ to be the reflexive-transitive closure of $\vdash$.

NPDA Computation Example 2.18



$$\begin{tikzcd}
(q_1, 0110, \varepsilon) \arrow[d] & & & & \\
(q_2, 0110, \$) \arrow[r] \arrow[d] & (q_2, 110, 0\$) \arrow[r] \arrow[d] & (q_2, 10, 10\$) \arrow[r] \arrow[d] & (q_2, 0, 110\$) \arrow[r] \arrow[d] & (q_2, \varepsilon, 0110\$) \arrow[d] \\
(q_3, 0110, \$) & (q_3, 110, 0\$) & (q_3, 10, 10\$) \arrow[d] & (q_3, 0, 110\$) & (q_3, \varepsilon, 0110\$) \\
 & & (q_3, 0, 0\$) \arrow[d] & & \\
 & & (q_3, \varepsilon, \$) \arrow[d] & & \\
 & & \mathbf{(q_4, \varepsilon, \varepsilon)} & &
\end{tikzcd}$$

Acceptance by *Final State* and by *Empty Stack*

☐ Acceptance by **final state**

$(q_0, w, \$) \vdash^* (p, \varepsilon, \gamma)$ p in F

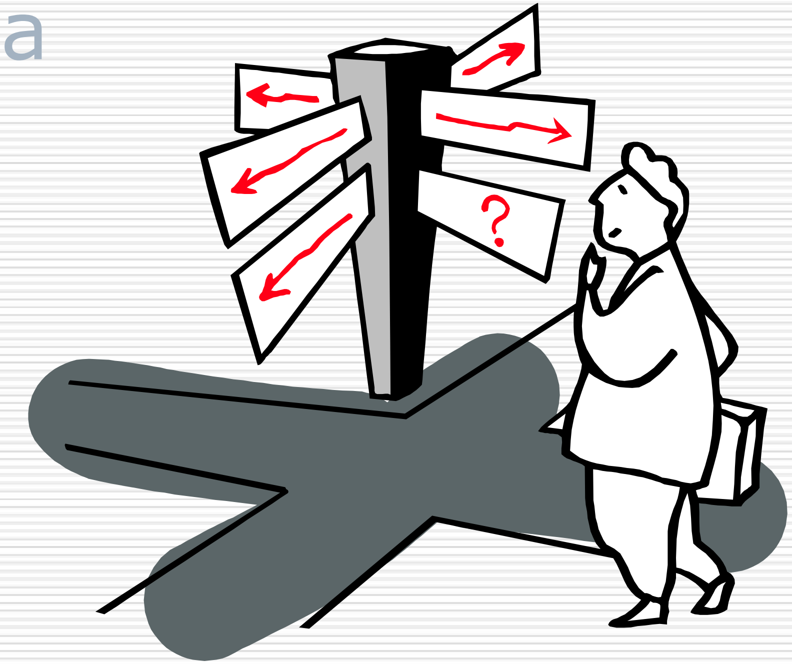
☐ Acceptance by **empty stack**

$(q_0, w, \$) \vdash^* (p, \varepsilon, \varepsilon)$

☐ They are equivalent

2.2 Pushdown Automata

- Pushdown Automata
- CFG = PDA
- Deterministic PDA



Equivalence of NPDA and CFG

Theorem 2.12: a language is context free iff some pushdown automaton recognizes it.

Must prove two directions:

- L is recognized by a NPDA ⇒ L is described by a CFG.
- L is described by a CFG ⇐ L is recognized by a NPDA.

From CFG to NPDA

Proof of (⇐): L is described by a CFG implies L is recognized by a NPDA.

an initial idea to design an NPDA from the CFG:

1. start from the start symbol; non-deterministically guess the derivation, and form it on the stack
2. compare the resulting terminal string with the input string; accept if they are identical

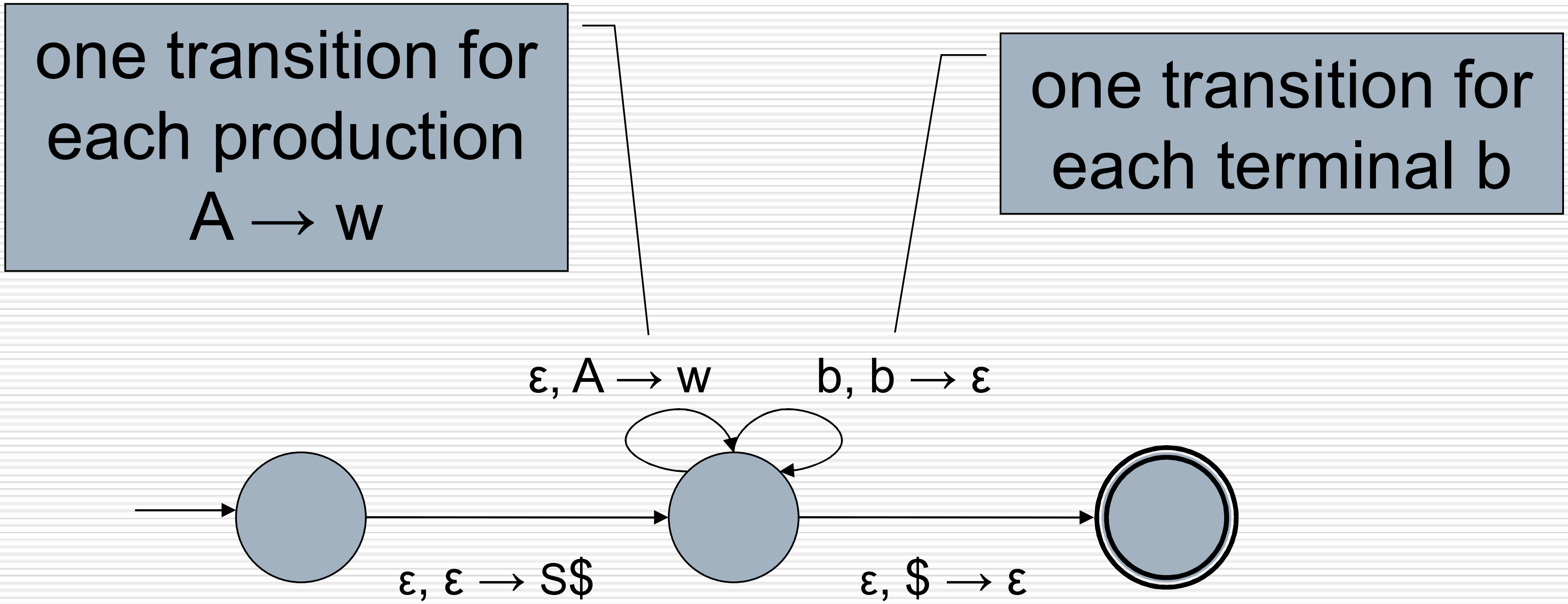
From CFG to NPDA

- What is wrong with this approach?
  - only have access to the top of stack
- Allow to match stack terminals with the input *during* the process of producing the derivation on the stack

From CFG to NPDA

- informal description of construction:
  - place $ and start symbol S on the stack
  - repeat:
    - if the top of the stack is a non-terminal A, pick a production with A on the lhs and substitute the rhs for A on the stack
    - if the top of the stack is a terminal b, read b from the input, and pop b from the stack.
    - if the top of the stack is $, enter the accept state.

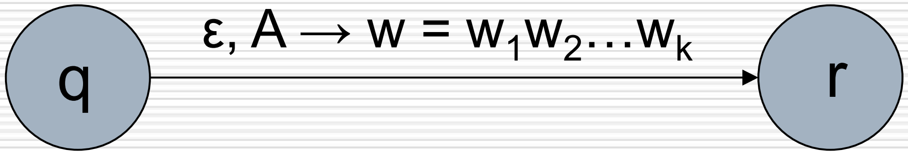
From CFG to NPDA



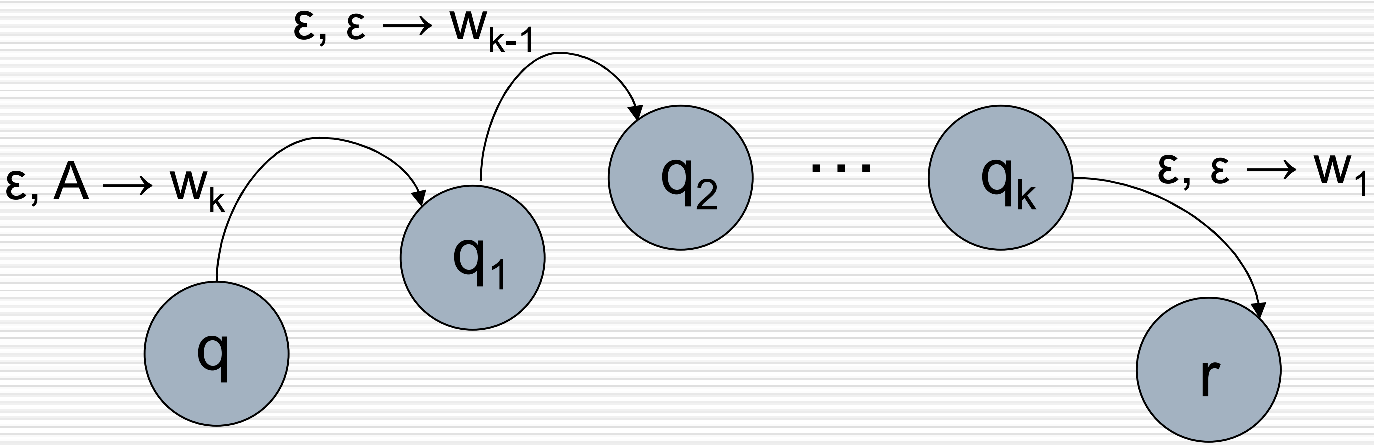
one transition for each production A → w

one transition for each terminal b

From CFG to NPDA



shorthand for:



Example: CFG → NPDA

$S \rightarrow aTb \mid b$

$T \rightarrow Ta \mid \varepsilon$

From NPDA to CFG

Proof of ($\Rightarrow$): L is recognized by a NPDA implies L is described by a CFG.

- harder direction
- first step: convert NPDA into “normal form”:
  - single accept state
  - empties stack before accepting
  - each transition *either* pushes *or* pops a symbol, but not both

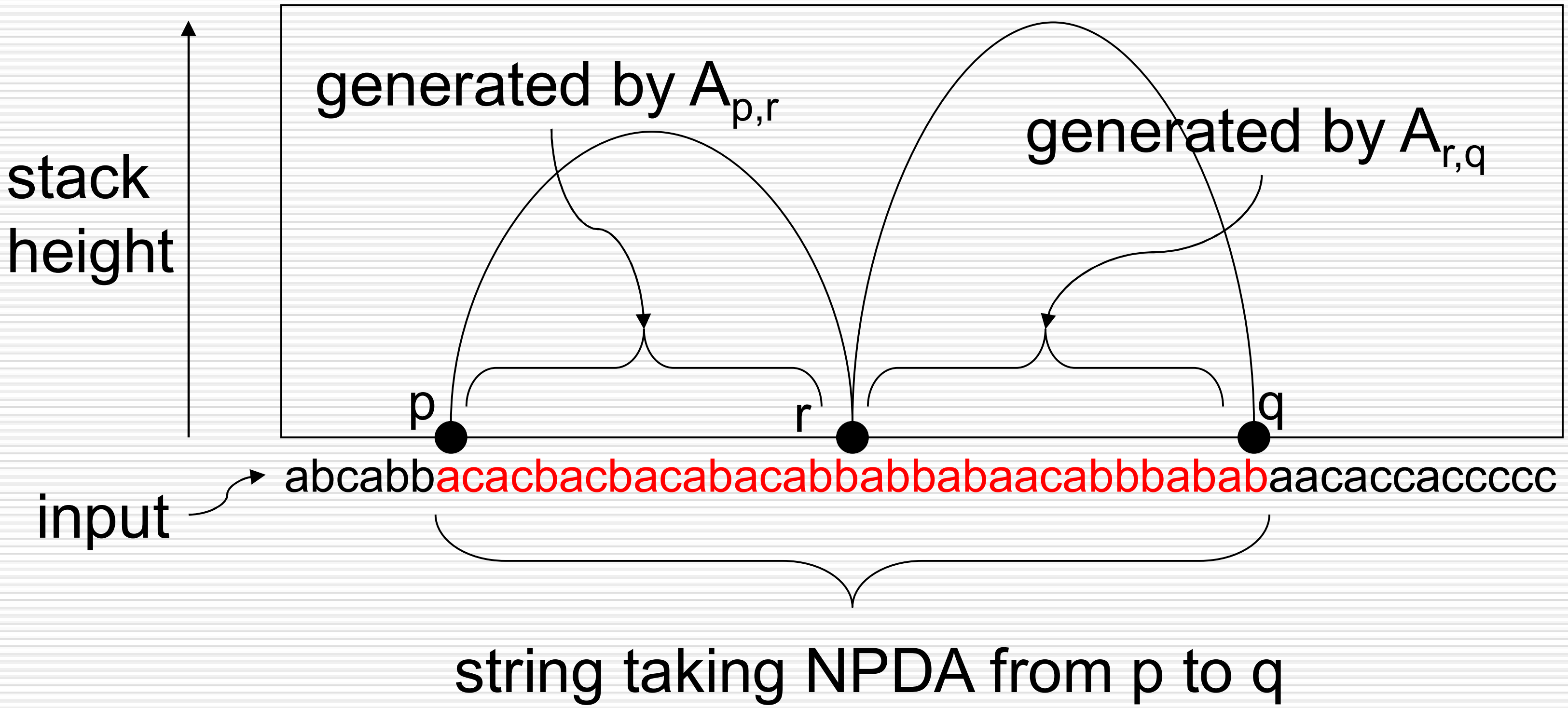
From NPDA to CFG

- **main idea**: non-terminal $A_{p,q}$ generates exactly the strings that take the NPDA from state p (with empty stack) to state q (with empty stack)

- then $A_{start, accept}$ generates all of the strings in the language recognized by the NPDA.

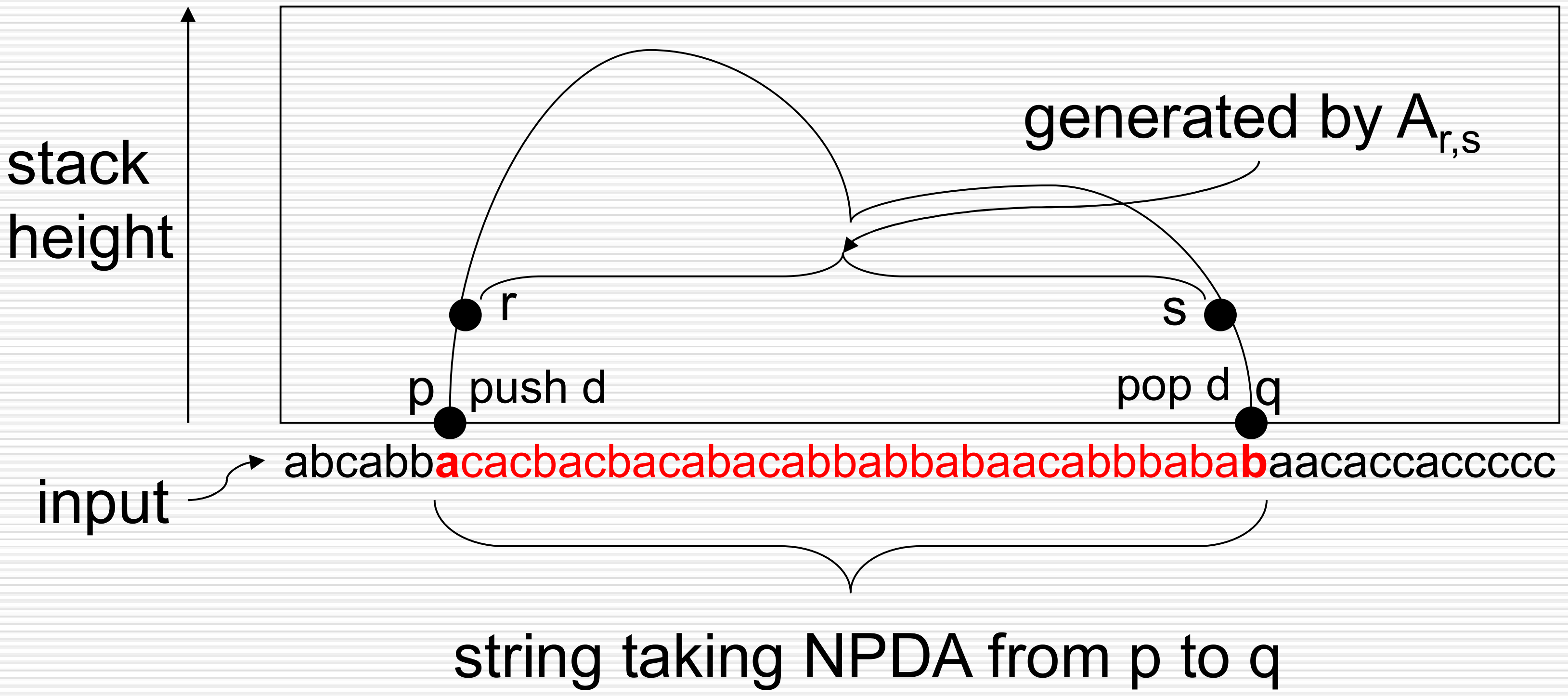
From NPDA to CFG

- To get from state p to q, case 1:



From NPDA to CFG

- To get from state p to q, case 2:



From NPDA to CFG

- NPDA $P = (Q, \Sigma, \Gamma, \delta, start, \{accept\})$
- CFG G:
  - non-terminals: $V = \{A_{p,q} \mid p, q \in Q\}$
  - start variable: $A_{start,\ accept}$
  - productions:

  Case 1

  for every p, r, q $\in$ Q, add the rule $A_{p,q} \rightarrow A_{p,r}A_{r,q}$;

  Case 2

  for every p, r, s, q $\in$ Q, d $\in \Gamma$, and a, b $\in (\Sigma \cup \{\varepsilon\})$
  if $(r, d) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, d)$, add the rule
  $A_{p,q} \rightarrow aA_{r,s}b$;

  for every p $\in$ Q, add the rule $A_{p,p} \rightarrow \varepsilon$

From NPDA to CFG

- ☐ Two claims to verify correctness:

1. if $A_{p,q}$ generates string x, then x can take NPDA P from state p (with empty stack) to q (with empty stack)
2. if x can take NPDA P from state p (with empty stack) to q (with empty stack), then $A_{p,q}$ generates string x

From NPDA to CFG

1. if $A_{p,q}$ generates string x, then x can take NPDA P from state p (with empty stack) to q (with empty stack)
   - induction on length of derivation of x.
   - Basis: one step derivation. must use rules that have only terminals on rhs. In G, must be productions of form $A_{p,p} \rightarrow \varepsilon$.
   - Induction: $A_{p,q} \Rightarrow^* x$.
     - ☐ verify case: $A_{p,q} \Rightarrow A_{p,r}A_{r,q} \Rightarrow^k x = yz$ ($A_{p,r} \Rightarrow^* y$, $A_{r,q} \Rightarrow^* z$)
     - ☐ verify case: $A_{p,q} \Rightarrow aA_{r,s}b \Rightarrow^k x = ayb$ ($A_{r,s} \Rightarrow^* y$)

From NPDA to CFG

2. if x can take NPDA P from state p (with empty stack) to q (with empty stack), then $A_{p,q}$ generates string x
   - induction on # of steps in P's computation
   - Basis: 0 steps. starts and ends at same state p. only has time to read empty string ε. G contains $A_{p,p} \rightarrow \varepsilon$.

From NPDA to CFG

2. if x can take NPDA P from state p (with empty stack) to q (with empty stack), then $A_{p,q}$ generates string x

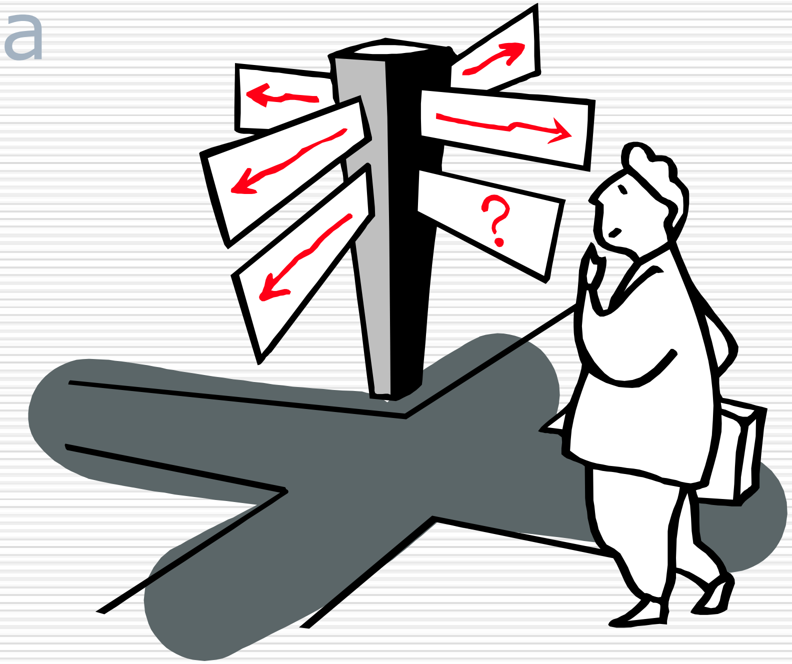
- Induction:
  - ☐ if stack becomes empty sometime in the middle of the computation (at state r)
    - y is read going from state p to r ($A_{p,r} \Rightarrow * y$)
    - z is read going from state r to q ($A_{r,q} \Rightarrow * z$)
    - conclude: $A_{p,q} \Rightarrow A_{p,r}A_{r,q} \Rightarrow^{*} yz = x$

From NPDA to CFG

2. if x can take NPDA P from state p (with empty stack) to q (with empty stack), then $A_{p,q}$ generates string x
   - ☐ if stack becomes empty only at beginning and end of computation.
     - first step: state p to r, read a, push d
     - go from state r to s, read string y ($A_{r,s} \Rightarrow * y$)
     - last step: state s to q, read b, pop d
     - conclude: $A_{p,q} \Rightarrow aA_{r,s}b \Rightarrow^* ayb = x$

2.2 Pushdown Automata

- ☐ Pushdown Automata
- ☐ CFG = PDA
- ☐ Deterministic PDA



Deterministic PDA

- Intuitively: never a choice of move.
  - δ(q, a, Z) is empty or a singleton for any q, a, Z (including a = ε).
  - If δ(q, ε , Z) is nonempty, then δ(q, a, Z) must be empty for all input symbols a.
- Parsers, as in YACC, are really DPDA's.
  - Thus, the question of what languages a DPDA can accept is really the question of what programming language syntax can be parsed conveniently.

Some Language Relationships

- If L is a regular language, then L is a DPDA language.
  - A DPDA can simulate a DFA, without using its stack (acceptance by final state).
- If L is a DPDA language, then L is a CFL that is not inherently ambiguous.
  - A DPDA yields an unambiguous grammar in the standard construction.