PDA DESIGN AND EQUIVALENCE

COMP 4200 – Formal Language

EXAMPLE

Design pushdown automata that recognizes the language $F = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i + j = k\}$

Logic: $a^i b^j c^{i+j}$

Read a push a

Read b push b

Read c pop a/b

Read a, push x

Read b, push x

Read c, pop x

$\delta(q_0, a, z_0) = \delta(q_1, xz_0)$

$\delta(q_1, a, x) = (q_1, xx)$ → loop for a

$\delta(q_1, b, x) = (q_2, xx)$

$\delta(q_2, b, x) = (q_2, xx)$ → loop for b

$\delta(q_2, c, x) = (q_3, \varepsilon)$

$\delta(q_3, c, x) = (q_3, \varepsilon)$ → loop for pop

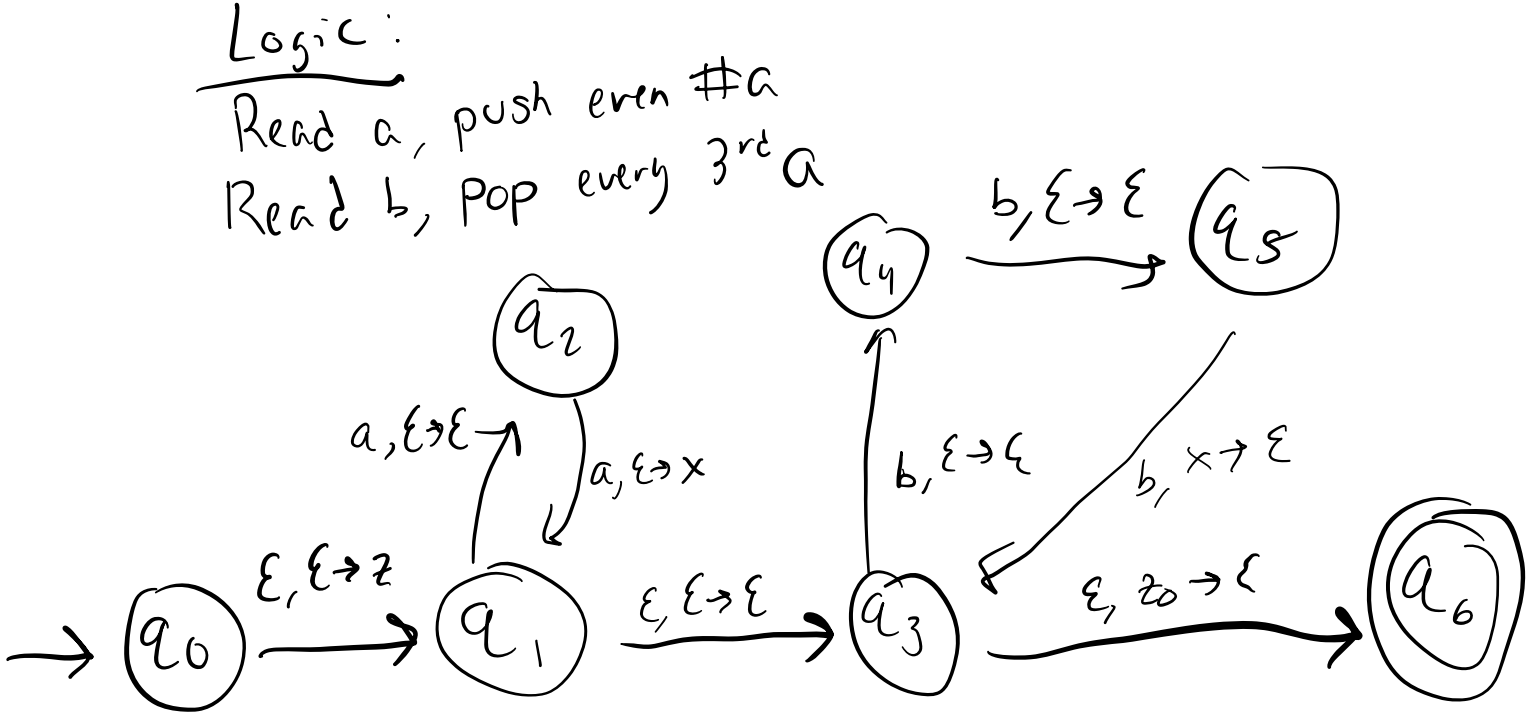
$\delta(q_3, \varepsilon, z_0) = (q_4, \varepsilon)$

EXAMPLE

a^2 b^3

a^4 b^6

Design pushdown automata that recognizes the language $G = \{a^{2n} b^{3n} \mid n \geq 0\}$.



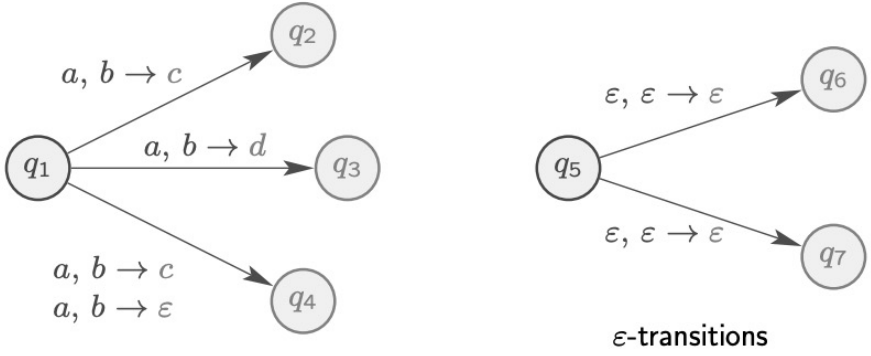
MORE EXAMPLES

- Design pushdown automata that recognizes the language G = $\{a^n b^{3n} \mid n \geq 0\}$.
- For the language L = $\{xcx^r / x \in \{a,b\}^*\}$ design a PDA (Push Down Automata) and trace it for string "abcba".

PDA MAY BE NONDETERMINISTIC

Can have multiple paths, & ε transitions

- PDA transition function allows for nondeterminism $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$



- Multiple choices when in state q_1, read $a \in \Sigma_\varepsilon$, and pop $b \in \Gamma_\varepsilon$;
- $\delta(q_1, a, b) = \{ (q_2, c), (q_3, d), (q_4, c), (q_4, \varepsilon) \}$

CFG TO PDA

equivalence of PDAs

CFG-PDA (End of exam stuff)

EQUIVALENCE WITH CONTEXT-FREE GRAMMARS

- Theorem 2.20
  - A language is context free iff(if and only if) some PDA recognizes it.

  - Showing this equivalence requires two steps.
    - Lemma 2.21 If A = L(G) for some CFG G, then A = L(M) for some PDA M.
    - Lemma 2.27 If A = L(M) for some PDA M, then A = L(G) for some CFG G.

We will only show how the first lemma works.

EQUIVALENCE WITH CFGS

If a language is context free, then some pushdown automaton recognizes it.

Let A be a CFL. P be a PDA. W be a string.

- The PDA P that we now describe will work by accepting its input w, if G generates that input, by determining whether there is a derivation for w.
- Each step yields intermediate string.
- We design P to determine whether some series of substitutions using the rules of G can lead from the start variable to w.
- Begins by writing the start variable on to the stack.
- Goes through the intermediate strings
- Eventually arrives at string with only terminal symbols
- P accepts if string identical to w.

HOW DOES PDA STORES INTERMEDIATE STRINGS?

- Is it stack?

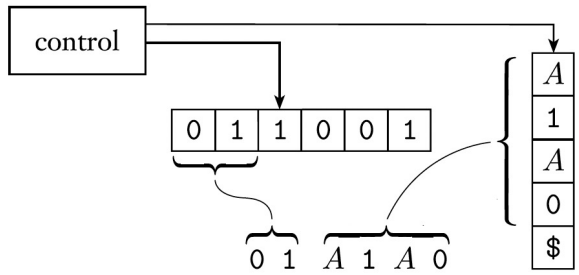
HOW DOES PDA STORES INTERMEDIATE STRINGS?

- Is it stack?
  - Not completely true!!
- PDA needs to find the variables in the intermediate string and make substitutions.
- The PDA can access only the top symbol, that may be terminal symbol instead of variable.
- Solution:
  - keep only part of the intermediate string on the stack: the symbols starting with the first variable in the intermediate string.
  - 01**A1A0** → intermediate string
  - Any terminal symbols appearing before the first variable are matched immediately

EQUIVALENCE WITH CFGS

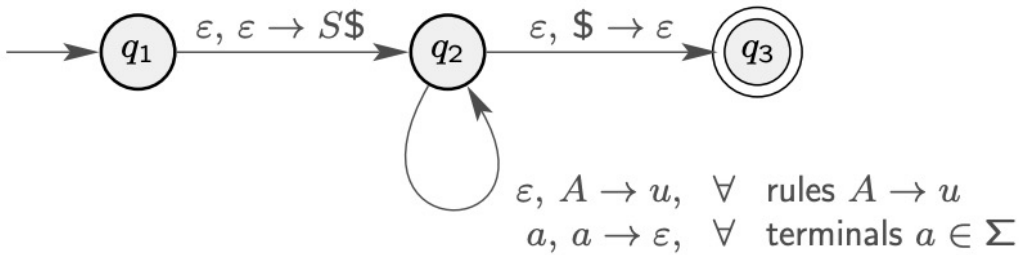
The following is an informal description of P.

- Place the marker symbol $ and the start variable on the stack.
- Repeat the following steps forever.
  - If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
  - If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
  - If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.



CONVERT CFG INTO PDA

- PDA works as follows:
  - Pushes $ and then S on the stack, where S is start variable.
  - Repeats following until stack empty
    - If top of stack is variable A ∈ V , then replace A by some u ∈ (Σ ∪ V)*, where A → u is a rule in R.
    - If top of stack is terminal a ∈ Σ and next input symbol is a, then read and pop a.
    - If top of stack is $, then pop it and accept.



Lemma 2.21:

If A = L(G) for some CFG G, then A = L(M) for some PDA M.

Proof Idea:

Given CFG G, convert it into PDA M with L(M) = L(G).

Basic idea: build PDA that simulates a leftmost derivation.

For example, consider CFG G = (V, Σ, R, S)

Variables V = {S, T}

Terminals Σ = {0, 1}

Rules: $S \rightarrow 0TS1 \mid 1T0, T \rightarrow 1$

Leftmost derivation of string 011101 ∈ L(G):

$$S \Rightarrow 0TS1 \Rightarrow 01S1 \Rightarrow 011T01 \Rightarrow 011101$$

EXAMPLE

$S \rightarrow 0TS1 \mid 1T0, T \rightarrow 1$

- PDA is non-deterministic.
- Input alphabet of PDA is the terminal alphabet of CFG
  - $\Sigma = \{0, 1\}$.
- Stack alphabet consists of all variables, terminals and "$"
  - $\Gamma = \{S, T, 0, 1, \$\}$.
- PDA simulates a leftmost derivation using CFG
  - Pushes RHS of rule in reverse order onto stack.

011101

$S \rightarrow 0TS1$

$\rightarrow 01S1$

$\rightarrow 011T01$

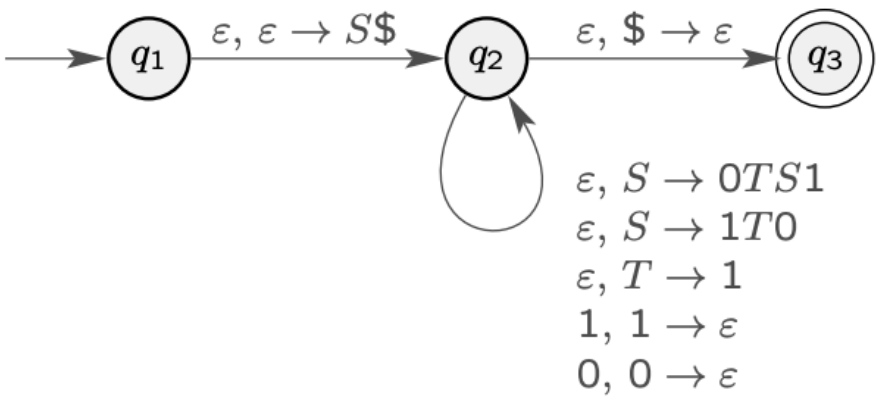
$\rightarrow 011101$

Recall leftmost derivation of string 011101 ∈ L(G):

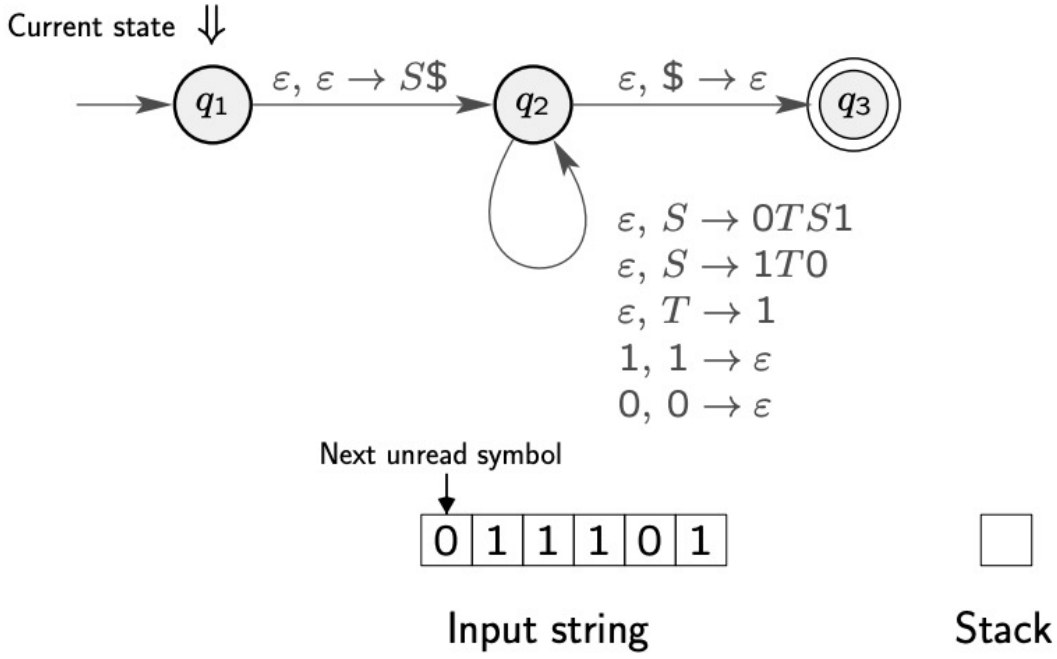
$$S \Rightarrow 0TS1 \Rightarrow 01S1 \Rightarrow 011T01 \Rightarrow 011101$$

Let's now process string 011101 on PDA.

When in state q_2, look at top of stack to determine next transition.



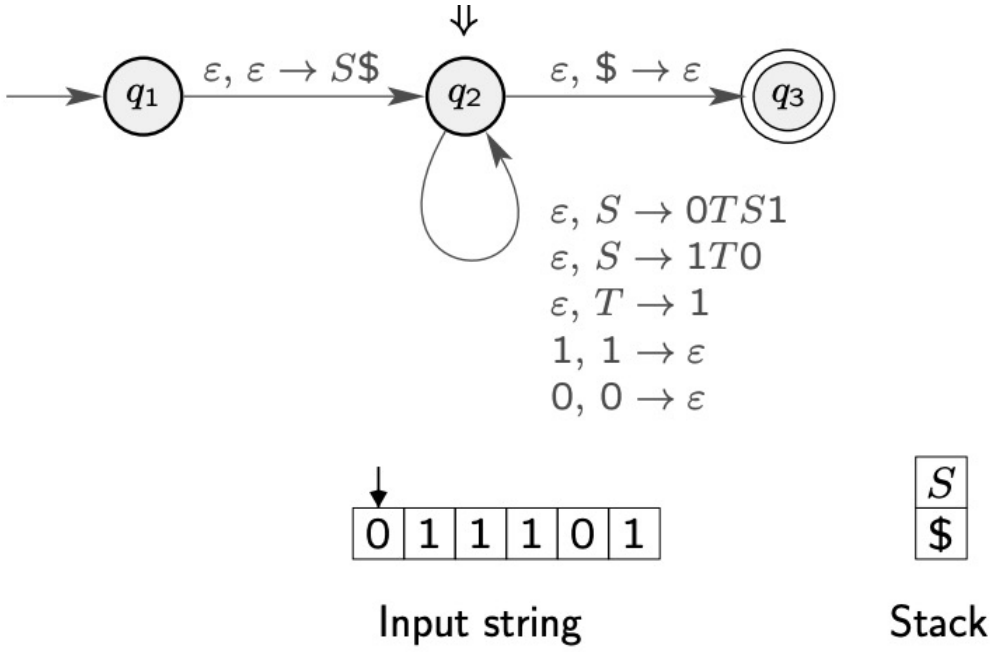
0. Start in state q_1 with 011101 on input tape and empty stack.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

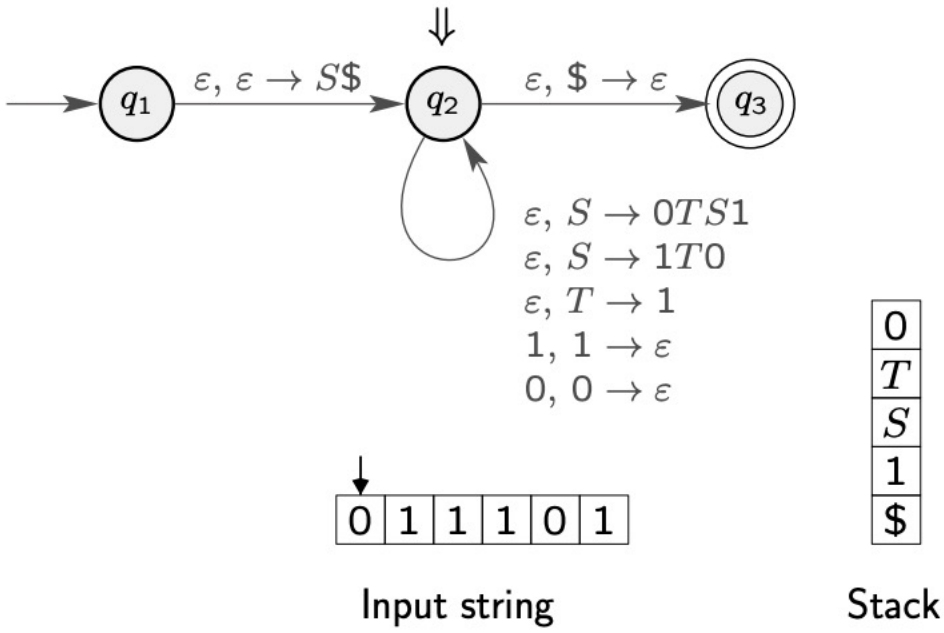
1. Read nothing, pop nothing, move to q_2, and push \$ and then S.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

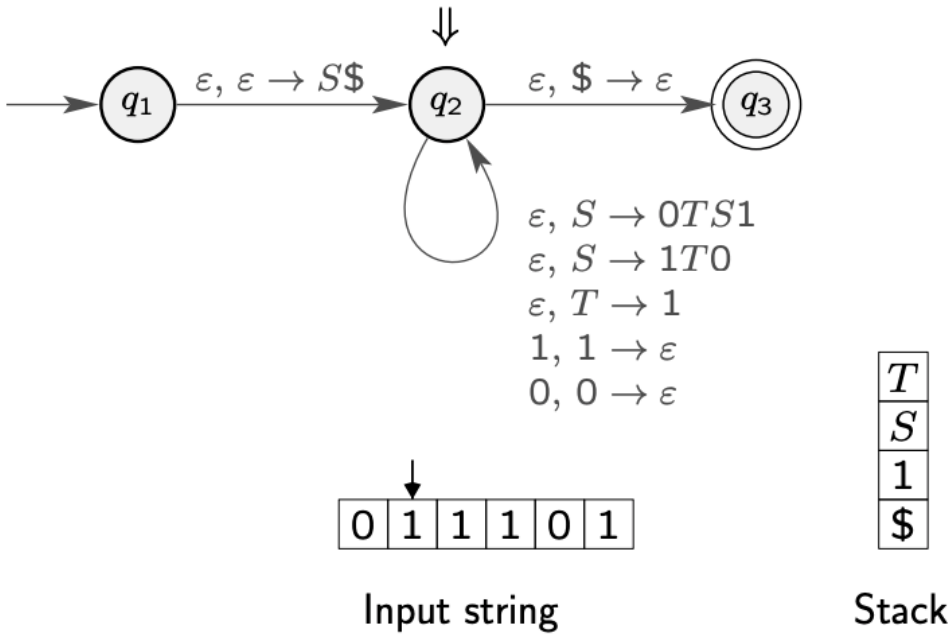
2. Read nothing, pop S, return to q_2, and push 0T S1.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

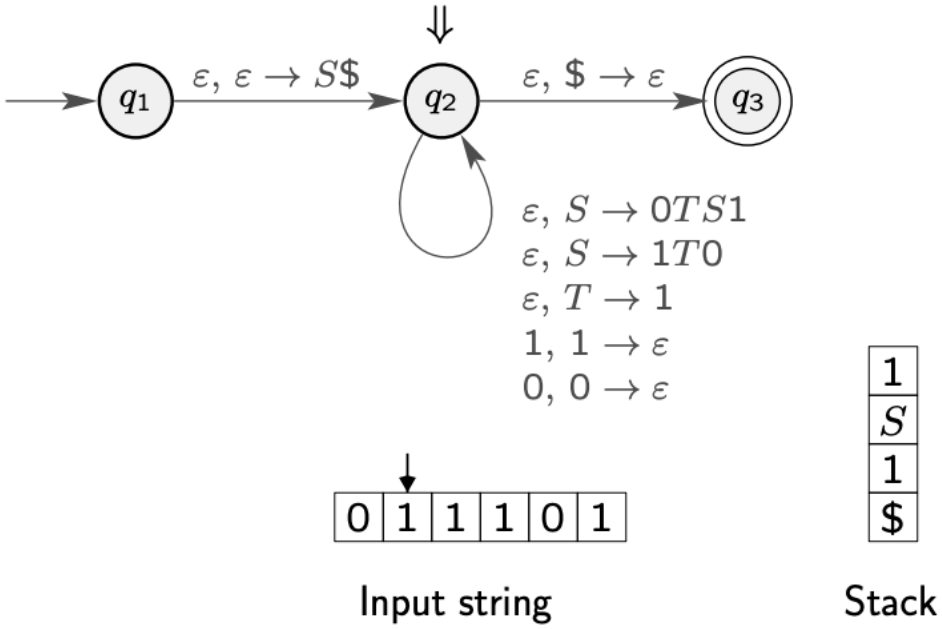
3. Read 0, pop 0, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

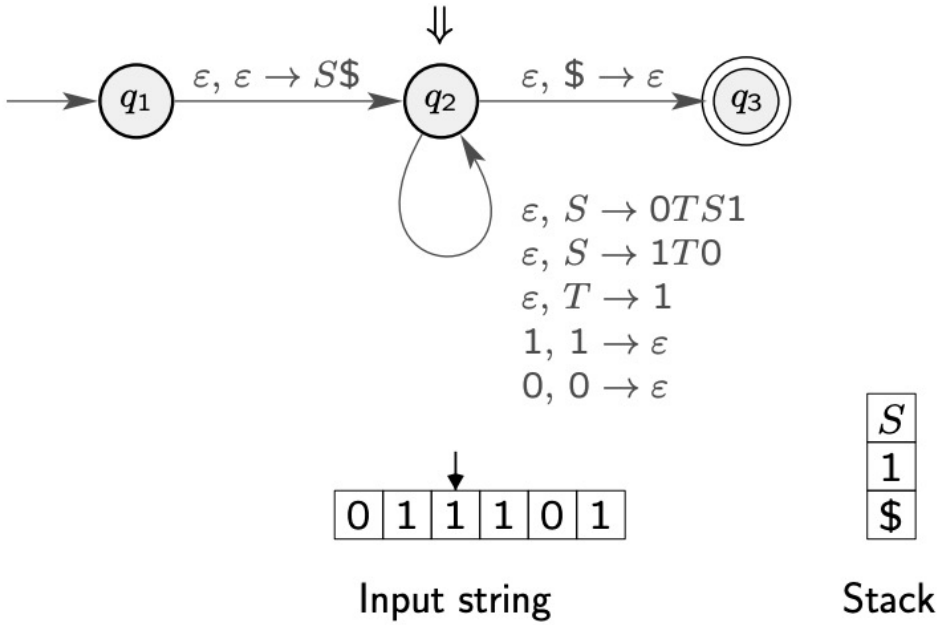
4. Read nothing, pop T, return to q_2, and push 1.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

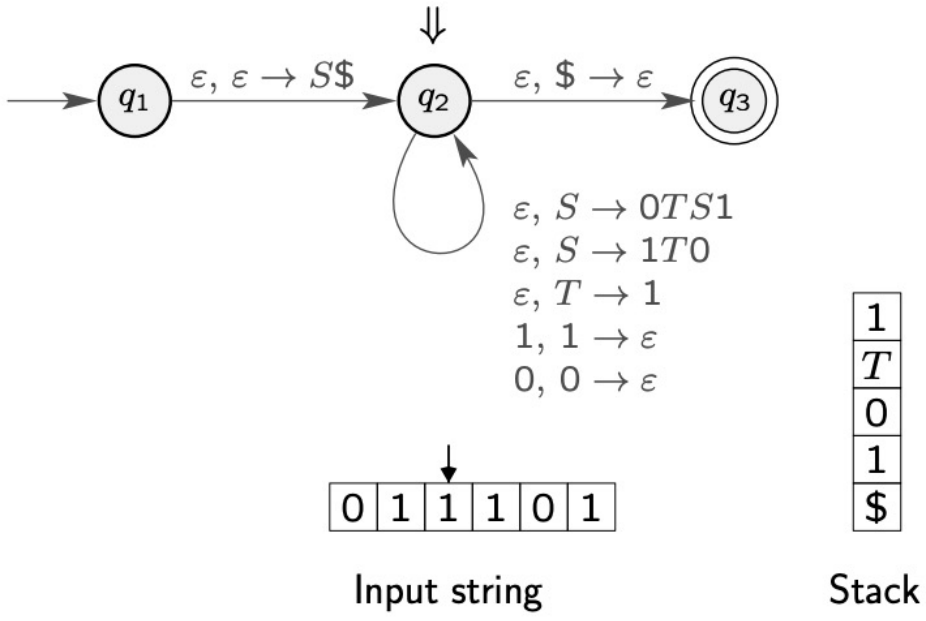
5. Read 1, pop 1, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

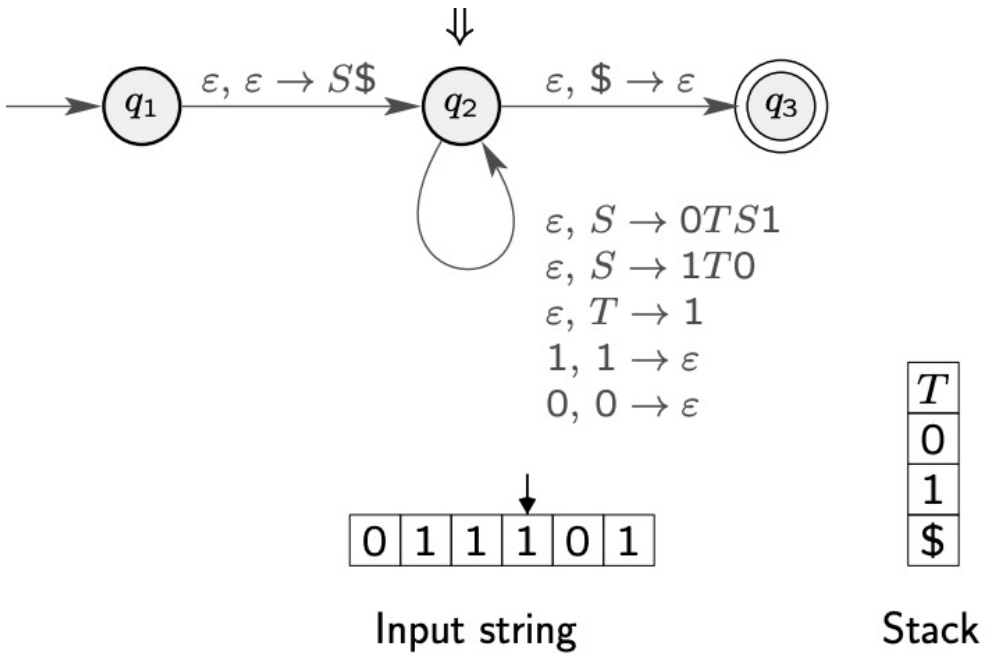
6. Read nothing, pop S, return to q_2, and push 1T0.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

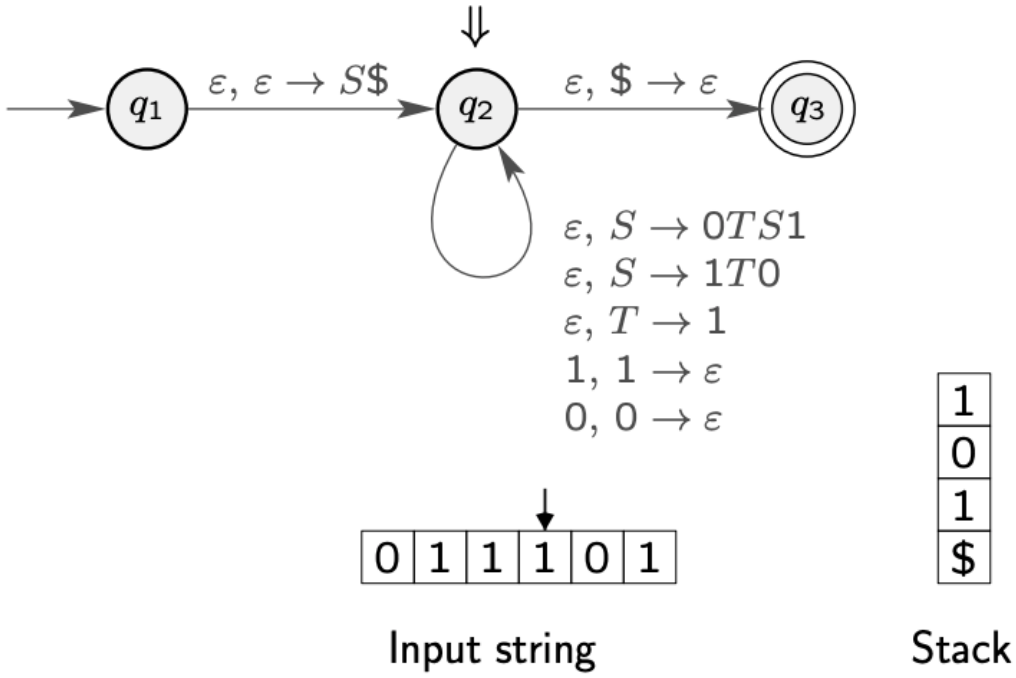
7. Read 1, pop 1, return to q_2, and push nothing



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

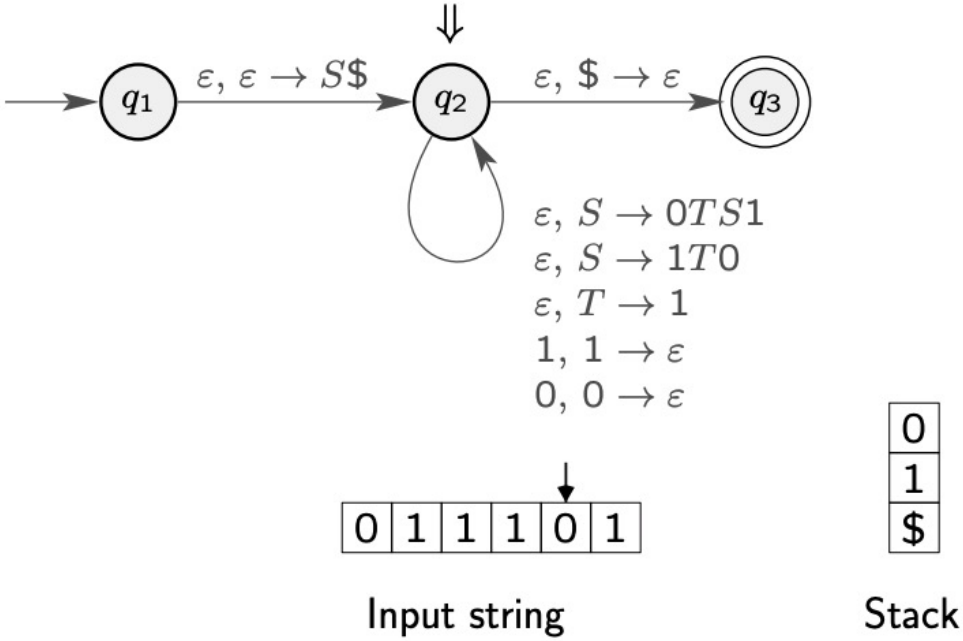
8. Read nothing, pop T, return to q_2, and push 1.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

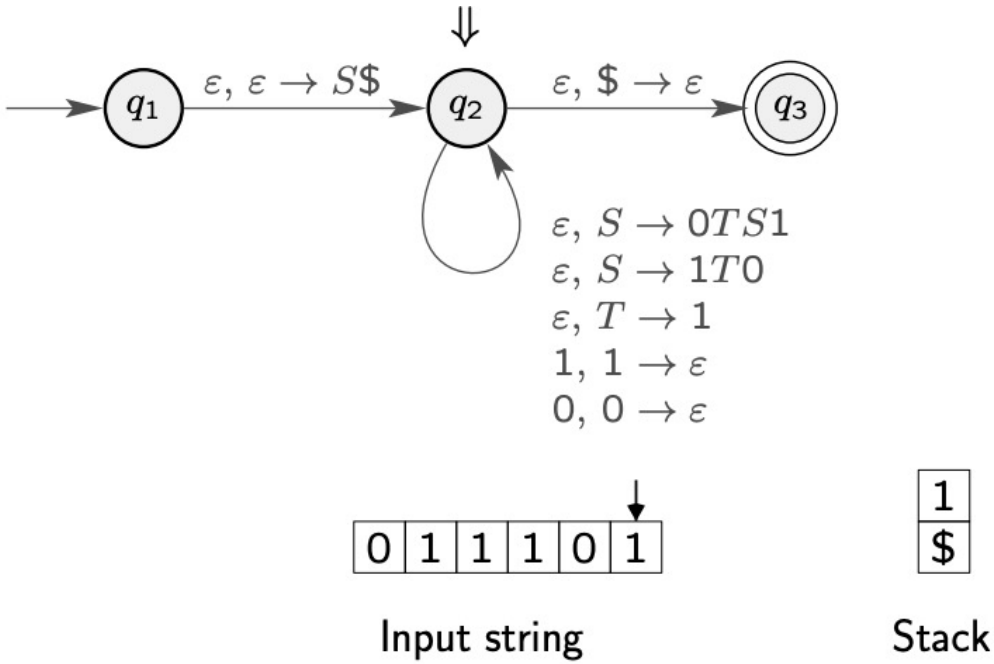
9. Read 1, pop 1, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

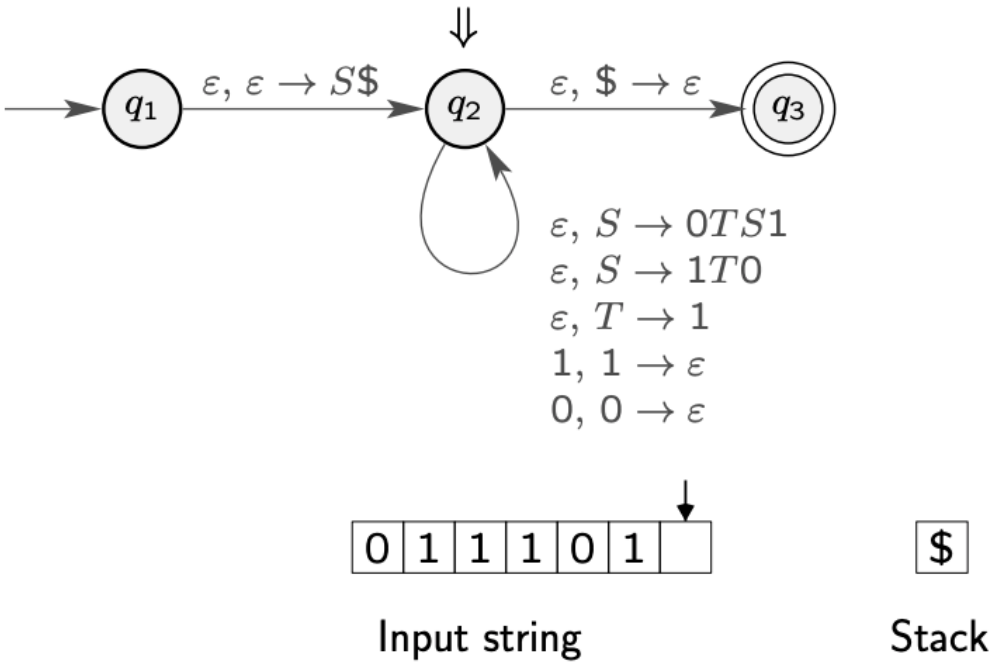
10. Read 0, pop 0, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

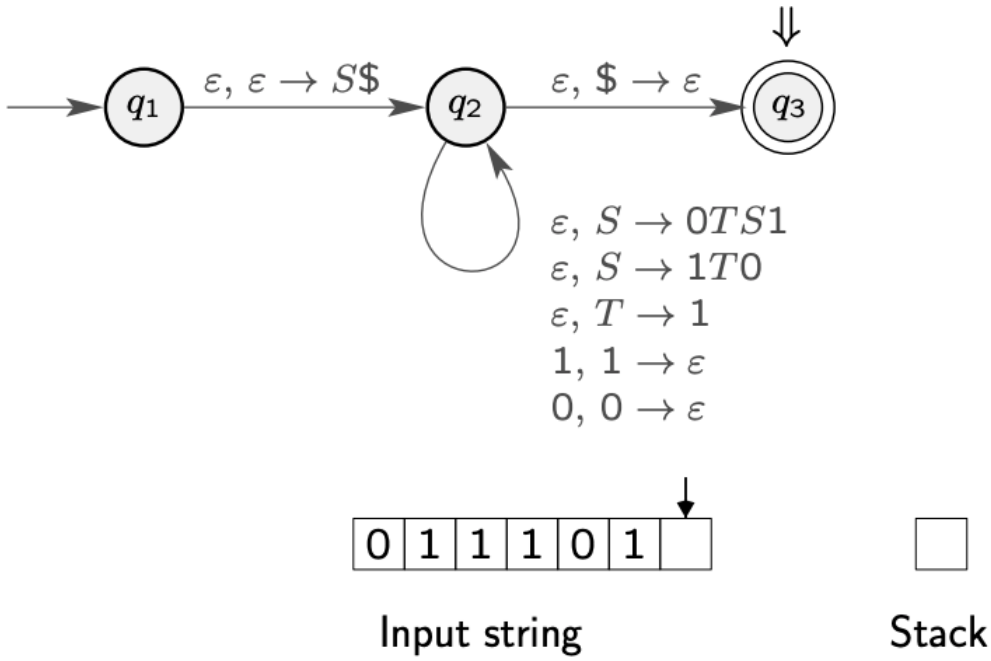
11. Read 1, pop 1, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

12. Read nothing, pop \$, move to q_3, push nothing, and accept.



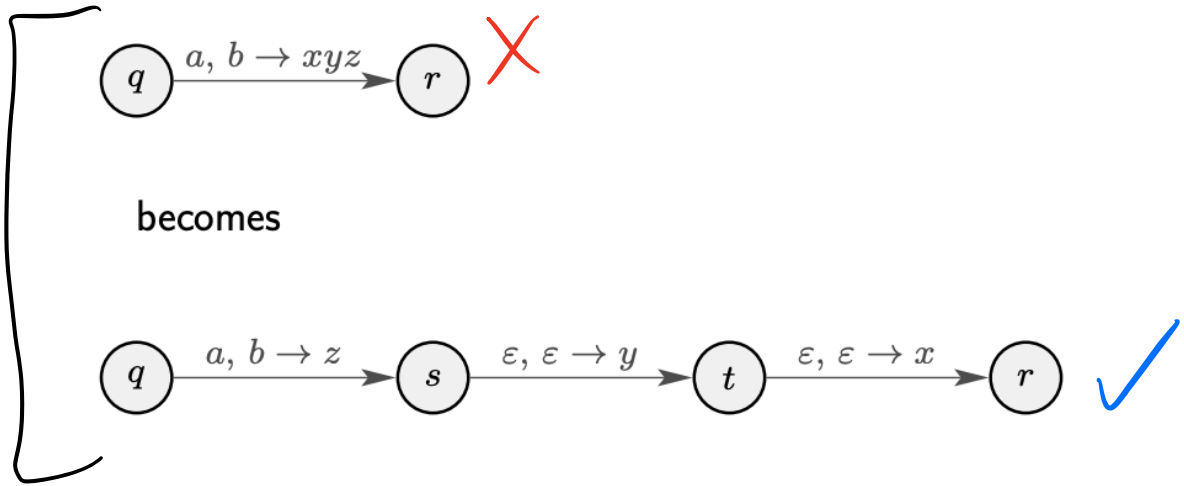
Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

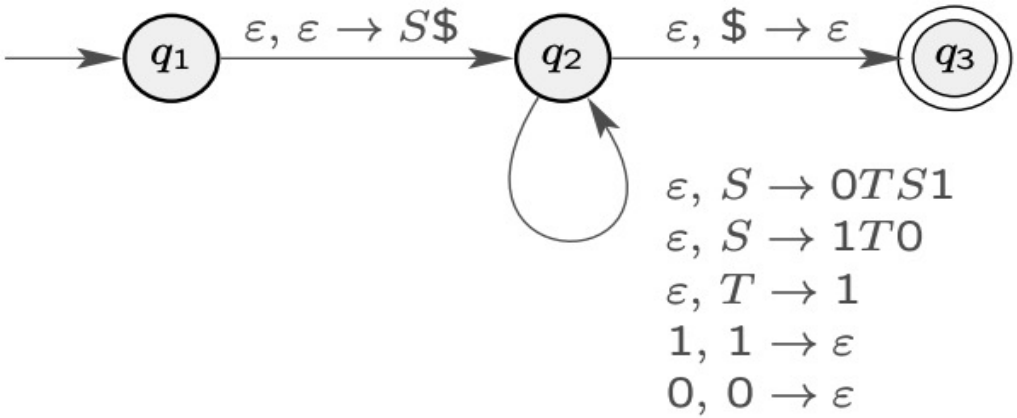
CONSTRUCTED PDA IS NOT COMPLIANT

- Problem: pushing strings onto stack instead of ≤ 1 symbols, which is not allowed in PDA specification.
- PDA transition function $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$
- Solution: Add extra states

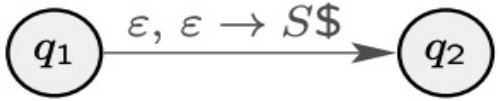
Can't push more than 1 element at once



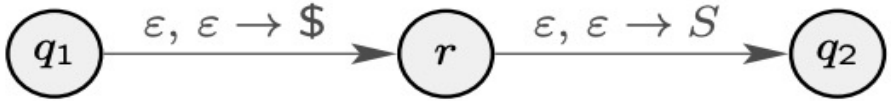
In our PDA,



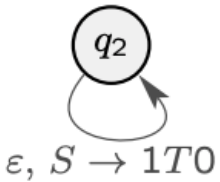
we replace



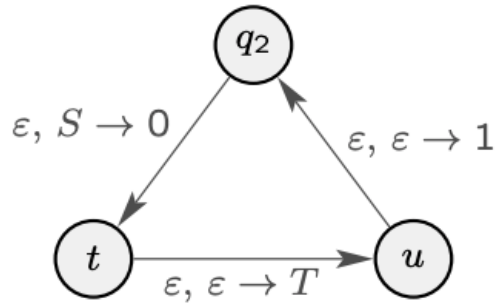
with



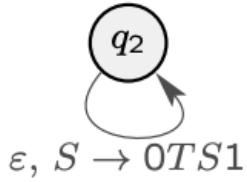
Also, replace



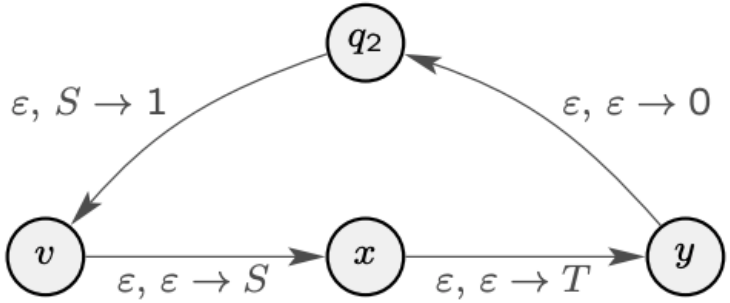
with



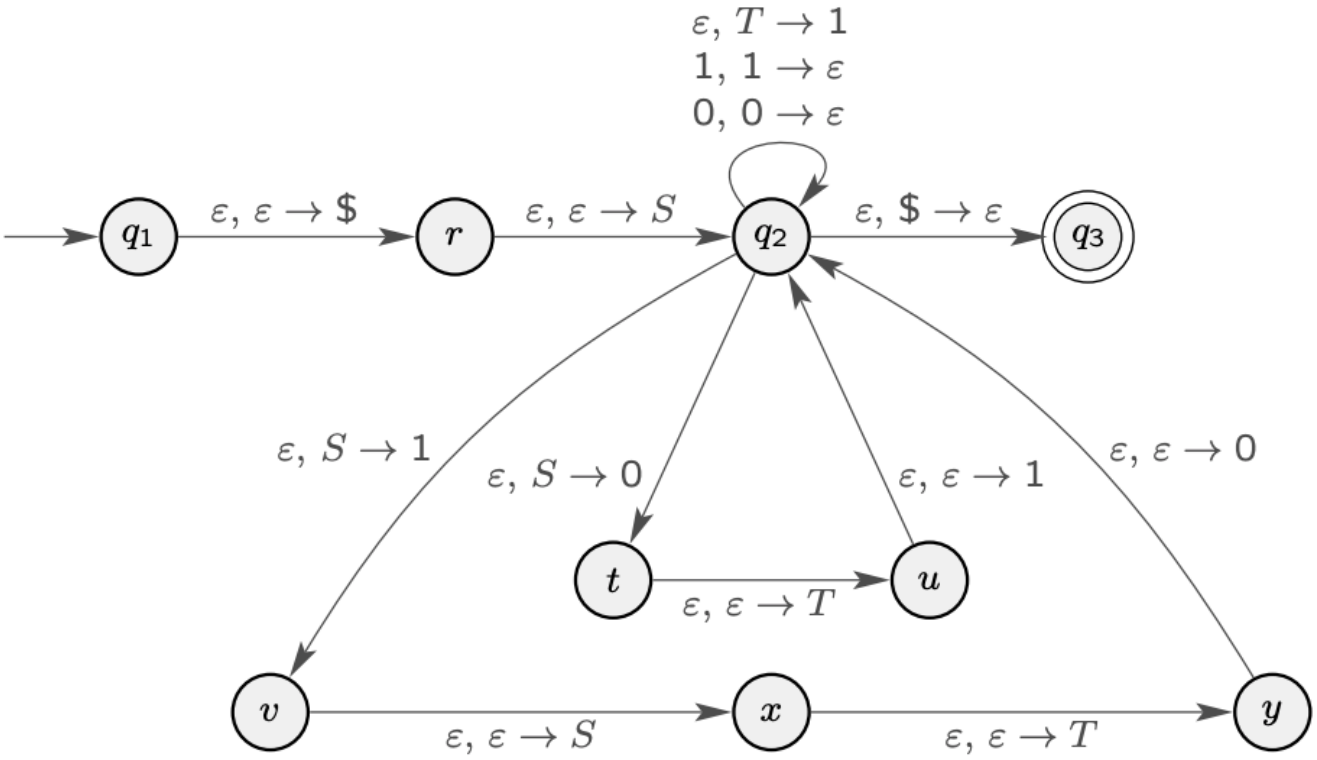
and replace



with



So our final PDA from the CFG is

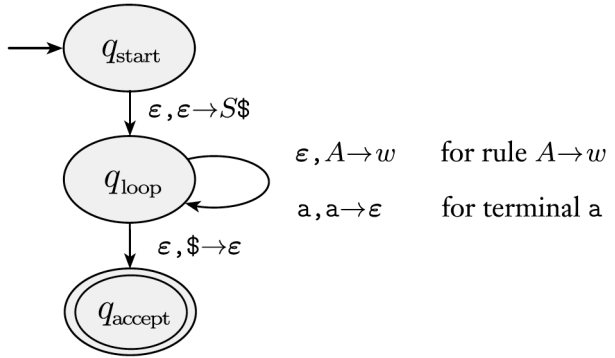


REVIEW

Recall, The following is an informal description of P.

- Place the marker symbol $ and the start variable on the stack.
- Repeat the following steps forever.
  - If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
  - If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
  - If the top of stack is the symbol $, enter the accept state. Doing so accepts the input if it has all been read.

The states of P are Q = $\{q_{start}, q_{loop}, q_{accept}\}$ U E, where E is the set of states we need for implementing the shorthand.



- The start state is q_{start}
- The only accept state is q_{accept}

The transition function is defined as:

- Initializing the stack to contain the symbols $ and S,

implementing step 1 in the informal description: $\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\ \$)\}$.

Then we put in transitions for the main loop of step 2.

- Case (a) wherein the top of the stack contains a variable:

  Let $\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) \mid \text{where } A \to w \text{ is a rule in R}\}$

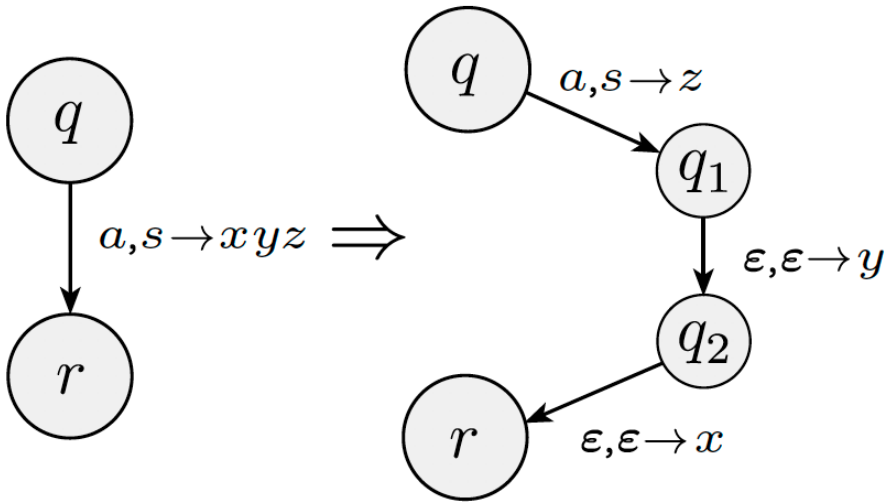
- Case (b) wherein the top of the stack contains a terminal.

  Let $\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\}$

- Case (c) wherein the empty stack marker $ is on the top of the stack.

  Let $\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)$

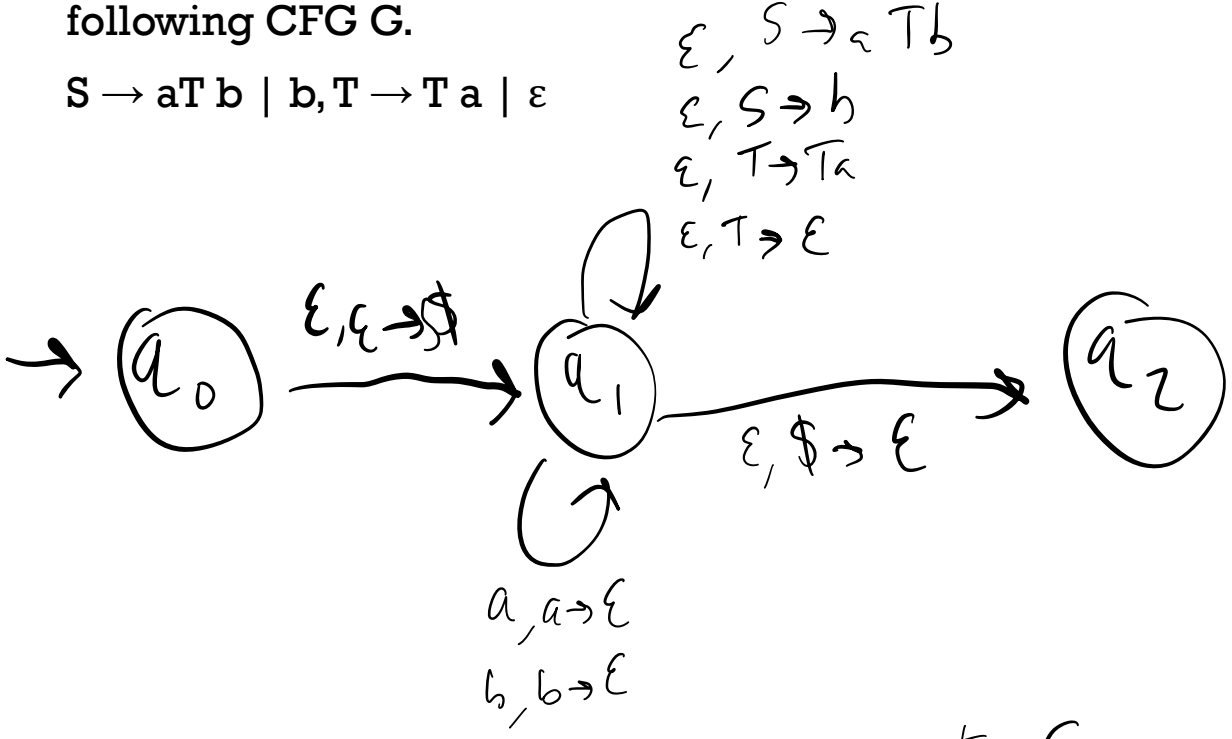
Implementing the shorthand (r, xyz) ∈ δ(q, a, s)



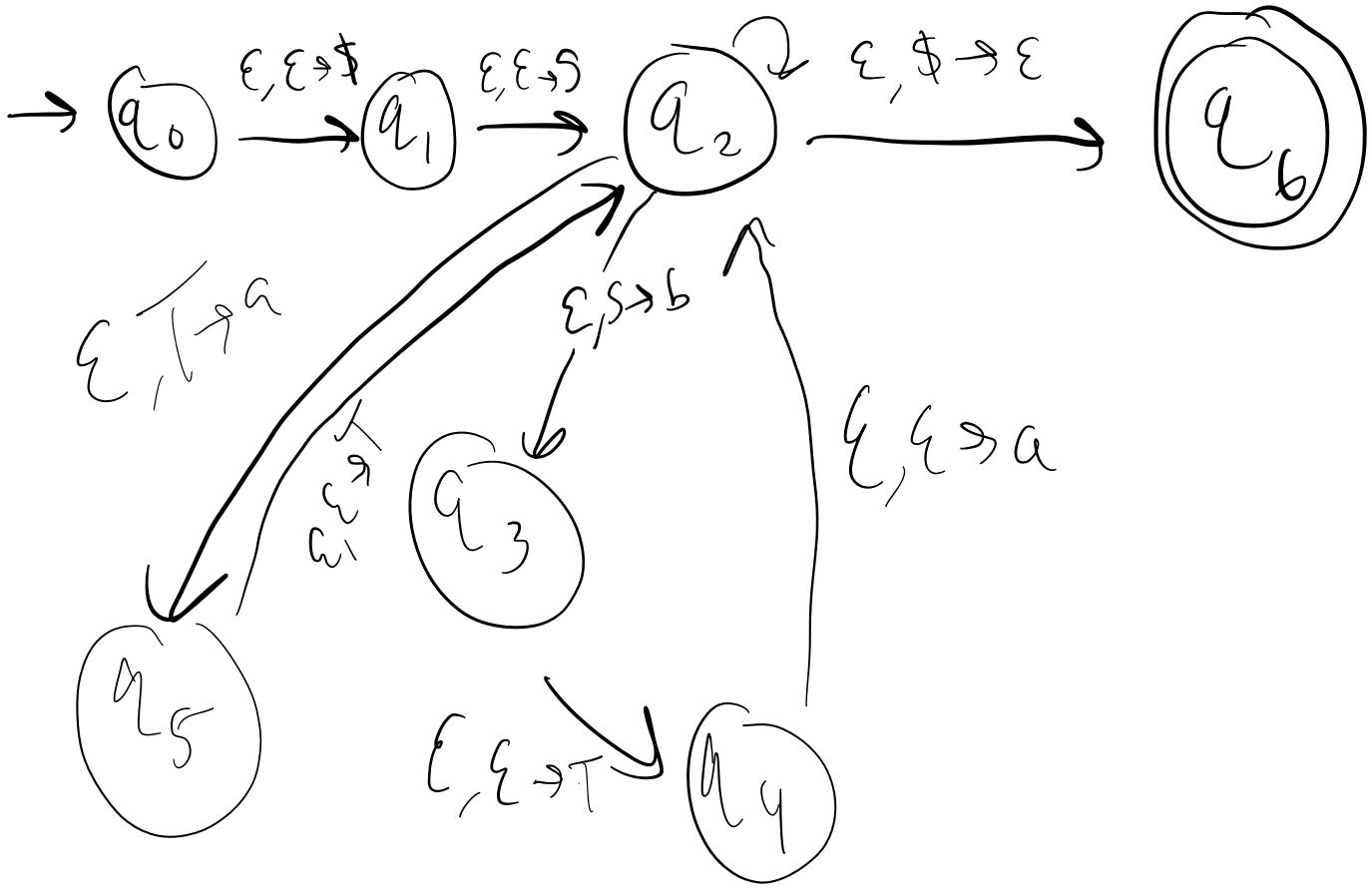
We use the notation (r, u) ∈ δ(q, a, s) to mean that when q is the state of the automaton, a is the next input symbol, and s is the symbol on the top of the stack, the PDA may read the a and pop the s, then push the string u onto the stack and go on to the state r.

We use the procedure developed in Lemma 2.21 to construct a PDA P_1 from the following CFG G.

$S \rightarrow aTb \mid b, T \rightarrow Ta \mid \varepsilon$



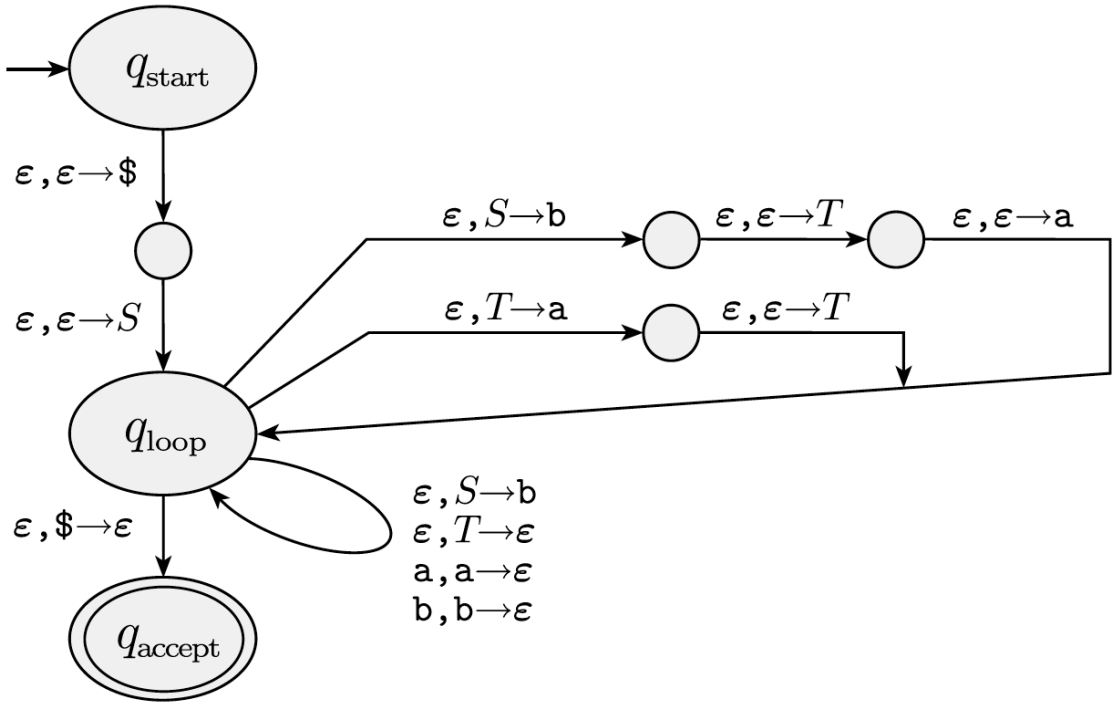
ε, T → ε
ε, S → b
a, a → ε
b, b → ε


q0
q1
q2
q3
q4
q5
q6
ε, ε → $
ε, ε → S
ε, $ → ε
ε, T → a
ε, S → b
ε, ε → T
ε, ε → a

We use the procedure developed in Lemma 2.21 to construct a PDA P_1 from the following CFG G.

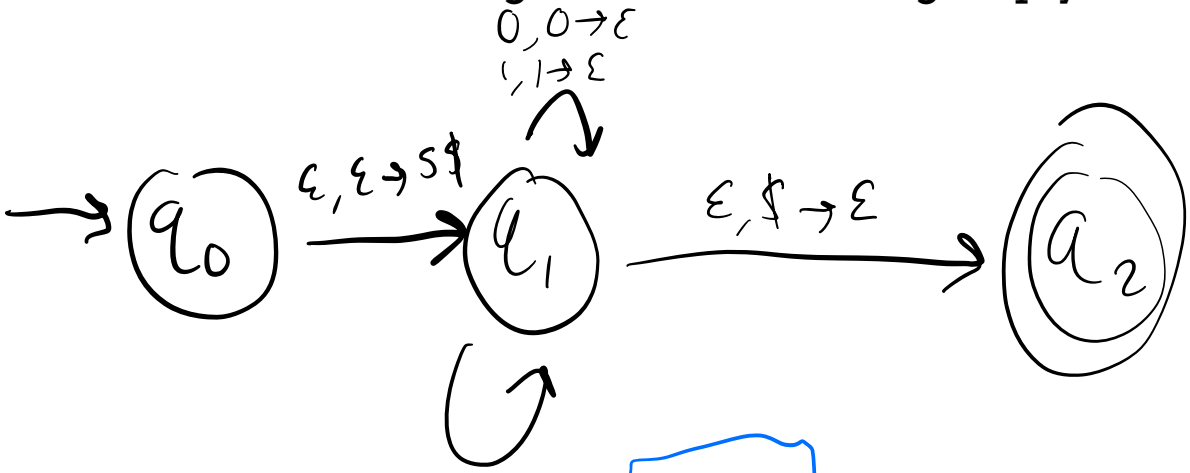
S → aT b | b, T → T a | ε

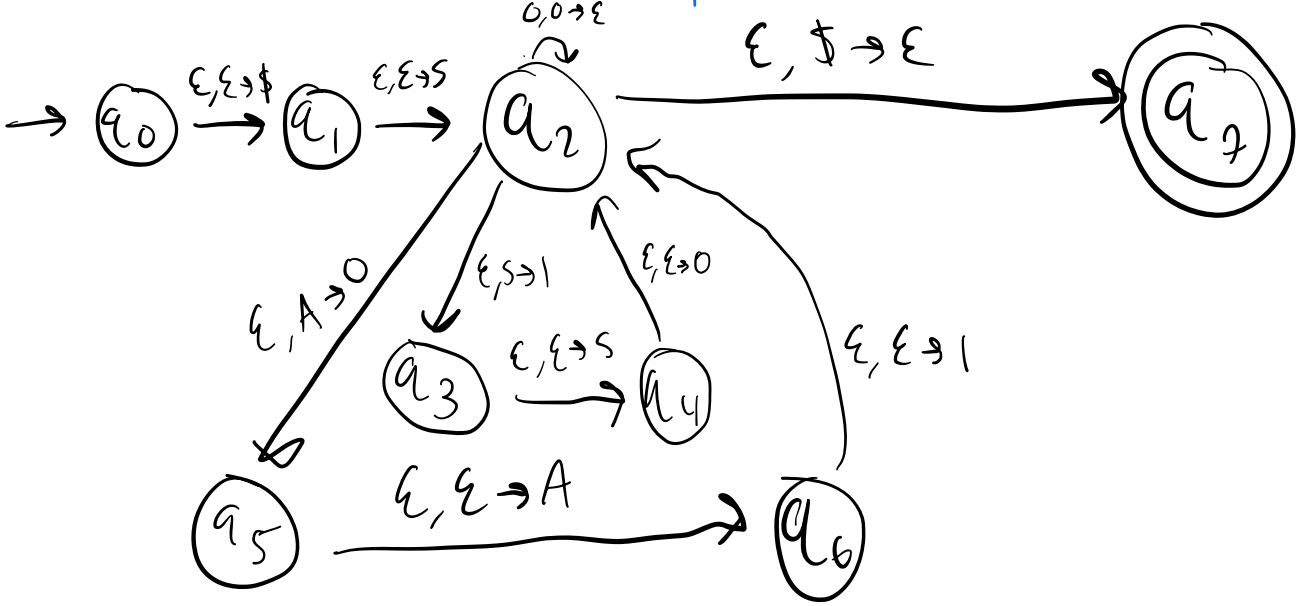
The transition function is shown in the following diagram.



EXAMPLE

Convert the following CFG into PDA using empty stack $S \rightarrow 0S1 \mid A \quad A \rightarrow 1A0 \mid S \mid \varepsilon$.

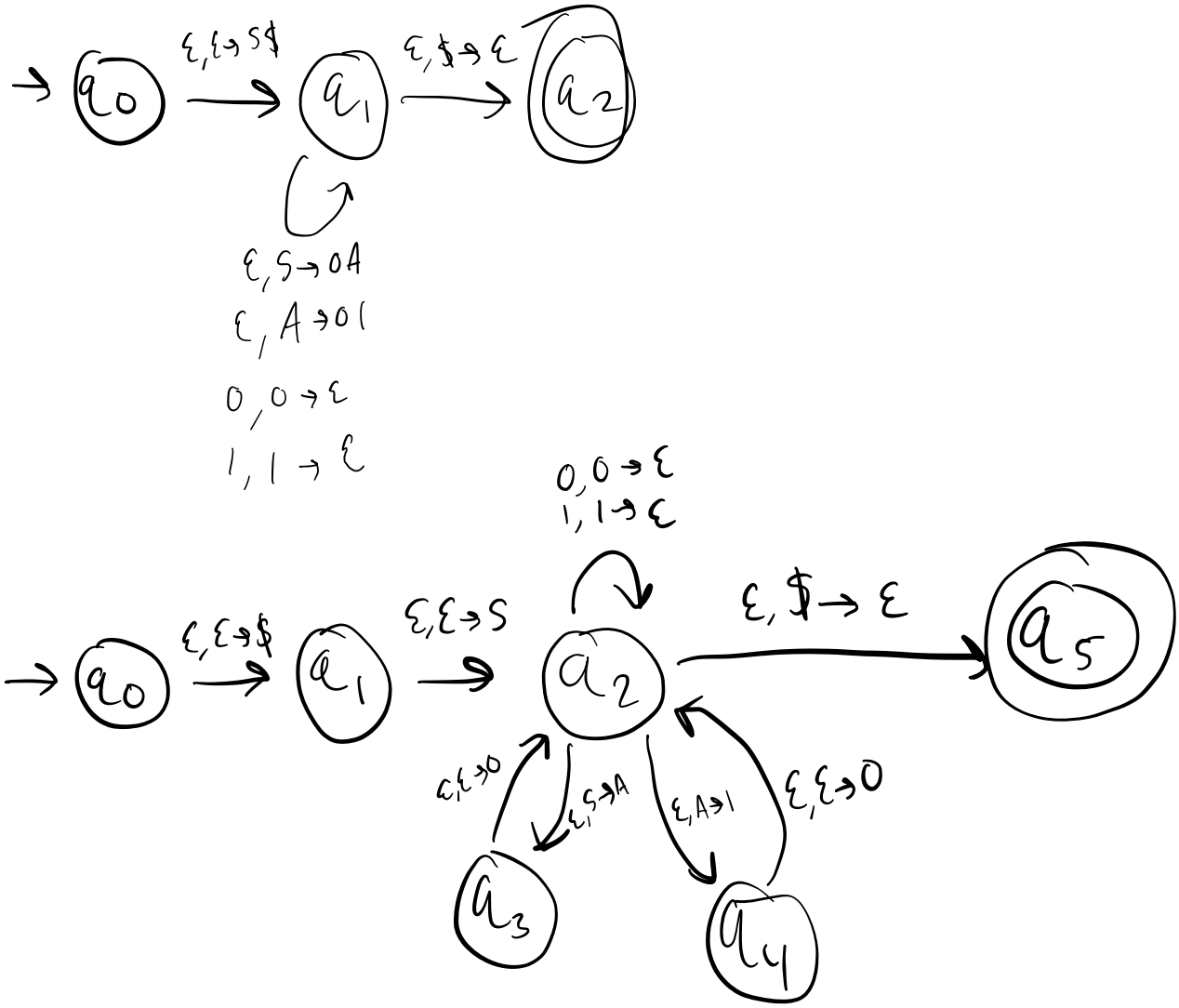



q0
q1
q2
q3
q4
q5
q6
q7
ε, ε → $
ε, ε → S
0, 0 → ε
ε, $ → ε
ε, A → 0
ε, S → 1
ε, ε → S
0 → ε
1 → ε
ε, ε → A

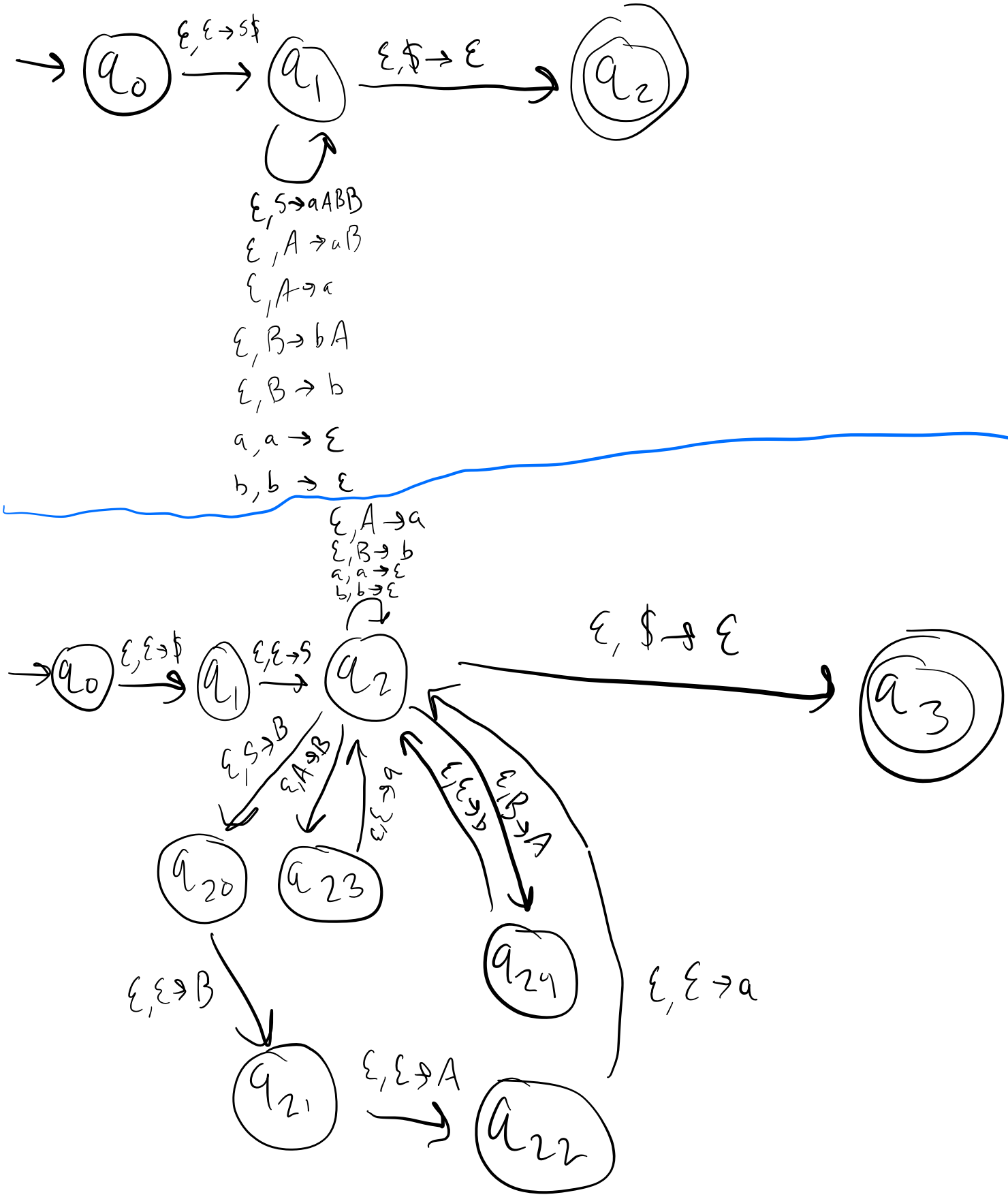
TRY YOURSELF!

1. Convert the following CFG into PDA S→0A A → 01
2. Convert the following CFG into PDA S →aABB A →aB/a B →bA/b
3. Convert the following CFG into PDA S → aAA A → aS|bS|a

1. Convert the following CFG into PDA S→0A A → 01



2. Convert the following CFG into PDA S →aABB A →aB/a B →bA/b



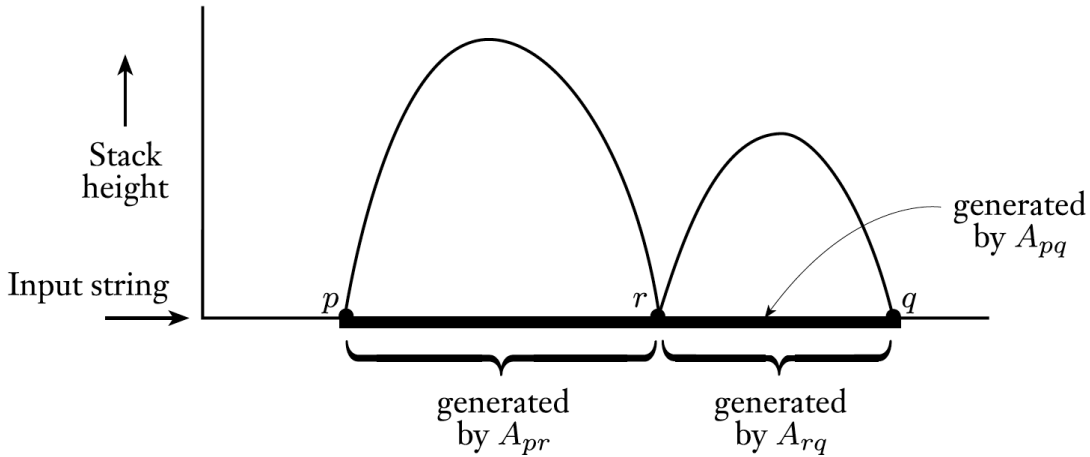
PDA TO CFG

IF A PUSHDOWN AUTOMATON RECOGNIZES SOME LANGUAGE, THEN IT IS CONTEXT FREE.

- For each pair of states p and q in P, the grammar will have a variable A_{pq}. This variable generates all the strings that can take P from p to q.
- Modifying P slightly to give it the following three features.
  - It has a single accept state, q_{accept}.
  - It empties its stack before accepting.
  - Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.
- Two possibilities occur during P's computation on x.
  - Either the symbol popped at the end is the symbol that was pushed at the beginning, or not. If so, the stack could be empty only at the beginning and end of P's computation on x.
  - If not, the initially pushed symbol must get popped at some point before the end of x and thus the stack becomes empty at this point.
- We simulate the with the rule $A_{pq} \rightarrow aA_{rs}b$, where a is the input read at the first move, b is the input read at the last move, r is the state following p, and s is the state preceding q.
- We simulate the latter possibility with the rule $A_{pq} \rightarrow A_{pr} A_{rq}$, where r is the state when the stack becomes empty.

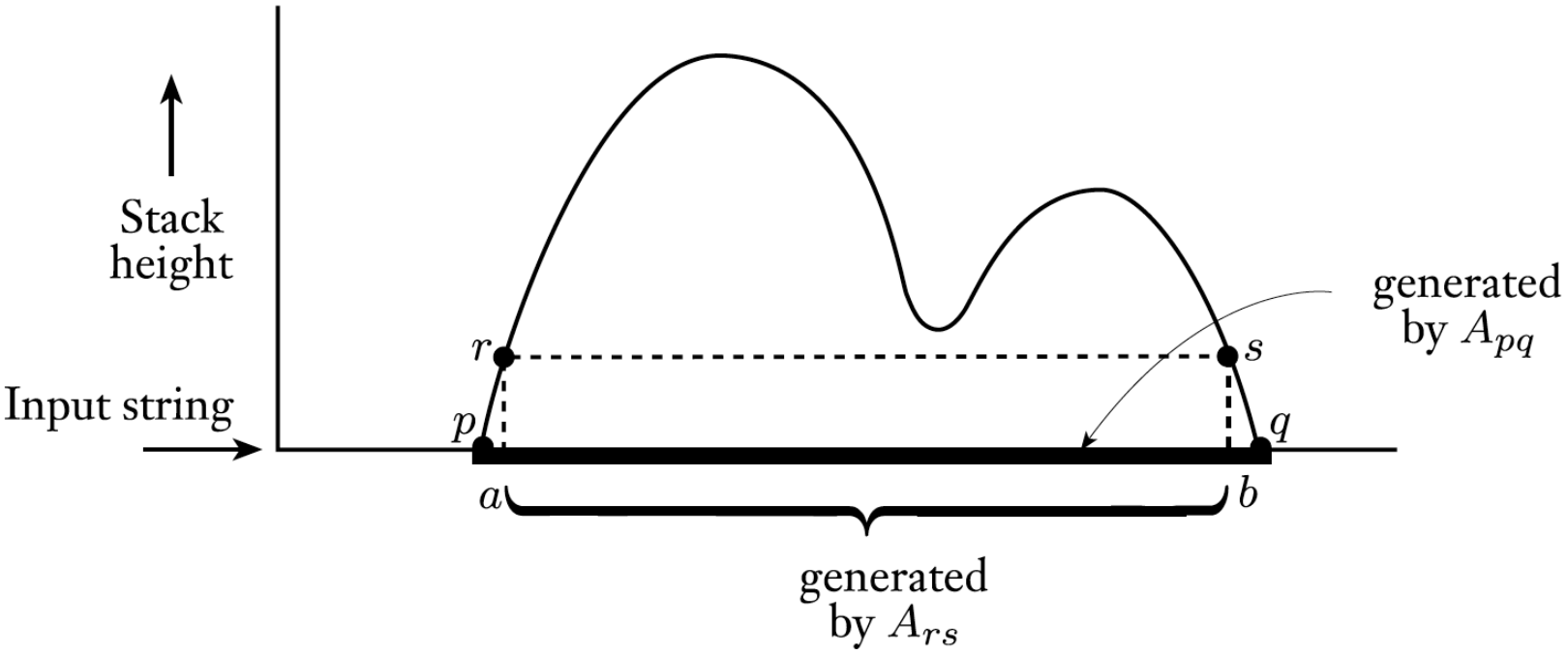
Proof:

- Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ and construct G.
- The variables of G are $\{A_{pq} \mid p, q \in Q\}$.
- The start variable is $A_{q0,q_{accept}}$.
- Now we describe G's rules in three parts.
  - For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \rightarrow aA_{rs}b$ in G.
  - For each $p, q, r \in Q$, put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G.
  - Finally, for each $p \in Q$, put the rule $A_{pp} \rightarrow \varepsilon$ in G.

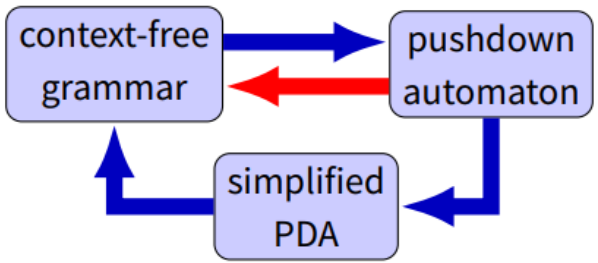


PDA computation corresponding to the rule $A_{pq} \rightarrow A_{pr}A_{rq}$

PDA computation corresponding to the rule $A_{pq} \rightarrow aA_{rs}b$



PDA TO CFG

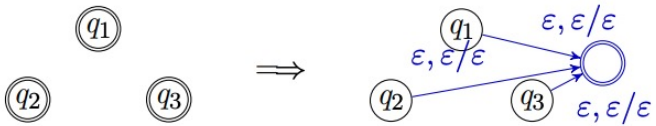


Simplified pushdown automaton:

- Has a single accepting state
- Empties its stack before accepting
- Each transition is either a push, or a pop, but not both

SIMPLIFYING

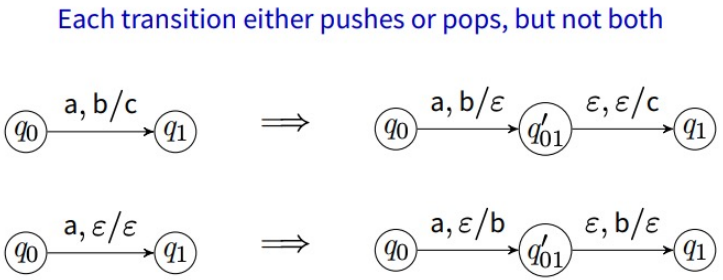
Single accepting state



Empties its stack before accepting

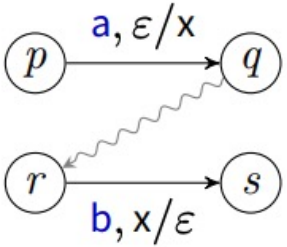
ε, a/ε for every stack symbol a

Each transition either pushes or pops, but not both

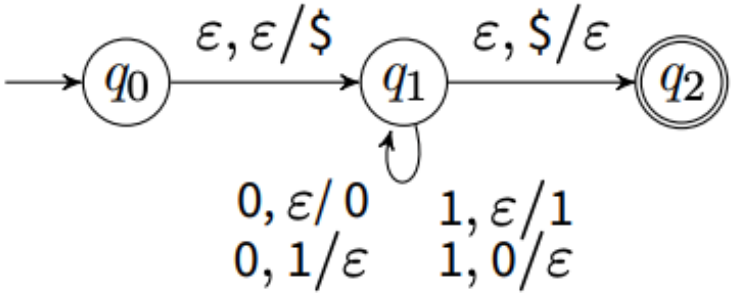


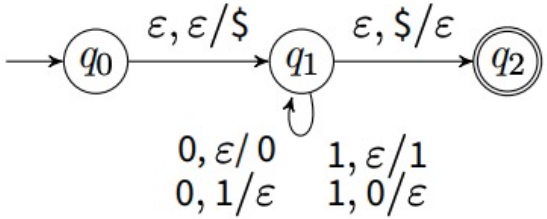
For every pair(q, r) of states in PDA, introduce variable A_{qr} in CFG

Intention: A_{qr} generates all strings that allow the PDA to go from q to r (with empty stack both at q and at r)

PDA	CFG
	$A_{qq} \to \varepsilon$
	$A_{pr} \to A_{pq}A_{qr}$
	$A_{ps} \to \mathrm{a}A_{qr}\mathrm{b}$ $\mathrm{a} = \varepsilon$ or $\mathrm{b} = \varepsilon$ allowed

EXAMPLE


$\varepsilon, \varepsilon/\$$
$\varepsilon, \$/\varepsilon$
q_0
q_1
q_2
$0, \varepsilon/0$
$1, \varepsilon/1$
$0, 1/\varepsilon$
$1, 0/\varepsilon$



variables: $A_{00}, A_{11}, A_{22}, A_{01}, A_{02}, A_{12}$

start variable: A_{02}

productions:

$A_{02} \to A_{01}A_{12}$

$A_{01} \to A_{01}A_{11}$

$A_{12} \to A_{11}A_{12}$

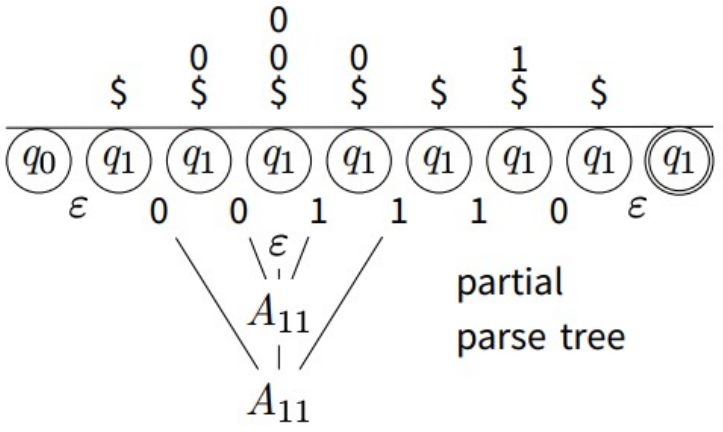
$A_{11} \to A_{11}A_{11}$

$A_{11} \to 0A_{11}1$

$A_{11} \to 1A_{11}0$

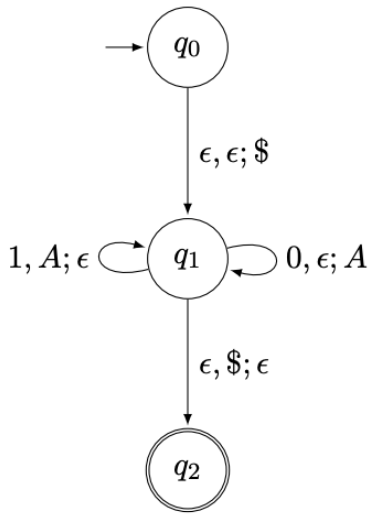
$A_{02} \to A_{11}$

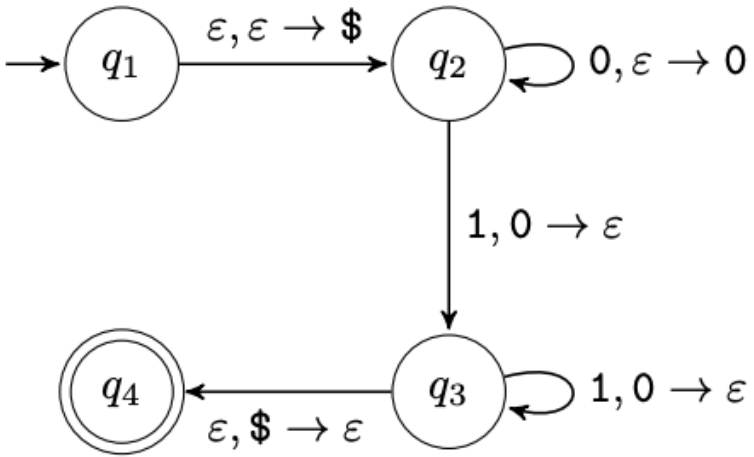
$A_{00} \to \varepsilon,\ A_{11} \to \varepsilon,$
$A_{22} \to \varepsilon$

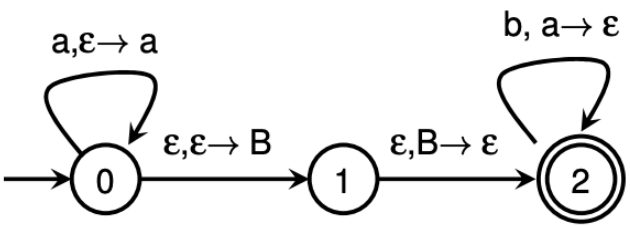


partial parse tree

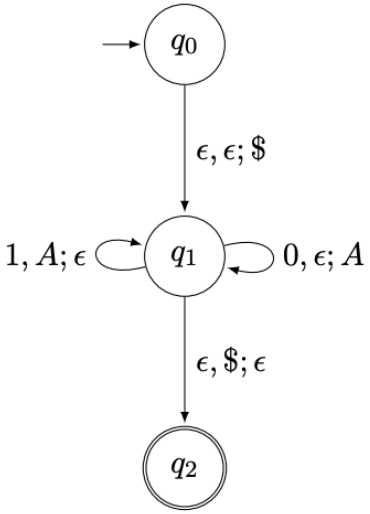
EXAMPLES TO TRY!







EXAMPLE


q_0
$\epsilon, \epsilon; \$$
$1, A; \epsilon$
q_1
$0, \epsilon; A$
$\epsilon, \$; \epsilon$
q_2

end up in q_2.

3. Introduce grammar rules corresponding to moves of the PDA of the form "push, compute, pop":

$$A_{02} \to A_{11}$$
$$A_{11} \to 0A_{11}1$$

4. Introduce grammar rules corresponding to moves of the PDA which drive it from states p to r and then from r to q:

$$A_{00} \to A_{00}A_{00} \mid A_{01}A_{10} \mid A_{02}A_{20}$$
$$A_{11} \to A_{10}A_{01} \mid A_{11}A_{11} \mid A_{12}A_{21}$$
$$A_{22} \to A_{20}A_{02} \mid A_{21}A_{12} \mid A_{22}A_{22}$$
$$A_{01} \to A_{00}A_{01} \mid A_{01}A_{11} \mid A_{02}A_{21}$$
$$A_{10} \to A_{10}A_{00} \mid A_{11}A_{10} \mid A_{12}A_{20}$$
$$A_{02} \to A_{00}A_{02} \mid A_{01}A_{12} \mid A_{02}A_{22}$$
$$A_{20} \to A_{20}A_{00} \mid A_{21}A_{10} \mid A_{22}A_{20}$$
$$A_{12} \to A_{10}A_{02} \mid A_{11}A_{12} \mid A_{12}A_{22}$$
$$A_{21} \to A_{20}A_{01} \mid A_{21}A_{11} \mid A_{22}A_{21}$$

5. Introduce grammar rules for ϵ:

$$A_{00} \to \epsilon$$
$$A_{11} \to \epsilon$$
$$A_{22} \to \epsilon$$

EXAMPLE