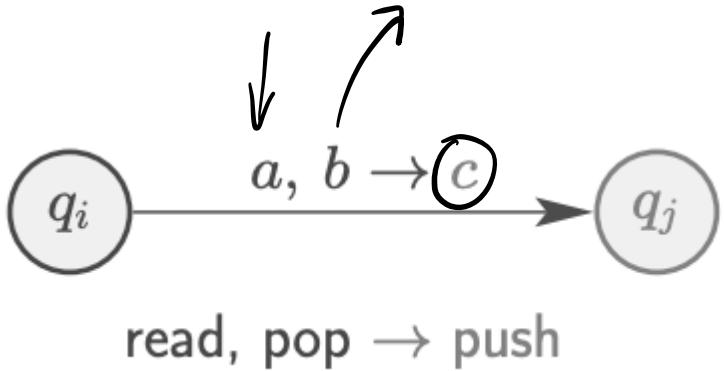
PDA DESIGN

COMP 4200 – Formal Language

Machine used to represent CFG

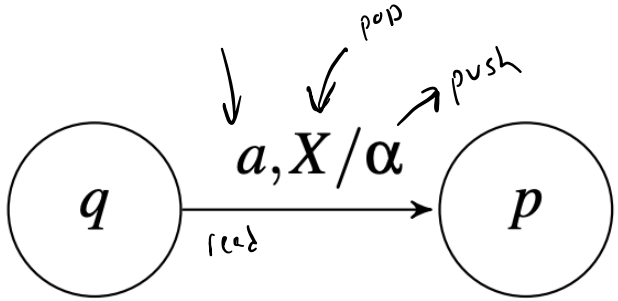
PDA TRANSITIONS

- If PDA
  - currently in state q_i,
  - reads $a \in \Sigma_\varepsilon$, and
  - pops $b \in \Gamma_\varepsilon$ off the stack,
- then PDA can
  - move to state q_j
  - push $c \in \Gamma_\varepsilon$ onto top of stack
- If $a = \varepsilon$, then no input symbol is read.
- If $b = \varepsilon$, then nothing is popped off stack.
- If $c = \varepsilon$, then nothing is pushed onto stack.



ANOTHER REPRESENTATION

- The nodes correspond to the states of the PDA.
- An arrow labelled start indicates the start state.
- Doubly circled states are accepting.
- If $\delta(q,a,X)$ contains a pair (p,α), then there is an arc from q to p labeled $a,X/\alpha$



- The only thing that the diagram does not tell us is which stack symbol is the start symbol. Conventionally, it is Z.

EXAMPLE

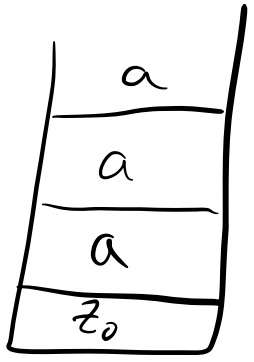
The diagramm to our PDA for $\{a^n b^n \mid n \geq 1\}$

Logic:

Read a push a on stack

Read b pop a's out of stack

Ex. String: aaabbb



$\rightarrow \delta(q_0, a, Z_0) = \delta(q_1, a\ Z_0)$ | 1st a read

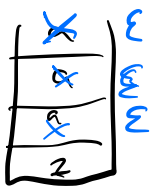
instaneous transition

$\rightarrow \delta(q_1, a, a) = \delta(q_1, aa)$ 2nd a read

$\rightarrow \delta(q_1, a, a) = \delta(q_1, aa)$ 3rd a read

$\delta(q_1, b, a) = \delta(q_2, \varepsilon)$ since popping no need to put a second one

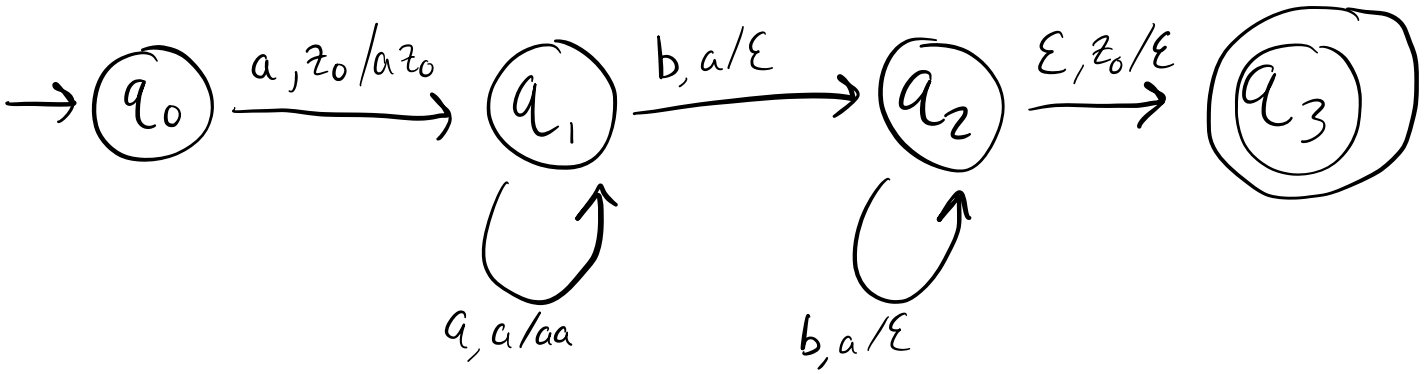
pop a (1st b)



$\delta(q_2, b, a) = \delta(q_2, \varepsilon)$ pop a (2nd b)

$\delta(q_2, b, a) = \delta(q_2, \varepsilon)$ pop a (3rd b)

$\delta(q_2, \varepsilon, z_0) = \delta(q_3, \varepsilon)$ popping the z_0



EXAMPLE

The diagramm to our PDA for $\{a^n b^n \mid n \geq 1\}$

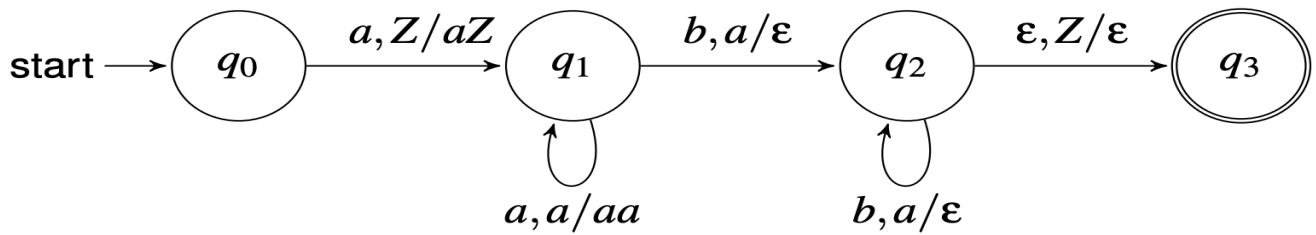
$\delta(q_0, a, Z) = \{(q_1, aZ)\}$ ··· push first a

$\delta(q_1, a, a) = \{(q_1, aa)\}$ ··· push further a's

$\delta(q_1, b, a) = \{(q_2, \varepsilon)\}$ ··· start popping a's

$\delta(q_2, b, a) = \{(q_2, \varepsilon)\}$ ··· pop further a's

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



EXAMPLE

- Transition Function:

1. $q0$ (initial state): Read a, push A onto the stack, move to state $q1$
2. $q1$: Read a, push A onto the stack, remain in state $q1$
3. $q1$: Read b, move to state $q2$ without pushing or popping anything from the stack
4. $q2$: Read b, move to state $q3$ without pushing or popping anything from the stack
5. $q3$: Read b, pop A from the stack, move back to state $q1$
6. $q3$: Read a, remain in state $q3$ if there are no more input symbols left
7. $q3$: If ϵ (empty stack) and no input symbols left, move to state $q4$
8. $q4$: If ϵ (empty stack) and no input symbols left, move to state $q5$ (final accepting state)

TRY YOURSELF!

- Build a PDA for the language L = $\{a^n b^{2n} \mid n \geq 1\}$
- Construct a PDA to accept L = (a, b)* with equal number of 'a' and 'b', i.e., $n_a(L) = n_b(L)$
- Build a PDA for the language L = $\{0^n 1^m 0^n \mid m,n \geq 1\}$
- Construct the PDA for the language L=$\{a^n b^m a^n / n,m >= 1\}$
- Construct a PDA to accept the language L = $\{a^n b^m c^m d^n$, where m, n ≥ 1$\}$.

EXAMPLE

Build a PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$

Logic:

1. Read a, push onto stack
2. Read odd b, do nothing
3. Read even b, pop a

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

$\delta(q_1, a, a) = \delta(q_1, aa)$

$\delta(q_1, a, a) = \delta(q_1, aa)$

$\delta(q_1, b, a) = \delta(q_2, a)$

$\delta(q_2, b, a) = \delta(q_1, \varepsilon)$

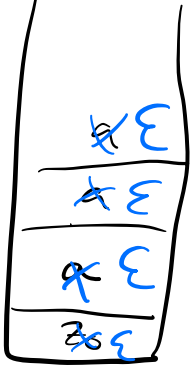
$\delta(q_1, b, a) = \delta(q_2, a)$

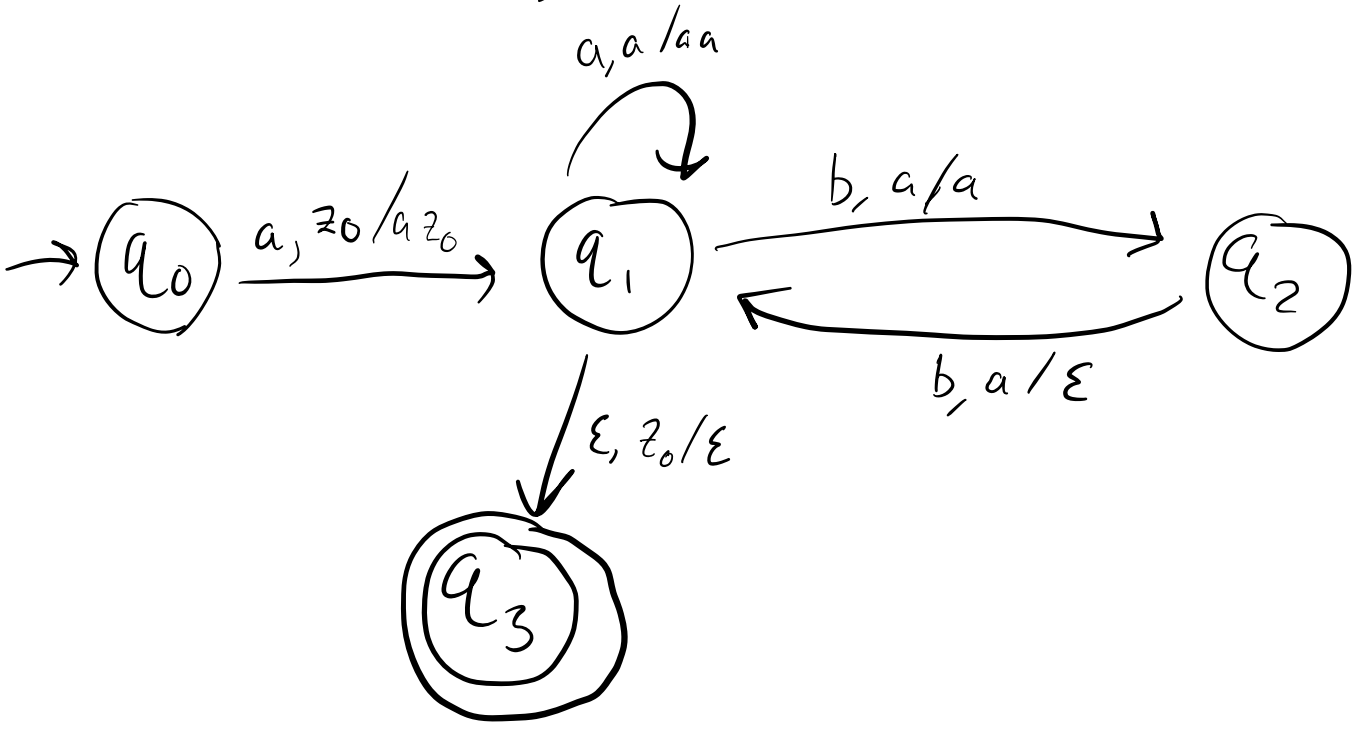
$\delta(q_2, b, a) = \delta(q_1, \varepsilon)$

$\delta(q_1, b, a) = \delta(q_2, a)$

$\delta(q_2, b, a) = \delta(q_1, \varepsilon)$

$\delta(q_1, \varepsilon, z_0) = \delta(q_3, \varepsilon)$





EXAMPLE

Build a PDA for the language $L = \{a^n b^{2n} \mid n \geq 1\}$

- Transition Function:

1. $q0$ (initial state): Read a, push A onto the stack, move to state $q1$
2. $q1$: Read a, push A onto the stack, remain in state $q1$
3. $q1$: Read b, move to state $q2$ without pushing or popping anything from the stack
4. $q2$: Read b, move to state $q3$ without pushing or popping anything from the stack
5. $q3$: Read b, pop A from the stack, move back to state $q1$
6. $q3$: Read a, remain in state $q3$ if there are no more input symbols left
7. $q3$: If ϵ (empty stack) and no input symbols left, move to state $q4$(final accepting state)

$0^n 1^n 2^n$
Not possible for PDA

TRY YOURSELF!

m = 2
n = 2

001122
m = 1
00122

Build a PDA for the language $L = \{0^n 1^m 2^n \mid m,n \geq 1\}$

Logic:
1. Read 0 push 0
2. Read 1 do nothing
3. Read 2 pop 0

$\delta(q_0, 0, z_0) = \delta(q_1, 0z_0)$

$\delta(q_1, 0, 0) = \delta(q_1, 00)$

$\delta(q_1, 0, 0) = \delta(q_1, 00)$

$\delta(q_1, 1, 0) = \delta(q_2, 0)$

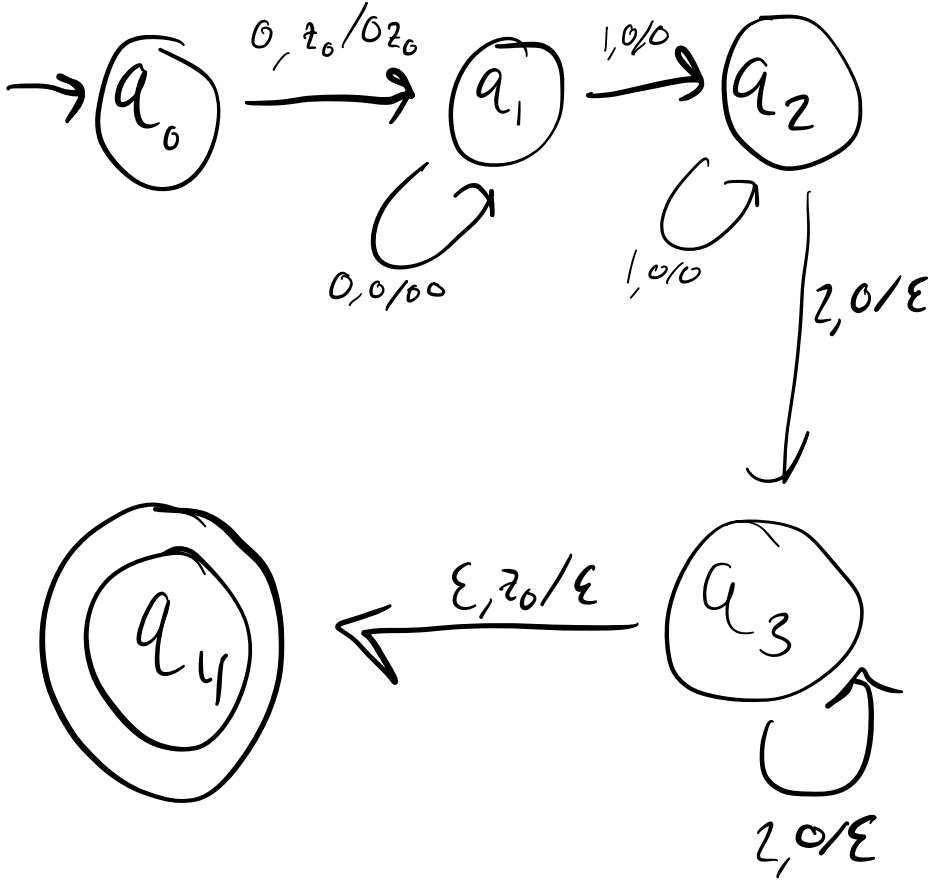
$\delta(q_2, 1, 0) = \delta(q_2, 0)$

$\delta(q_2, 2, 0) = \delta(q_3, \varepsilon)$

$\delta(q_3, 2, 0) = \delta(q_3, \varepsilon)$

$\delta(q_3, 2, 0) = \delta(q_3, \varepsilon)$

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



- Construct a PDA to accept the language L = {$a^n b^m c^m d^n$, where m, n ≥ 1}.

Logic:

1. Read a push a
2. Read b push b
3. Read c pop b
4. Read d pop a

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

$\delta(q_1, a, a) = \delta(q_1, aa)$

$\delta(q_1, b, a) = \delta(q_2, ba)$

$\delta(q_2, b, b) = \delta(q_2, bb)$

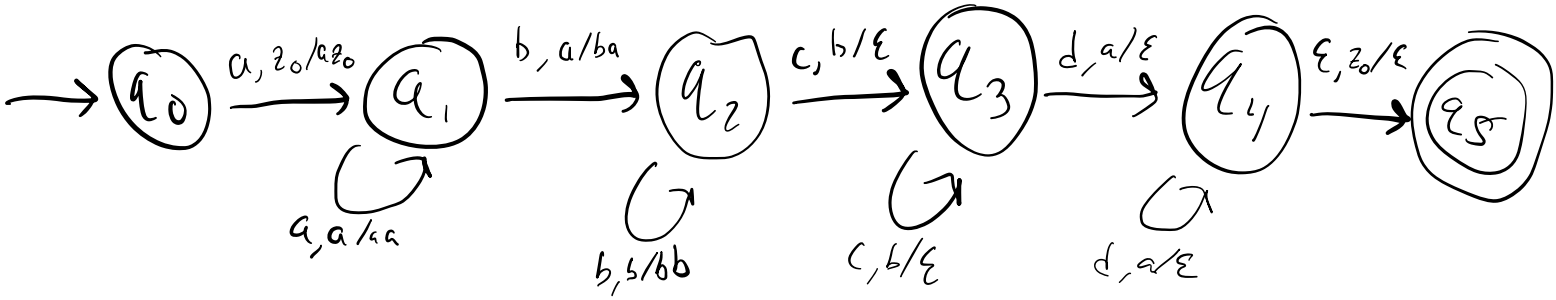
$\delta(q_2, c, b) = \delta(q_3, \varepsilon)$

$\delta(q_3, c, b) = \delta(q_3, \varepsilon)$

$\delta(q_3, d, a) = \delta(q_4, \varepsilon)$

$\delta(q_4, d, a) = \delta(q_4, \varepsilon)$

$\delta(q_4, \varepsilon, z_0) = \delta(q_5, \varepsilon)$



CONSTRUCT A PDA TO ACCEPT L = (A, B)* WITH EQUAL NUMBER OF 'A' AND 'B', I.E., $N_A(L) = N_B(L)$

Logic:

1. Read a push a
2. Read b pop a

or

1. Read b push b
2. Read a pop b

CONSTRUCT A PDA TO ACCEPT L = (A, B)* WITH EQUAL NUMBER OF 'A' AND 'B', I.E., $N_A(L) = N_B(L)$

Logic: The δ function for these two cases are

a. If string starts with a, and a comes after a then the PDA is in state q0 with the stack top a.

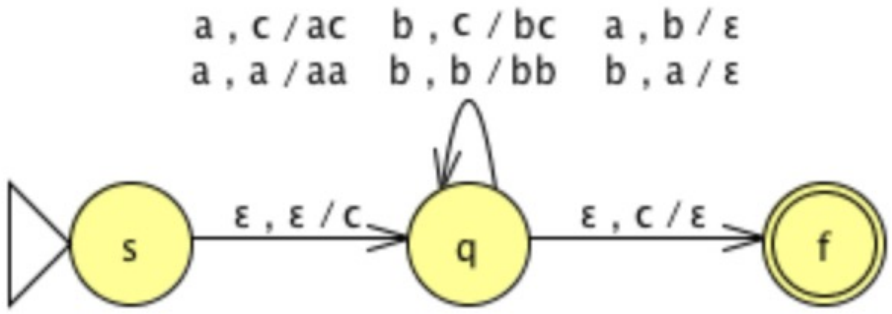
b. If the string starts with 'a' and 'b' comes after 'a', then one match of 'a' and 'b' is got and the stack top, i.e., a will be popped from the stack.

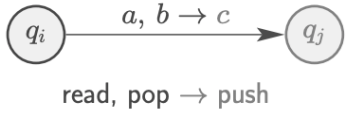
c. If string starts with b, and b comes after a then the PDA is in state q0 with the stack top b.

d. If the string starts with 'b' and 'a' comes after 'b', then one match of 'b' and 'a' is got and the stack top, i.e., b will be popped from the stack.

- The transitions will be,
- $\delta(q0, a, z_0) = (q0, az_0)$
- $\delta(q0, b, z_0) = (q0, bz_0)$
- $\delta(q0, a, a) = (q0, aa)$
- $\delta(q0, b, a) = (q0, \varepsilon)$
- $\delta(q0, b, b) = (q0, bb)$
- $\delta(q0, a, b) = (q0, \varepsilon)$
- $\delta(q0, \varepsilon, z_0) = (q0, \varepsilon)$

CONSTRUCT A PDA TO ACCEPT L = (A, B)* WITH EQUAL NUMBER OF 'A' AND 'B', I.E., $N_A(L) = N_B(L)$





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

$a^i b^i c^k \mid a^i b^j c^j$

Logic:

1. read a push a OR 1. read a do nothing
2. read b pop a 2. read b push b
3. read c do nothing 3. read c pop b

- Case 1: i=j

logic -

- Read a, push it onto the stack.
- When you start reading b, pop one a for every single b.
- When you start reading c, the stack must be empty Just go on reading c because there are any number of c's in the language.

- $\delta(q0, a, z_0) = (q0, az_0)$
- $\delta(q0, a, a) = (q0, aa)$
- $\delta(q0, b, a) = (q1, \varepsilon)$
- $\delta(q1, b, a) = (q1, \varepsilon)$
- $\delta(q1, c, z_0) = (q2, z_0)$
- $\delta(q2, c, z_0) = (q2, z_0)$
- $\delta(q2, \varepsilon, z_0) = (q3, \varepsilon)$

Current state	Current input	Next state	Stack
Start			Z_0
q_0	a	q_0	aZ_0
q_0	a	q_0	aa
q_0	b	q_1	ε (i.e. pop a)
q_1	b	q_1	ε (i.e. pop a)
q_1	c	q_2	Z_0
q_2	c	q_3	Z_0
q_3 ACCEPT STATE	Δ	q_3	Z_0

- Case 2:j=k
- Logic:
  - reading a's.
  - When you read out b, just push it onto the stack.
  - As soon as c is read pop b for each corresponding c.

- $\delta(q0, a, z_0) = (q0, z_0)$
- $\delta(q0, b, z_0) = (q1, bz_0)$
- $\delta(q1, b, b) = (q1, bb)$
- $\delta(q1, c, b) = (q2, \varepsilon)$
- $\delta(q2, c, b) = (q2, \varepsilon)$
- $\delta(q2, \varepsilon, z_0) = (q3, \varepsilon)$

Current state	Current input	Next state	Stack
Start			Z_0
q_0	a	q_0	Z_0
q_0	b	q_1	bZ_0
q_1	b	q_1	bb
q_1	c	q_2	ε i.e. pop b
q_2	c	q_2	ε
q_2	ε	q_3	Z_0

PDA FOR LANGUAGE $\{WW^R \mid W \in \{0, 1\}^*\}$

W^R: reverse of string

1101

1. $\delta(q_0,0, Z_0)=\{(q_0,0Z_0)\}$
2. $\delta(q_0,1, Z_0)=\{(q_0,1Z_0)\}$

First symbol push on stack

3. $\delta(q_0,0, 0)=\{(q_0,00)\}$
4. $\delta(q_0,0, 1)=\{(q_0,01)\}$
5. $\delta(q_0,1, 0)=\{(q_0,10)\}$
6. $\delta(q_0,1, 1)=\{(q_0,11)\}$

Grow the stack by pushing new symbols on top of old (w-part)

Showing going into popping mode

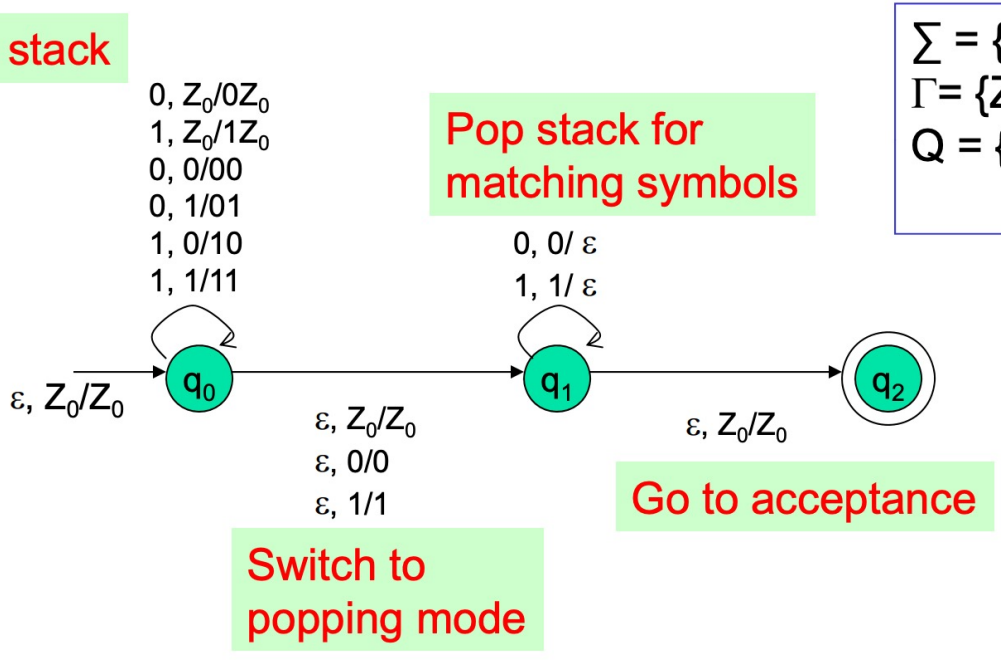
7. $\delta(q_0, \varepsilon, 0)=\{(q_1, 0)\}$
8. $\delta(q_0, \varepsilon, 1)=\{(q_1, 1)\}$
9. $\delta(q_0, \varepsilon, Z_0)=\{(q_1, Z_0)\}$

Switch to popping mode, nondeterministically (boundary between w and w^R)

10. $\delta(q_1,0, 0)=\{(q_1, \varepsilon)\}$
11. $\delta(q_1,1, 1)=\{(q_1, \varepsilon)\}$

Shrink the stack by popping matching symbols (w^R-part)

PDA FOR LANGUAGE $\{WW^R \mid W \in \{0, 1\}^*\}$



EXAMPLE

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

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

Logic:

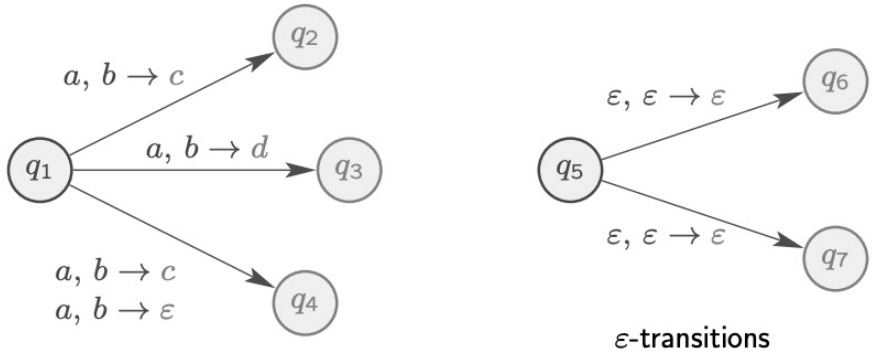
1. Read a push a
2. Read b push b
3. Read c pop a or b

EXAMPLE

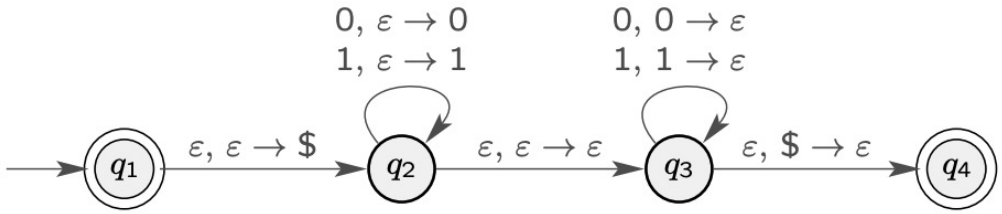
Design pushdown automata that recognizes the language $G = \{a^{2n}\, b^{3n} \mid n \geq 0\}$.

PDA MAY BE NONDETERMINISTIC

- 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 $b \in \Gamma_\varepsilon$;
- $\delta(q_1, a, b) = \{ (q_2, c), (q_3, d), (q_4, c), (q_4, \varepsilon) \}$



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

PDA works as follows:

$q_1 \rightarrow q_2$: First pushes $ on stack to mark bottom

$q_2 \rightarrow q_2$: Reads in first half w of string, pushing it onto stack

$q_2 \rightarrow q_3$: Guesses that it has reached middle of string

$q_3 \rightarrow q_3$: Reads second half w^R of string, matching symbols from first half in reverse order (recall: stack LIFO)

$q_3 \rightarrow q_4$: Makes sure that no more input symbols on stack

CFG TO PDA

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?

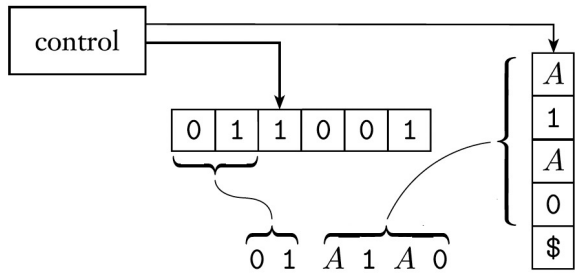
HOW DOES PDA STORES INTERMEDIATE STRINGS?

- 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

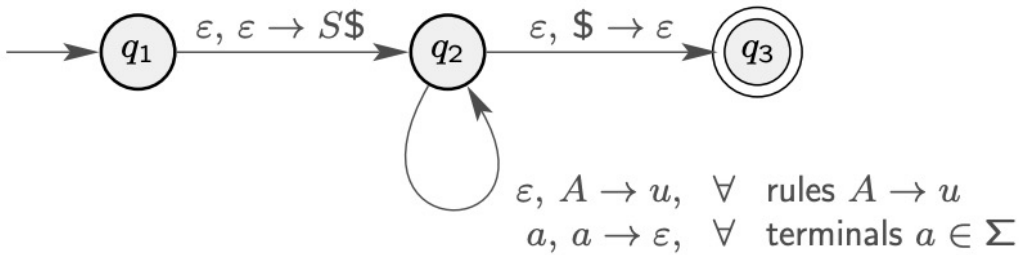
The following is an informal description of P.

- 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

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

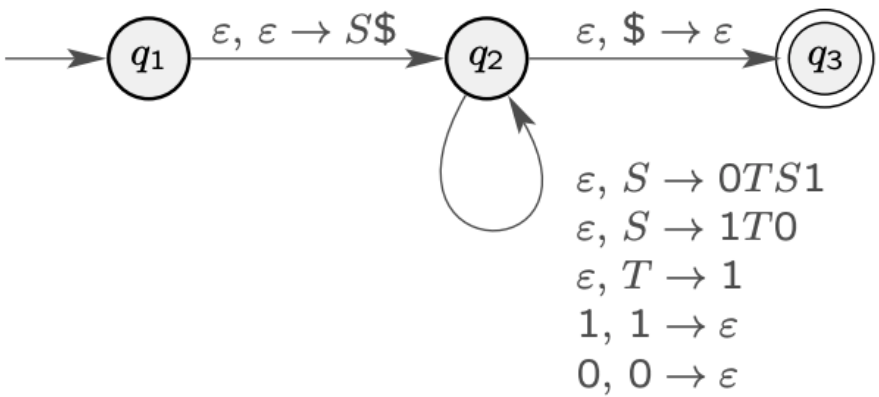
- PDA is non-deterministic.
- Input alphabet of PDA is the terminal alphabet of CFG
  - $\Sigma = \{0, 1\}$.
- Stack alphabet consists of all variables, terminals and "$"
  - $\Gamma = \{S, T, 0, 1, \$\}$.
- PDA simulates a leftmost derivation using CFG
  - Pushes RHS of rule in reverse order onto stack.

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

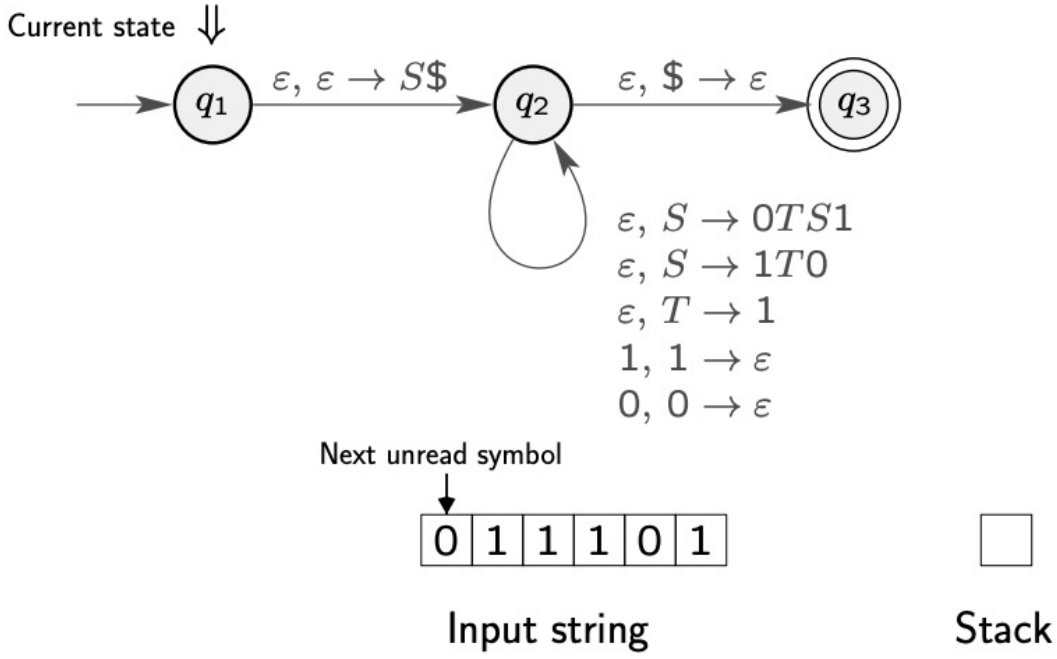
$$S \Rightarrow 0TS1 \Rightarrow 01S1 \Rightarrow 011T01 \Rightarrow 011101$$

Let's now process string 011101 on PDA.

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



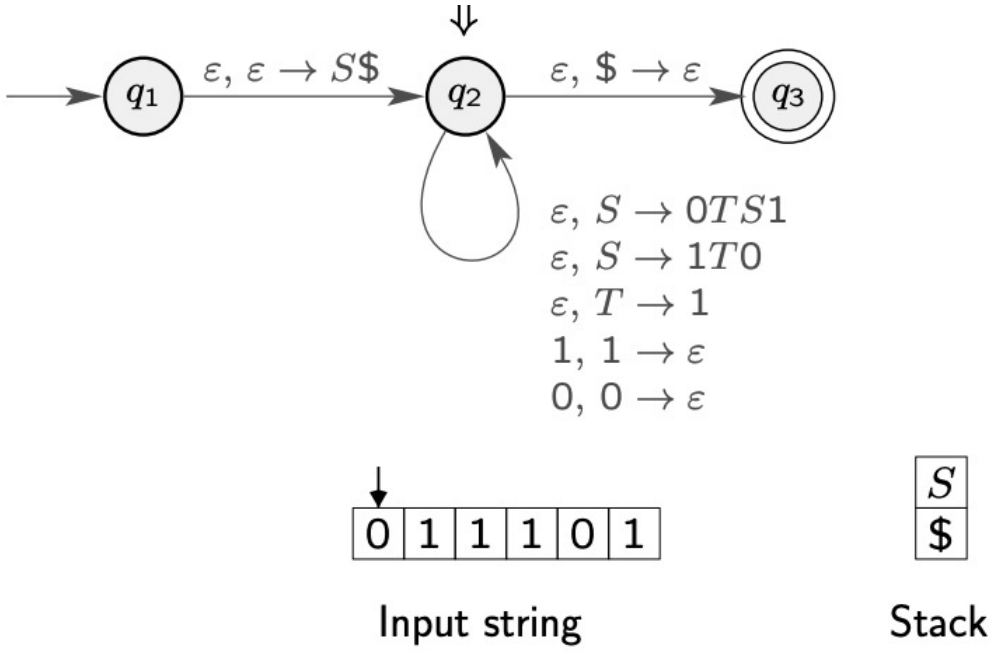
0. Start in state q_1 with 011101 on input tape and empty stack.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

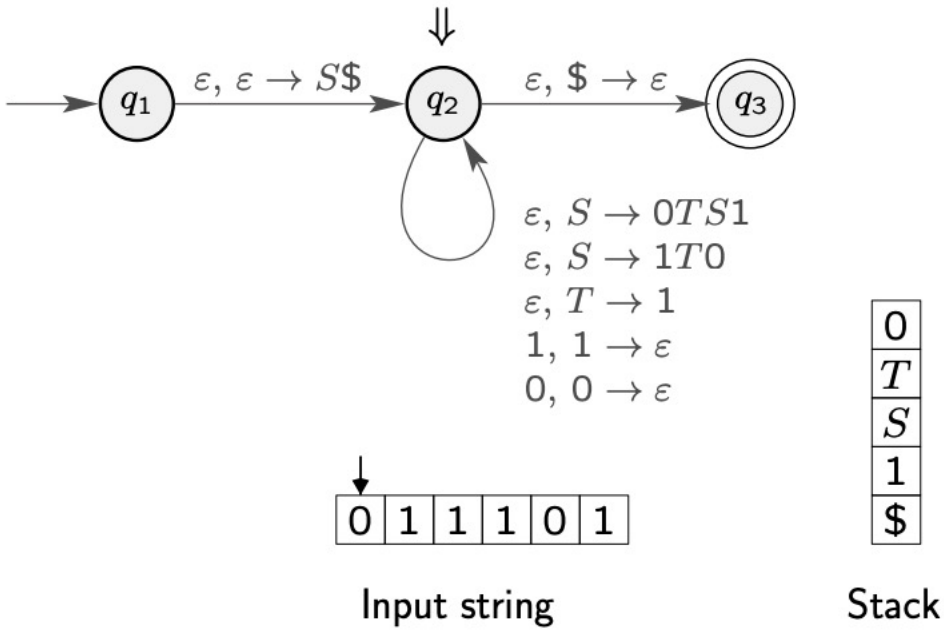
1. Read nothing, pop nothing, move to q_2, and push \$ and then S.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

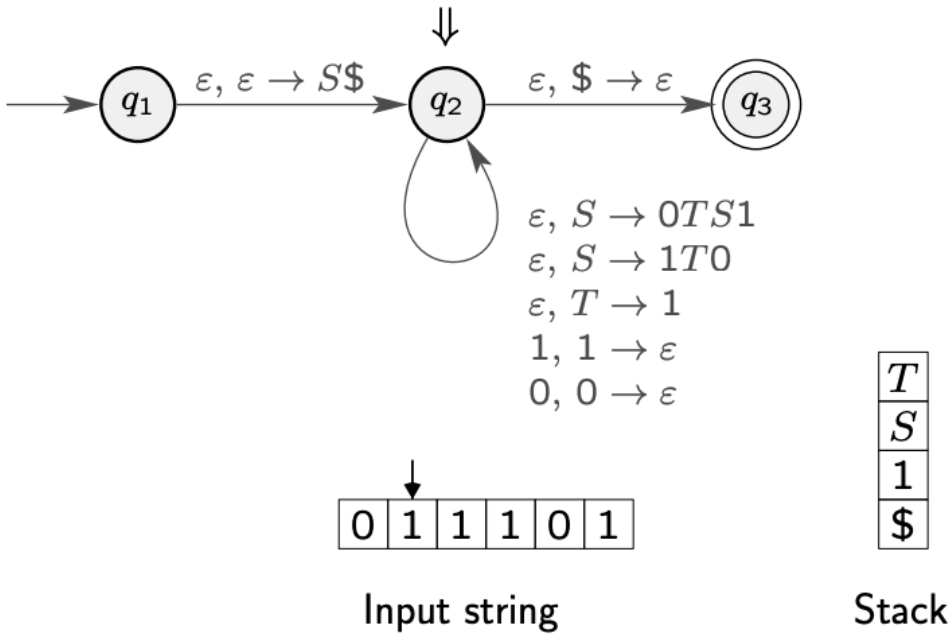
2. Read nothing, pop S, return to q_2, and push 0T S1.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

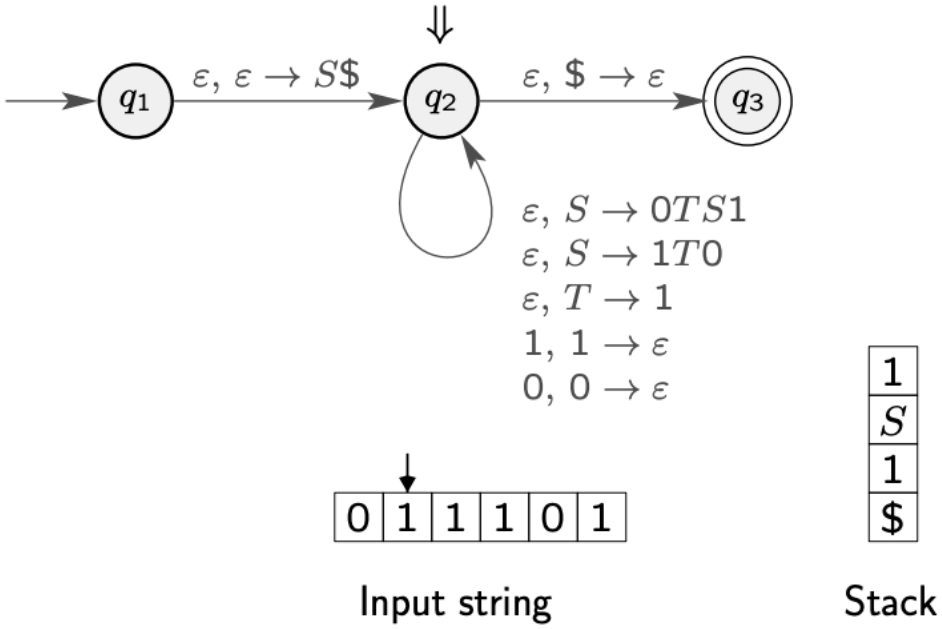
3. Read 0, pop 0, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

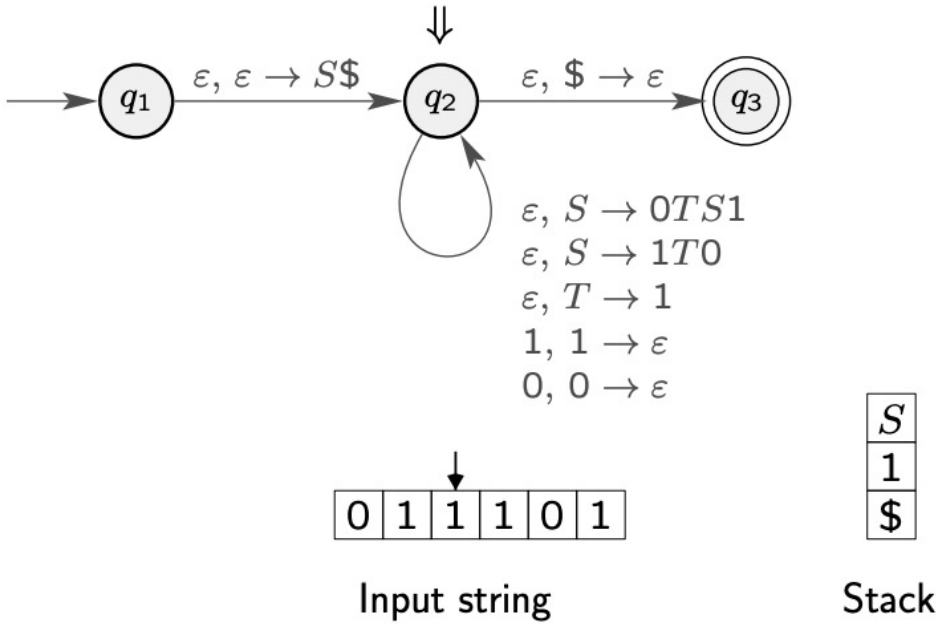
4. Read nothing, pop T, return to q_2, and push 1.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

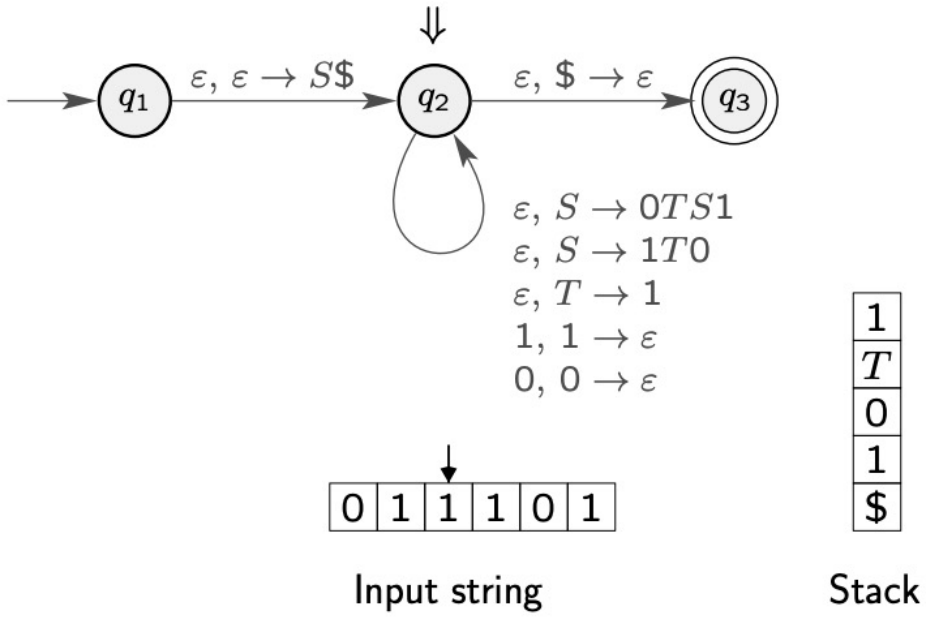
5. Read 1, pop 1, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

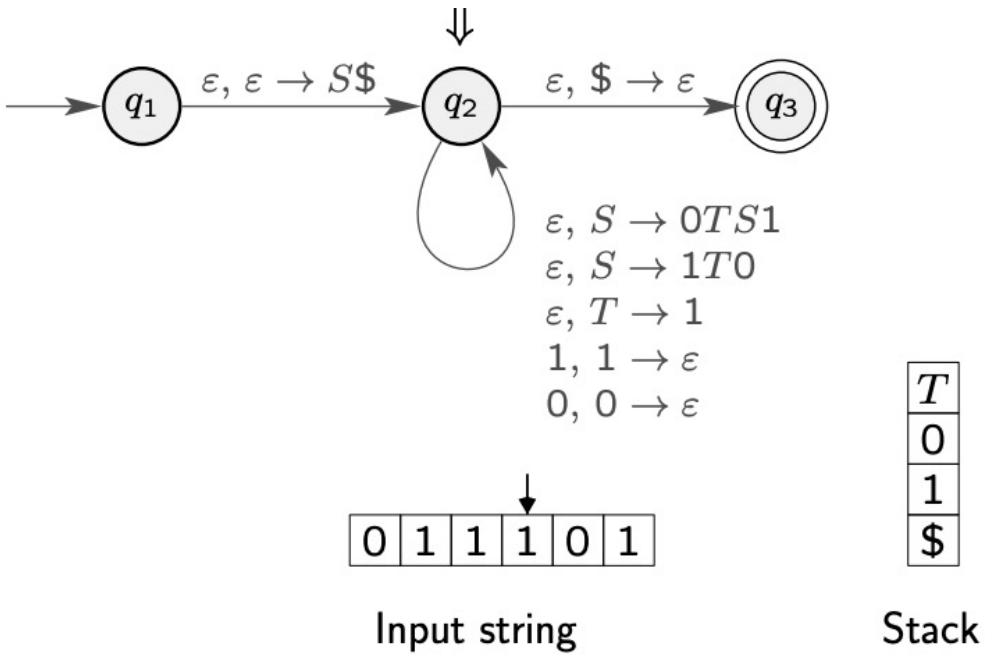
6. Read nothing, pop S, return to q_2, and push 1T0.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

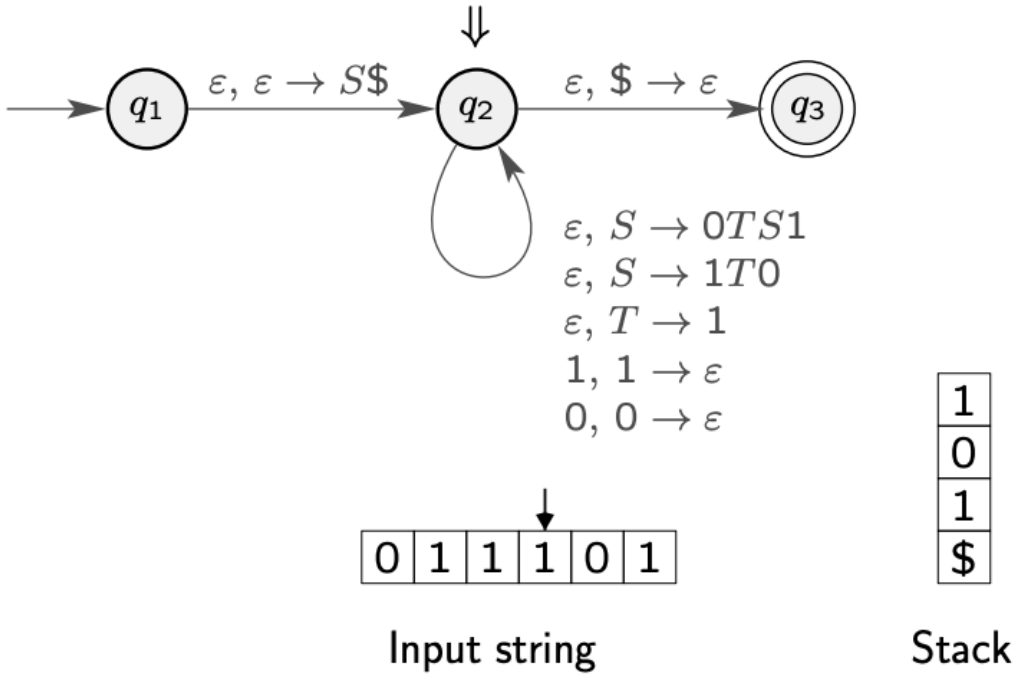
7. Read 1, pop 1, return to q_2, and push nothing



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

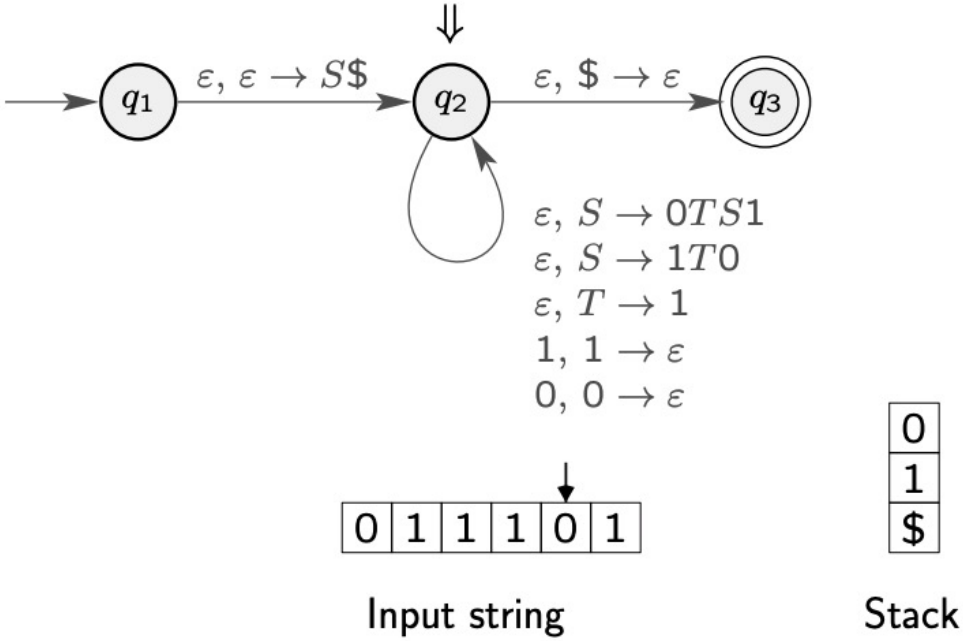
8. Read nothing, pop T, return to q_2, and push 1.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

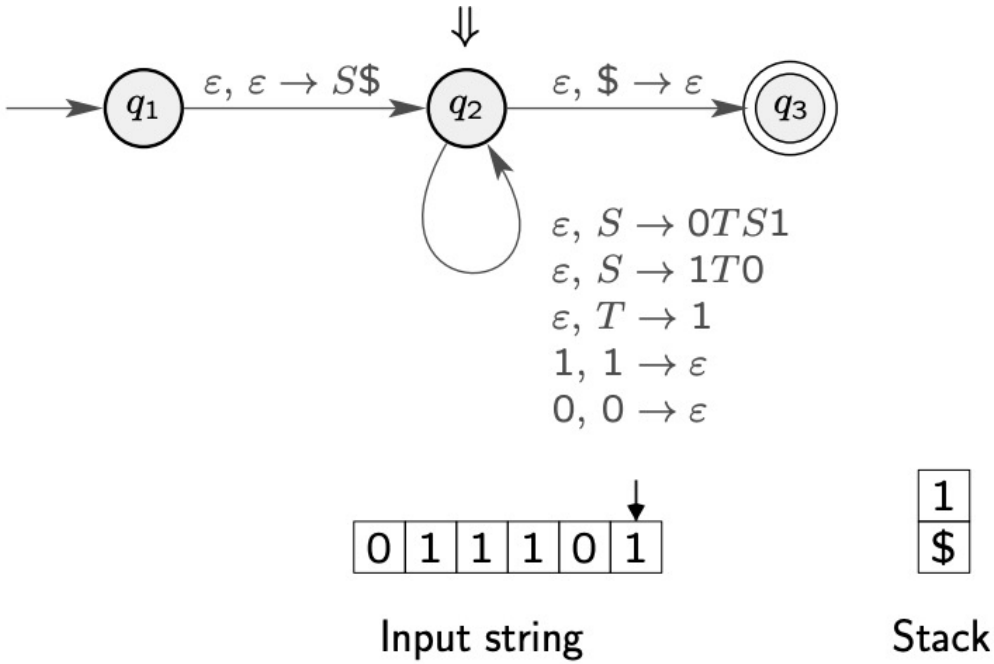
9. Read 1, pop 1, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

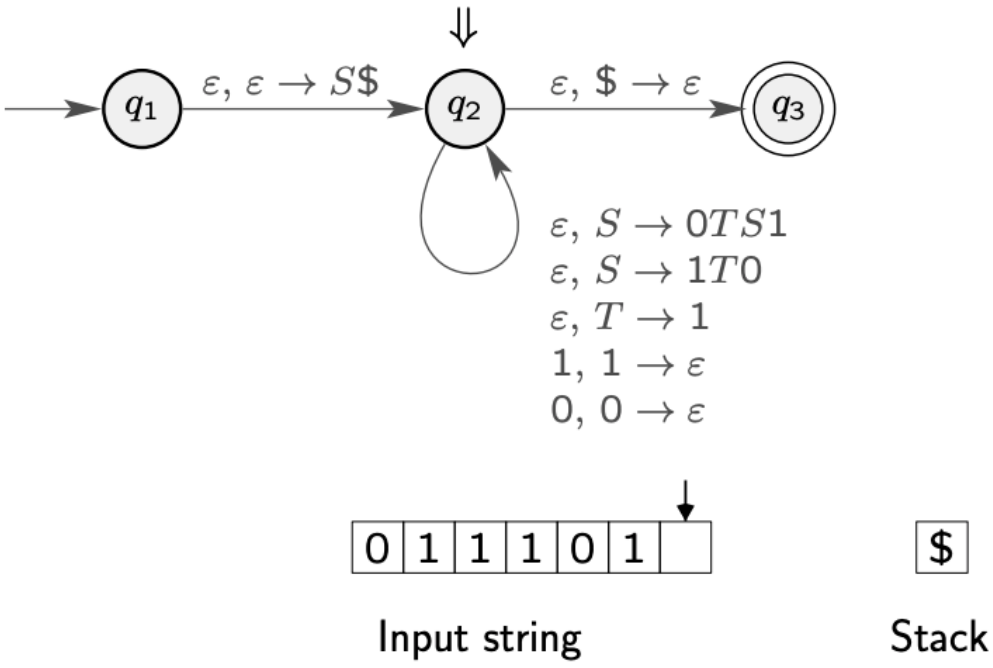
10. Read 0, pop 0, return to q_2, and push nothing.



Leftmost derivation of string 011101 ∈ L(G):

S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

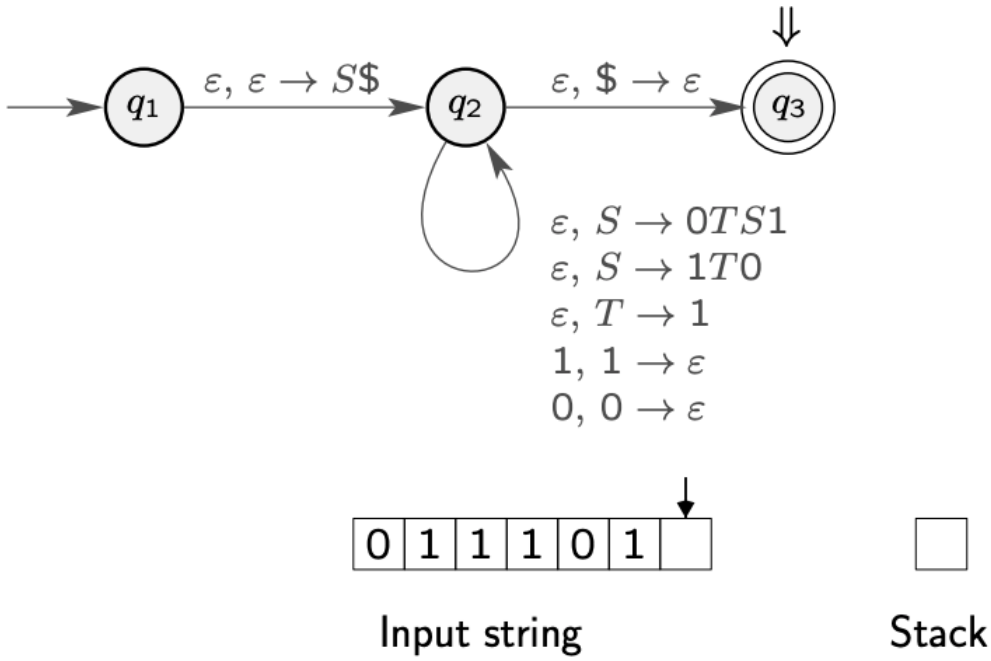
11. Read 1, pop 1, return to q_2, and push nothing.



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.

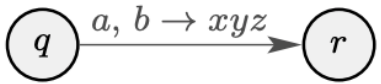


Leftmost derivation of string 011101 ∈ L(G):

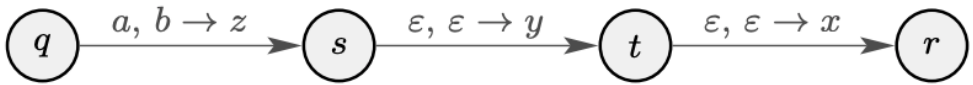
S ⇒ 0T S1 ⇒ 01S1 ⇒ 011T01 ⇒ 011101

CONSTRUCTED PDA IS NOT COMPLIANT

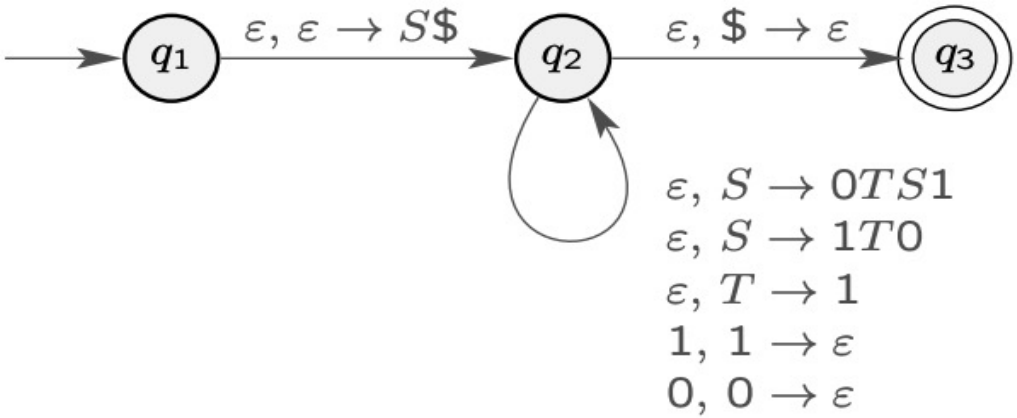
- Problem: pushing strings onto stack instead of ≤ 1 symbols, which is not allowed in PDA specification.
- PDA transition function $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$
- Solution: Add extra states



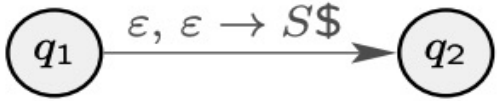
becomes



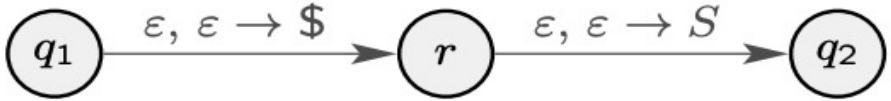
In our PDA,



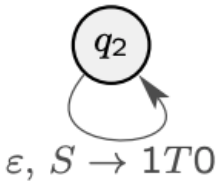
we replace



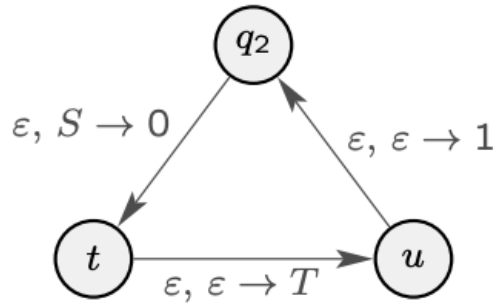
with



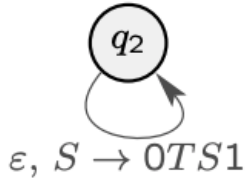
Also, replace



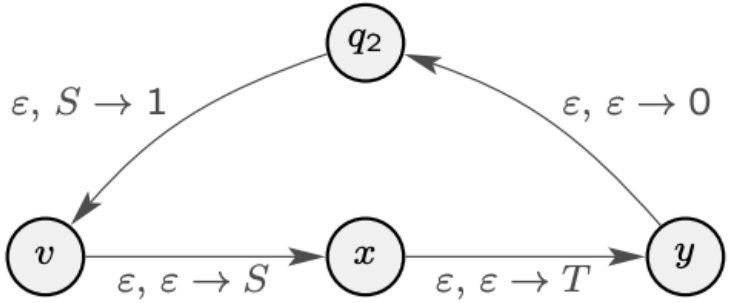
with



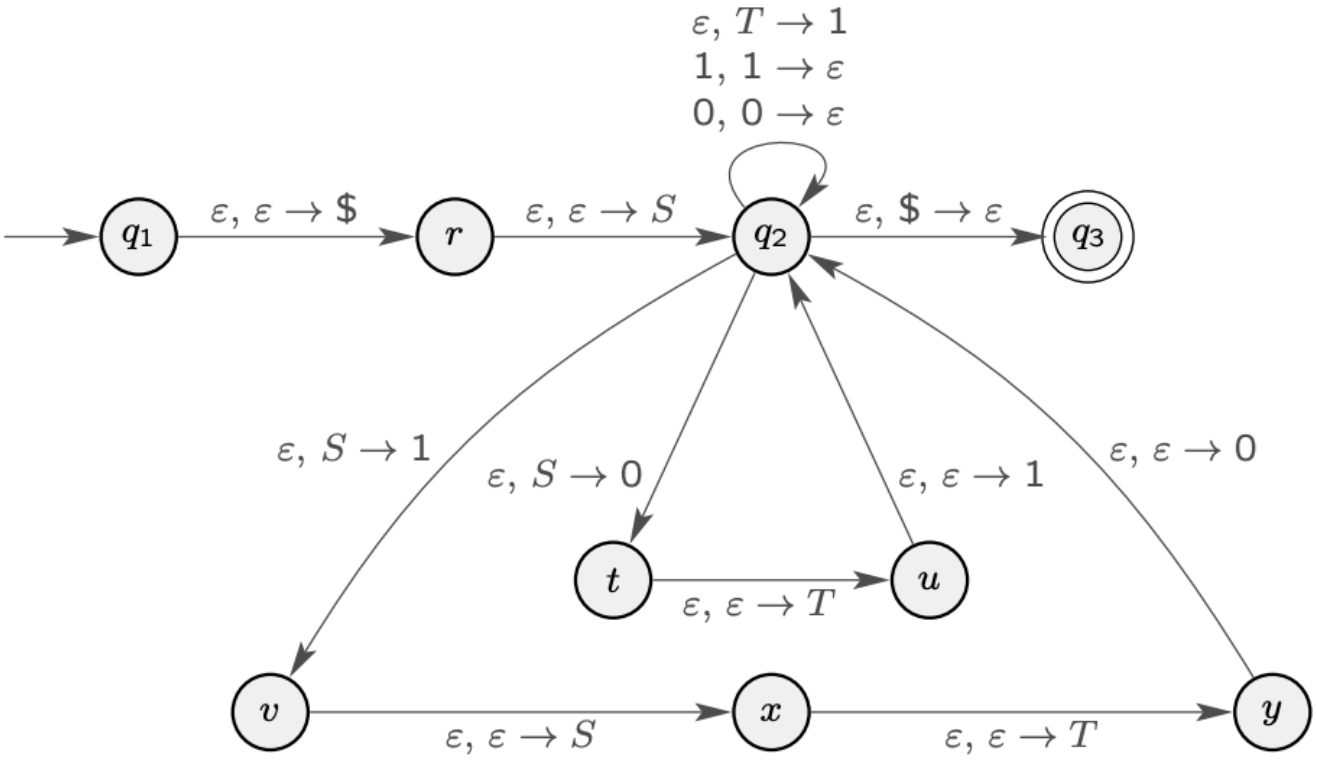
and replace



with



So our final PDA from the CFG is

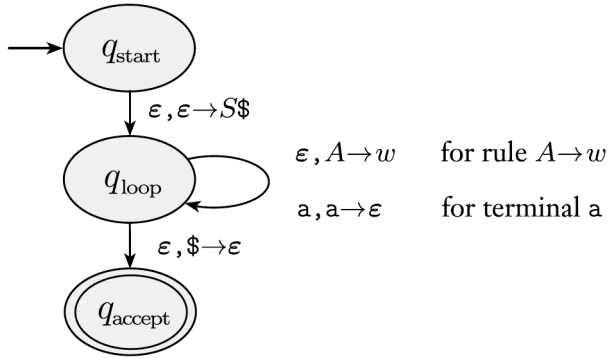


REVIEW

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

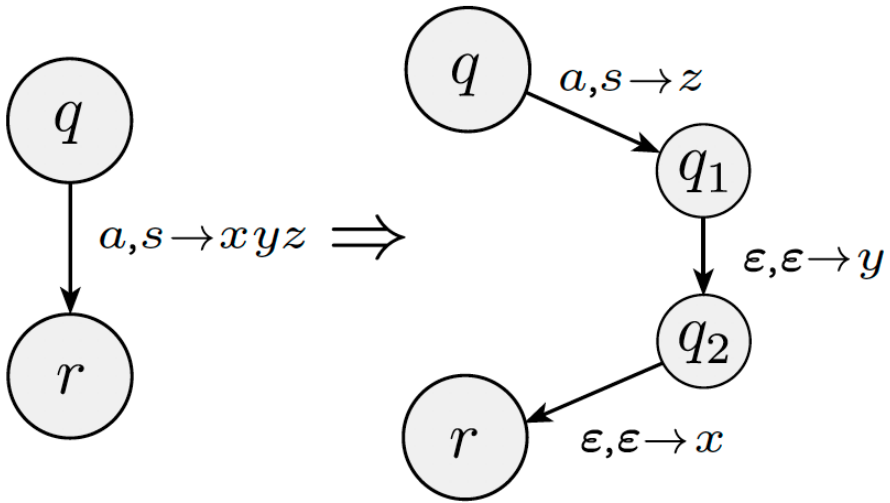
- Case (b) wherein the top of the stack contains a terminal.

  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.

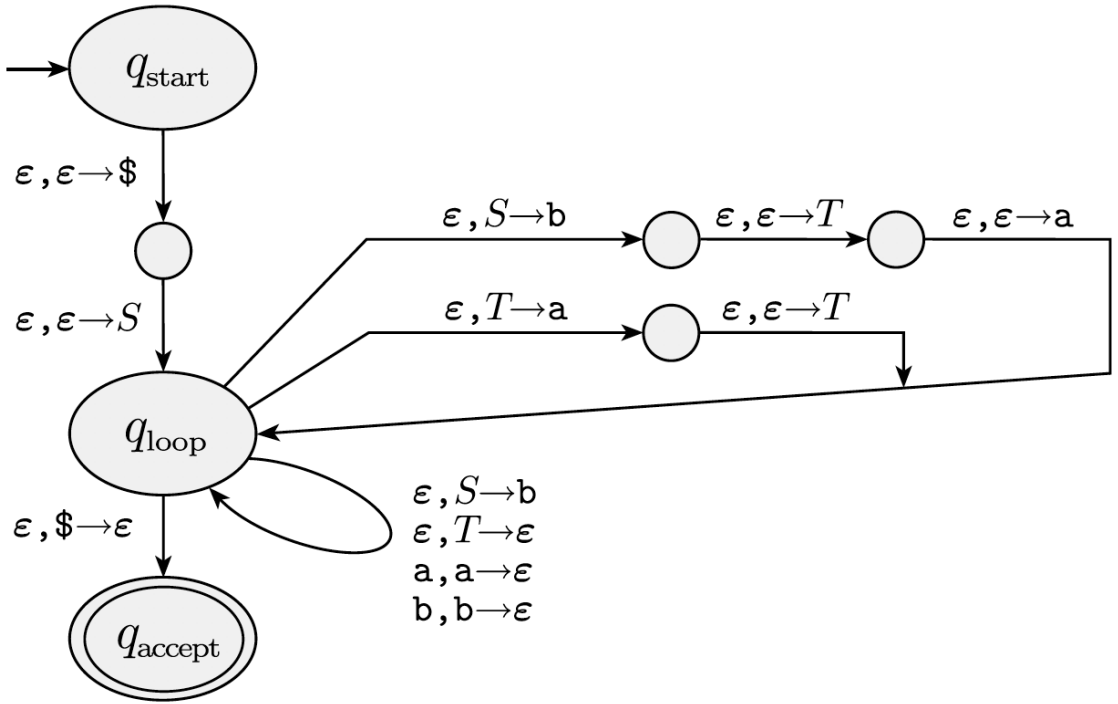
We use the procedure developed in Lemma 2.21 to construct a PDA P_1 from the following CFG G.

$S \rightarrow aT\,b \mid b, T \rightarrow T\,a \mid \varepsilon$

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 $S \rightarrow 0S1 \mid A \quad A \rightarrow 1A0 \mid S \mid \varepsilon$.

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