Equivalence of PDA and CFG/CFL

Theorem 2.20

A language is context free if and only if some pushdown automaton recognizes it.

PDA $\Leftrightarrow$ CFL/CFG

Lemma 2.21

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

CFG $\Rightarrow$ PDA

Lemm 2.21 Proof idea:

We need a way to map derivations of strings using CFG to PDA.

Idea1:

One idea would be to keep the intermediate string that we produce in the stack.

Why?

PDA needs to find the variables in the intermediate string to make substitutions.

Idea2:

Keep only a part of intermediate string.

Keep any symbols starting with the first variable in the intermediate string.

$$S \longrightarrow aSbcTa$$

Any terminal symbols appearing before the first variable are matched immediately with symbols in the input string.

Here is the general idea for converting the CFG to PDA.

1. Place the marker symbol $ and the start variable on the stack.

2. Repeat the following steps:

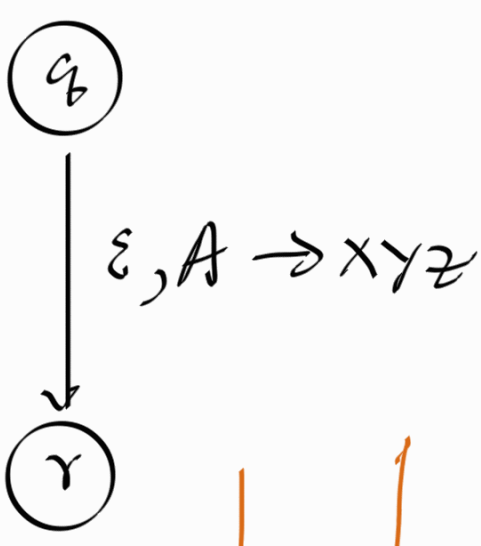
   a) If the top of the stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right handside of the rule.

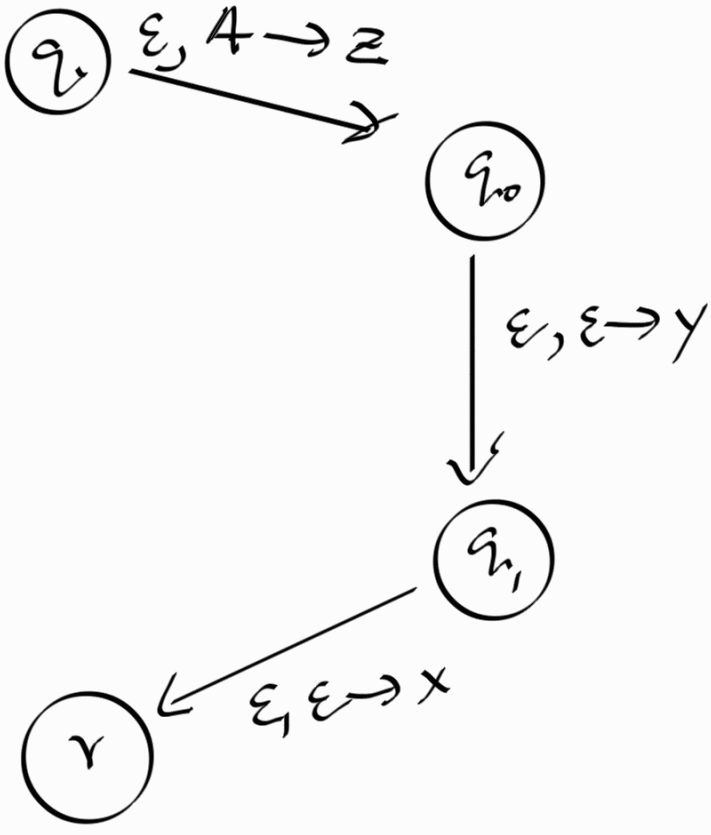
   b) If the top of the stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.

   c) If the top of the stack is symbol $, enter the accept state. Doing so accepts the string if all of it has been read.

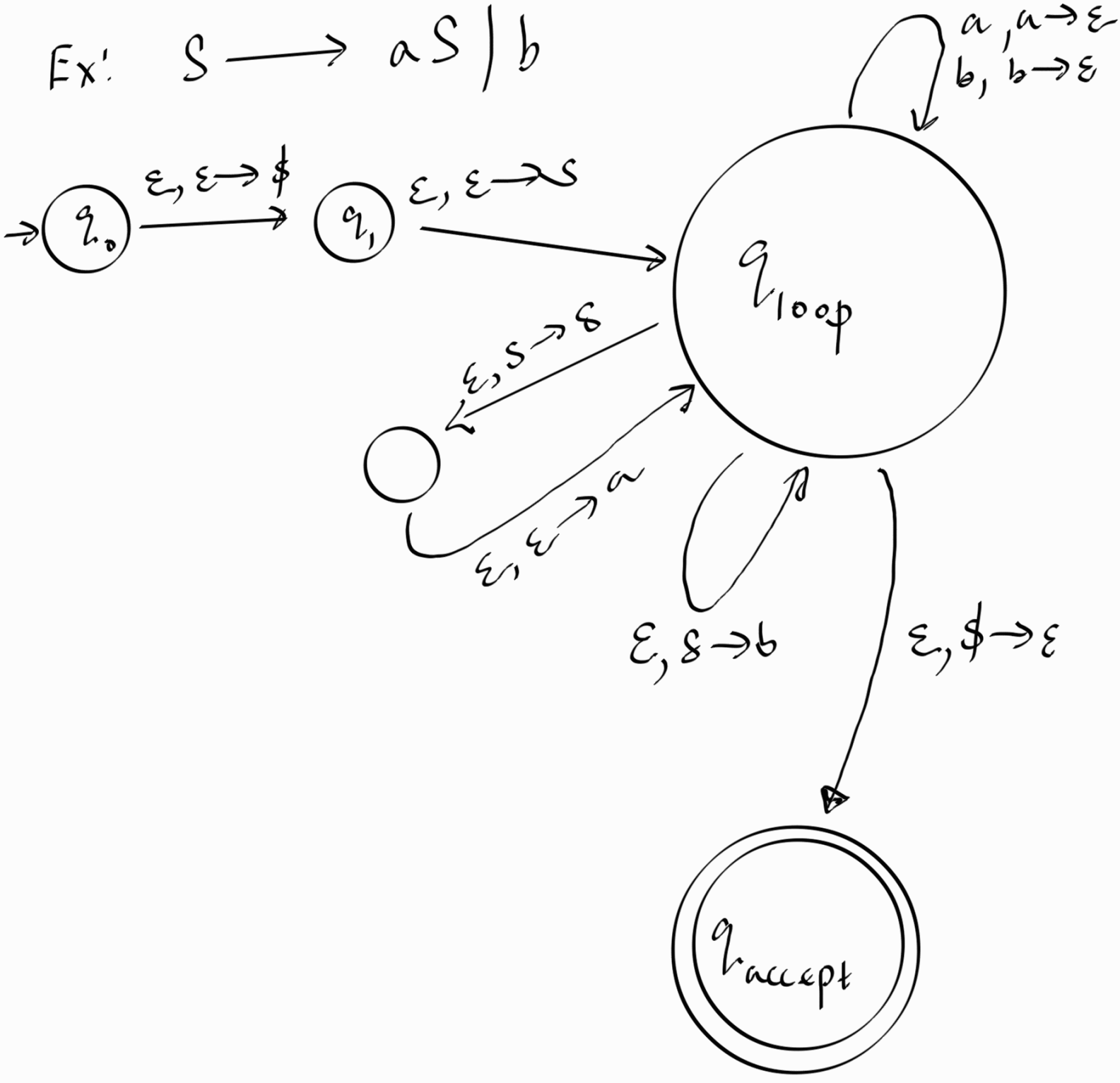
Let's briefly look at how we can implement part a).

$A \longrightarrow xyz$

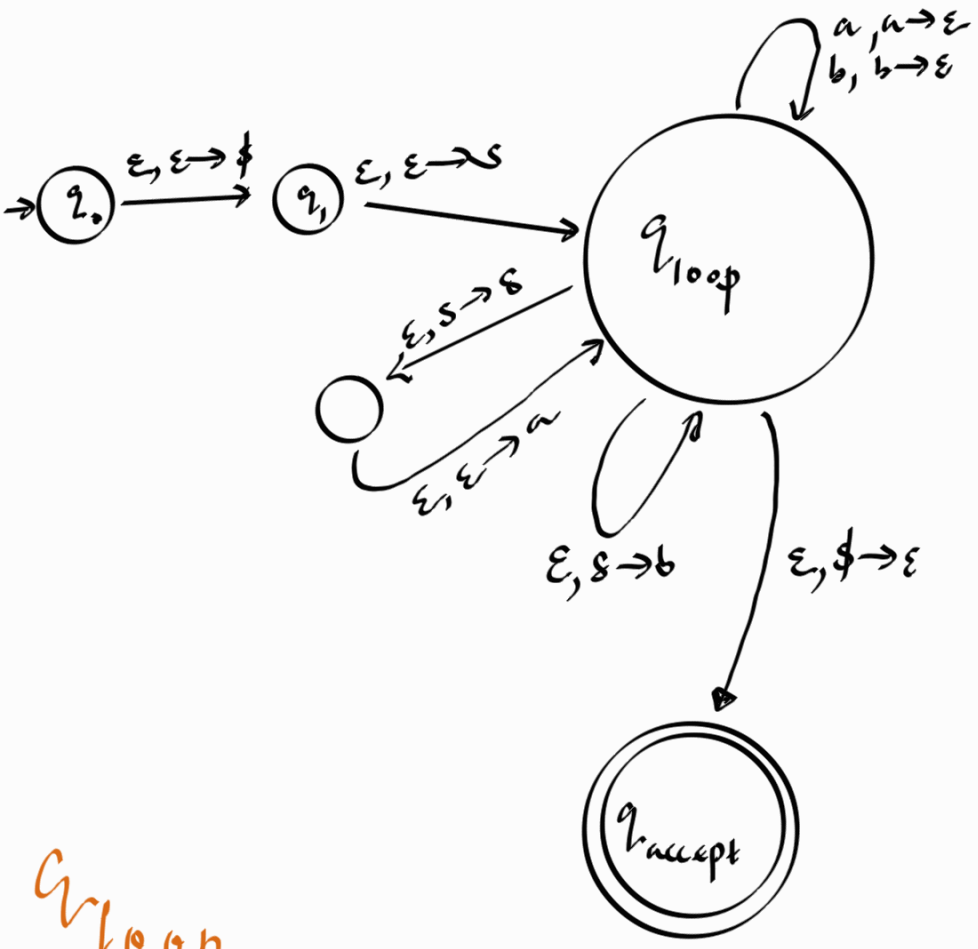


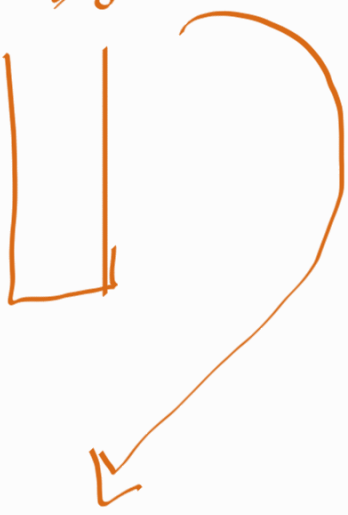


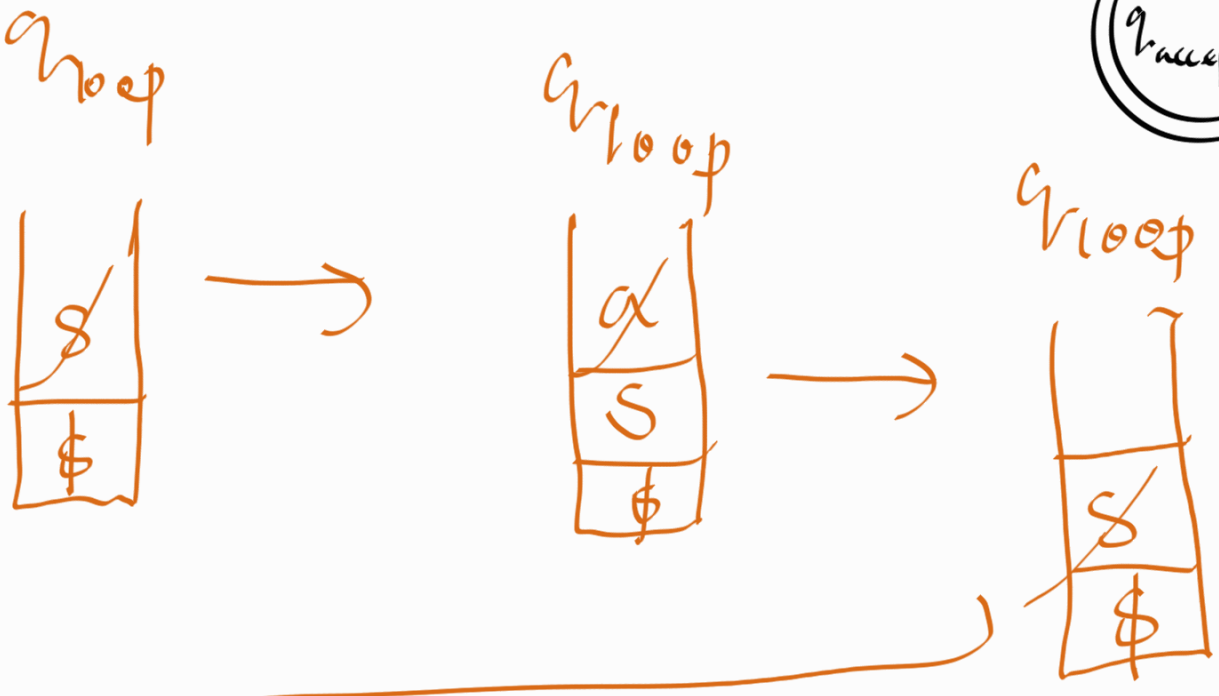
Ex: $S \longrightarrow aS \mid b$

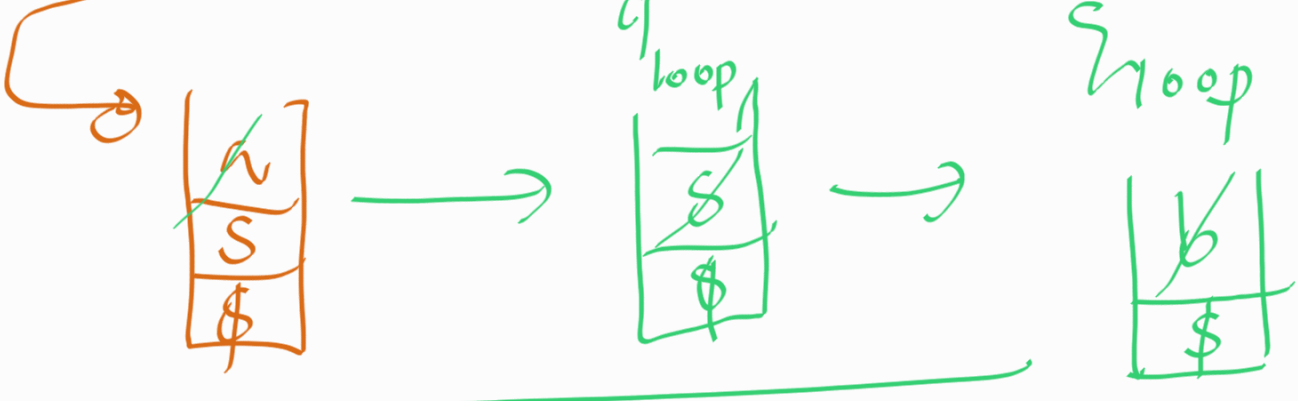


$S = aSb$


a, a → ε
b, b → ε
ε, ε → $
ε, ε → S
q0
q1
qloop
ε, S → b
ε, ε → a
ε, S → b
ε, $ → ε
qaccept


q0


qloop
S
$
qloop
a
S
$
qloop
S
$

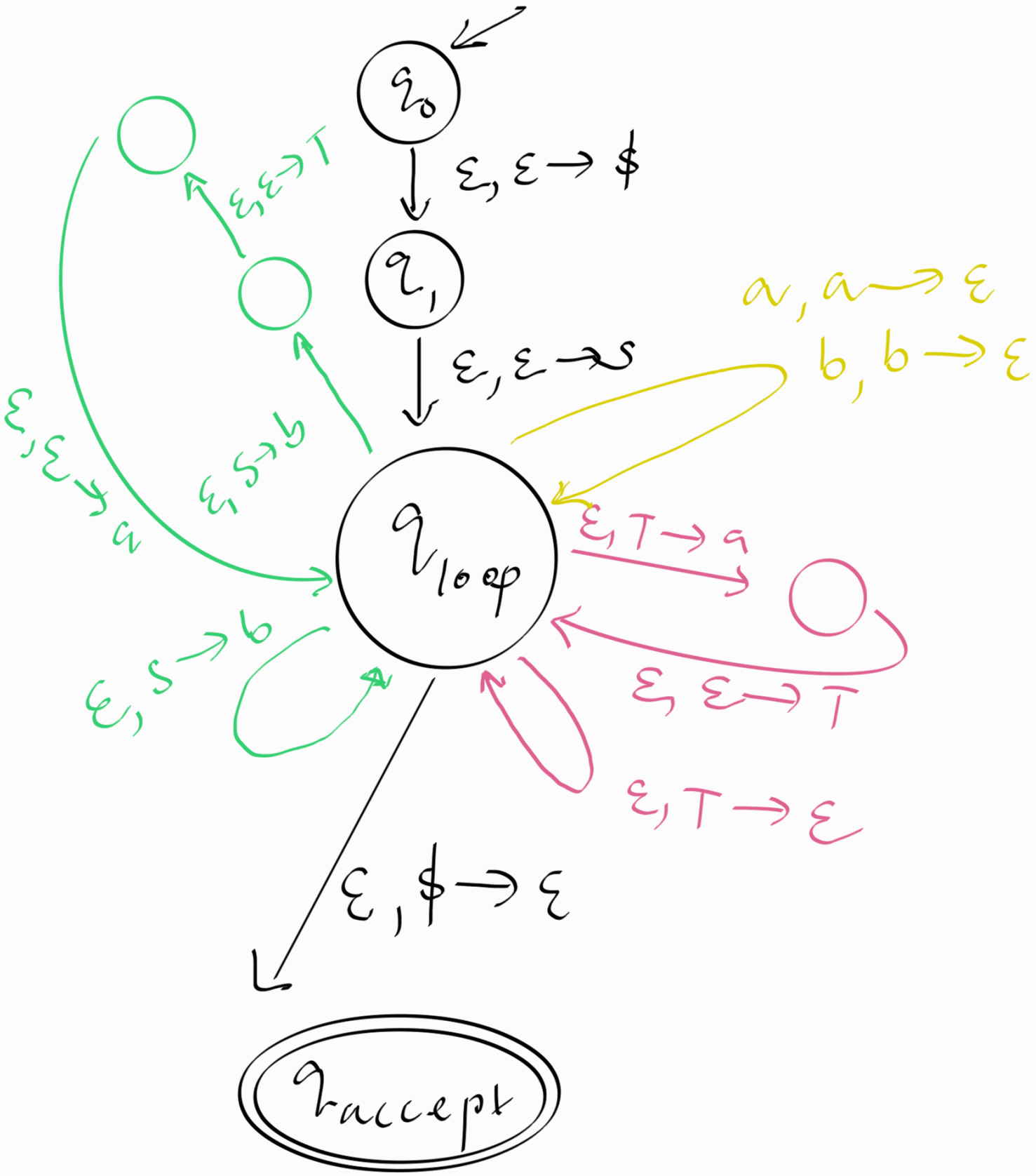

a
S
$
qloop
S
$
qloop
b
$


$

CFG to PDA

$S \rightarrow aTb \mid b$

$T \rightarrow Ta \mid \varepsilon$



Lemma 2.27

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

Sketch of the proof:

We want our grammar to simulate the PDA M.

This is harder to do.

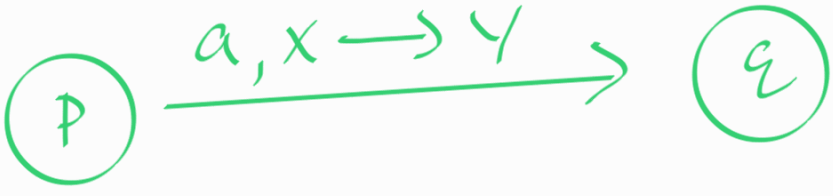
We define variables for G as follows:

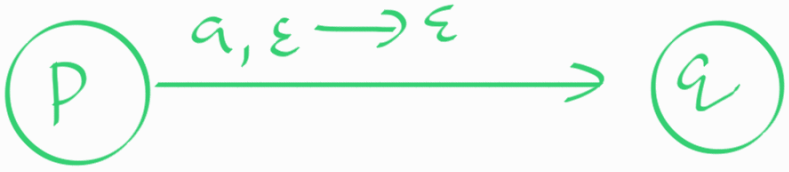
A_{pq} for all pairs of states P, Q.

This variable generates all strings that can take the PDA M from P with an empty stack to Q with an empty stack.

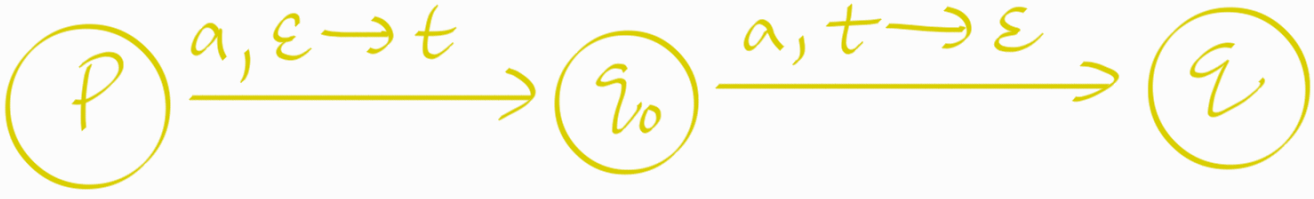
We modify M as follows.

1. Modify M to have a single accept state.
2. Empty the stack before accepting.
3. Each transition either pushes a symbol into the stack or pops one off the stack, but does not do both at the same time.





$t \in \Gamma$



Here is the algorithm

Suppose PDA $P = (Q, \Sigma, \Gamma, \delta, q_0)$ and we construct grammar G.

The variables of G are

$$\{A_{pq} \mid p, q, \in Q\}$$

The start variable is $A_{q_0, q_{accept}}$

1. 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

$$A_{pq} \longrightarrow a A_{rs} b .$$

2. For each $p, q, r \in Q$, put the rule, $A_{pq} \longrightarrow A_{pr} A_{rq}$

3. Finally, for every $p \in Q$, put the rule $A_{pp} \longrightarrow \varepsilon$ in G.

We need to show that this construction works by showing

A_{pq} generates string x $\iff$ x can bring the PDA P from p with an empty stack to q with an empty stack.

In the proof, you need to show

$\Rightarrow$

1. If A_{pq} generates x, then x can bring the PDA from P with an empty stack to q with an empty stack.

$\Leftarrow$

2. If x can bring PDA from p with an empty stack to q with an empty stack, then A_{pq} generates x.