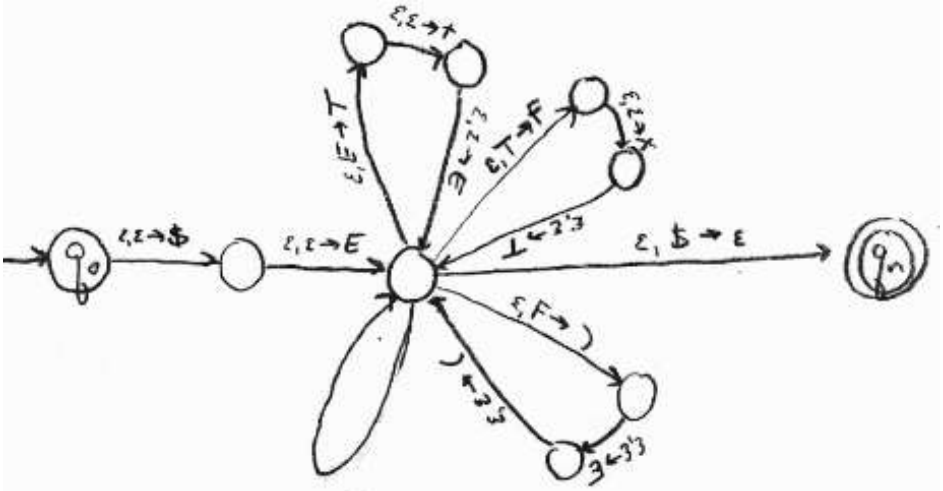
Theory of Computation, CSCI 438 spring 2022
Equivalence of Non-deterministic pdas and Context Free Grammars, pages 115-119
Feb. 25

2.11 Convert the CFG G_4 given in Exercise 2.1 to an equivalent PDA, using the procedure given in Theorem 2.20.

$E \rightarrow E + T \mid T$
$T \rightarrow T \times F \mid F$
$F \rightarrow (E) \mid a$

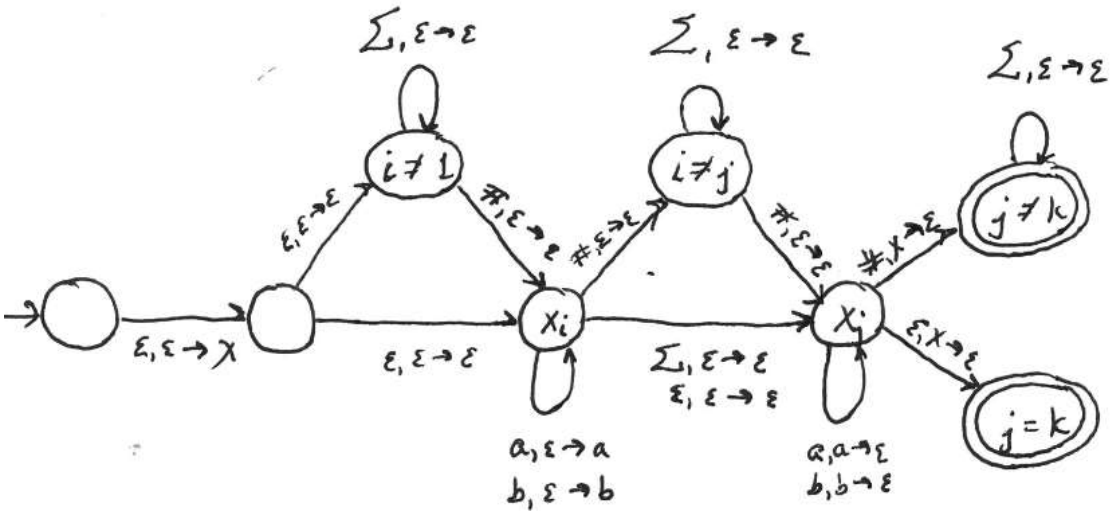
Answer:



2.6 d Give a PDA for

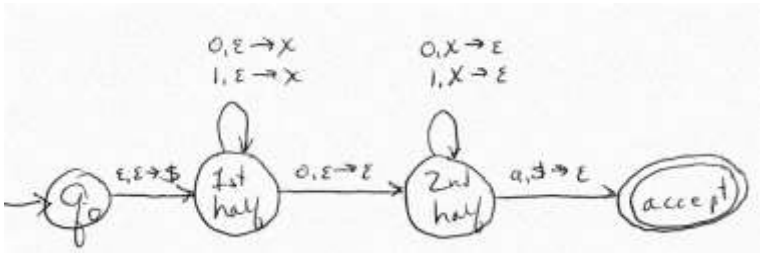
$L = \{x_1\#x_2\#...\#x_k \mid k \geq 1, \text{ each } x_i \in \{a,b\}^*, \text{ and for some i and j, } x_i = x_j^R\}$

Mark the bottom of the stack. Consume a's, b's and #'s until the beginning of x_i is non-deterministically found. If $i \neq 1$, consume the # preceding x_i. Read x_i placing a's and b's onto the stack. Consume the # following x_i, if $i \neq j$. Consume a's, b's and #'s until the beginning of x_j is non-deterministically found (note that if i=j, no elements are consumed here). Consume the # preceding x_j if $i \neq j$. Consume x_j, matching a's and b's. Consume the # following x_j if $i \neq k$, and consume the remaining a's, b's and #'s.



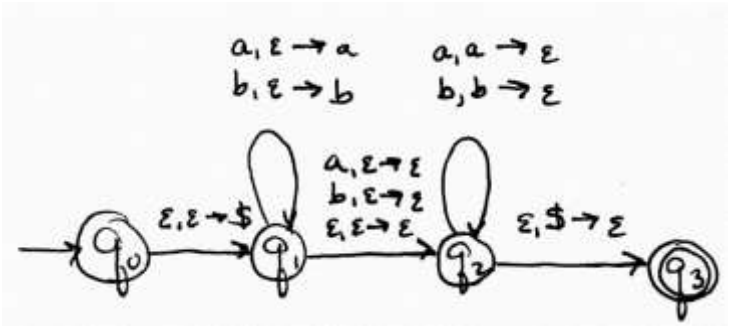
2. Create a pda for the language described in exercise 2.4 d. That is, for Σ={0,1}
{w | the length of w is odd and its middle symbol is a 0}

Plan:
Mark the bottom of the stack. Consume half of the string, pushing x's to count the symbols. Consume the middle '0'. Consume the last half of the string, popping x's. Accept if the bottom of the stack is found (and the string was consumed).



3. Define a pda for L={w | $w=w^R$ for $w \in$ {a,b} *} i.e. w is a palindrome

Mark the bottom of the stack. Consume a's and b's, pushing them onto the stack until the first half of the string is nondeterministically found. Possibly consume a middle a or b. Consume the last half of the string by popping matching symbol from the stack. When the end of stack is found, accept.



4. Let Σ ={a,b}. Consider the language Σ consisting of all strings with the same number of a's as b's. That is, L={w | $n_a(w)=n_b(w)$} (This is very different from a^nb^n, since in L the a's and b's can appear in any order.)

a. Create a PDA for L.

Plan for the PDA for L:

This machine can have three modes:

1. Same number of a's and b's.
2. More a's
3. More b's

The mode tells same, more a's or more b's. The stack records how many more, beyond the first extra.

Mark the bottom of the stack with $ and start in mode 1.

In mode 1.

- See '_', accept
- See 'a', go to mode 2.
- See 'b', go to mode 3.

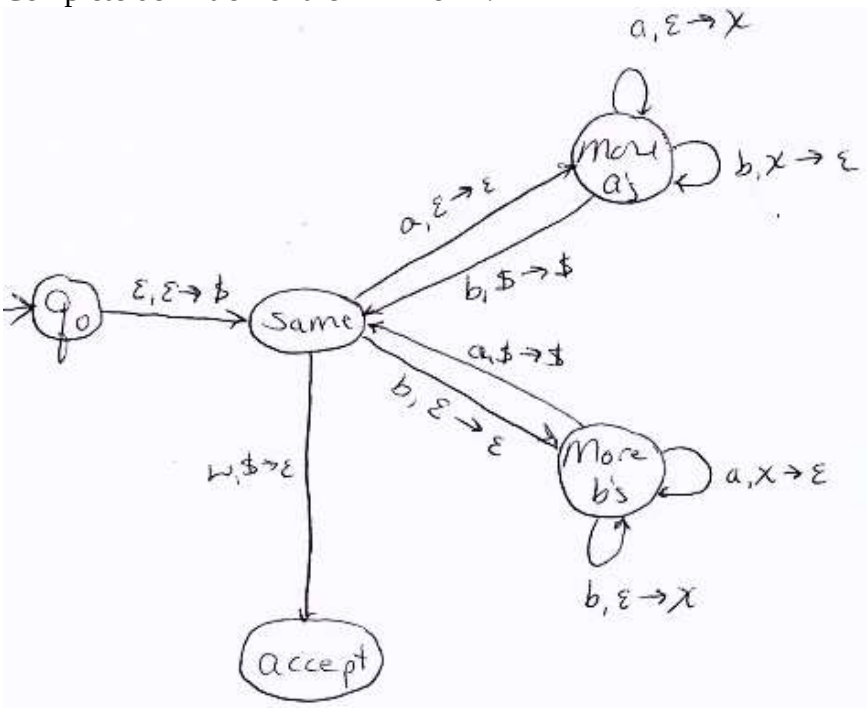
In mode 2.

- See '_', reject # More a's than b's.
- See 'a', push 'x' and stay in mode 2.
- See 'b', pop 'x' and stay in mode 2.
- See 'b', pop '$', same number a's and b's so replace $ and go to mode 1.

In mode 3.

- See '_', reject # More b's than a's.
- See 'a', pop 'x' and stay in mode 3.
- See 'a', pop '$', same number a's and b's so replace $ and go to mode 1.
- See 'b', push 'x' and stay in mode 3.

Complete definition of the PDA for L:



b. Create a grammar for the language.

S→ a S b | b S a | S S | ε