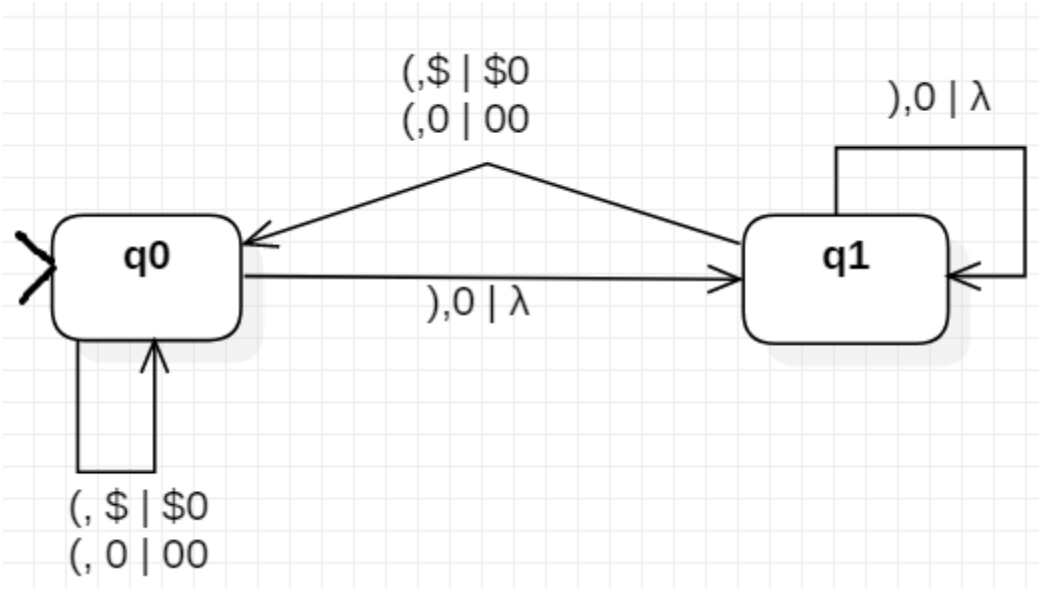
CS332 Mod 03 HW5

1. (15 pts) In the previous module you designed a language for properly nested parentheses. Design the PDA to recognize that language. Present the machine in both its graphical form and the 7-tuple, M = {Q, Σ, Γ, q_0, Z, F, δ }…



M = {Q, Σ, Γ, q_0, Z, F, δ }
Q = { q_0, q_1 }
Σ = { (,) }
Γ = { 0, 1, $}
q_0 = q_0
Z = $
F = Ø
δ =

	State Transitions						Pushed On Stack					
	(			)			(			)		
State	$	0	1	$	0	1	$	0	1	$	0	1
q0	q0	q0	R	R	q1	R	$0	00	N/A	N/A	λ	N/A
q1	q0	q0	R	R	q1	R	$0	00	N/A	N/A	λ	N/A
R = rejected string												

2. (10 pts) Provide the list of states and status of the stack for your machine when processing the string u = (( ) ( )). Is the string accepted or rejected, and why?

u = (() ( ))

Beginning State	Existing Stack Contents	Remaining String	Input Symbol	Popped From Stack	New State	Pushed to Stack	New Stack Contents	Current String Contents
q0	$	(( ) ( ))	(	$	q0	$0	0$	(
q0	0$	( ) ( ))	(	0	q0	00	00$	((
q0	00$	) ( ))	)	0	q1	λ	0$	(( )
q1	0$	( ))	(	0	q0	00	00$	(( ) (
q0	00$	))	)	0	q1	λ	0$	(( ) ( )
q1	0$	)	)	0	q1	λ	$	(( ) ( ))
q1	$ = ACCEPT	λ						

The string is accepted because string u is in the language defined by the machine. It allows all 'properly' nested parentheses, which means every open parentheses, (, must have exactly one corresponding close parentheses,), following it somewhere in the string; which is the case for this string.

3. (5 pts) It is a common exercise to design a PDA that recognizes palindromes of the form *ucv*, where 'c' is a single element in the alphabet and string *v* is the reverse of string *u*. For example, *abbcbba*, *aabbbacabbbaa*, and *aca* are all palindromes. It's fairly easy to do this. Suppose you were assigned the task of designing a PDA to recognize strings of the form *vcv*, where the string after the 'c' matches the string before the 'c.' For example, *abbcabb*, *aabbbacaabbba*, and *aca* are all of this form.
   State with justification whether designing the second PDA is easier, harder, or the same as designing the PDA that recognizes palindromes.

If I were to consider the problem without attempting to design the state machines, I would assume designing the first PDA (palindrome version) would be simpler as it is very similar in function to the parentheses PDA that we just designed in question 1. It would have one key difference from the parentheses PDA in that it would have a center part (c) that once inputted would require that no new patterns be inputted, and you just start popping from stack until the string is mirrored.

In summary, the main reason the palindrome PDA would be simpler would be the fact that it practically requires direct usage of the key trait of PDAs: last-in-first-out memory. It would require this trait since it requires the first symbol in the left string (first to be placed in memory and last to be popped out) to be the last symbol in the right string.