Partners: Michael Sanchez and Simrun Heir
I pledge my honor that I have abided by the Stevens Honor System.

Problem 1 A 2-stack PDA (2PDA for short) is a pushdown automaton with two stacks. In one step each stack can be popped or pushed independently. For example, the 2PDA may read an input symbol, pop the first stack, ignore the second stack and, based on the values seen, change state, push different symbols on the two stacks. The input tape is read only. Describe a 2-stack PDA that accepts the language L = {a^i b^k c^i d^k : i, k ≥ 0}. You do not need to give a PDA diagram – a high-level, but complete, description of how your PDA works will suffice.

Answer:
In each stack, the first thing to push will be a $ so we can tell if we are at the end of the stack. In the first stack, for every a we see in the input, we will push an a to stack 1, and then for every b we see in the input, we will push a b to stack 2. Next, for every c we see, we pop stack 1, and if stack 1 pops a $ and there are still c's in the input, we know we have exited the language and reject the input. If the stack still has a's, but the input does not read a c, we exit the language, so again, we reject. Similarly for stack 2, we pop the stack for every d we read and if stack 2 pops a $ and there are still d's in the input, we reject, or if the stack contains more b's but reaches the end of the input, we reject.

Problem 2 In Problem Set 6 you showed that the language L_{mult} = {$a^i b^{ij} c^j$: i,j ≥ 0} is not context free. Either show that this language can be recognized by a 2PDA or explain why that is not possible.

Note:
2PDA simulates a Turing machine

- Transition function for 2PDA (with no diagram)
    - currentState, input,(stack 1, stack 2)-->newState, (stack 1, stack 2)

Construct turing machine for language
Write proof that states 2PDA simulates action of turing machine
Write formal definition construction of both PDA and Turing Machine then write out in english a description of how it works

Answer:
Let P be a 2-PDA = (Q, Σ, Γ, δ, q_0, F)
Let T be a Turing Machine = (Q, Σ, Γ, δ, q0, q_{accept}, q_{reject})
According to the definition of a Turing machine:
1. A Turing machine can both write on the tape and read from it.
2. The read–write head can move both to the left and to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately

By pushing the entire input onto stack 1, we can emulate moving to the end of the input tape on a Turing Machine. To emulate moving to the left, we pop from stack 1 and push that symbol onto stack 2, and to emulate moving right we do the opposite. To emulate reading the

location under the tape head, we can read the symbol at the top of stack 2. To emulate writing the location under the tape head, we pop one symbol from stack 2 and push a new symbol onto stack 2.
By this logic, a 2PDA is Turing complete, and has the capabilities of simulating a Turing machine.
Let the turing machine T represent the 2PDA for L_{mult}
For turing machine T,

1. Cross off one a
2. Then as many b's as there are c's.
3. Uncross all c's
4. Repeat until all a's are crossed off
5. If all c's are crossed and there are any b's uncrossed, or all b's are crossed and there are any c's uncrossed, reject
6. Otherwise, accept

Therefore, L_{mult} can be recognized by T, and by extension can be recognized by P.

Problem 3 Prove that the intersection of a CFL and a Regular language is always context free.

Notes:

- Regular languages are closed under $\cap$, $\cup$, complement, concatenation, and *
- Corollary 2.23: every regular language is context-free
    - All finite languages are regular, therefore all finite languages are context free
    - Not every CFL is regular
- Thm 2.20: a language is context free iff some PDA recognizes it
- Thm 2.9: any CFL is generated by a CFG in CNF
- language of the grammar is $\{w \in \Sigma* | S * \Rightarrow w\}$

Language of intersection is cross product of L and Rs languages
Construct a PDA that reads that language

Answer:
Let R be a regular language, and let M be the DFA that accepts R.
Formally M = (Q1, Σ, δ1, q1, F1) be a DFA such that its language is R
Let L be a context-free language, and let P be the PDA that accepts L.
Formally P = (Q2, Σ, Γ, δ2, q2, F2) be a PDA such that its language is L
Any strings accepted by both P and M make the language of the intersection.
This can be represented by a PDA Y that models M $\times$ P
This can be modeled by a PDA Y where Y = (Q, Σ, Γ, δ, q0, F)

- Q = Q1 $\times$ Q2
- Γ is the stack alphabet
- q0 = (q1,q2)
- F = F1 $\times$ F2
- $\delta((p,q),x,a) = \{ ((p',q'),b) : p' = \delta1(p,x) \text{ and } (q',b) \in \delta2(q,x,a) \}$

    - Note that the symbol that is popped, as well as pushed, is dependent on the transition function for P

Y accepts strings that are accepted by both P and M.
Since Y is a PDA, the language it recognizes is context free, meaning, since the intersection of a regular language and a CFL can be modeled by a PDA that represents M $\times$ P, therefore the language is context-free.
It cannot be explicitly said that the intersection of a CFL and a regular language is always regular because not every CFL is regular.

Optional Problem 4 Prove that the language A\B = {w: wx $\in$ A, x $\in$ B}, where A is a CFL and B is regular is a CFL

Notes:
Series of symbols of in A then series of symbols in B
Non-deteministic PDA, guess where halfway point is

Answer:
The language of A\B produces a string that contains a number of symbols from language A followed by a number of symbols from language B
The language of A\B recognizes strings that are in both A and B, with the form of strings from A then strings from B, meaning A\B is the concatenation of the language A and B.
This can be simulated by a PDA P where P = (Q, Σ, Γ, δ, q0, F).

- Q is the set of states,
    - Let there be one distinct start state, and accept state
    - Let there be one state, qA, that represents the strings that can be produced from language A
    - Let there be one state, qB that represents the strings that can be produced from language B
- Σ = union of the alphabets of A and B,
- Γ is the stack alphabet,
- $\delta : Q \times \Sigma\varepsilon \times \Gamma\varepsilon \rightarrow P(Q \times \Gamma\varepsilon)$ is the transition function,
- q0 $\in$ Q is the start state
- F is a single accept state, and is a member of Q.

Description of PDA P

- first P doesn't read the input or pop from the stack and pushing $ on the stack, and transitions from q0 to qA
- While in qA two things can occur
    - If the symbol that is read from the tape belongs to the alphabet A, w, push the symbol 'a' on to the stack without popping anything from the stack and transition to qA
    - If the string is in the middle, the portion of the string where it switches from being in alphabet A to being in alphabet B, x, don't read the tape, or pop/push anything from the stack or onto the stack, and transition to state qB
- While in qB two things can occur

- If the symbol that is read from the tape belongs to alphabet B, pop ‘a’ from the stack don’t push anything onto the stack and transition back to state qB
- If the only symbol remaining in the stack is $, do not read from the input, pop $, do not push anything onto the stack, and transition to the accept state
    - If the string is done being read at this point it is accepted, if not then reject

Since the language A\B can be described by the PDA P, the language is context-free

Optional Problem 5 A queue automaton is like a push-down automaton except that the stack is replaced by a queue. A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we’ll call it a push) adds a symbol to the left-hand end of the queue and each read operation (we’ll call it a pull) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at any time. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable.

Note:
Show every part of a queue automaton can be simulated by a Turing machine and vice-versa
Queue automaton can only access the oldest input element

- Read everything then push on blank symbol
- Then go series of pulling to check input

Turing-recogonizable

- Accepts in finite time

Turing-decidable

- Accepts and rejects in finite time

Pushing of queue automaton is going down input tape

Answer:
Let T be a Turing Machine where T = $(Q, \Sigma, \Gamma, \delta, q0, q_{accept}, q_{reject})$

Defining a Queue Automaton Formally:
A queue automaton is a six-tuple.
Let queue automaton M = $(Q, \Sigma, \Gamma, \$, s, \delta)$ where

- Q is a finite set of states
- $\Sigma \subset \Gamma$ is the finite set of the input alphabet
- Γ is the finite queue alphabet
- $\$ \in \Gamma - \Sigma$ is the initial queue symbol
- $s \in Q$ is the start state
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow P(Q \times (\Gamma \cup \{\varepsilon\}))$ is the transition function

The configuration of of M is an ordered pair of its state and queue contents $(q,k) \in Q \times \Gamma^*$

- Note that Γ^* is the Kleene closure of Γ

The starting configuration of an input string w is defined as (s,w$).
The transition function is defined by $(p,Aa) \rightarrow (1,M)\ (q,ak)$

- a is a sequence of queue symbols ($a \in \Gamma^*$)
- $(q,k) = \delta(p, A)$

M accepts a string $w \in \Sigma^*$ if after a finite number of transitions the starting configuration changes and becomes the null string, ε, or $(s,w\$) \rightarrow (*,M)\ (q,\varepsilon)$.

M is capable of simulating T:

- Note that in order to simulate T, the queue of M must always contain a copy of T's contents
    - There is a specific marker in the queue for the head position of T
        - This is done by using a '$' to mark the beginning of the tape
        - At all time the queue will contain symbols from the queue alphabet and a '$'
    - The queue will follow some form
        - xy$z
            - $x \in \Sigma$
                - x is the symbol currently representing the head
            - $y \in \Sigma^*$
                - Contents of tape to the right of the head
            - $z \in \Sigma^*$
                - Contents of tape to the left of the head
                    - The tail of the queue

T is capable of simulating M:

- Let T be a multiple tape turing machine
    - Note that a multiple tape turing machine is equivalent to a single-tape turing machine
- One tape is used to read the input tape
- One tape is used to store the queue