Haris Spahic [illegible]

"I pledge my honor I have abided by the Stevens Honor System"

CS 334 ~ PS 6

 # 1. We design our 2 stack PDA to do the following.

<u>Step 1</u>: push \$ to both stacks

<u>Step 2</u>: while (!#):
  read & push character to stack 1

<u>Step 3</u>: @ #, don't push & move to step 4

<u>Step 4</u>: while (pop(stack 1) != \$):
  don't read
  push the popped symbol to stack 2

<u>Step 5</u>: When \$ popped, start reading each symbol again, popping from stack 2 as you go.

If (pop != read):
  <u>reject.</u> ✗
else if (pop(\$)):
  <u>accept.</u> ✓
else:
  repeat reading & popping

#1b. Exact same description as 1a, except instead of checking for # each time @ step 2; non-deterministically transition to step 4 after each read & push.

#1c. We construct a 2PDA for each $n \in \mathbb{Z}^+$, let this 2PDA be A_n where A_n accepts a^{n^2}.

We construct A_n as follows.

Step 1: Push \$ to both stacks.

Step 2: Read & push an a on stack 1, then non-deterministically (ε transition) go to step 3

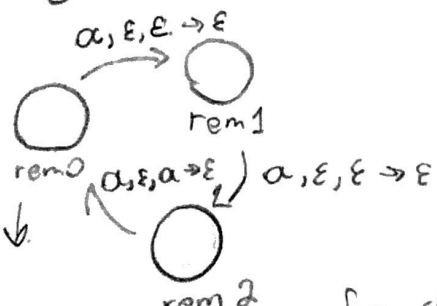
Step 3: Pop an a off stack 1. Do nothing with it.

Step 4: Pop a's off stack 1, and push them to stack 2 until you pop the \$.

Step 5: At this point, we have n-1 a's in stack 2. ← And n-1 a's is in our language

From this "state" we have n states each taking a read, representing each possible remainder of n & following a cycle back to our state of rem 0.

Ex: A_3



If we go from rem n-1 → rem0 pop from stack 2.

← from stack 2

If we pop a \$ in rem0, we accept. Otherwise reject. This will only accept when we pop n-1 multiples of a, in other words our remaining $(a \times n-1)$ symbols.

Our final 2PDA will take the union of each A_n, or $L = \bigcup_{n=1}^{\infty} A_n$, and since PDA's are closed under union L will accept $\{a^{\lambda^2} : \lambda \geq 1\}$.

(The union just nondeterministically goes to each of those PDA's, and accepts if one of them (the one for λ) accepts)

1do Key is 2 stacks of PDA2 can be used to hold all previous terminals & all terminals to the right of current we've already seen.

If we can show a PDA2 can simulate each state & transition of a TM, then it will accept the same language.

We do this as follows:

1) Start by pushing $ to left stack. This will be used to represent the first left space.
2) Have the 2PDA push every element to the left stack, Once it reaches the last symbol push the $ to the right stack.

At this point our 2PDA has all its symbols in the left stack, plus $ to represent spaces in each. Pop all the elements in the left stack & push them to the right. Now pretend the stacks are the tape of the TM. The top of each stack represent the elements left & right of the tape head respectively. As of now, our stacks are oriented, to represent the TM head at its start.

We can represent the state of our turing machine as

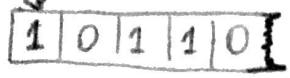
$uaq_i bv$, where a, b are the top elements of left & right stack respectfully, while u & v are the rest of the strings in both stacks respectfully.

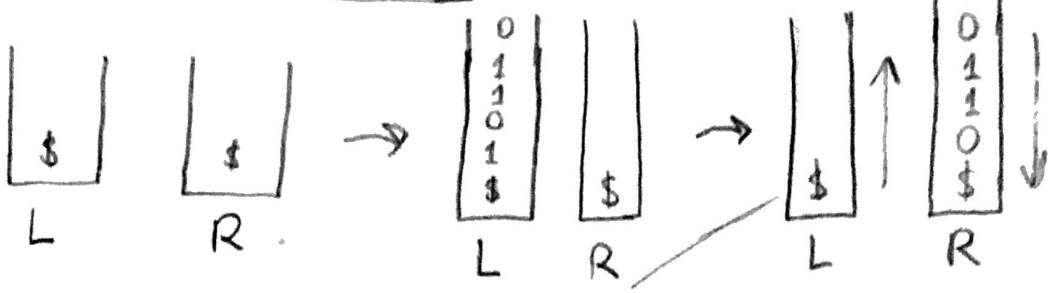
When the TM takes any left transition, pop the top element of the left stack, and push the newly overwritten value. Unless it's $, then pop the right stack & replace it with the newly written value, pushing the $ back onto the left stack.

Same idea just reversed for right transition.

I know this is formal & complicated, here's an example, →

Start:

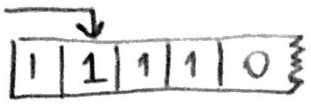




String: $q 10110$

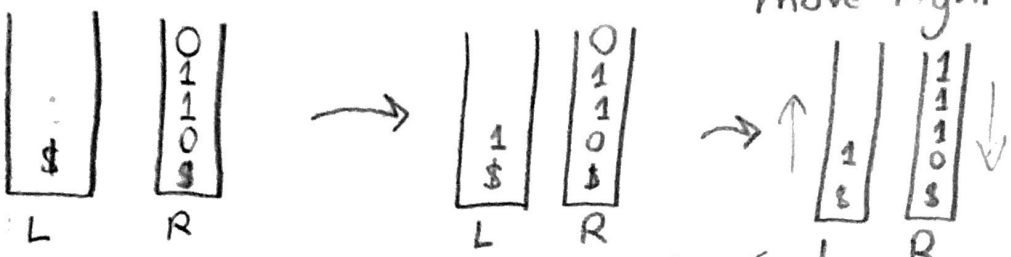
$ is space / end

Move right:



write = 1
read = 0
move right

String $1 q 1110$

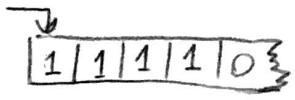


(pops from right, PDA can see popped symbol)

(push to left)

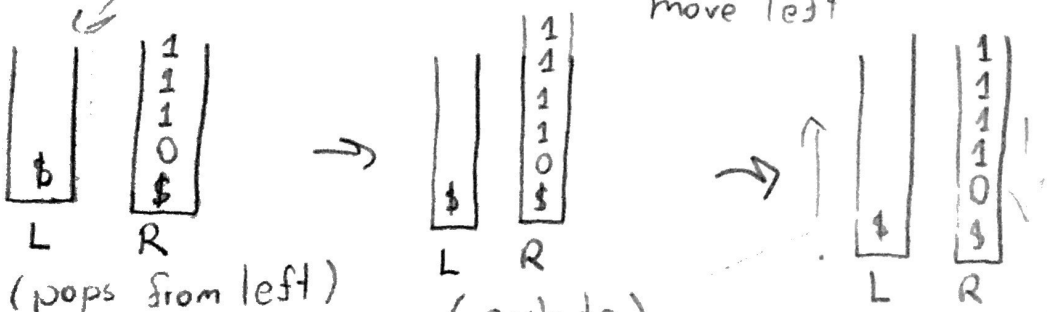
(Pop & push new symbol to R)

Move left:



write=1
read=1
move left

String: $q 1110$



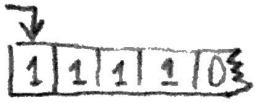
(pops from left)

(push to right)

(Pop $, push new symbol to R)

Same for space just in reverse

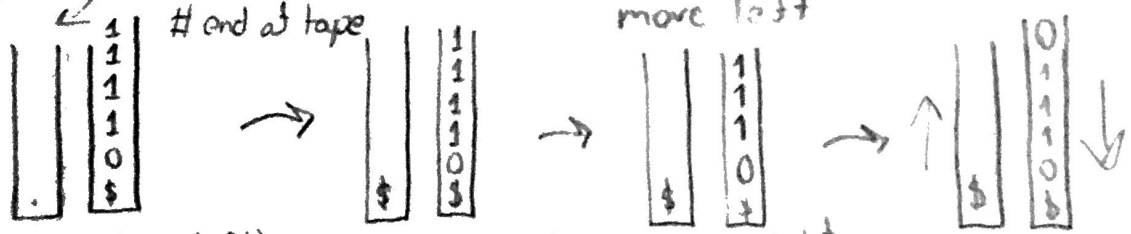
Move left:



end of tape

write=0
read=1
move left

String: $q0111$



(pops from left) reads $

(pushes $ back left)

(pops right, new read)

(push new symbol right)

#2) Every regular language can be represented by a FSA which accepts it. Every CFL is accepted by a PDA. The intersection of a regular language & a CFL accepts when <u>both</u> the regular language & CFL is accepted.

Show that intersection is also a CFL.

Proof by construction:

We make a PDA to accept the intersection of both languages. We'll do this in a similar fashion as converting NFA's to DFA's. We define $(Q, \Sigma, T, \delta, q_0, F)$ as follows:

Let R be the nfa of our regular language & C the PDA of our CFL.

1. $Q = \{Q_C \times Q_R\}$ or a combination of each of C's & R's states.
2. $\Sigma = \Sigma_C \cup \Sigma_R$
3. $T = T_C$
4. We define δ as follows, let q_c be the current Q_C state in Q. Likewise let q_r be the current Q_R in Q. So $Q = \{q_c \times q_r\}$

Then, $\delta(Q, \Sigma, T_\varepsilon) = \begin{cases} (q_c' \times q_r') \text{ if both 1 \& 2 true.} \\ 1) (q_c \times q_r) \text{ if } \delta_C(q_c, a, \varepsilon) \to \{q_c', \varepsilon\} \\ 2) (q_c \times q_r') \text{ if } \delta_R(q_R, b) \to \{q_R'\} \\ 3) (q_c' \times q_r) \text{ if } \delta_C(q_c, a, d) \to \{q_c', T\} \\ (q_c \times q_r) \text{ otherwise} \end{cases}$

with $q_c' \in Q_C$, $q_R' \in Q_R$, $a \in \Sigma_C \cup \varepsilon$, $b \in \Sigma_R \cup \varepsilon$, $d \in T$.

That is if a transition input takes C to a new state, our PDA transitions to the cross with that new state. If transition input doesn't pop, & takes R to a new state, the PDA transitions to the cross with the new state. And If both C & R transition, we transition to the state with both new states. Else C & R go nowhere on input, & our PDA stays in the same state.

5. $q_0 = \{ q_{C_0} \times q_{R_0} \}$

6. $F = \{ q_{C_{accept}} \times q_{R_{accept}} \}$ → So only accept if both accept.

Thus we created a PDA for the intersection of both a regular & Context Free Language → That language must be CFL. ■

3.

#3) Show TM-decidable languages are closed under:

i) Union: Given a string $w \in L_1 \cup L_2$, we run w on two tapes with TM_1 on the first & TM_2 on the second. If either accepts, our new TM accepts. Else we reject. Since a 2-tape TM can be converted into a one-tape TM this is valid.

Test TM_1 & TM_2 decide L_1 & L_2 respectively

ii) Concatenation: Suppose we have string $a \in L_1$, & $b \in L_2$ where L_1 & L_2 are decided by TM_1 & TM_2.

→ Construct new TM to accept ab.

We run a on TM_1, then nondeterministically guess when we reach b. At b, we run TM_2. If it accepts we accept, else reject.

Check if A is accepted, then split. If not reject.

iii) Star: Suppose we have $a \in L_1^*$, where TM_1 decides just L_1. We construct a TM to accept L_1^* as follows.

If the string is empty, accept. (start reading ⊔)

Else, run TM_1 on a. If it accepts, nondeterministically guess where the next iteration of L_1 will be. Run TM_1 again on the new iteration of L_1. Repeat this until end. If TM accepts all strings we partition, accept.

iv) Intersection: Same as union, but only ~~accept~~ decides if both TM_1 & TM_2 accept.

U. Compliment: Suppose $w \in L^c$ s.t L is decided by TM_1. Run TM_1 on w. If TM_1 accepts, reject & vice versa. Since L_1 is decidable, we know TM_1 will reach one of these 2 states.

3b. Show Turing Recognizable Languages are closed under

Union: Suppose we have L_1 & L_2 which are recognized by Tm_1 & Tm_2 respectively. Let $w \in L_1 \cup L_2$. Construct a new Tm to recognize w.

→ Run Tm_1 & Tm_2 on a 2 tape Tm, s.t each tape holds a copy of w. If either accept, then accept. If both machines halt & reject, then reject.

catenation: Suppose we have L_1 & L_2 ... Let $a \in L_1$ & $b \in L_2$. Show $ab \in L$ is Turing recognizable. Construct TM as follows-

→ Run TM_1 on ab. If TM_1 accepts, we nondeterministically guess the location of b & run TM_2. If TM_1 or TM_2 halts and rejects, reject. If TM_1 & TM_2 accept, then accept.

Star: Suppose we have $a \in L^*$, where TM_1 recognizes L. Show L^* is Turing recognizable.

→ If tape is empty, accept.
Else, run TM_1 on a . If it accepts, non-deterministically guess the next iteration of L. Repeat until string a ends ($\sqcup$). If every partition of a is accepted, accept. Else reject.

Intersection: Same as unions just if both accept then accept.

Compliment: Not closed. Proof by contradiction.

Suppose we have a language L that is recognized by TM_1. Assume L^c is also recognizable. Then L^c must be recognized by some TM_2, call it TM_2. We ~~show the implies L^c must be decidable~~ Run TM_1 & TM_2 separately on 2 tapes, reading some input string w. We alternate reads between each Turing Machine.

→ If $w \in L$, TM_1 will accept w in some n moves. Our 2 tape TM then accepts w in 2n moves.

→ If $w \notin L$, then TM_2 will accept w in some m moves. Our 2 tape TM then rejects w in some finite moves 2m.

However, our 2 tape TM now is able to decide the language of L, which is only Turing Recognizable. This is a contradiction.

Thus Turing recognizable languages are not closed under compliment.