CS 181 Midterm

Sriram Balachandran

TOTAL POINTS

264 / 302

QUESTION 1

Question 1: FFAs 130 pts

1.1 1a 25 / 30

✓ **- 5 pts** **No explanation**

We required at least a short explanation of intuition (1-2 sentences). Your machine does correctly decide the language.

1.2 1b 20 / 20

✓ **- 0 pts** **Correct**

1.3 1c 49 / 50

✓ **- 1 pts** **FFAs are nondeterministic, so there may be multiple computational paths. You want to consider the execution on some accepting path, not "the" path.**

Swapping fetch transitions with normal transitions produces an NFA, not a DFA.

1.4 1d 25 / 30

✓ **- 5 pts** **|ab| could be less than the pumping length n. We don't necessarily have |ab| = n.**

QUESTION 2

Question 2: TFAs 120 pts

2.1 2a 20 / 20

✓ **- 0 pts** **Correct**

2.2 2b 20 / 20

✓ **- 0 pts** **Correct**

2.3 2c 30 / 30

✓ **- 0 pts** **Correct**

2.4 2d 30 / 50

Construction Errors

✓ **- 10 pts** **Accepting states need to have an additional transition to consume the final #.**

✓ **- 10 pts** **You cannot allow a throw on all symbols on the first loop in the transformation because this creates computation paths where you've fetched the next-next or in general the $$i$$th occurrence of your fetched symbol.**

QUESTION 3

3 Extra Credit: FPDA 45 / 50

+ 50 pts Correct

- 10 pts No explanation

+ 50 pts Correct, except that your machine doesn't accept the empty string.

✓ **+ 45 pts** **Partial Credit: Correct, except that your machine does not accept strings with an equal number of 0's and 1's in the first half. Recall that 0 is the tie-breaker for majority.**

+ 45 pts Partial Credit: Correct, except that if there are an equal number of 0's and 1's in the first half, your machine can accept if the second half consists of entirely 1's. Recall that 0 is the tie-breaker for majority.

+ 10 pts Partial credit: concrete construction reflects a high level idea with potential.

+ 10 pts Partial credit: Correct intuition, but no concrete construction

+ 0 pts No attempt/I don't know/vague idea only

+ 0 pts Incorrect

QUESTION 4

4 Extra Credit: Automadoggos 0 / 2

✓ **- 2 pts** **No Submission**

QUESTION 5

5 Honor Code Agreement 0 / 0

✓ **- 0 pts** **Correct**

Playing fetch with your pet automadog.[1]

Problem 1. A *Fetch Finite Automata (FFA)* is exactly like an NFA, except that:

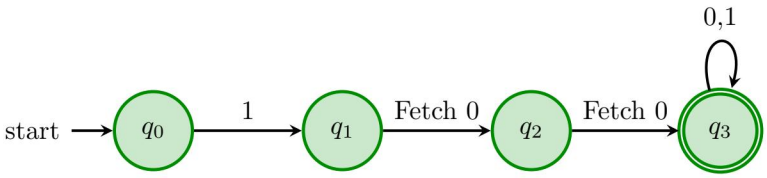
1. We do not allow any epsilon transitions. (All transitions must consume input.)
2. There exist special Fetch transitions.

On a fetch transition Fetch σ for $\sigma \in \Sigma$, your pet automadog goes and fetches the next occurrence of σ in the input string and brings it back to you. You will then consume the fetched σ to complete the Fetch transition.

Formally, a Fetch σ transition works as follows: If the symbol σ exists anywhere in the remaining input, then the transition may be taken. Otherwise, it cannot be taken. If the transition is taken, then the next occurrence of σ in the remaining input string is consumed and you move to the specified state. The transition function is $\delta : Q \times (\Sigma \cup (\{\mathsf{Fetch}\} \times \Sigma)) \to \mathcal{P}(Q)$. We say that a language L is *Fetching* if there exists an FFA F such that $L(F) = L$.

Example: The following FFA decides language

$$L = \{x \in \{0,1\}^* \mid x \text{ starts with a 1 and contains at least two 0's.}\}$$

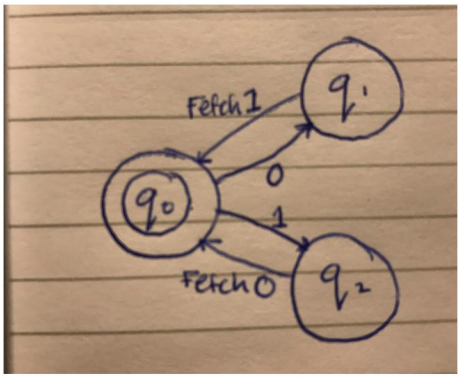


a) **(30 points).** Prove that

$$L_{EQ} = \{x \in \{0,1\}^* \mid x \text{ has an equal number of 0's and 1's}\}$$

is *Fetching*. (i.e. Prove that there exists an FFA that decides L_{EQ}.)

Answer:



[1] Automadogs are dogs that also like playing with finite automata.

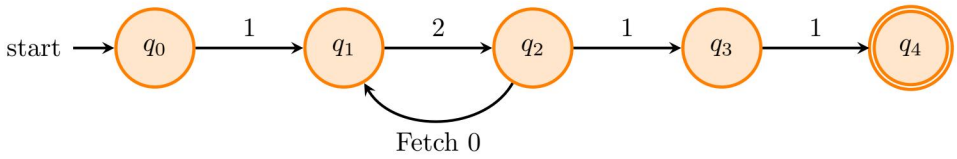
1.1 1a **25 / 30**

✓ **- 5 pts** **No explanation**

We required at least a short explanation of intuition (1-2 sentences). Your machine does correctly decide the language.

Definition 1 (Permutation)**.** *A permutation of the string x is a string formed by reordering the elements of x. For example, if $x = 0012$, then 1020 is a permutation of x. The set S of permutations of $x = 0012$ is $S = \{0012, 0021, 0102, 0201, 1002, 2001, 0120, 0210, 1020, 2010, 1200, 2100\}$.*

b) **(20 points).** Consider the following FFA F:



String $x = 122211000$ is in $L(F)$. Write three distinct permutations of x (other than x itself) that are also in $L(F)$.

Answer:

1202020211,1222000211,1220200211

c) **(50 points).** Prove the following pumping lemma about *Fetching* languages:

Lemma 1. *For all* Fetching *languages L, there exists an $n \in \mathbb{N}$ such that for all $x \in L$ with $|x| \geq n$, there exist strings a, b, c such that $w = abc \in L$ where w is a permutation of x and*

i. $|ab| \leq n$

ii. $|b| > 0$

iii. $\forall i \geq 0, ab^i c \in L$

Hint: Part (1b) should help you develop intuition for this.

Answer:

Let us consider the fetching language L, and some $w \in L$. Let F be the FFA that decides L. A fetch σ transition in F will always fetch the first occurrence of σ from the remaining input. Let us then permute w such that whenever F has a fetch σ transition, σ is the next character in the input. In other words, our FFA will always only have to look at the first character of the remaining input whenever it executes a fetch transition. Since in such a case the fetch operation would always be processing the next letter in the input, the 'fetch' operation is effectively redundant - it can be rewritten as a standard transition. If we apply this transformation to all fetch transitions in F, since F has no ϵ transitions, F becomes a DFA that we shall label as D.

For all $w \in L$, it is trivial to see there exists a permutation such that the fetch transition will simply fetch the next character. Let us label the language of fetch-eliminating permutations of words in L as $L_{permute}$. D is then the DFA that decides $L_{permute}$. Since we are able to create a DFA that decides $L_{permute}$, then $L_{permute}$ must be regular. Now let us prove the pumping lemma for the regular language $L_{permute}$ that can be formed for any Fetching language L.

(Proof of pumping lemma for regular language cited from textbook)

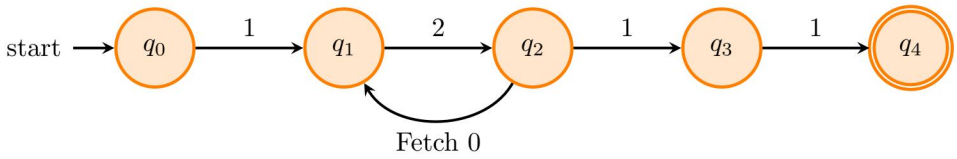
Let p be the number of states in D. Let $s = s_1 s_2 ... s_n$ be a string in $L_{permute}$ of length n, where $n >= p$. Let $r1, ..., r_{n+1}$ be the sequence of states that D enters while processing s, so $r_{i+1} = \delta(r_i, s_i)$ for $1 <= i <= n$. This sequence has length $n + 1$, which at least $p + 1$. Among the first $p + 1$ elements in the sequence, two must be the same state, by the pigeonhole principle. We call the first of these r_j and the second r_l. Because r_l occurs among the first $p + 1$ places in a sequence starting at r_1, we have $l <= p + 1$. Now let $a = s_1 ... s_{j-1}, b = s_j ... s_{l-1}$ and $c = s_l ... s_n$.

1.2 1b 20 / 20

✓ - 0 pts Correct

Definition 1 (Permutation)**.** *A permutation of the string x is a string formed by reordering the elements of x. For example, if $x = 0012$, then 1020 is a permutation of x. The set S of permutations of $x = 0012$ is $S = \{0012, 0021, 0102, 0201, 1002, 2001, 0120, 0210, 1020, 2010, 1200, 2100\}$.*

b) **(20 points).** Consider the following FFA F:



String $x = 122211000$ is in $L(F)$. Write three distinct permutations of x (other than x itself) that are also in $L(F)$.

Answer:

1202020211,1222000211,1220200211

c) **(50 points).** Prove the following pumping lemma about *Fetching* languages:

Lemma 1. *For all* Fetching *languages L, there exists an $n \in \mathbb{N}$ such that for all $x \in L$ with $|x| \geq n$, there exist strings a, b, c such that $w = abc \in L$ where w is a permutation of x and*

i. $|ab| \leq n$

ii. $|b| > 0$

iii. $\forall i \geq 0, ab^i c \in L$

Hint: Part (1b) should help you develop intuition for this.

Answer:

Let us consider the fetching language L, and some $w \in L$. Let F be the FFA that decides L. A fetch σ transition in F will always fetch the first occurrence of σ from the remaining input. Let us then permute w such that whenever F has a fetch σ transition, σ is the next character in the input. In other words, our FFA will always only have to look at the first character of the remaining input whenever it executes a fetch transition. Since in such a case the fetch operation would always be processing the next letter in the input, the 'fetch' operation is effectively redundant - it can be rewritten as a standard transition. If we apply this transformation to all fetch transitions in F, since F has no ϵ transitions, F becomes a DFA that we shall label as D.

For all $w \in L$, it is trivial to see there exists a permutation such that the fetch transition will simply fetch the next character. Let us label the language of fetch-eliminating permutations of words in L as $L_{permute}$. D is then the DFA that decides $L_{permute}$. Since we are able to create a DFA that decides $L_{permute}$, then $L_{permute}$ must be regular. Now let us prove the pumping lemma for the regular language $L_{permute}$ that can be formed for any Fetching language L.

(Proof of pumping lemma for regular language cited from textbook)

Let p be the number of states in D. Let $s = s_1 s_2 ... s_n$ be a string in $L_{permute}$ of length n, where $n >= p$. Let $r1, ..., r_{n+1}$ be the sequence of states that D enters while processing s, so $r_{i+1} = \delta(r_i, s_i)$ for $1 <= i <= n$. This sequence has length $n + 1$, which at least $p + 1$. Among the first $p + 1$ elements in the sequence, two must be the same state, by the pigeonhole principle. We call the first of these r_j and the second r_l. Because r_l occurs among the first $p + 1$ places in a sequence starting at r_1, we have $l <= p + 1$. Now let $a = s_1 ... s_{j-1}, b = s_j ... s_{l-1}$ and $c = s_l ... s_n$.

As a takes D from r_1 to r_j, b takes D from r_j to r_j, and c takes D from r_j to r_{n+1}, which is an accept state, D must accepts ab^ic for $i >= 0$. We know that $j! = l$, so $|y| > 0$; and $l <= p + 1$, so $|xy| <= p$. Thus we have satisfied all the conditions of Lemma 1, since a is the permutation of some $w \in L$.

d) **(30 points).** Use **Lemma 1** from (1c) to prove that

$$L_{count} = \{0^n 1^n \mid n \geq 0\}$$

is not *Fetching*.

Answer:

Let us consider $w \in L_{count}$, of form $0^n 1^n$. The only permutations of w that will remain in L_{count} are permutations of the form $0^n 1^n$. So let us apply Lemma 1 to L_{count}, letting $w_{lemma} = w$. There are 3 possible partitions of w in the form ab^ic.

(a)

$$b = 0^r | r <= n \tag{1}$$

$$a = 0^{n-r}, c = 1^n \tag{2}$$

It is trivial to see that if i is set to an arbitrarily high number, then the word will not be of form $0^n 1^n$ anymore, meaning it will not be in the language and thus cause a contradiction.

(b)

$$b = 1^r | r <= n \tag{3}$$

$$a = 0^n, c = 1^{n-r} \tag{4}$$

It is trivial to see that if i is set to an arbitrarily high number, then the word will not be of form $0^n 1^n$ anymore, meaning it will not be in the language and thus cause a contradiction.

(c)

$$b = 0^q 1^r | r <= n, q <= n \tag{5}$$

$$a = 0^{p-q}, c = 1^{p-r} \tag{6}$$

It is trivial to see that if i is greater than 1, then the word will not be of form $0^n 1^n$ anymore, meaning it will not be in the language and thus cause a contradiction.

Since there is no possible partition of the form ab^ic that will allow for $w \in L_{count}$ to remain in L_{count}, then by the Lemma 1, L_{count} is not fetching.

1.3 1c 49 / 50

✓ **- 1 pts FFAs are nondeterministic, so there may be multiple computational paths. You want to consider the execution on some accepting path, not "the" path.**

Swapping fetch transitions with normal transitions produces an NFA, not a DFA.

As a takes D from r_1 to r_j, b takes D from r_j to r_j, and c takes D from r_j to r_{n+1}, which is an accept state, D must accepts ab^ic for $i >= 0$. We know that $j! = l$, so $|y| > 0$; and $l <= p + 1$, so $|xy| <= p$. Thus we have satisfied all the conditions of Lemma 1, since a is the permutation of some $w \in L$.

d) **(30 points).** Use **Lemma 1** from (1c) to prove that

$$L_{count} = \{0^n 1^n \mid n \geq 0\}$$

is not *Fetching*.

Answer:

Let us consider $w \in L_{count}$, of form $0^n 1^n$. The only permutations of w that will remain in L_{count} are permutations of the form $0^n 1^n$. So let us apply Lemma 1 to L_{count}, letting $w_{lemma} = w$. There are 3 possible partitions of w in the form ab^ic.

(a)

$$b = 0^r | r <= n \tag{1}$$

$$a = 0^{n-r}, c = 1^n \tag{2}$$

It is trivial to see that if i is set to an arbitrarily high number, then the word will not be of form $0^n 1^n$ anymore, meaning it will not be in the language and thus cause a contradiction.

(b)

$$b = 1^r | r <= n \tag{3}$$

$$a = 0^n, c = 1^{n-r} \tag{4}$$

It is trivial to see that if i is set to an arbitrarily high number, then the word will not be of form $0^n 1^n$ anymore, meaning it will not be in the language and thus cause a contradiction.

(c)

$$b = 0^q 1^r | r <= n, q <= n \tag{5}$$

$$a = 0^{p-q}, c = 1^{p-r} \tag{6}$$

It is trivial to see that if i is greater than 1, then the word will not be of form $0^n 1^n$ anymore, meaning it will not be in the language and thus cause a contradiction.

Since there is no possible partition of the form ab^ic that will allow for $w \in L_{count}$ to remain in L_{count}, then by the Lemma 1, L_{count} is not fetching.

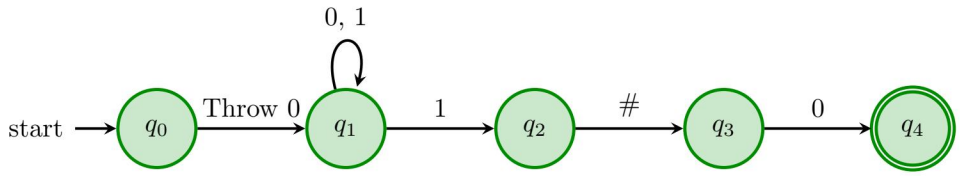
1.4 1d **25 / 30**

✓ **- 5 pts** |ab| could be less than the pumping length n. We don't necessarily have |ab| = n.

Example: The following TFA T decides language

$$L = \{x \in \{0,1\}^* \mid x \text{ starts with a 0 and ends with a 1}\}$$

This is because if x starts with a 0 and ends with a 1, then when processing x, T will first start by appending # to the end of x to get $x\#$. Then, T will throw the initial 0 to the end of the string after the #.

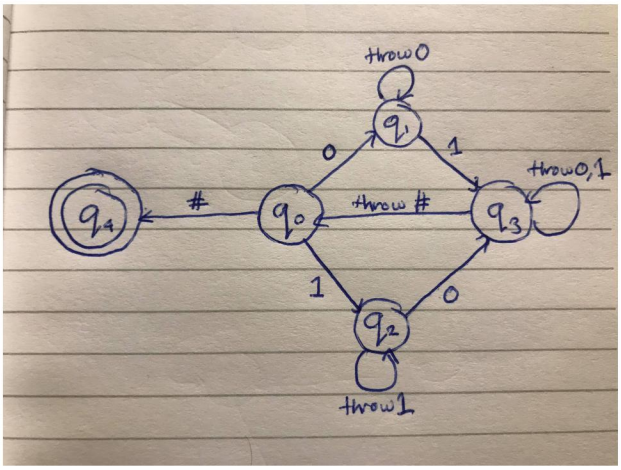


a) **(20 points).** Draw a TFA that decides

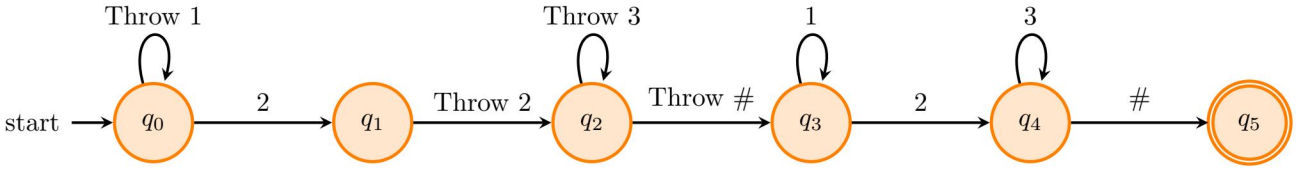
$$L_{EQ} = \{x \in \{0,1\}^* \mid x \text{ has an equal number of 0's and 1's}\}$$

where $\Sigma = \{0,1\}$.

Answer:



b) **(20 points).** Consider the following TFA T over $\Sigma = \{1,2,3\}$:



What is $L(T)$?

Answer:

$L(T) = 1 * 223*$. The shortest word in $L(T)$ is 22, if we avoid all loopback transitions. If we account for the loopback transitions, we can first have an arbitrary number of 1s before the 22, and then have an arbitrary number of 3s after the 22.

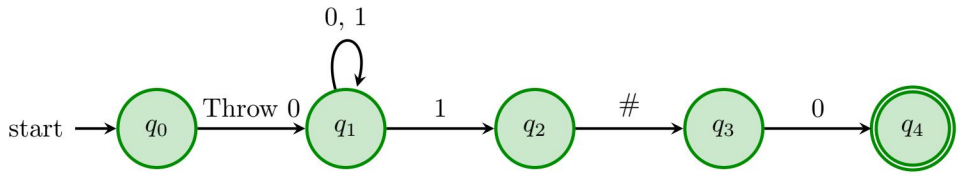
2.1 2a **20 / 20**

✓ **- 0 pts** Correct

Example: The following TFA T decides language

$$L = \{x \in \{0,1\}^* \mid x \text{ starts with a 0 and ends with a 1}\}$$

This is because if x starts with a 0 and ends with a 1, then when processing x, T will first start by appending # to the end of x to get $x\#$. Then, T will throw the initial 0 to the end of the string after the #.

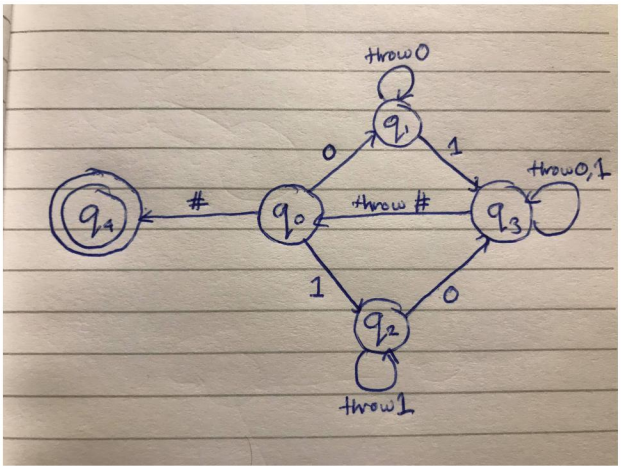


a) **(20 points).** Draw a TFA that decides

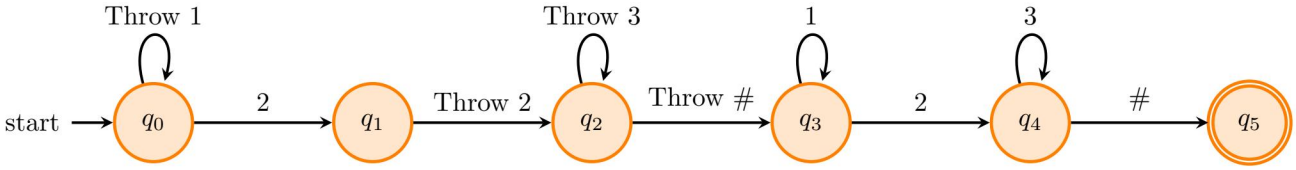
$$L_{EQ} = \{x \in \{0,1\}^* \mid x \text{ has an equal number of 0's and 1's}\}$$

where $\Sigma = \{0,1\}$.

Answer:



b) **(20 points).** Consider the following TFA T over $\Sigma = \{1,2,3\}$:



What is $L(T)$?

Answer:

$L(T) = 1 * 223*$. The shortest word in $L(T)$ is 22, if we avoid all loopback transitions. If we account for the loopback transitions, we can first have an arbitrary number of 1s before the 22, and then have an arbitrary number of 3s after the 22.

2.2 2b 20 / 20

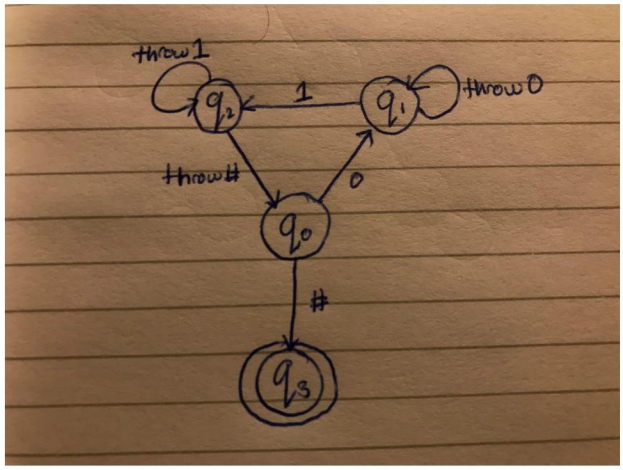
✓ - 0 pts Correct

c) **(30 points).** Prove that for $\Sigma = \{0, 1\}$ then

$$L_{count} = \{0^n 1^n \mid n \geq 0\}$$

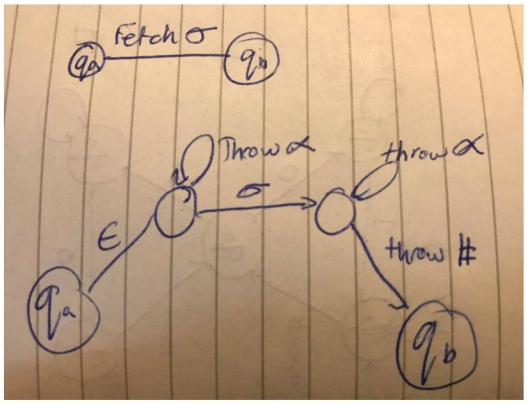
is *Thrown*. (i.e. Prove that there exists a TFA that decides L_{count}.)
Hint: Problem (2b) may help you develop intuition for this.

Answer:



d) **(50 points).** Prove that if L is *Fetching*, then L is *Thrown*.
Hint: Show how to implement Fetch *transitions using a combination of normal transitions,* Throw *transitions, and the end-of-string-symbol #.*
Answer:

First let us consider how to implement a Fetch transition in a TFA. Let $q_a \to q_b$ be a Fetch σ transition. Let α be any letter in the alphabet of the original FFA. We can implement the transition as follows:



This translation throws all the letters before the first instance of σ, then processes σ, then throws all the letters after the first instance of σ, so that the input q_b receives has had the first σ "fetched".

To prove correctness, consider the input string $x@y$, where x and y are arbitrary strings, and x does not contain any @ characters. Let us consider a Fetch @ transition from state q_a to q_b. Using the actual fetch transition, q_b will receive the input xy. Let us consider our fetch transition implementation. First we will epsilon transition. Then, since we know that x does not contain any @ characters, we will throw the substring x using our loopback transition. Then we will process @

2.3 2c 30 / 30

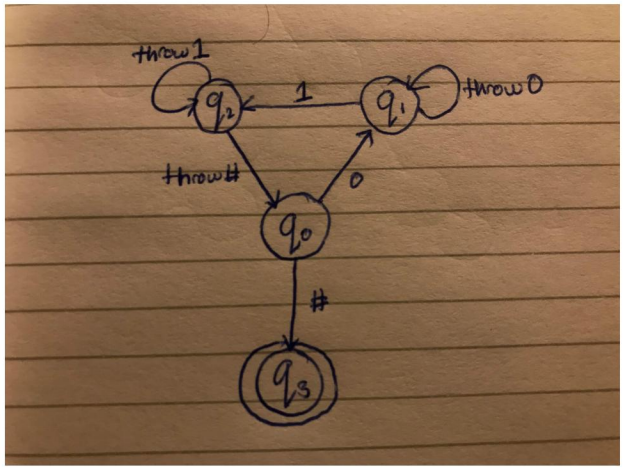
✓ - 0 pts Correct

c) **(30 points).** Prove that for $\Sigma = \{0, 1\}$ then

$$L_{count} = \{0^n 1^n \mid n \geq 0\}$$

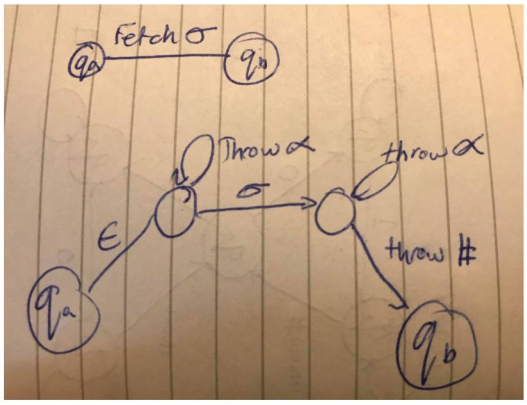
is *Thrown*. (i.e. Prove that there exists a TFA that decides L_{count}.)
Hint: Problem (2b) may help you develop intuition for this.

Answer:



d) **(50 points).** Prove that if L is *Fetching*, then L is *Thrown*.
Hint: Show how to implement Fetch *transitions using a combination of normal transitions,* Throw *transitions, and the end-of-string-symbol #.*
Answer:

First let us consider how to implement a Fetch transition in a TFA. Let $q_a \to q_b$ be a Fetch σ transition. Let α be any letter in the alphabet of the original FFA. We can implement the transition as follows:



This translation throws all the letters before the first instance of σ, then processes σ, then throws all the letters after the first instance of σ, so that the input q_b receives has had the first σ "fetched".

To prove correctness, consider the input string $x@y$, where x and y are arbitrary strings, and x does not contain any @ characters. Let us consider a Fetch @ transition from state q_a to q_b. Using the actual fetch transition, q_b will receive the input xy. Let us consider our fetch transition implementation. First we will epsilon transition. Then, since we know that x does not contain any @ characters, we will throw the substring x using our loopback transition. Then we will process @

and transition to the next state. We will then process the y substring and then Throw to finally transition to q_b. The input that q_b will receive in our case will be xy, and the can be ignored. Thus our implementation of Fetch using TFA operations is a valid substitution.

Let $F = (Q, \Sigma, \delta, q_0, F)$ be the FFA that decides L, the fetching language. Let m be the number of Fetch transitions in F, and let the 2 additional states we need to recreate the ith fetch transition in a TFA be q_{ai} and q_{bi}. We can then create a TFA $T = (Q^{'}, \Sigma^{'}, \delta^{'}, q_0^{'}, F^{'})$ that decides L as follows.

$$Q^{'} = Q \cup \{q_{ai}, q_{bi} \text{ for } 0 < i <= m\} \tag{7}$$

$$\Sigma^{'} = \Sigma \tag{8}$$

$$q_0^{'} = q_0 \tag{9}$$

$$F^{'} = F \tag{10}$$

$$\text{For non-fetch transitions } \delta^{'}(x, a) = \delta(x, a) \tag{11}$$

I'm not sure how to formally express this but we should also replace all fetch transitions with our TFA substitution for a fetch transition.

Since we have proven that our TFA fetch substitution is functionally the same as a FFA fetch transition, one can construct a TFA T that decides L from F. Since we were able to construct such a TFA for an arbitrary fetching language L, if L is fetching, then L is thrown.

2.4 2d 30 / 50

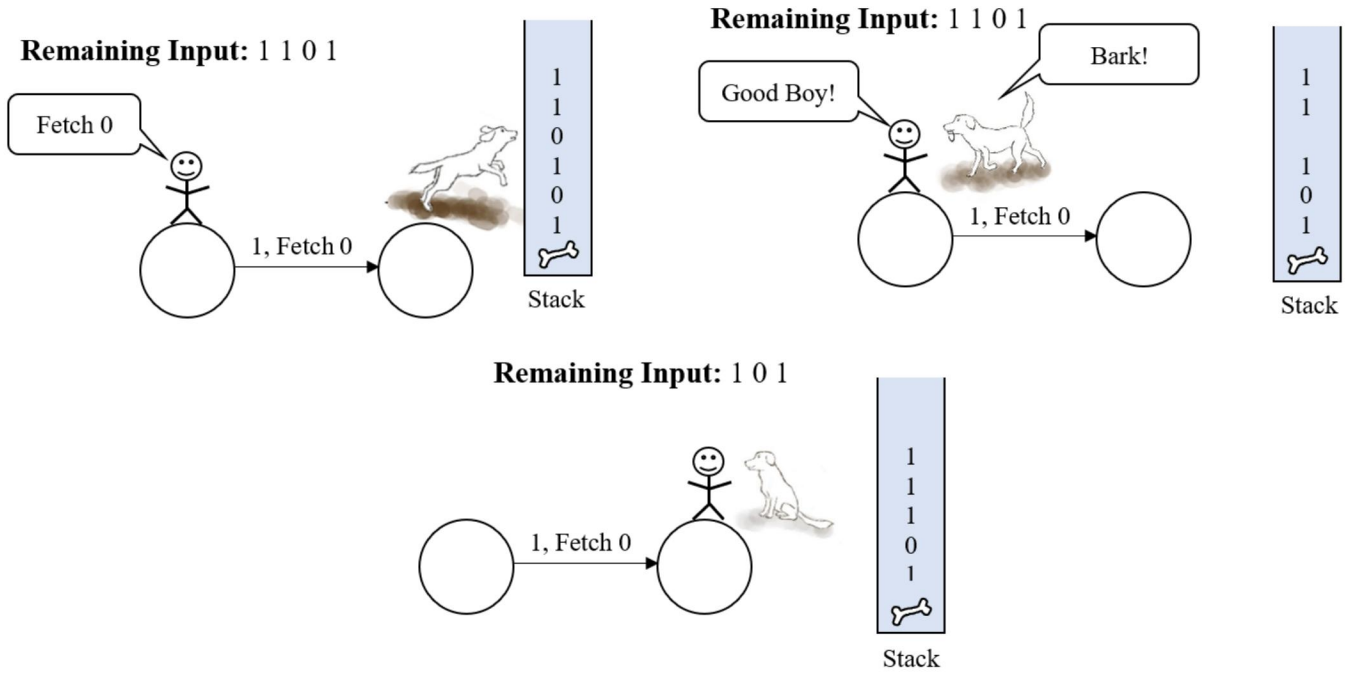
Construction Errors

✓ **- 10 pts** Accepting states need to have an additional transition to consume the final #.

✓ **- 10 pts** You cannot allow a throw on all symbols on the first loop in the transformation because this creates computation paths where you've fetched the next-next or in general the $$i$$th occurrence of your fetched symbol.

Playing fetch but now with stacks.

Extra Credit (50 extra credit points). A Fetch-PDA (FPDA) is like a PDA except that there also exist special Fetch **stack** operations (in addition to push, pop, or $b \to c$ stack operations). On a Fetch transition $(\sigma, \textsf{Fetch } \gamma)$ for $\sigma \in \Sigma \cup \{\epsilon\}$, $\gamma \in \Gamma$, if your next input symbol is σ and there exists a symbol γ anywhere in your stack, then your pet automadog goes and fetches the first γ in your stack (starting from the top of the stack) and brings it to you, removing it from the stack.



(In the above diagram, we use a bone symbol 🦴 *instead of* \$ *to mark the bottom of the stack.)*

Formally, a $(\sigma, \textsf{Fetch } \gamma)$ transition works as follows: If your next input symbol is σ and there exists a γ anywhere in the stack, then the transition may be taken. Otherwise, it cannot be taken. If the transition is taken, you consume input σ, remove the topmost occurrence of γ from wherever it is in the stack (without disturbing the rest of the stack), and transition to the specified state. Note that after removing the γ, everything that was above the γ in the stack shifts down a spot so that there is not an empty position in your stack.

Note that **you may only fetch symbols from the stack**. Your pet automadog will no longer fetch symbols from the input string for you.

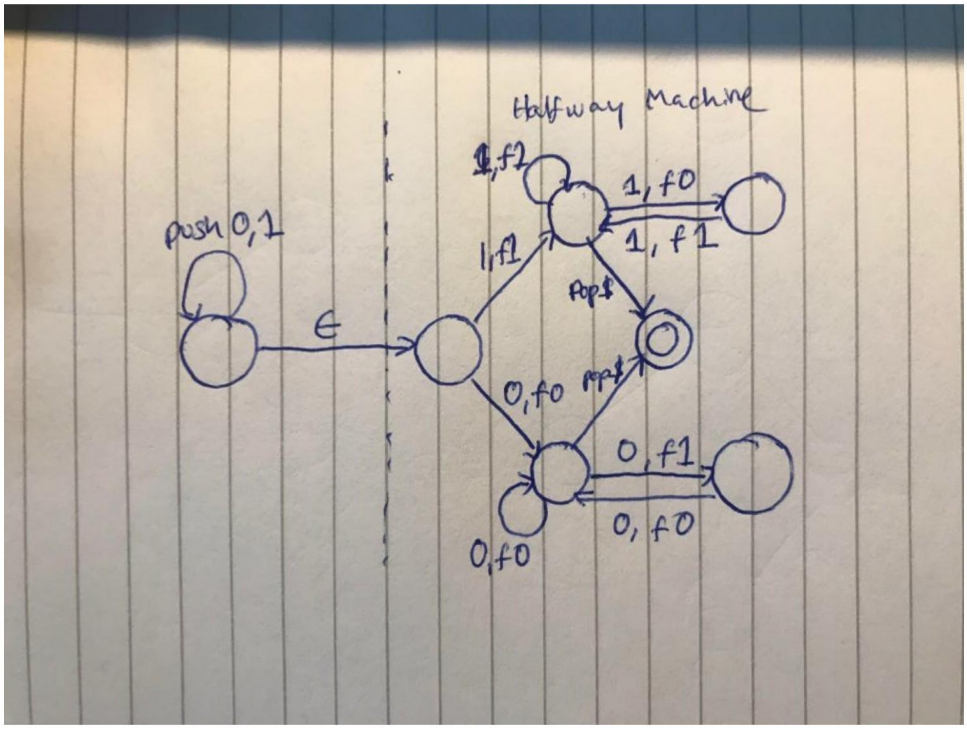
Prove that there is an FPDA that decides

$$L_{\text{ZACH}} = \{x_1 x_2 \dots x_{2n} \in \{0,1\}^* \mid n \geq 0 \text{ and } x_i = \textsf{maj}(x_1, \dots, x_n) \text{ for all } i \in [n+1, 2n]\}$$

where $\textsf{maj}(x_1, \dots, x_n)$ is the majority function defined by

$$\textsf{maj}(x_1, \dots, x_n) = \begin{cases} 0 & \text{if half or more of } x_1, \dots, x_n \text{ are 0} \\ & \text{(i.e. } |\{j \mid 1 \leq j \leq n \text{ and } x_j = 0\}| \geq \tfrac{n}{2}) \\ 1 & \text{else} \end{cases}$$

Answer:



The intuition behind this machine is relatively straightforward. There are 2 portions. We start at a state that just pushes the letters in the input word onto the stack. The second portion is something I've labeled the "Halfway Machine", and its purpose is to determine if the majority property has been filled. Since we have no idea when we've actually reached the halfway point of our input string, we epsilon transition to the halfway machine every time we push onto the stack. The halfway machine uses similar intuition to 2a. The top branch represents the case where 0 is the majority, in which case we need to read n 0s and keep alternating fetching 0s and 1s until the 1s run out. The bottom brnach is the case where 1 is the majority, and we simply flip the top branch logic.
I'm not sure how to formally prove this.

3 Extra Credit: FPDA **45 / 50**

+ **50 pts** Correct

- **10 pts** No explanation

+ **50 pts** Correct, except that your machine doesn't accept the empty string.

✓ **+ 45 pts** **Partial Credit: Correct, except that your machine does not accept strings with an equal number of 0's and 1's in the first half. Recall that 0 is the tie-breaker for majority.**

+ **45 pts** Partial Credit: Correct, except that if there are an equal number of 0's and 1's in the first half, your machine can accept if the second half consists of entirely 1's. Recall that 0 is the tie-breaker for majority.

+ **10 pts** Partial credit: concrete construction reflects a high level idea with potential.

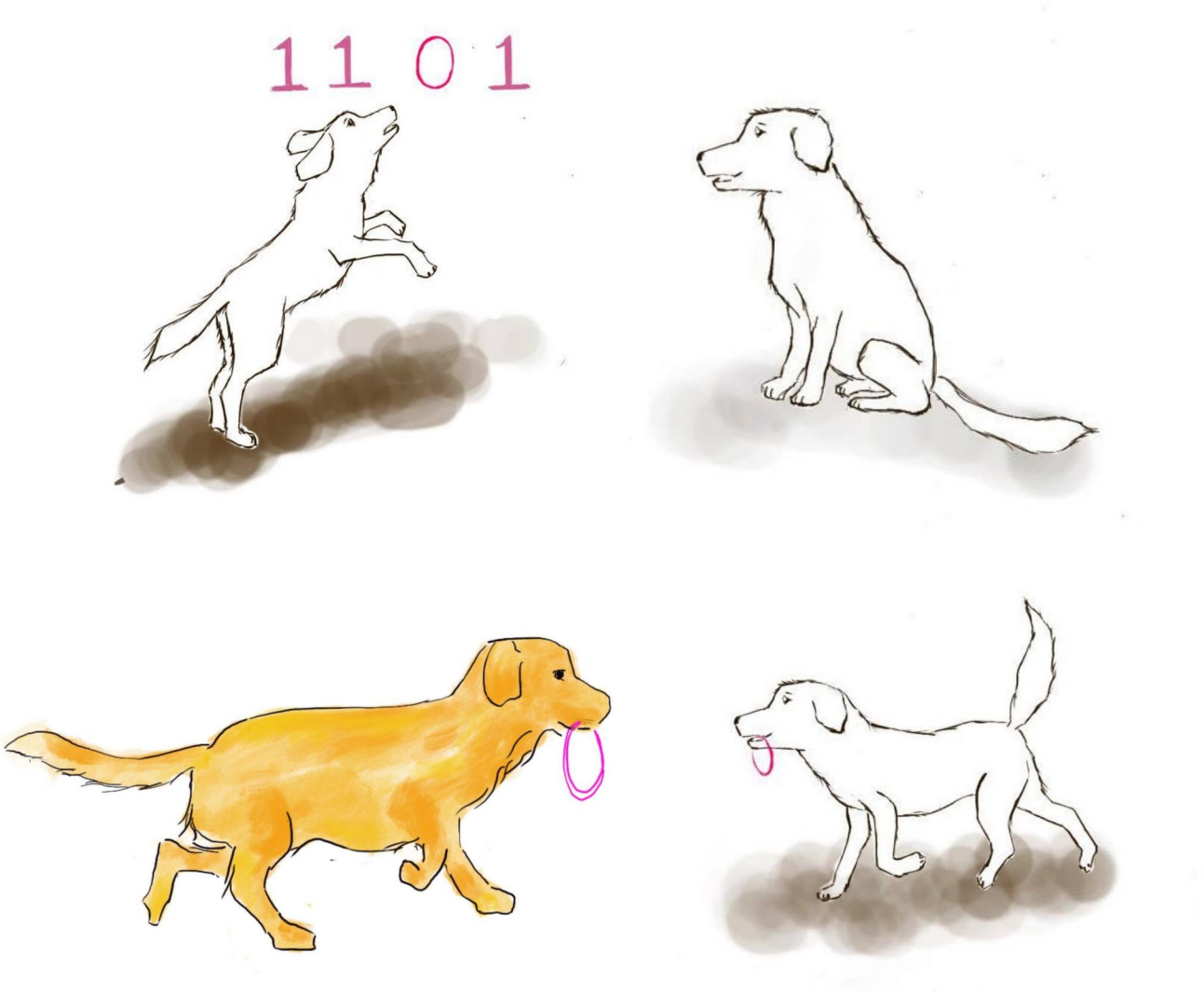
+ **10 pts** Partial credit: Correct intuition, but no concrete construction

+ **0 pts** No attempt/I don't know/vague idea only

+ **0 pts** Incorrect

More automadoggos.[2]

Extra Credit (2 extra credit points). Draw an automadog. Bad drawings okay if art is not your forte. (Any time spent drawing automadoggos does not count against your three hour time limit.)

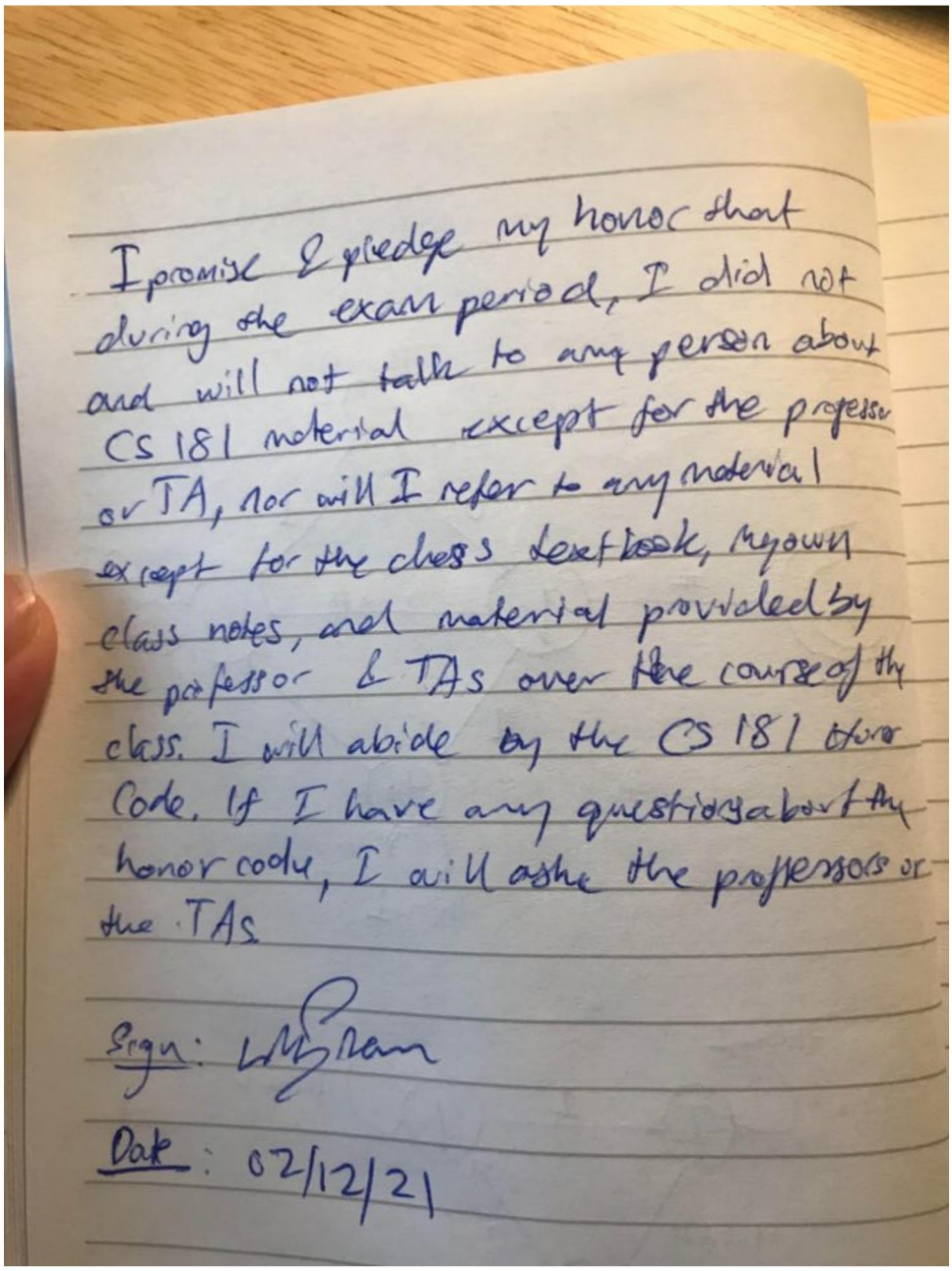


[2]All automadog art created by your TA Paul Lou.

Good luck!

4 Extra Credit: Automadoggos 0 / 2

✓ **- 2 pts** **No Submission**


I promise & pledge my honor that during the exam period, I did not and will not talk to any person about CS 181 material except for the professor or TA, nor will I refer to any material except for the class textbook, my own class notes, and material provided by the professor & TAs over the course of the class. I will abide by the CS 181 Honor Code. If I have any questions about the honor code, I will ask the professors or the TAs

Sign: [signature]

Date: 02/12/21

5 Honor Code Agreement 0 / 0

✓ **- 0 pts** **Correct**