CS 181 Final

Sriram Balachandran

TOTAL POINTS

296 / 372

QUESTION 1

k-hat-CFGs 75 pts

1.1 1a 15 / 15

✓ **- 0 pts Correct**

I will assume that rule (3) had a typo and you intended to write A -> 0A1 | epsilon.

1.2 1b 14 / 20

✓ **- 6 pts Error with majority functions**

hat-A and hat-C do not generate the correct majority functions. For example, hat-A cannot generate 011010110 = 0110 1 0110. Consider where the 1 of E1E must be. Since each E must contain at least as many 1's as 0's, then the 1 must be one of the three central 1's in 011010110. But this means that at least one of the E's has 0's on both endpoints.

1.3 1c 25 / 25

✓ **- 0 pts Correct**

1.4 1d 15 / 15

✓ **- 0 pts Correct**

QUESTION 2

Infinite Dog Paradise 92 pts

2.1 2a 20 / 20

✓ **- 0 pts Correct**

2.2 2b 30 / 30

✓ **- 0 pts Correct**

2.3 2c 40 / 40

✓ **- 0 pts Correct**

pairing(list) will

2.4 Extra Credit (Infinite Dog Paradise) 2 / 2

✓ **- 0 pts Excellent!**

QUESTION 3

Undecidability and Unrecognizability 70 pts

3.1 3a 20 / 20

✓ **- 0 pts Correct**

3.2 3b.i 30 / 30

✓ **- 0 pts Correct**

3.3 3b.ii 15 / 20

✓ **- 5 pts Two machines are circularly defined. This is a crucial point to identify for problem 3b.**

QUESTION 4

Oracle Machines 85 pts

4.1 4a 20 / 20

✓ **- 0 pts Correct**

4.2 4b 20 / 30

✓ **- 10 pts Your helper machine does not take into account the fact that M1 or M2 may loop.**

4.3 4c 15 / 20

✓ **- 5 pts Needs more explanation**

You need a stronger explanation for why your decider is correct.

4.4 4d **15 / 15**

✓ **- 0 pts** **Correct**

QUESTION 5

5 Extra Credit (Narcissist) 0 / 50

✓ **- 50 pts** **No Submission**

QUESTION 6

6 Honor Code 0 / 0

✓ **- 0 pts** **Correct**

a) **(15 points).** Prove that $L_{0123} = \{0^n 1^n 2^n 3^n \mid n \geq 0\}$ is a 2-hat-CFL.

Answer:

We construct a 2-hat-CFL with the following rules:

$$S \to \widehat{X}\widehat{Y}|\epsilon \tag{1}$$
$$\widehat{X} \to 0A1|\epsilon \tag{2}$$
$$A \to 0C1|\epsilon \tag{3}$$
$$\widehat{Y} \to 2B3|\epsilon \tag{4}$$
$$B \to 2B3|\epsilon \tag{5}$$

Due to the rule defining $\widehat{X}$, we can see $\widehat{X}$ will only accept strings of the form $0^n 1^n$, with length $2n$. Similarly, $\widehat{Y}$ will only accept strings of the form $2^m 3^m$, with length $2m$. Now if we look at the start symbol, S, we see that X and Y are hatted, meaning they must be of the same length. Thus $2n = 2m$, and $n = m$. This means the only strings accepted by the described grammar will be of the form $0^n 1^n 2^n 3^n$, and that all words in L_{0123} will be accepted by the constructed grammar. Thus L_{0123} is a 4-hat-CFL.

b) **(20 points).** Prove that the following language is a 4-hat-CFL

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

where

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

Answer:

We construct a 4-hat-CFL with the following rules:

$$S \to \widehat{A}\widehat{B}|\widehat{C}\widehat{D} \tag{6}$$
$$\widehat{A} \to E1E|X \tag{7}$$
$$\widehat{C} \to F0F|Y \tag{8}$$
$$E \to 0E1|1E0|1E1|\epsilon \tag{9}$$
$$F \to 0F1|1F0|0F0|\epsilon \tag{10}$$
$$\widehat{B} -> 1X \tag{11}$$
$$X -> 1X|\epsilon \tag{12}$$
$$\widehat{D} -> 0Y \tag{13}$$
$$Y -> 0Y|\epsilon \tag{14}$$

Let us first consider the first possible parse of the start symbol S, $\widehat{A}\widehat{B}$. $\widehat{A}$ is constructed such that it will only accept a string that has at least one more 1 than it has 0s. It does so by effectively processing 0s and 1s simultaneously until only 1s are left. $\widehat{B}$ only accepts strings of

1.1 1a **15 / 15**

✓ **- 0 pts** **Correct**

I will assume that rule (3) had a typo and you intended to write A -> 0A1 | epsilon.

a) **(15 points).** Prove that $L_{0123} = \{0^n 1^n 2^n 3^n \mid n \geq 0\}$ is a 2-hat-CFL.

Answer:

We construct a 2-hat-CFL with the following rules:

$$S \to \widehat{X}\widehat{Y}|\epsilon \tag{1}$$
$$\widehat{X} \to 0A1|\epsilon \tag{2}$$
$$A \to 0C1|\epsilon \tag{3}$$
$$\widehat{Y} \to 2B3|\epsilon \tag{4}$$
$$B \to 2B3|\epsilon \tag{5}$$

Due to the rule defining $\widehat{X}$, we can see $\widehat{X}$ will only accept strings of the form $0^n 1^n$, with length $2n$. Similarly, $\widehat{Y}$ will only accept strings of the form $2^m 3^m$, with length $2m$. Now if we look at the start symbol, S, we see that X and Y are hatted, meaning they must be of the same length. Thus $2n = 2m$, and $n = m$. This means the only strings accepted by the described grammar will be of the form $0^n 1^n 2^n 3^n$, and that all words in L_{0123} will be accepted by the constructed grammar. Thus L_{0123} is a 4-hat-CFL.

b) **(20 points).** Prove that the following language is a 4-hat-CFL

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

where

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

Answer:

We construct a 4-hat-CFL with the following rules:

$$S \to \widehat{A}\widehat{B}|\widehat{C}\widehat{D} \tag{6}$$
$$\widehat{A} \to E1E|X \tag{7}$$
$$\widehat{C} \to F0F|Y \tag{8}$$
$$E \to 0E1|1E0|1E1|\epsilon \tag{9}$$
$$F \to 0F1|1F0|0F0|\epsilon \tag{10}$$
$$\widehat{B} -> 1X \tag{11}$$
$$X -> 1X|\epsilon \tag{12}$$
$$\widehat{D} -> 0Y \tag{13}$$
$$Y -> 0Y|\epsilon \tag{14}$$

Let us first consider the first possible parse of the start symbol S, $\widehat{A}\widehat{B}$. $\widehat{A}$ is constructed such that it will only accept a string that has at least one more 1 than it has 0s. It does so by effectively processing 0s and 1s simultaneously until only 1s are left. $\widehat{B}$ only accepts strings of

1s. Because A and B are hatted together in the definition of S, they are guaranteed to have the same length. We can then see $\widehat{A}\widehat{B}$ will only accept even-length strings where the first half of the string contains more 1s than 0s (majority 1s), and then the second half of the string contains only 1s. It is obvious that any such word that can be parsed as $\widehat{A}\widehat{B}$ will be in L_{ZACH}. The second possible parse of the start symbol, $\widehat{C}\widehat{D}$ is almost identical to the first parse, except it expects a majority of 0s in the first half and only 0s in the second half of the string. Thus any word that can be parsed as $\widehat{C}\widehat{D}$ will also be in L_{ZACH}. By its definition, all words in L_{ZACH} are of the form where the second half of the string consists only of the majority letter of the first half of the string. This means all words in L_{ZACH} will be accepted by the grammar we constructed, meaning it is a 4-hat-CFL.

c) **(25 points).** Prove that $L_{abc} = \{a^i b^j c^k \mid 0 \leq i < j < k\}$ is a 3-hat-CFL.

Answer:

We construct a 3-hat-CFL with the following rules:

$$S \to \widehat{A}\widehat{B}bGcc\widehat{C} \tag{15}$$

$$\widehat{A} \to D \tag{16}$$

$$D \to aD|\epsilon \tag{17}$$

$$\widehat{B} \to E \tag{18}$$

$$E \to bE|\epsilon \tag{19}$$

$$\widehat{C} \to F \tag{20}$$

$$F \to cF|\epsilon \tag{21}$$

$$G \to HF \tag{22}$$

$$H \to bHc|\epsilon \tag{23}$$

Looking at the start symbol S, we see that A, B, and C are the hatted symbols. These symbols accepts a continuous string of a's, b s, and c's, respectively. Because they are hatted, we can see that there needs to be at least the same number of b's and c's as there are a's. By manually adding in 1 b and 2c's into the start symbol rule, we force the number of b's to be at least 1 greater than the number of a's, and we force the number of c's to be at least 1 greater than the number of the c's. By looking at this alone, we can see that all words accepted by this grammar will be of the form $a^i b^j c^k \mid 0 \leq i < j < k$. The purpose of the G symbol is to allow for flexibility in how many more b's than a's there can be, and also for flexibility in how many more c's than b's there can be. With the G symbol in place, we can then confidently claim that all words in L_{abc} will be accepted by our constructed grammar, meaning L_{abc} is a 3-hat-CFL.

d) **(15 points).** Suppose that we tried to prove a pumping lemma for 2-hat-CFLs.

Lemma 1 (2-hat-CFL Pumping Lemma (false))**.** *If L is a* 2-hat-CFL*, then there exists $n \in \mathbb{N}$ such that $\forall w \in L$ where $|w| \geq n$, then there exist strings $a, b, c, d, e \in \Sigma^*$ such that $w = abcde$ and*

1) $|bcd| \leq n$

2) $|bd| > 0$

3) $\forall i \geq 0$, $ab^i cd^i e \in L$

1.2 1b 14 / 20

✓ **- 6 pts** **Error with majority functions**

hat-A and hat-C do not generate the correct majority functions. For example, hat-A cannot generate 011010110 = 0110 1 0110. Consider where the 1 of E1E must be. Since each E must contain at least as many 1's as 0's, then the 1 must be one of the three central 1's in 011010110. But this means that at least one of the E's has 0's on both endpoints.

1s. Because A and B are hatted together in the definition of S, they are guaranteed to have the same length. We can then see $\widehat{A}\widehat{B}$ will only accept even-length strings where the first half of the string contains more 1s than 0s (majority 1s), and then the second half of the string contains only 1s. It is obvious that any such word that can be parsed as $\widehat{A}\widehat{B}$ will be in L_{ZACH}. The second possible parse of the start symbol, $\widehat{C}\widehat{D}$ is almost identical to the first parse, except it expects a majority of 0s in the first half and only 0s in the second half of the string. Thus any word that can be parsed as $\widehat{C}\widehat{D}$ will also be in L_{ZACH}. By its definition, all words in L_{ZACH} are of the form where the second half of the string consists only of the majority letter of the first half of the string. This means all words in L_{ZACH} will be accepted by the grammar we constructed, meaning it is a 4-hat-CFL.

c) **(25 points).** Prove that $L_{abc} = \{a^i b^j c^k \mid 0 \leq i < j < k\}$ is a 3-hat-CFL.

Answer:

We construct a 3-hat-CFL with the following rules:

$$S \to \widehat{A}\widehat{B}bGcc\widehat{C} \tag{15}$$

$$\widehat{A} \to D \tag{16}$$

$$D \to aD|\epsilon \tag{17}$$

$$\widehat{B} \to E \tag{18}$$

$$E \to bE|\epsilon \tag{19}$$

$$\widehat{C} \to F \tag{20}$$

$$F \to cF|\epsilon \tag{21}$$

$$G \to HF \tag{22}$$

$$H \to bHc|\epsilon \tag{23}$$

Looking at the start symbol S, we see that A, B, and C are the hatted symbols. These symbols accepts a continuous string of a's, b s, and c's, respectively. Because they are hatted, we can see that there needs to be at least the same number of b's and c's as there are a's. By manually adding in 1 b and 2c's into the start symbol rule, we force the number of b's to be at least 1 greater than the number of a's, and we force the number of c's to be at least 1 greater than the number of the c's. By looking at this alone, we can see that all words accepted by this grammar will be of the form $a^i b^j c^k \mid 0 \leq i < j < k$. The purpose of the G symbol is to allow for flexibility in how many more b's than a's there can be, and also for flexibility in how many more c's than b's there can be. With the G symbol in place, we can then confidently claim that all words in L_{abc} will be accepted by our constructed grammar, meaning L_{abc} is a 3-hat-CFL.

d) **(15 points).** Suppose that we tried to prove a pumping lemma for 2-hat-CFLs.

Lemma 1 (2-hat-CFL Pumping Lemma (false))**.** *If L is a* 2-hat-CFL*, then there exists $n \in \mathbb{N}$ such that $\forall w \in L$ where $|w| \geq n$, then there exist strings $a, b, c, d, e \in \Sigma^*$ such that $w = abcde$ and*

1) $|bcd| \leq n$

2) $|bd| > 0$

3) $\forall i \geq 0$, $ab^i cd^i e \in L$

Now, the 2-hat-CFL Pumping Lemma is false because otherwise L_{0123} would not be a 2-hat-CFL. Suppose, however, that we tried to prove the 2-hat-CFL Pumping Lemma using the same proof that we used to prove the normal CFL Pumping Lemma. Consider the following proof attempt:

Proof Attempt.

1) Let L be a 2-hat-CFL. Then there exists a 2-hat-CFG $G = (V, \Sigma, R, S)$ for L.
2) Let $n = k^{|V|+1} + 1$ where $|V|$ is the number of variables in G and k is the maximum number of symbols on the RHS of any rule in G.
3) Consider any string $x \in L$ with $|x| \geq n$.
4) Let T be a minimal parse tree for x.
   (If T is a minimal parse tree for x, then there does not exist another parse tree for x with fewer nodes than T.)
5) Since $|x| \geq k^{|V|+1} + 1 \geq k^{|V|} + 1$, then the height of the parse tree must be at least $|V| + 1$.
   (A tree of height h with branching factor k cannot have more than k^h leaf nodes.)
6) Then, there must exist a path from root to leaf in T with at least $|V| + 1$ variables.
7) Consider the bottom-most $|V| + 1$ variables in such a path. By the pigeonhole principle, at least one of these variables must repeat. Let us call the repeated variable A.

1.3 1c **25 / 25**

✓ **- 0 pts** Correct

13) Finally, $|bcd| \leq n$. This is because we chose our repeated variable A from among the bottom-most $|V| + 1$ variables, so the height of the larger parse tree generated by A is at most $|V| + 1$. But this means that the larger parse tree can have at most $k^{|V|+1} < n$ leaf nodes, which means $|bcd| \leq n$.

□

Name the first step where the proof attempt above fails, and explain why that step fails.

Hint: The proof attempt above would be a valid proof for the normal CFL *pumping lemma. You should be looking for statements that are no longer true when we try to apply them to* 2-hat-CFL*s instead of normal* CFL*s.*

Answer:

Step 10 is the first step where the proof attempt above fails. One of A's subtrees could potentially be in the subtree of a hatted variable - if we were to substitute one subtree of A for another, we would change the length of the parse of one of the hatted variables. This would then cause our two hatted variables to not be of the same length, and thus we would contradict the rules of the 2-hat-CFL.

1.4 1d 15 / 15

✓ - 0 pts Correct

Answer:

On day j, dog i should take path $i + j$. Since j strictly increases, we can see that dog i will never be assigned to the same path on different days, so this assignment satisfies the adventure-seeking requirement. Consider path k on the jth day. Based on our assignment rule, only dog $k - j$ can be assigned to path k on day j. This hold for all paths across all days, meaning that no 2 dogs will be assigned to the same path on the same day, satisfying the privacy-concerned requirement.

b) **(30 points).** During the second ∞-month on the job, the dogs are lonely and want to try speed dating in pairs. Let each day of the second ∞-month be labeled $(b_1, b_2, b_3, \ldots)$.

i. Each dog is *love-seeking* and wants to see new faces, so they must be assigned a completely new partner for each day of the second ∞-month. That is, if d_i is paired with d_j for day b_k, then d_i cannot be paired with d_j for any other day of the second ∞-month. Every dog needs to be paired on every day.

ii. Moreover, each dog remains *adventure-seeking*. If a pair of dogs (d_i, d_j) is assigned to path p_k on day b_ℓ, both dog d_i and dog d_j cannot be assigned path p_k on any other day of the second ∞-month.

iii. Finally, each pair is *privacy-concerned* so on any given day there is a single pair of dogs assigned to any path. However, not every path must be used on each day.

Describe a way to assign pairs and paths to the dogs for every day of the second ∞-month such that the assignment satisfies the *love-seeking*, *adventure-seeking*, and *privacy-concerned* desiderata.

2.1 2a **20 / 20**

✓ **- 0 pts** Correct

Answer:

On day j, dog i should take path $i + j$. Since j strictly increases, we can see that dog i will never be assigned to the same path on different days, so this assignment satisfies the adventure-seeking requirement. Consider path k on the jth day. Based on our assignment rule, only dog $k - j$ can be assigned to path k on day j. This hold for all paths across all days, meaning that no 2 dogs will be assigned to the same path on the same day, satisfying the privacy-concerned requirement.

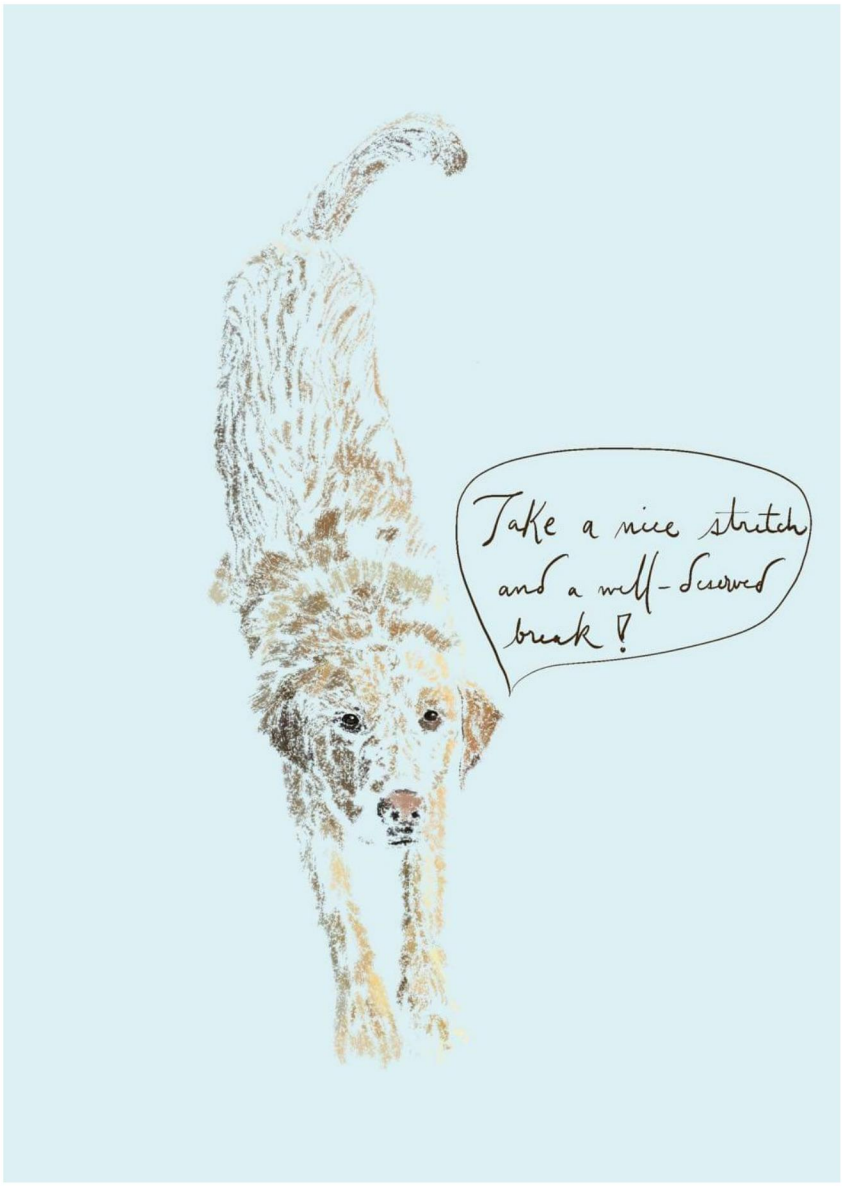
b) **(30 points).** During the second ∞-month on the job, the dogs are lonely and want to try speed dating in pairs. Let each day of the second ∞-month be labeled $(b_1, b_2, b_3, \ldots)$.

i. Each dog is *love-seeking* and wants to see new faces, so they must be assigned a completely new partner for each day of the second ∞-month. That is, if d_i is paired with d_j for day b_k, then d_i cannot be paired with d_j for any other day of the second ∞-month. Every dog needs to be paired on every day.

ii. Moreover, each dog remains *adventure-seeking*. If a pair of dogs (d_i, d_j) is assigned to path p_k on day b_ℓ, both dog d_i and dog d_j cannot be assigned path p_k on any other day of the second ∞-month.

iii. Finally, each pair is *privacy-concerned* so on any given day there is a single pair of dogs assigned to any path. However, not every path must be used on each day.

Describe a way to assign pairs and paths to the dogs for every day of the second ∞-month such that the assignment satisfies the *love-seeking*, *adventure-seeking*, and *privacy-concerned* desiderata.



Drawn by Paul Lou

Answer:

Let P be a bijective function mapping $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$. An example of such a function can be derived from the following logic. There exists a unique prime factorization for all natural numbers. Also consider the fact that all prime numbers except for 2 are odd, and that the product of odd numbers is always odd. From these facts we can derive the following:

$$n = 2^a(2b - 1) \tag{24}$$

All natural numbers can be uniquely expressed as the product of a power of 2 and an odd number. Thus our function P to uniquely map a tuple of natural number to a single natural number is:

$$P(a, b) = 2^{a-1}(2b - 1) \tag{25}$$

Now that we have defined P, we can describe the method for assigning pairs and paths to the dogs for every day of the month.

This is the process to get the pairings and path assignments for all dogs on day m. Start with an empty set, called paired. This will hold all the dogs that have already been paired on day m, so we avoid pairing dogs that have already been paired. If dog i is not in the paired set, pair dog i with dog $j = P(i, m)$. Then push dog i and dog j into the paired set. We then assign the dog pair (i, j) to path $k = P(i, j)$.

Due to the unique mapping nature of the pairing function P, we know that $j = P(i, m)$ will result in dog i never being paired with the same dog more than once, satisfying the love-seeking requirement. Additionally, since $P(i, j)$ always maps to a unique number, we know that the only possible way for dog i to be assigned to the same path more than once is if it is paired with dog j again. However, we have proved that that is not possible, and thus our assignment method also satisfies the adventure-seeking requirement. Under the same logic, since $k = P(i, j)$ always maps to a unique number, we know that it is impossible for 2 pairs to get mapped to the same path on a given day, satisfying the privacy-concerned requirement.

2.2 2b 30 / 30

✓ - 0 pts Correct

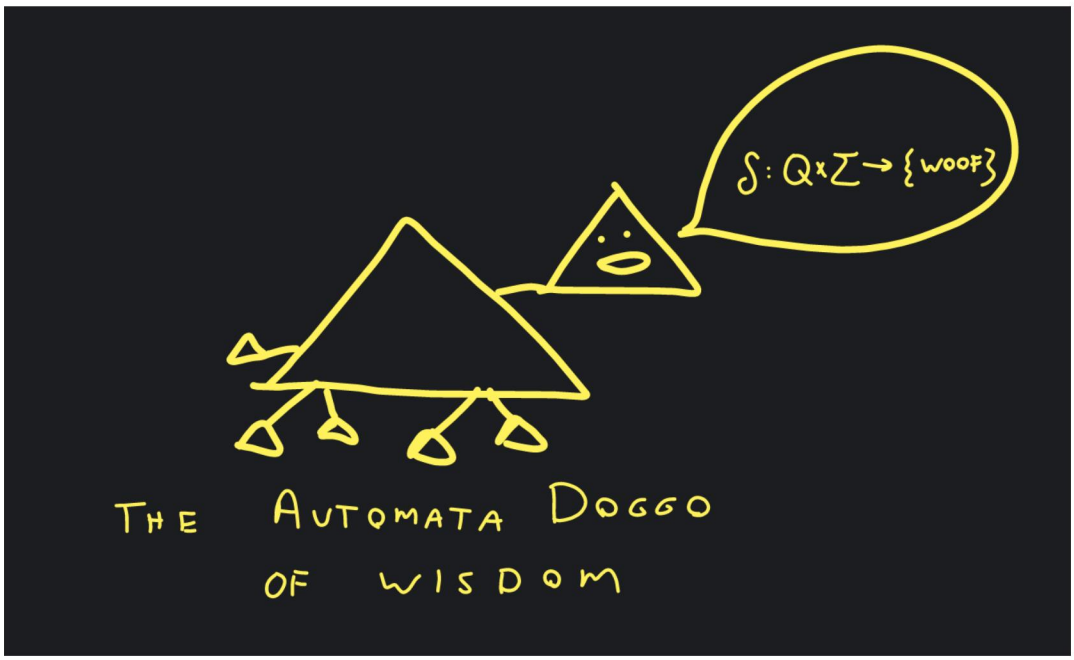
c) **(40 points).** During the third ∞-month on the job, the dogs have speed-dated and now they want to grow their social circle. Let each day of the third ∞-month be labeled $(c_1, c_2, c_3, \ldots)$.

i. Each dog is *socializing* and wants to be assigned to a friend circle of k dogs on day c_k. However, any two distinct dogs d_i and d_j can be part of the same friend circle only once. That is, if dogs d_i, d_j are in the same friend circle on day c_k, then they cannot be in the same friend circle on any other day of the third ∞-month.

ii. Each dog remains *adventure-seeking*. If a circle of dogs (d_i, d_j, d_k) is assigned to path p_ℓ on day c_3, dogs d_i, d_j, d_k cannot be assigned path p_ℓ on any other day of the third ∞-month.

iii. Each friend circle is *privacy-concerned* so on any given day there is a single friend circle of dogs assigned to any path. However, not every path must be used on each day.

Describe a way to assign friend circles and paths to the dogs for every day of the third ∞-month such that the assignment satisfies the *socializing*, *adventure-seeking*, and *privacy-concerned* desiderata.



Drawn by Professor Sahai

Answer:

We can assign friend circles using the algorithm described below:

```
def pairing(list):
    if len(list) == 1:
        return list[0]
    if len(list) == 2:
        return P(list[0], list[1])
    return P(list[0], pairing(list[1:])

paired = set()
for k in days:
    for i in dogs:
        if i not in paired:
            circle = [k, i]
```

```
    for j in range(k-1): #k-1 since we include dog i
        circle.append(pairing(circle))
    circle = [1:]
    # get rid of k before adding to paired
    paired += set(circle)
```

The path for a circle c is simply $pairing(c)$.

Note the function "pairing" is simply a recursive extension of the pairing function we defined previously, P. It allows us to uniquely map $\mathbb{N}^M \to \mathbb{N}$. The first dog in a circle, i, and the day, k, act as the "seed" values for a circle. From the same logic explained in the previous part, we can see why dog i on day k would never generate the same "next dog" more than once. Each additional dog added to the circle is a function of all the dogs in the circle so far. Intuitively, we can see that as long as the seed values never generate the same dog more than once on a given day, which they don't, then the rest of the circle will remain unique as well, meaning none of the dogs will have previously met each other. This satisfies the socializing requirement. The path for a circle c, on day k, is determined from $pairing(c)$. Since all circles will be size k on day k, $pairing$ uniquely maps $\mathbb{N}^k \to \mathbb{N}$. Thus we can be assured that no two circles will be assigned to the same path on a given day, satisfying the privacy-concerned requirement.

It is trivial to see that if one pairing function maps $\mathbb{N}^x \to \mathbb{N}$ and another pairing function maps $\mathbb{N}^y \to \mathbb{N}$ such that $x \neq y$, it will not be possible for $P_x(c1) = P_y(c2)$ unless $|c1 \cap c2| >= min(|c1|, |c2|)$. However, since we satisfy the socializing property, we know that this will never be true. Thus a dog will never be assigned to the same path more than once, satisfying the adventure-seeking property.

Hint: In the above problems it may be helpful to recall that all natural numbers have a unique prime factorization.

At the end of your internship after the third ∞-month, there is a giant party for all of the dogs where they can hang out with all of the new friends they made over the summer.

(a) **(2 Extra Credit Points). Review your internship at *Infinite Dog Paradise*. Draw a picture of your time, tell us about what role you'd like to have full-time, and/or tell us how you envision the future of *Infinite Dog Paradise*.**

Please include the words "I give the CS 181 instructors permission to share my extra credit answer with the class" somewhere in your response if you would like to give us permission to compile your extra credit answer into an anonymous compilation to be shared with the class.

I give the CS 181 instructors permission to share my extra credit answer with the class.

2.3 2c **40 / 40**

✓ **- 0 pts** **Correct**

pairing(list) will


Pointeur of le doggos

2.4 Extra Credit (Infinite Dog Paradise) 2 / 2

✓ **- 0 pts** **Excellent!**

3. **Undecidability and Unrecognizability (70 points).**

(a) **(20 points).** Define $\Sigma = \{a, b, c, \ldots, z\}$ be the set of letters in the English alphabet. Prove that the following language is undecidable:

$$\textsf{Cats-VS-Dogs} = \{\langle M \rangle \mid \text{Either “cats” } \in L(M) \text{ or “dogs” } \in L(M), \text{ but not both.}\}.$$

Answer:

Suppose, for sake of contradiction, that Cats-VS-Dogs is decidable. Then, there exists a decider D for Cats-VS-Dogs. Construct a TM M as follows:

$M(x)$:

1) Let $z = \langle M \rangle$, as obtained from the recursion theorem.
2) Run $D(z)$.
3) If D accepts then output **accept**.
4) If D rejects then output **accept** if x="cats" else output **reject**.

Consider the case where D accepts. Then we accept all x, and the language of M becomes Σ^*, which contradicts D accepting M.

Then consider the case where D rejects. Then we only accept the word "cats", and reject all other inputs. This means the language of M is {"cats"}, which means $< M >$ should be in Cats-VS-Dogs, contradicting the decision of the decider D.

Since we arrive at contradiction in all possible cases, we have that Cats-VS-Dogs is undecidable.

(b) Consider the following language called best friends forever (BFFs):

$$\textsf{BFFs} = \{(\langle M_1 \rangle, \langle M_2 \rangle) \mid L(M_1) = \{\langle M_2 \rangle\} \text{ and } L(M_2) = \{\langle M_1 \rangle\}\}.$$

i. **(30 points).** Prove that BFFs is undecidable.

Answer:

Suppose, for sake of contradiction, that BFFs is decidable. Then, there exists a recognizer D for BFFs. Construct TMs M_1 and M_2 as follows:

Let M_2 be a TM such that $L(M_2) = \langle M_1 \rangle$

$M_1(x)$:

1) Let $z = \langle M_1 \rangle$, as obtained from the recursion theorem.
2) Run $D(z, \langle M_2 \rangle)$.
3) If D accepts then output **accept**.
4) If D rejects then output **accept** if x=$\langle M_2 \rangle$ else output **reject**.

Consider the case where D accepts. Then we accept all x, and the language of M_1 becomes Σ^*, which contradicts D accepting M_1, M_2, since $L(M_1)$ should've only been $\{\langle M_2 \rangle\}$.

Then consider the case where D rejects. Then we only accept the description of M_2, and reject all other inputs. This means the language of M_1 is $\{\langle M_2 \rangle\}$, which means M_1, M_2 should be in BFFs, contradicting the decision of the decider D.

Since we arrive at contradiction in all possible cases, we have that BFFs is undecidable.

ii. **(20 points).** Prove that BFFs is unrecognizable.

Answer:

Suppose, for sake of contradiction, that BFFs is recognizable. Then, there exists a recognizer R for BFFs. Construct TMs M_1 and M_2 as follows:

Let M_2 be a TM such that $L(M_2) = \langle M_1 \rangle$

$M_1(x)$:

3.1 3a **20 / 20**

✓ **- 0 pts** Correct

3. **Undecidability and Unrecognizability (70 points).**

(a) **(20 points).** Define $\Sigma = \{a, b, c, \ldots, z\}$ be the set of letters in the English alphabet. Prove that the following language is undecidable:

$$\textsf{Cats-VS-Dogs} = \{\langle M \rangle \mid \text{Either “cats” } \in L(M) \text{ or “dogs” } \in L(M), \text{ but not both.}\}.$$

Answer:

Suppose, for sake of contradiction, that Cats-VS-Dogs is decidable. Then, there exists a decider D for Cats-VS-Dogs. Construct a TM M as follows:

$M(x)$:

1) Let $z = \langle M \rangle$, as obtained from the recursion theorem.
2) Run $D(z)$.
3) If D accepts then output **accept**.
4) If D rejects then output **accept** if x="cats" else output **reject**.

Consider the case where D accepts. Then we accept all x, and the language of M becomes Σ^*, which contradicts D accepting M.

Then consider the case where D rejects. Then we only accept the word "cats", and reject all other inputs. This means the language of M is {"cats"}, which means $< M >$ should be in Cats-VS-Dogs, contradicting the decision of the decider D.

Since we arrive at contradiction in all possible cases, we have that Cats-VS-Dogs is undecidable.

(b) Consider the following language called best friends forever (BFFs):

$$\textsf{BFFs} = \{(\langle M_1 \rangle, \langle M_2 \rangle) \mid L(M_1) = \{\langle M_2 \rangle\} \text{ and } L(M_2) = \{\langle M_1 \rangle\}\}.$$

i. **(30 points).** Prove that BFFs is undecidable.

Answer:

Suppose, for sake of contradiction, that BFFs is decidable. Then, there exists a recognizer D for BFFs. Construct TMs M_1 and M_2 as follows:

Let M_2 be a TM such that $L(M_2) = \langle M_1 \rangle$

$M_1(x)$:

1) Let $z = \langle M_1 \rangle$, as obtained from the recursion theorem.
2) Run $D(z, \langle M_2 \rangle)$.
3) If D accepts then output **accept**.
4) If D rejects then output **accept** if x=$\langle M_2 \rangle$ else output **reject**.

Consider the case where D accepts. Then we accept all x, and the language of M_1 becomes Σ^*, which contradicts D accepting M_1, M_2, since $L(M_1)$ should've only been $\{\langle M_2 \rangle\}$.

Then consider the case where D rejects. Then we only accept the description of M_2, and reject all other inputs. This means the language of M_1 is $\{\langle M_2 \rangle\}$, which means M_1, M_2 should be in BFFs, contradicting the decision of the decider D.

Since we arrive at contradiction in all possible cases, we have that BFFs is undecidable.

ii. **(20 points).** Prove that BFFs is unrecognizable.

Answer:

Suppose, for sake of contradiction, that BFFs is recognizable. Then, there exists a recognizer R for BFFs. Construct TMs M_1 and M_2 as follows:

Let M_2 be a TM such that $L(M_2) = \langle M_1 \rangle$

$M_1(x)$:

3.2 3b.i 30 / 30

✓ - 0 pts Correct

3. **Undecidability and Unrecognizability (70 points).**

(a) **(20 points).** Define $\Sigma = \{a, b, c, \ldots, z\}$ be the set of letters in the English alphabet. Prove that the following language is undecidable:

$$\textsf{Cats-VS-Dogs} = \{\langle M \rangle \mid \text{Either “cats” } \in L(M) \text{ or “dogs” } \in L(M), \text{ but not both.}\}.$$

Answer:

Suppose, for sake of contradiction, that Cats-VS-Dogs is decidable. Then, there exists a decider D for Cats-VS-Dogs. Construct a TM M as follows:

$M(x)$:

1) Let $z = \langle M \rangle$, as obtained from the recursion theorem.
2) Run $D(z)$.
3) If D accepts then output **accept**.
4) If D rejects then output **accept** if x="cats" else output **reject**.

Consider the case where D accepts. Then we accept all x, and the language of M becomes Σ^*, which contradicts D accepting M.

Then consider the case where D rejects. Then we only accept the word "cats", and reject all other inputs. This means the language of M is {"cats"}, which means $< M >$ should be in Cats-VS-Dogs, contradicting the decision of the decider D.

Since we arrive at contradiction in all possible cases, we have that Cats-VS-Dogs is undecidable.

(b) Consider the following language called best friends forever (BFFs):

$$\textsf{BFFs} = \{(\langle M_1 \rangle, \langle M_2 \rangle) \mid L(M_1) = \{\langle M_2 \rangle\} \text{ and } L(M_2) = \{\langle M_1 \rangle\}\}.$$

i. **(30 points).** Prove that BFFs is undecidable.

Answer:

Suppose, for sake of contradiction, that BFFs is decidable. Then, there exists a recognizer D for BFFs. Construct TMs M_1 and M_2 as follows:

Let M_2 be a TM such that $L(M_2) = \langle M_1 \rangle$

$M_1(x)$:

1) Let $z = \langle M_1 \rangle$, as obtained from the recursion theorem.
2) Run $D(z, \langle M_2 \rangle)$.
3) If D accepts then output **accept**.
4) If D rejects then output **accept** if x=$\langle M_2 \rangle$ else output **reject**.

Consider the case where D accepts. Then we accept all x, and the language of M_1 becomes Σ^*, which contradicts D accepting M_1, M_2, since $L(M_1)$ should've only been $\{\langle M_2 \rangle\}$.

Then consider the case where D rejects. Then we only accept the description of M_2, and reject all other inputs. This means the language of M_1 is $\{\langle M_2 \rangle\}$, which means M_1, M_2 should be in BFFs, contradicting the decision of the decider D.

Since we arrive at contradiction in all possible cases, we have that BFFs is undecidable.

ii. **(20 points).** Prove that BFFs is unrecognizable.

Answer:

Suppose, for sake of contradiction, that BFFs is recognizable. Then, there exists a recognizer R for BFFs. Construct TMs M_1 and M_2 as follows:

Let M_2 be a TM such that $L(M_2) = \langle M_1 \rangle$

$M_1(x)$:

1) Let $z = \langle M_1 \rangle$, as obtained from the recursion theorem.
2) If $x = \langle M_2 \rangle$ then output **accept**.
3) Run $R(z, \langle M_2 \rangle)$.
4) If R accepts then output **accept**.
5) If R rejects then output **reject**.
6) If R loops then pass

Consider the case where R accepts. Then we accept all x, and the language of M_1 becomes Σ^*, which contradicts R accepting M_1, M_2, since $L(M_1)$ should've only been $\{\langle M_2 \rangle\}$.
Then consider the case where R rejects. Then we only accept the description of M_2, and reject all other inputs. This means the language of M_1 is $\{\langle M_2 \rangle\}$, which means M_1, M_2 should be in BFFs, contradicting the decision of the decider R.
Finally, consider the case where R loops. Then we only accept the description of M_2, and loop on all other inputs. Thus the language of M_1 is $\{\langle M_2 \rangle\}$, which means M_1, M_2 should be in BFFs, contradicting the decision of the decider R (it should've accepted instead of looping).
Since we arrive at contradiction in all possible cases, we have that BFFs is unrecognizable.

3.3 3b.ii **15 / 20**

✓ **- 5 pts** Two machines are circularly defined. This is a crucial point to identify for problem 3b.

4. **Oracle Machines (85 points).**

Intuitively, an oracle machine is a Turing machine that has access to a magical "oracle" that can instantly answer questions of a certain form. The TM operates like a normal TM except that it can ask questions to the oracle and receive an immediate response.

Remark 1. *We refer to the TM model you learned in lecture as a standard TM.*

Remark 2. *If A is a set, we use the notation $|A| = \infty$ to mean the set cardinality is infinite, and the notation $|A| < \infty$ to mean the set cardinality is finite.*

Infinite Overlap Turing Machines
We define an infinite overlap Turing machine (iO-TM) to be a Turing machine with an "infinite-overlap" oracle. Given two TM descriptions $\langle M_1 \rangle$ and $\langle M_2 \rangle$, the infinite-overlap oracle will tell the infinite-overlap TM whether $|L(M_1) \cap L(M_2)| = \infty$. More formally,

Definition 1. *We define an infinite-overlap Turing machine (iO-TM) to be a four-tape TM that has a single all-purpose input/work/output tape, two query input tapes, and a query result tape.*

- *An iO-TM must have the states $q_{start}, q_{accept}, q_{reject}$ and a special state q_{query}.*
- *An iO-TM M can write the description of two standard TMs $\langle M_1 \rangle$ and $\langle M_2 \rangle$ onto the two query tapes, each tape containing a single description.*
- *If M reaches the special query state q_{query} where the string $\langle M_1 \rangle$ is the content of the first query input tape and the string $\langle M_2 \rangle$ is the content of the second query input tape for any arbitrary standard TMs M_1, M_2, then in a single atomic step all three query tapes are wiped clean and a single bit is written on the query output tape. The single bit written is 1 if*

$$|L(M_1) \cap L(M_2)| = \infty$$

and 0 otherwise.

- *In all other aspects, an iO-TM operates normally as a standard TM would.*

Definition 2. *A language $L \subseteq \{0,1\}^*$ is infinite-overlap decidable (iO-decidable) if there exists an iO-TM M that decides L. That is, $M(x)$ accepts if and only if $x \in L$ and $M(x)$ rejects if and only if $x \notin L$.*

a) **(20 points).** Prove that the following language is iO-decidable:

$$\mathsf{HALT}_\epsilon = \{\langle M \rangle : M(\varepsilon) \text{ halts}, M \text{ is a standard TM}\}.$$

Answer:
We can construct the following iO-TM to decide this language as follows. Let $val(x)$ be the integer value of a binary string x.
$\underline{I(< M >)}$:
$\underline{K(x)}$:
1) run $M(\epsilon)$ for $val(x)$ steps
2) if M has not halted, then output **accept**
3) else output **reject**

$\underline{J(x)}$: output **accept**

1) Run oracle($\langle K \rangle$, $\langle J \rangle$)
2) If oracle returns 0, output **accept**
3) If oracle returns 1, output **reject**

If M halts on epsilon in k steps, then $|L(K)| = k$. When we pass K into the oracle with the description of machine J, whose language is all possible words, the overlap will be finite only if $L(K)$ is finite, meaning M halts on epsilon.

b) **(30 points).** Prove that the following language is iO-decidable:

$$\mathsf{DISJOINT} = \{(\langle M_1 \rangle, \langle M_2 \rangle) : |L(M_1) \cap L(M_2)| = 0,\ M_1 \text{ and } M_2 \text{ are standard TMs}\}.$$

Answer:

We can construct the following iO-TM to decide this language as follows.

$\underline{I(< M_1 >, < M_2 >)}$:

$\underline{K(x)}$:

1) for y in range(0,x): if $M_1(y)$=accept and $M_2(y)$=accept then output **accept**
2) output **reject**

$\underline{J(x)}$: output **accept**

1) Run oracle($\langle K \rangle$, $\langle J \rangle$)
2) If oracle returns 0, output **accept**
3) If oracle returns 1, output **reject**

If $|M_1 \cap M_2| > 0$, then by the construction of K, $|L(K)| = infinite$. When we pass K into the oracle with the description of machine J, whose language is all possible words, the overlap will be finite only if $L(K)$ is finite, meaning $|M_1 \cap M_2| = 0$.

4.1 4a 20 / 20

✓ - 0 pts Correct

1) Run oracle($\langle K \rangle$, $\langle J \rangle$)
2) If oracle returns 0, output **accept**
3) If oracle returns 1, output **reject**

If M halts on epsilon in k steps, then $|L(K)| = k$. When we pass K into the oracle with the description of machine J, whose language is all possible words, the overlap will be finite only if $L(K)$ is finite, meaning M halts on epsilon.

b) **(30 points).** Prove that the following language is iO-decidable:

$$\mathsf{DISJOINT} = \{(\langle M_1 \rangle, \langle M_2 \rangle) : |L(M_1) \cap L(M_2)| = 0,\ M_1 \text{ and } M_2 \text{ are standard TMs}\}.$$

Answer:

We can construct the following iO-TM to decide this language as follows.

$\underline{I(< M_1 >, < M_2 >)}$:

$\underline{K(x)}$:

1) for y in range(0,x): if $M_1(y)$=accept and $M_2(y)$=accept then output **accept**
2) output **reject**

$\underline{J(x)}$: output **accept**

1) Run oracle($\langle K \rangle$, $\langle J \rangle$)
2) If oracle returns 0, output **accept**
3) If oracle returns 1, output **reject**

If $|M_1 \cap M_2| > 0$, then by the construction of K, $|L(K)| = infinite$. When we pass K into the oracle with the description of machine J, whose language is all possible words, the overlap will be finite only if $L(K)$ is finite, meaning $|M_1 \cap M_2| = 0$.

4.2 4b 20 / 30

✓ **- 10 pts** Your helper machine does not take into account the fact that M1 or M2 may loop.

Counting Turing Machines

We now consider a Turing machine with a different oracle. A counting Turing machine (counting-TM) is a Turing machine with a "counting" oracle. Given a TM description $\langle N \rangle$, the counting oracle will tell the counting-TM the size of $L(N)$. More formally,

Definition 3. *We define a counting Turing machine (counting-TM) to be a three-tape TM that has a single all-purpose input/work/output tape, a query input tapes, and a query result tape.*

- *A counting-TM must have the states $q_{start}, q_{accept}, q_{reject}$ and a special state q_{query}.*
- *A counting-TM M can write the description of a standard TM $\langle N \rangle$ onto the query tape.*
- *If M reaches the special query state q_{query} and the query input tape content is $\langle N \rangle$ for any arbitrary standard TM N, then in a single atomic step both query tapes are wiped clean and the cardinality $|L(N)|$ is written to the query output tape. If $|L(N)|$ is finite, then the cardinality is written in unary as $1^{|L(N)|}$. If $|L(N)|$ is infinite, then a special ∞ symbol is written instead.*
- *In all other aspects, a counting-TM operates normally as a standard TM would.*

Definition 4. *A language $L \subseteq \{0,1\}^*$ is counting-decidable if there exists a counting-TM M that decides L. That is, $M(x)$ accepts if and only if $x \in L$ and $M(x)$ rejects if and only if $x \notin L$.*

c) **(20 points).** Prove that the following language is counting-decidable:

$$\text{FINITE-EQUIV} = \{(\langle M_1 \rangle, \langle M_2 \rangle) : L(M_1) = L(M_2),\ |L(M_1)| < \infty,\ |L(M_2)| < \infty,\ M_1 \text{ and } M_2 \text{ are standard TMs}\}.$$

Answer: We can construct a counting-TM to decide this language as follows.

$\underline{C(< M_1 >, < M_2 >)}$:

1) run oracle($< M_1 >$) and oracle($< M_2 >$)
2) If either oracle outputs infinity or the sizes are not the same, then output **reject**

   $\underline{K(x)}$:

   1) if $M_1(y)$=accept and $M_2(y)$=accept then output **accept**
   2) else output **reject**
3) If oracle($< K >$)=oracle($< M_1 >$)=oracle($< M_2 >$) then output **accept**
4) Else output **reject**

First, we check that both languages are finite, and that they have the same cardinality. Then we construct K such that $L(K) = L(M_1) \cup L(M_2)$. If $L(M_1) = L(M_2)$, then the cardinality of $L(K)$ will be equivalent to the cardinality of $L(M_1)$ and $L(M_2)$. We can do this final check by consulting the counting oracle again.

d) **(15 points).** Prove that there exists a language that is not counting-decidable.

Note: We have not proven a recursion theorem for counting-TMs.

Answer:

(This is cited from class resources). We first note that every counting machine has a finite description: It can be described by the tuple $(Q, \Sigma, \Gamma, q_0, q^*, \delta)$ where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet, q_0 is the start state, q^* is the special non-halting accept state, and δ is the transition function. Thus, there exists an encoding of each counting machine in binary which consists of writing down the tuple notation for a counting machine and converting it into binary using ASCII.

Let P be the set of all counting machines. We first claim that $|P| = |\Sigma^*|$ for $\Sigma = \{0,1\}$.

4.3 4c 15 / 20

✓ **- 5 pts** **Needs more explanation**

You need a stronger explanation for why your decider is correct.

Counting Turing Machines

We now consider a Turing machine with a different oracle. A counting Turing machine (counting-TM) is a Turing machine with a "counting" oracle. Given a TM description $\langle N \rangle$, the counting oracle will tell the counting-TM the size of $L(N)$. More formally,

Definition 3. *We define a counting Turing machine (counting-TM) to be a three-tape TM that has a single all-purpose input/work/output tape, a query input tapes, and a query result tape.*

- *A counting-TM must have the states $q_{start}, q_{accept}, q_{reject}$ and a special state q_{query}.*
- *A counting-TM M can write the description of a standard TM $\langle N \rangle$ onto the query tape.*
- *If M reaches the special query state q_{query} and the query input tape content is $\langle N \rangle$ for any arbitrary standard TM N, then in a single atomic step both query tapes are wiped clean and the cardinality $|L(N)|$ is written to the query output tape. If $|L(N)|$ is finite, then the cardinality is written in unary as $1^{|L(N)|}$. If $|L(N)|$ is infinite, then a special ∞ symbol is written instead.*
- *In all other aspects, a counting-TM operates normally as a standard TM would.*

Definition 4. *A language $L \subseteq \{0,1\}^*$ is counting-decidable if there exists a counting-TM M that decides L. That is, $M(x)$ accepts if and only if $x \in L$ and $M(x)$ rejects if and only if $x \notin L$.*

c) **(20 points).** Prove that the following language is counting-decidable:

$$\text{FINITE-EQUIV} = \{(\langle M_1 \rangle, \langle M_2 \rangle) : L(M_1) = L(M_2),\ |L(M_1)| < \infty,\ |L(M_2)| < \infty,\ M_1 \text{ and } M_2 \text{ are standard TMs}\}.$$

Answer: We can construct a counting-TM to decide this language as follows.

$\underline{C(< M_1 >, < M_2 >)}$:

1) run oracle($< M_1 >$) and oracle($< M_2 >$)
2) If either oracle outputs infinity or the sizes are not the same, then output **reject**

   $\underline{K(x)}$:

   1) if $M_1(y)$=accept and $M_2(y)$=accept then output **accept**
   2) else output **reject**
3) If oracle($< K >$)=oracle($< M_1 >$)=oracle($< M_2 >$) then output **accept**
4) Else output **reject**

First, we check that both languages are finite, and that they have the same cardinality. Then we construct K such that $L(K) = L(M_1) \cup L(M_2)$. If $L(M_1) = L(M_2)$, then the cardinality of $L(K)$ will be equivalent to the cardinality of $L(M_1)$ and $L(M_2)$. We can do this final check by consulting the counting oracle again.

d) **(15 points).** Prove that there exists a language that is not counting-decidable.

Note: We have not proven a recursion theorem for counting-TMs.

Answer:

(This is cited from class resources). We first note that every counting machine has a finite description: It can be described by the tuple $(Q, \Sigma, \Gamma, q_0, q^*, \delta)$ where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite tape alphabet, q_0 is the start state, q^* is the special non-halting accept state, and δ is the transition function. Thus, there exists an encoding of each counting machine in binary which consists of writing down the tuple notation for a counting machine and converting it into binary using ASCII.

Let P be the set of all counting machines. We first claim that $|P| = |\Sigma^*|$ for $\Sigma = \{0,1\}$.

i. Consider $g : P \to \Sigma^*$ which maps a counting machine $p \in P$ to the encoding in Σ^* of the description of p (as described above). This is an injection since every counting machine has a different typle notation, and thus a different binary encoding.

ii. Consider $h : \Sigma^* \to P$ which maps a string $x \in \Sigma^*$ to the counting machine p_x which enters q^* an infinte number of times on string x and never enters q^* on any other string $y \neq x$. Then this is clearly an injection.

Since there is an injection in both directions, then there is a bijection $f : P \to \Sigma^*$ and thus $|P| = |\Sigma^*|$.

Let L be the set of all languages. Since we know that $|L| > |\Sigma^*| = |P|$, then by the infinite pigeonhole principle, there cannot be a surjection from $|P|$ to $|L|$. Thus, the function $f : P \to L$ defined by $f(p) = \{x | p(x) = \text{accept}\}$. So there must be some language $l \in L$ that is not equal to $f(P)$ for any $p \in P$. This means that there exists a language that is not computable by any counting machine.

4.4 4d 15 / 15

✓ - 0 pts Correct

5 Extra Credit (Narcissist) **0 / 50**

✓ **- 50 pts** **No Submission**

6 Honor Code 0 / 0

✓ - 0 pts Correct