CS 121 Homework 4: Fall 2020

Some policies: (See the course policy at `http://madhu.seas.harvard.edu/courses/Fall2020/policy.html` for the full policies.)

- **Collaboration:** You can collaborate with other students that are currently enrolled in this course (or, in the case of homework zero, planning to enroll in this course) in brainstorming and thinking through approaches to solutions but you should write the solutions on your own and cannot share them with other students.
- **Owning your solution:** Always make sure that you "own" your solutions to this other problem sets. That is, you should always first grapple with the problems on your own, and even if you participate in brainstorming sessions, make sure that you completely understand the ideas and details underlying the solution. This is in your interest as it ensures you have a solid understanding of the course material, and will help in the midterms and final. Getting 80% of the problem set questions right on your own will be much better to both your understanding than getting 100% of the questions through gathering hints from others without true understanding.
- **Serious violations:** Sharing questions or solutions with anyone outside this course, including posting on outside websites, is a violation of the honor code policy. Collaborating with anyone except students currently taking this course or using material from past years from this or other courses is a violation of the honor code policy.
- **Submission Format:** The submitted PDF should be typed and in the same format and pagination as ours. Please include the text of the problems and write **Solution X:** before your solution. Please mark in Gradescope the pages where the solution to each question appears. Points will be deducted if you submit in a different format.
- **Late Day Policy:** To give students some flexibility to manage your schedule, you are allowed a net total of **eight** late days through the semester, but you may not take more than **two** late days on any single problem set. No exceptions to this policy.

By writing my name here I affirm that I am aware of all policies and abided by them while working on this problem set:

Your name: Todd Morrill

Collaborators: (List here names of anyone you discussed problems or ideas for solutions with)

No. of late days used on previous psets (not including Homework Zero): 1
No. of late days used after including this pset: 1

Questions

Please solve the following problems. Some of these might be harder than the others, so don't despair if they require more time to think or you

bonus questions if you have the time to do so. If you don't have a proof for a certain statement, be upfront about it. You can always explain clearly what you are able to prove and the point at which you were stuck. You can always simply write **"I don't know"** and you will get 15 percent of the credit for this problem. If you are stuck on this problem set, you can use Piazza to send a private message to all staff.

Note on reading the textbook: If you are stuck on some of the problems, try consulting the book to 1) understand the concepts the question is referencing, and 2) review the way similar theorems are proved in the book.

Problem 0 (5 points): True or False: I have completed the midterm feedback survey. (True worth 5 points, False, or I don't know worth 0 points.)

Solution 0: True

Problem 1 (15 points): Let T be the Turing Machine with alphabet $\Sigma = \{\triangleright, 0, 1, \phi\}$ and transition function $\delta : [1] \times \Sigma \to [1] \times \Sigma \times \{L, R, S, H\}$ given by $\delta(0, \triangleright) =$ invalid, $\delta(0, 0) = (-, 1, H)$, $\delta(0, 1) = (0, 0, R)$ and $\delta(0, \phi) = (-, 1, H)$. Describe the function computed by T. (Hint: two cases that you may want to consider: (1) if the input and output of T are interpreted as integers written with least significant digit first (so $(x_0 \cdots x_{n-1})$ represents the integer $X = \sum_{i=0}^{n-1} x_i 2^i$), what is the output of T on input 121? (2) What is the output of T on input 000?)

Solution 1:

The function computed by T is $f : x \to x + 1$, where $x \in \mathbb{N}$. In other words, it is the function that increments its input by one.

Suppose we interpret the inputs to T as natural numbers written with least significant digit first. The Turing Machine T reads bits from left to right and considers two cases:

1) the machine reads a 1, in which case a 0 is written to the tape to signify that a carry is required (i.e. 1+1 results in 0 with a carry). This process repeats until either a 0 or a ϕ are read by the machine.

2) the machine reads a 0, in which case a 1 is written to the tape to signify that either a carried digit is being written *or* the first bit read by the machine is a 0 (i.e. the least significant bit). In either case, the input value is incremented by 1.

Once the machine reads a 0 or a ϕ, it halts because it has finished incrementing the input value by one. To be sure, in solving this problem, we're using the convention that once T halts, it returns all characters in $\{0, 1\}$ between the start symbol $\triangleright$ and the first ϕ.

Problem 2: In this problem we work with functions over a ternary alphabet, i.e., $f : \{0, 1, \#\}^* \to \{0, 1, \#\}^*$. What this means is that

1. An # symbol on the tape is a valid input symbol, and
2. The output of the machine is the concatenation of all the symbols from $\{0, 1, \#\}$ on the tape when the machine halts.

(Outside this problem, we normally use # as a blank character which is not part of the output.)

Problem 2.1 (25 points): Give a Turing Machine with alphabet $\Sigma = \{\triangleright, 0, 1, \#, \phi\}$ to compute the function $\text{Clean}_\# : \{0, 1, \#\}^* \to \{0, 1, \#\}^*$ which "erases" all the #'s on the tape. Specifically if $x_0, \ldots, x_{n-1} \in \{0, 1, \#\}$ and $0 \leq i_0 < i_1 < \cdots < i_{k-1} < n$ are indices such that for all $j \in [k]$, $x_{i_j} \in \{0, 1\}$ and for all $\ell \notin \{i_0, \ldots, i_{k-1}\}$, $x_\ell = \#$, then $\text{Clean}_\#(x_0, \ldots, x_{n-1}) = x_{i_0} \cdots x_{i_{k-1}}$.

Solution 2.1:

A rough sketch of the machine before showing the implementation details is as follows. The Turing Machine T shown below reads characters from the tape from left to right. When the machine encounters a 0 or a 1, it keeps moving to the right. When the machine encounters a #, it scans to the right looking for a 0 or 1. If it finds a 0 or a 1, it replaces that 0 or 1 with a #, remembers which character in $\{0, 1\}$ that it read and scans back to the left to replace the # with the character it remembered. This process is repeated until the terminating character ϕ is encountered. This means that all characters in $\{0, 1\}$ have been moved to the left. At this point the machine scans back to left, skipping all the #s until it reads a 0 or a 1 and halts. The final step is to replace all the #s on the right side of the tape with ϕs. To be sure, in solving this problem, we're using the convention that once T halts, it returns all characters in $\{0, 1, \#\}$ between the start symbol $\triangleright$ and the first ϕ symbol.

Alphabet: $\Sigma = \{\triangleright, 0, 1, \#, \phi\}$

States:
0 - Start state, scanning right for #
1 - Read #, scanning right for 0 or 1
2 - Read 0 after #, scanning left for 0 or 1
3 - Write 0
4 - Read 1 after #, scanning left for 0 or 1
5 - Write 1
6 - Read ϕ, scan left to halt

State/Input	$\triangleright$	0	1	#	ϕ
0	invalid	(0, 0, R)	(0, 1, R)	(1, #, R)	(6, ϕ, L)
1	invalid	(2, #, L)	(4, #, L)	(1, #, R)	(6, ϕ, L)
2	(3, $\triangleright$, R)	(3, 0, R)	(3, 1, R)	(2, #, L)	invalid
3	invalid	invalid	invalid	(0, 0, R)	invalid
2	(5, $\triangleright$, R)	(5, 0, R)	(5, 1, R)	(4, #, L)	invalid
3	invalid	invalid	invalid	(0, 1, R)	invalid
6	(-, $\triangleright$, H)	(-, 0, H)	(-, 1, H)	(6, ϕ, L)	invalid

Problem 2.2 (15 points): Let $f : \{0,1\}^* \to \{0,1\}^*$ be computed by a Turing Machine M with k states. Let $g : \{0,1\}^* \to \{0,1\}^*$ be computed by a Turing Machine N with ℓ states. Show that the function $h(x) = g(f(x))$ can be computed by a Turing Machine with $k + \ell + 1000000$ states.

Solution 2.2:

We can compose two turing machines M_f and M_g, where M_f and M_g denote the turing machines that compute the functions f and g, respectively, to generate a Turing Machine M that computes $h(x) = g(f(x))$. The procedure to compose two turing machines is as follows.

The input $x \in \{0,1\}^*$ to M will be the input to M_f. Machine M will start in M_f's start state and compute function f. For all transitions that cause M_f to halt, the machine will transition to a new state called "left 1" that simply moves the read head to the left to $i = 1$ where i is the index of the read head (i.e. move left to the start symbol $\triangleright$ and move one position to the right). The Turing Machine M can then transition to the start state of $M_{2.1}$, which is the machine desribed in problem 2.1 above, which clears the tape of all elements other than 0 or 1 using 7 states. For completeness, $M_{2.1}$ can be modified so that instead of being specific to erasing # it can be generalized to erase anything other than 0 or 1 on the tape by using the alphabet of M.

For all transitions that cause $M_{2.1}$ to halt, the machine will transition to a new state "left 2" that simply moves the read head to the left to $i = 1$. At this point, the tape is clean, the only thing remaining on the tape is the return value from M_f, and the read head is in position to start computing M_g. Machine M can then transition to the start state of M_g, which can run as it normally would.

In all, we used k states for M_f, ℓ states for M_g, 2 states for the "left *" states, and 7 states for $M_{2.1}$, which is $k + \ell + 9$ states, which is less than $k + \ell + 1000000$ states.

Problem 2.3 (10 points): Assume there exists a function $U : \{0,1\}^* \to \{0,1\}$ that is not computable. Let $f : \{0,1\}^* \to \{0,1\}^*$ be a computable function. For $g : \{0,1\}^* \to \{0,1\}^*$, let $h(x) = g(f(x))$. Prove or disprove the following statements:

1. For all such f, g, and h, if g is not computable, then h is not computable.
2. For all such f, g, and h, if h is not computable, then g is not computable.

Solution 2.3:

Solution 2.3.1:

The approach is to provide an example function h can be computed both with and without an uncomputable function g to disprove the claim.

Suppose h is the constant zero function that returns zero on all inputs, f is a function that for any input returns a string representation of a Turing Machine that loops infinitely on any input, and g is the uncomputable function $HALT : \{0,1\}^* \to \{0,1\}$ that returns 1 if the TM passed to $HALT$ halts on the passed input and returns 0 otherwise. As defined, h will always return 0 because g always returns 0 on a TM that loops infinitely.

Composing f and g to compute h we have:

```
def p_h(x):
    x = p_f(x) # returns string of TM that runs infinite while loop
    return p_g(x) # HALT(x, x), always returns 0
```

but h can be computed without relying on an uncomputable function as follows:

```
def p_h(x):
    x = p_f(x) # returns string of TM that runs infinite while loop
    return p_g'(x) # constant zero function
```

This shows that h can be computed even if g is not computable, which disproves the claim.

Solution 2.3.2:

We will prove the claim by contradiction. Suppose h couldn't be computed and g could be computed. We know that $h(x) = g(f(x))$. We also know from **solution 2.2** that two computable TMs can be composed by adding a constant number of states. Since f and g are computable and can be composed, we can compute h. But this is a contradiction because we assumed that h couldn't be computed, which is what we needed to show. From this we conclude that if h is not computable, then g is not computable.

Problem 3: In this multi-part question you will construct a Turing Machine to multiply integers. It will be convenient to work with functions over a ternary alphabet, i.e., $f : \{0, 1, @\}^* \to \{0, 1, @\}^*$. What this means is that

1. An @ symbol on the tape is a valid input symbol, and
2. The output of the machine is the concatenation of all the symbols from $\{0, 1, @\}$ on the tape when the machine halts.

We will also view strings in $\{0, 1\}^*$ as non-negative integers with least significant bit first. (E.g. we equate the string 011 with the integer 6 etc., and integer operations on strings will mean "Take the integer corresponding to this string, perform the given operation, and then reconvert the result to a string".)

Problem 3.1 (15 points): Let $f_1 : \{0, 1, @\}^* \to \{0, 1, @\}^*$ be the partial function given by $f_1(x@y@z) = (x - 1)@y@z$ where $x, y, z \in \{0, 1\}^*$ and $x \geq 1$. Give a Turing machine M_1 to compute f_1.

Solution 3.1:

A rough sketch of the machine before showing the implementation details is as follows. The Turing Machine M_1 shown below reads characters from the tape from left to right. When the machine encounters a 0 it replaces it with a 1, and when the machine encounters a 1 it replaces it with a 0 and halts. We strictly assume that the input to M_1 is greater than or equal to zero, hence the invalid result marked under the @ symbol. To be sure, in solving this problem, we're using the convention that once T halts, it returns all characters in $\{0, 1, @\}$ between the start symbol $\triangleright$ and the first ϕ symbol.

Alphabet: $\Sigma = \{\triangleright, 0, 1, @, \phi\}$

States:
0 - Start state, scanning right for 0 or 1

State/Input	$\triangleright$	0	1	@	ϕ
0	invalid	(0, 1, R)	(-, 0, H)	invalid	invalid

Problem 3.2 (Bonus, 0 points): Let $f_2 : \{0, 1, @\}^* \to \{0, 1, @\}^*$ be the partial function given by $f_2(x@y@z) = (x - 1)@y@(z + y)$ where $x, y, z \in \{0, 1\}^*$ and $x \geq 1$. Give a Turing machine M_2 to compute f_2. You may use the machines claimed in previous problems even if you have not constructed them.

Solution 3.2:

A rough sketch of the machine before showing the implementation details is as follows. The Turing Machine M_2 reuses the functionality of M_1 to decrement x, and then adds new states to add y to z. This machine adds bits one at a time from the least signficant bit of y and z and to the most signifcant bits of y and z. It uses an expanded alphabet that includes $\#_0$ and $\#_1$ to keep track of which bits from y have already been added to z. This process is repeated until all the bits of y have been added to z. At this point the machine enters a cleanup routine to restore y to its original value and halts. To be sure, in solving this problem, we're using the convention that once M_1 halts, it returns all characters in $\{0, 1\}$ between the start symbol $\triangleright$ and the first ϕ symbol.

Alphabet: $\Sigma = \{\triangleright, 0, 1, @, \#_0, \#_1, \phi\}$

States:
0 - Start state, scanning right for 0 or 1 (same as **Solution 3.1**)
1 - scan right for first @
2 - start state for 3.2, scan for 0 or 1
3 - read a 0 bit from y, scan right for second @
4 - read 0 from y, scanning right through z for 0 or 1
5 - read a 1 bit from y, scan right for second @
6 - read 1 from y, scanning right through z for 0 or 1
7 - scan left for second @
8 - scan left for a 0 or 1 in y
9 - scan left for first @ to stop
10 - clean right, replacing all $\#_0, \#_1$ with 0, 1, respectively, and halt

State/Input	$\triangleright$	0	1	$\#_0$	$\#_1$	@	ϕ
0	invalid	(0, 1, R)	(1, 0, R)	invalid	invalid	invalid	invalid
1	invalid	(1, 0, R)	(1, 1, R)	invalid	invalid	(2, @, R)	invalid
2	invalid	(3, $\#_0$, R)	(5, $\#_1$, R)	invalid	invalid	(9, @, L)	invalid
3	invalid	(3, 0, R)	(3, 1, R)	invalid	invalid	(4, @, R)	invalid
4	invalid	(7, $\#_0$, L)	(7, $\#_1$, L)	(4, $\#_0$, R)	(4, $\#_1$, R)	invalid	(7, $\#_0$, L)
5	invalid	(5, 0, R)	(5, 1, R)	invalid	invalid	(6, @, R)	invalid
6	invalid	(7, $\#_1$, L)	(6, $\#_0$, R)	(6, $\#_0$, R)	(6, $\#_1$, R)	invalid	(7, $\#_1$, L)
7	invalid	invalid	invalid	(7, $\#_0$, L)	(7, $\#_1$, L)	(8, @, L)	invalid
8	invalid	(8, 0, L)	(8, 1, L)	(2, $\#_0$, R)	(2, $\#_1$, R)	invalid	invalid
9	invalid	invalid	invalid	(9, $\#_0$, L)	(9, $\#_1$, L)	(10, @, R)	invalid
10	invalid	(10, 0, R)	(10, 1, R)	(10, 0, R)	(10, 1, R)	(10, @, R)	(-, ϕ, H)

Problem 3.3 (20 points): Describe in English the ingredients of a Turing Machine to compute the partial function $\times : \{0, 1, @\}^* \to \{0, 1\}^*$ given by $\times(x@y) = xy$. You may use the machines claimed in previous problems even if you have not constructed them.

Solution 3.3:

The solution relies on the TM specified in **solution 3.2**, which in turn relies on the solution specified in **solution 3.1**. A new start state is required to scan to the end of the input string and add a second @ symbol. Another state is needed to write a 0 after the newly added @ symbol, which acts as the z in the previous parts of problem 3. This is to handle the case when x is 0 (i.e. $xy = 0$). A third state called "scan left" is needed to scan back to the start of the input string (i.e. scan back to $\triangleright$). At this point the machine is in state 0 from M_2 above and starts one iteration of decrementing x and adding y to z at the end of the tape. Where M_2 halts, we should instead transition to the "scan left" state, write the character that was read, and move left. This completes one iteration of the loop.

This machine loops until $x = 0$, which implies that y has been added to z x times and $z = xy$. This is even the case when $y = 0$, since adding 0 x times is still 0. We know that $x = 0$ when @ is read from state 0 of M_2, which requires a small modification to M_2. If this occurs, we should transition to a new state that scans left to the start symbol, replacing everything it encounters with @ symbols (with the exception of $\triangleright$). Then transition to a new state that scans to the end of the newly written @s, and transitions to another state that replaces everything it encounters with @ symbols as it moves right, until it reads the second @ symbol, denoting the start of z. At this point, we can recycle the Turing Machine specified in **solution 2.1** that erases everything except 0 and 1, which requires 7 states, with a slight modification to the alphabet. Instead of erasing #s as in **solution 2.1**, we are erasing @ symbols and moving our solution $z = xy$ to the start of the tape. Once the machine describe in **solution 2.1** halts, the only thing remaining on the tape is $z = xy$ and we return the correct result.

Problem 3.4 (Bonus, 0 points): Give a complete Turing Machine to compute the partial function $\times$ described in Problem 3.3.

Solution 3.4:

Problem 3.5 (Bonus, 0 points): Give a Turing Machine to compute the partial function $\times$ described in Problem 3.3 for which, on an input of length n, the number of transitions it makes before halting is $O(n^{10})$. (You'll need a strategy different from the one suggested by 3.1 and 3.2.)

Solution 3.5:

Problem 4:

Problem 4.1 (15 points): Prove that there exists a function $f : \{0,1\}^{1000} \to \{0,1\}$ which is not computed by any Turing Machine with alphabet $\{\triangleright, 0, 1, \phi\}$ and at most 10 states. (Hint: Give an upper bound on the number of Turing Machines with k states as a function of k.)

Solution 4.1:

We compare the set of unique functions $F = \{f | f : \{0,1\}^{1000} \to \{0,1\}\}$ to the set of unique configurations of Turing Machines $C = \{\delta | \delta : [10] \times \Sigma \to [10] \times \Sigma \times [4]\}$ where 10 is the number of states and 4 corresponds to the set of actions $\{L, R, S, H\}$. We will now compare the cardinalities of F and C to show that there are more functions than Turing Machine configurations.

$$|F| = 2^{2^{1000}}$$
$$|C| = (10 \cdot |\Sigma| \cdot 4)^{10 \cdot |\Sigma|} = (10 \cdot 4 \cdot 4)^{10 \cdot 4} = 160^{40}$$

Since log is a monotonically increasing function, the result of taking the log of both sides retains their relation and makes the comparison easier.

$$\log_2 2^{2^{1000}} = 2^{1000}$$

$$\log_2 160^{40} = 40 \log_2 160 \approx 292.88$$
$$2^{1000} > 292.88$$

Since there are more functions than Turing Machine configurations (i.e. $|F| > |C|$), there exists a function $f : \{0,1\}^{1000} \to \{0,1\}$ which is not computed by any Turing Machine with alphabet $\{\triangleright, 0, 1, \phi\}$ and at most 10 states.

Problem 4.2 (5 points): Prove that every function $f : \{0,1\}^{1000} \to \{0,1\}$ is computed by a Turing Machine with alphabet $\{\triangleright, 0, 1, \phi\}$.

Solution 4.2:

We construct a Turing Machine that hardcodes all possible functions $f : \{0,1\}^{1000} \to \{0,1\}$. The key idea is to represent states as the bits read up to index $j \in [1000]$. For example, if the first 3 bits of the input to the Turing Machine are 010, then the machine is in state 010. This implies that a set of states is required for every for every $j \in [1000]$. In particular, we have sets indexed by j as follows: $s_0 = \{0,1\}^0, s_1 = \{0,1\}^1, s_2 = \{0,1\}^2, \cdots, s_{999} = \{0,1\}^{999}$. The set of states q for whole Turing Machine is the union of all of these sets as follows.

$$q = \bigcup_{j=0}^{999} \{0,1\}^j$$

The machine reads the input one bit at a time, transitioning through the states defined above, and when the machine is in a state in $s_{999} = \{0,1\}^{999}$, the next bit read (i.e. the 1000th bit) will determine which value is returned, 0 or 1, since this return value is hardcoded into the machine. If the return value is 0, the machine transitions to the "Return Zero" state that overwrites the inputs with ϕs and returns 0. The machine can do the same thing for the "Return One" state, which will return 1.