CSCC63 Assignment 1

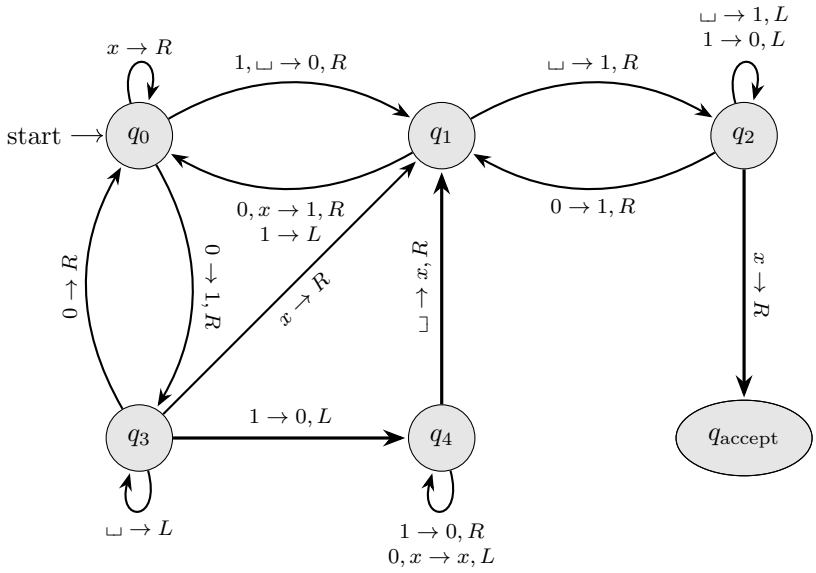
Turing Machines and Reductions

Due 11:59pm, June 07

Warning: *For this assignment you may work either alone or in pairs. Your electronic submission of a PDF to Crowdmark affirms that this assignment is your own work and that of your partner, and no one else's, and is also in accordance with the University of Toronto Code of Behaviour on Academic Matters, the Code of Student Conduct, and the guidelines for avoiding plagiarism in CSCC63. Note that using Google or any other online resource is considered plagiarism.*

1. (20 marks)

   (a) (15 marks) Let M_2 be the following TM. Suppose that M_2 is given the empty string as an input, so that the first configuration given this input is q_0. Write the first fifteen configurations that M_2 will take when you run it (this includes the initial configuration).



Solution:

Configuration 0	$q_0\sqcup$	Configuration 5	$1q_101$	Configuration 10	$11q_2001$
Configuration 1	$0q_1\sqcup$	Configuration 6	$11q_01$	Configuration 11	$111q_101$
Configuration 2	$01q_2\sqcup$	Configuration 7	$110q_1\sqcup$	Configuration 12	$1111q_01$
Configuration 3	$0q_211$	Configuration 8	$1101q_2\sqcup$	Configuration 13	$11110q_1\sqcup$
Configuration 4	q_2001	Configuration 9	$110q_211$	Configuration 14	$111101q_2\sqcup$

   (b) (5 marks) What is the is the largest number of characters by which any two neighbouring configurations C_i and C_{i+1} can differ for this TM? Why?

Solution:

The maximum number of characters by which two configurations can differ is three: if the TM writes a character nd moves left, e.g., in the arrow from q_3 to q_4, then the two configurations would be, e.g.,

$$\begin{array}{ccccc} \ldots & x & q_3 & 1 & \ldots \\ & & \Downarrow & & \\ \ldots & q_4 & x & 0 & \ldots \end{array}$$

and this gives us a difference of three. We can't get a difference of more than three, since the only changes we can ever see are on or immediately adjecent to the TM head, and there are only three characters of that sort.

Note: if you always write a single blank symbol at the end of the configuration you can technically get four changes if you overwrite the blank character and move left, e.g., in the q_4 to q_1 arrow. This could give us a pair of configurations that go, e.g.,

$$\begin{array}{cccccc} \ldots & x & q_4 & \sqcup \\ & & \Downarrow & \\ \ldots & q_1 & x & x & \sqcup \end{array}$$

We sometimes write our configurations this way (we effectively do this in our PCP reduction), and sometimes we don't (in our later proof of the Cook-Levin theorem). Since there's a small amount of ambiguity here, we'll accept either three of four as an answer, so long as the justification is reasonable.

2. (10 marks) Consider the language L_2:

$$L_2 = \left\{ \langle M \rangle \,\middle|\, \exists n \in \mathbb{N}, 0^n 1^n \notin L(M) \right\}.$$

Show that L_2 is not recognizable.

Solution:

We'll reduce from $\overline{\text{HALT}}$.

Let P = "On input $\langle M, w \rangle$:

1. Let M_0 = "On input $\langle x \rangle$:
   1. If $x \neq 01$ *accept.*
   2. Run M on w.
   3. *Accept.*
2. Return $\langle M_0 \rangle$.

Since we're writing M_0 it's OK to have it take inputs of the form $\{0,1\}^*$. We'll proceed under that assumption.

Proof that $\langle M, w \rangle \in \overline{\text{HALT}}$ iff $\langle M_0 \rangle \in L_2$:

$\Rightarrow$) Suppose that $\langle M, w \rangle \in \overline{\text{HALT}}$.
Then M loops on w.
Now, if $x \neq 01$, M will accept X in line 1.
On the other hnd, if $x = 01$, M_0 will loop on line 2. and not progress to line 3.
This means that M_0 will not accept the string $x = 01 = 0^1 1^1$.
So there is some $0^n 1^n$ not accepted by M_0.
$\Rightarrow \langle M_0 \rangle \in L_2$.

$\Leftarrow$) Suppose that $\langle M, w \rangle \notin \overline{\text{HALT}}$.
Then M halts on w.
This means that if we run M_0 on $x = 01$, it will run M on w on line 2. and proceed to line 3. When it does so it will accept.
On the other hand, if we run M_0 on any $x \neq 01$ it will accept on line 1.
So M_0 will accept all of its inputs, and particular will accept every input $x \in \left\{ 0^n 1^n \,\middle|\, n \in \mathbb{N} \right\}$.
$\Rightarrow \langle M_0 \rangle \notin L_2$.

■

3. (10 marks) Consider the language L_3:

$$L_3 = \{\langle M \rangle \mid \exists n \in \mathbb{N}, 0^n 1^n \in L(M)\}.$$

Show that L_3 is not co-recognizable.

Solution:

We'll reduce from HALT.

Let P = "On input $\langle M, w \rangle$:
1. Let M_0 = "On input $\langle x \rangle$:
   1. Run M on w.
   2. If $x = 01$, *accept*, otherwise *reject*.
2. Return $\langle M_0 \rangle$.

Proof that $\langle M, w \rangle \in$ HALT iff $\langle M_0 \rangle \in L_3$:

$\Rightarrow$) Suppose that $\langle M, w \rangle \in$ HALT.
Then M halts on w.
So for any input x, M_0 will finish running line 1. and progress to line 2.
In particular, this means that M_0 will accept the input $x = 01 = 0^1 1^1$.
So there is some $0^n 1^n$ accepted by M_0.
$\Rightarrow \langle M_0 \rangle \in L_3$.

$\Leftarrow$) Suppose that $\langle M, w \rangle \notin$ HALT.
Then M loops on w.
This means that if we run M_0 on any input x it will loop on line 1. and not proceed to line 2.
So M_0 will not accept any input.
In particular, M_0 will not accept accept any input $x \in \{0^n 1^n \mid n \in \mathbb{N}\}$.
$\Rightarrow \langle M_0 \rangle \notin L_3$.

■

4. (20 marks) Consider the language L_4:

$$L_4 = \{\langle M, k \rangle \mid \exists w \in \Sigma^*, \langle M, w, k \rangle \in \text{ACC}\}.$$

Is L_4 recognizable, co-recognizable, neither, or decidable? Prove your answer.

Solution:

This language is actually decidable – it is both recognizable and co-recognizable. To see why, let $\langle M, k \rangle$ be an instance of L_4 and consider the action of M on any string of length $|w| > k$.

We can write any such w as $a \cdot b$, where a is a string of length k and b is non-empty string.

Now, as we run M it will move its tape head over the different cells in its tape, but in any one step if can only move its head by one step at a time. So:

- In the first step M will look at the first cell in its tape (index 0).
- In the second step M might look at either the first or second cell in its tape. It can't look any further, since it can only have moved one step from its initial point (the maximum index is 1).
- In the third step M may be looking at the first, second, or third tape cell. The furthest right it can be is one step right from its furthest possible point in step two (the maximum index is 2).
- ...

Continuing inductively we see that in step i of its run M must have its tape head in index j, where $j < i$. In particular, if we run M for k steps it can only ever look at the first k cells on its tape.

So when we run M on $w = a \cdot b$, it will never see b within its first k steps. Since the behaviour of M depends only on the contents of the tape cells it sees, the action of M will be the same on its first k steps if we run it on $w = a \cdot b$ as if we were to run it on a alone. In particular, if M does not accept a within k steps, then neither will it accept $w = a \cdot b$. In order to check whenther $\langle M, k\rangle \in L_4$ we only need to check the strings up to length k.

Recognizer for L_4:

Let $w_0, w_1, \ldots$ be an effective enumeration of Σ^*.

Let $R =$ "On input $\langle M, k\rangle$:
1. For $i = 0$ to ∞:
2. Run M on w_i for k steps.
3. If it accepts, *accept*."

If $\langle M, k\rangle \in L_4$:

Then there is some $j \in \mathbb{N}$ such that M accepts w_j within k steps.
If R has not accepted by the loop iteration $i = j$, then it will run M on w_j for k steps in line 2 and accept in line 3.
So R will accept $\langle M, k\rangle$.

If $\langle M, k\rangle \notin L_4$:

Then there is no $j \in \mathbb{N}$ such that M accepts w_j within k steps.
So for any loop iteration $i = j$, R run M on w_j for k steps in line 2 and not accept in line 3.
Since this is true for every loop iteration, and since this is the only way that R can accept, R will never accept $\langle M, k\rangle$.

■

Recognizer for $\overline{L_4}$:

Let $w_0, w_1, \ldots$ be the shortlex enumeration of Σ^*.

Let $R =$ "On input $\langle M, k\rangle$:
1. For $i = 0$ to ∞:
2. If $|w_i| > k$, *accept*.
3. Else:
4. Run M on w_i for k steps.
5. If it accepts, *reject*."

If $\langle M, k\rangle \in \overline{L_4}$:

Then $\langle M, k\rangle \notin L_4$.
There is some $j \in \mathbb{N}$ such that M accepts w_j within k steps.
In particular, there is no $j \in \mathbb{N}$ such that $|w_j| \leqslant k$ and M accepts w_j within k steps.
This means that on every loop iteration of R, M will not accept w_i within k steps.
This means that R will not reject in any of its loop iterations and so will eventually reach a loop iteration in which $|w_i| > k$, t which point it will accept in line 2.
So R will accept $\langle M, k\rangle$.

If $\langle M, k\rangle \notin \overline{L_4}$:

Then $\langle M, k\rangle \in L_4$.
So there is some $j \in \mathbb{N}$ such that M accepts w_j within k steps.
We have two possibilities: either (1) $|w_j| < k$ or (2) there is some $j' \in \mathbb{N}$ and non-empty $b \in \Sigma^*$ such that $|w_{j'}| = k$ and $w_j = w_{j'} \cdot b$.

(1) If $|w_j| \leqslant k$ then R will reach the loop iteration j before it reaches any i such that $|w_i| > k$ (note that we're using a shortlex order).
So R cannot accept before reaching the loop iteration $i = j$, at which point it will run M on w_j for k steps and reject in line 5.

(2) If $|w_j| > k$ then the operation of M on its first k steps is the same as it would be for $w_{j'}$.
In particular, if M accepts w_j then it also accepts $w_{j'}$.
This means that the first case will apply, and so R will reject $\langle M, k\rangle$.

In either case we see that R will reject $\langle M, k\rangle$.

■

5. (20 marks) Consider the language L_5:

$$L_5 = \big\{\langle M\rangle \,\big|\, \overline{L(M)} \subseteq \{0^n1^n \,|\, n \in \mathbb{N}\}\big\}.$$

Is L_5 recognizable, co-recognizable, neither, or decidable? Prove your answer.

Solution:

This question if mostly a test as to how well you can read and understand a somewhat convoluted lnguage description. Let's start our answer by looking at what, exactly, L_5 is. A TM description $\langle M\rangle$ is in L_5 iff $\overline{L(M)} \subseteq \{0^n1^n \,|\, n \in \mathbb{N}\}$. That is, for ny input x:

- If x is *not* of the form 0^n1^n, we must accept, otherwise $\overline{L(M)}$ would contain an x not of the form 0^n1^n.
- If x is of the form 0^n1^n, we really don't care what we do with it. The language description asserts that every string we don't accept is of the form 0^n1^n, but not that we can't accept any string of this form.

So a perhaps simpler way of writing L_5 would be

$$\begin{aligned} L_5 &= \big\{\langle M\rangle \,\big|\, \forall x \in \Sigma^*, (x \in \{0^n1^n \,|\, n \in \mathbb{N}\}) \vee (x \in L(M)\big)\big\} \\ &= \big\{\langle M\rangle \,\big|\, \forall x \in \Sigma^*, \exists k \in \mathbb{N}, (x \in \{0^n1^n \,|\, n \in \mathbb{N}\}) \vee (\langle M, x, k\rangle \in \mathrm{ACC})\big\}. \end{aligned}$$

Note that we can decide whether x is of the form 0^n1^n, so we don't include a quantifier for that.

If we look t it this way, we'd guess that it's likely neither recognizable nor co-recognizable. Let's try to prove this.

Proof that L_5 is not recognizable:

We'll reduce from $\overline{\mathrm{HALT}}$:
Let P = "On input $\langle M, w\rangle$:
1. Let M_0 = "On input $\langle x\rangle$:
    1. Let $k = |x|$.
    2. Run M on w for k steps.
    3. If it hapts, *loop*, otherwise, *accept*."
2. Return $\langle M_0\rangle$.

Proof that $\langle M, w\rangle \in \overline{\mathrm{HALT}}$ iff $\langle M_0\rangle \in L_5$:

$\Rightarrow$) Suppose that $\langle M, w\rangle \in \overline{\mathrm{HALT}}$:
Then M loops on w.
So for any $x \in \Sigma^*$, M will not halt on w within $|x|$ steps.
This means that for any input x, M_0 will accept on line 3.
In particular, if $x \notin \{0^n1^n \,|\, n \in \mathbb{N}\}$, then M will accept x.
So $\overline{L(M)} \subseteq \{0^n1^n \,|\, n \in \mathbb{N}\}$.
$\Rightarrow \langle M_0\rangle \in L_5$.

⇐) Suppose that $\langle M, w\rangle \notin \overline{\text{HALT}}$:
Then M halts on w within k steps for some $k \in \mathbb{N}$.
Suppose we run M_0 on $x = 1^{k+1}$.

Note: We don't really need the +1, actually, since we state that TMs always run for at least one step. But it's there in principle in case $k = 0$, since then $1^k = \varepsilon = 0^k 1^k$. I'm putting it here so people reading the solutions know it's something that might come up in other questions.

Then M_0 will run M on w for $k + 1$ steps in line 2.
M_0 will then loop on line 3, so it will not accept x.
But x is not of the form $0^n 1^n$ for any $n \in \mathbb{N}$.
So $\overline{L(M)} \not\subseteq \{0^n 1^n \mid n \in \mathbb{N}\}$.
$\Rightarrow \langle M_0\rangle \notin L_5$.

■

Proof that L_5 is not co-recognizable:

We'll reduce from HALT.

Let $P =$ "On input $\langle M, w\rangle$:
1. Let $M_0 =$ "On input $\langle x\rangle$:
    1. If $x \neq 1111$, *accept*.
    2. Run M on w.
    3. *Accept*.
2. Return $\langle M_0\rangle$.

Proof that $\langle M, w\rangle \in$ HALT iff $\langle M_0\rangle \in L_5$:

⇒) Suppose that $\langle M, w\rangle \in$ HALT:
Then M halts on w.
So if we run M_0 on any input $x \neq 1111$, M_0 will halt and accept on line 1.
On the other hand, if $x = 1111$, M will halt on line 2. and M_0 will accept in line 3.
So M_0 will accept every input.
In particular, it will accept every $x \notin \{0^n 1^n \mid n \in \mathbb{N}\}$.
$\Rightarrow \langle M_0\rangle \in L_5$

⇐) $\langle M, w\rangle \notin$ HALT:
Then M loops on w.
So if we run M_0 on the input $x = 1111$ (note that $x \notin \{0^n 1^n \mid n \in \mathbb{N}\}$), M_0 will loop on line 2.
So M_0 will not accept $x = 1111$, and so there is some $x \notin \{0^n 1^n \mid n \in \mathbb{N}\}$ that M_0 will not accept.
$\Rightarrow \langle M_0\rangle \notin L_5$

■

6. (10 marks) Suppose that you come across a new type of TM – a *plane TM*. A plane TM P is a lot like a regular TM in that it has a set Q of internal states, an input and tape alphabet Σ and Γ, a start state q_0, and designated *accept* and *reject* states q_A and q_R.

However, the tape of a plane TM P is a 4-way infinite two-dimensional plane: there is a cell for every x, y in $\mathbb{Z}^2$. Moreover, each of these cells contains a string in Γ^* instead of a single character. When we run P on n input $w \in \Sigma^*$, we set the string in cell $(0, 0)$ to w and every other cell to ε.

Consider any step of P in which we are in a state q_a and cell (i, j) of the tape.

- If P is in the accept state q_A it halts and accepts. If it is in the reject state q_R it halts and rejects.
- Otherwise, it reads the *final* character of the strings in the cells $(i-1, j-1), (i, j-1), (i+1, j-1), (i-1, j), (i, j), (i+1, j), (i-1, j+1, i, j+1)$, and $(i+1, j+1)$ (i.e., in the 3×3 square centered at (i, j)). The transition function δ of P can depend on all of these cells, as well as the state q_a.

  When reading the final character of the empty string ε, we treat it as a blank character $\sqcup$.

- The transition function can tell P to either write any charcter in Γ to the end of the string in (i, j), or to erase the final character in that string. If we try to erase the final character in an empty string we just get the empty string again.

  After writing or erasing a charcter, P can change its state to a new state $q_{a'}$ and move its tape head to a new cell in the 3×3 square centered at (i, j) (so it can move to any of the cells it has just read from).

  As usual, if there is no transition available for P to take it just goes into the reject state.

To answer this question, show how, given a plane TM P, you would write a regular TM M such that on any input w:

- If P accepts w, then so does M.
- If P rejects w, then so does M.
- If P loops on w, then so does M.

Your answer should include the pseudocode for the TM M and a justification (not a full proof) that your code is correct. If you write a multitape TM, describe what the different tapes are used for and how the data on those tapes will be formatted.

Solution:

Given a plane TM P, we'll write a multitape TM M. We'll give it four tapes:

- The first tape will contain each cell of the plane that we have visited (i.e., that the head of P has directly visited in its run), ordered by a bijection $f : \mathbb{N} \mapsto \mathbb{Z}^2$. We will describe this bijection shortly. If $(i, j) = f(n)$ is written on the tape and if $(i', j') = f(n')$ for some $n < n'$, then we will write (i', j') to our tape as well. The tape will be formatted as follows:

  $$\#0\$w_0\#1\$w_w\# \ldots \#i\$w_i\# \ldots,$$

  where i is the index of the cell (the number n denoting which cell we are looking at), and w_i is the string we keep in the cell.

  The bijection $f : \mathbb{N} \mapsto \mathbb{Z}^2$ will be defined first by giving a bijection $f_1 : \mathbb{N} \mapsto \mathbb{N}^2$, and then by composing this with a bijection $f_2 : \mathbb{N} \mapsto \mathbb{Z}$.

  Let's start with the bijection f_1. Given an $n \in \mathbb{N}$, let $t(n) = \sum_{i=0}^{n} i$, and let $s(n)$ be the largest number in $\mathbb{N}$ such that $t(s(n)) \leqslant n$. So the first few values of $t(n)$ and $s(n)$ are:

n	0	1	2	3	4	5	6	7	8	9	10	...
$s(n)$	0	1	1	2	2	2	3	3	3	3	4	...
$t(s(n))$	0	1	1	3	3	3	6	6	6	6	10	...

  We note that for all $n \in \mathbb{N}, t(s(n)) \leqslant n < t(s(n) + 1)$ (this follows immediately from the definition of $s(n)$).

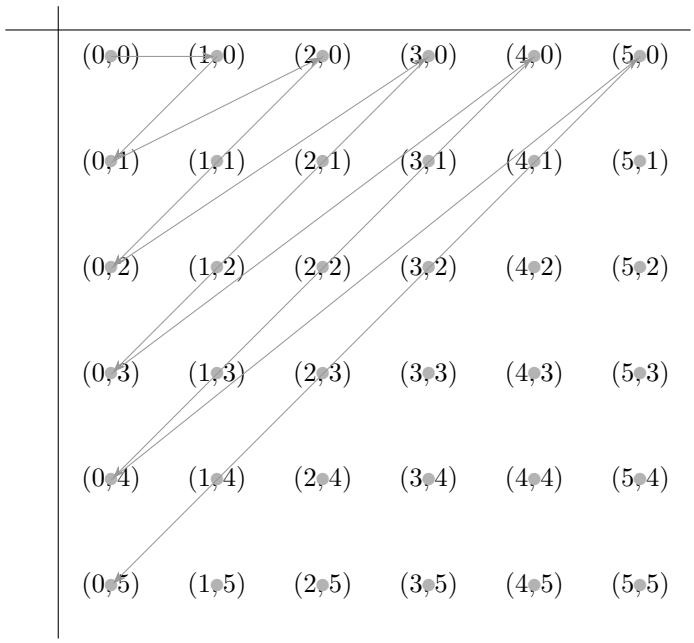
  Furthermore, since $t(s(n) + 1) - t(s(n)) = s(n) + 1$ (by the definition of $t(n)$), we can see that there are exactly $s(n) + 1$ numbers m such that $t(s(n)) \leqslant m < t(s(n) + 1)$: the numbers $\{t(s(n)), t(s(n)) + 1, \ldots, t(s(n)) + s(n)\}$. for all of these numbers, $t(s(m)) = t(s(n))$.

  So, given $n \in \mathbb{N}$, we can define the numbers $a(n)$ and $b(n)$ as follows:

  $$a(n) = t(s(n)) + s(n) - n,$$
  $$b(n) = n - t(s(n)).$$

  We note that for any $n \in \mathbb{N}$, $a(n) + b(n) = s(n)$.

  Our bijection $f_1 : \mathbb{N} \mapsto \mathbb{N}^2$ is defined as $f_1 = (a(n), b(n))$. Basically, the function f_1 enumerates the pairs of natural numbers by iteratively listing the diagonals:



Our second function f_2 will map $\mathbb{N}$ to $\mathbb{Z}$:

$$f_2(n) = \begin{cases} \frac{n}{2}, & n \text{ is even,} \\ -\frac{n+1}{2}, & n \text{ is odd.} \end{cases}$$

So our overall f will be

$$f(n) = (f_2(a(n)), f_2(b(n))).$$

In the question we ask for a justifiction of correctness and not a full proof, so we'll skip the proof that this is indeed a bijection. You are encouraged to check this property for yourselves, though.

- The second tape will hold the current address of the planar TM head.
- The third tape will hold the address on the plane that we wish to read.
- The fourth tape will be used for scratch work.

We'll generally remember the state of P by using the states of M: there will be one copy of the TM that updates P for every state in P, where the copy we're currently using indicates the current state of P.

Suppose that we want to emulate a run of P on the input w. To initialize our TM M, we set the first tape to #0\$$w$# and the second tape to 0. The third and fourth tapes will be empty, and we will initialize P to the start state q_0.

Recall that M will use its states to remember the current state of P. We'll have one copy of the following steps for every such state. To simulate one step of P, we will follow these steps:

1) Let n be the current address in tape two. For each pair p in

$$\{(-1,1),(0,1),(1,1),(-1,0),(0,0),(1,0),(-1,-1),(0,-1),(1,-1)\}:$$

- We copy $f^{-1}(f(n)+p)$ into tape three. We'll always use this order.
  Also remember that we allow arithmetic operations, so we'll be able to implement these steps.
- For each of these new addresses, we move the tape head of tape one to its left-hand side (note that we can find the left side of the tape by looking for the address 0).

- We then walk right on the tape until we see an address corresponding to the string on tape three. If such an address if found we move to the string tht is located at this address and read its last character. If no such address is found we read the blank character.

We remember which characters we've seen as we go, as well as the current state of P, by using states in M (this is the same process as we used in the multitape to single-tape construction).

2) Once we've read all of the characters (and remember the current state), we have enough information to determine which rule in δ to follow.
   - We again move the head for tape one to the left-hand side, and we move it right until we see the address i for the current cell. We then move to the associated string w_i.
   - If the rule from δ tells us to append a character to the end of the current cell, we add a character to w_i and shift every character to the right of w_i on tape one right by one.

     If the rule from δ tells us to delete from w_i and $w_i \neq \varepsilon$, we shift every character to the right of w_i on tape one left by one.

     If the rule from δ tells us to delete from w_i and $w_i = \varepsilon$, we don't change w_i.
   - In order to change the position of the head cell, we read the next move p in

     $$\{(-1,1),(0,1),(1,1),(-1,0),(0,0),(1,0),(-1,-1),(0,-1),(1,-1)\}$$

     from δ and we copy $f^{-1}(f(n)+p)$ into tape two.
     We then move the head of tape one to the left-hand side and then move it to the right until we see an address that matches tape two. If no such address is found by the end of tape one we add blank cells until we reach the desired address, using tape three to keep track of the current rightmost address on tape one.
   - We then delete the contents of tapes three and four, set M so that it remembers the new state of P, and are ready to start the next step.

   Note that the question asks for a justification of the program correctness, not a full proof. So what we do here will sketch the steps we'd take to formally prove correctness, but we won't go into as much detail as we might otherwise require.

We can justify the correctness of this approach as follows: suppose that M has a recorded state for P, an address in tape two, and a set of strings in tape one that correctly model the configuration of P at the beginning of its i^{th} step. Then after running our update steps, M will correctly hold the state, tape two address, and plane strings in tape one to encode P in its $(i+1)^{st}$ step. Since M starts by encoding the correct initial state of P into M, we can see inductively that M will correctly encode every configuration of P. In particular, if P every enters its accept state then so does M, if P ever enters its reject state then so does M, and if P never enters either of these states then neither does M. So M correctly emulates P, as required.

□

7. (10 marks) **Note: for this question you'll need to have covered oracles and the Arithmetic Hierarchy. If the notes for this aren't up they'll be up soon.**

In class we've said that problems like the halting problem are not solvable using algorithms, where algorithms are programs that must halt and return the correct answer in a finite number of steps. But what happens if we allow TMs to loop?

First of all, how can a TM even return an answer if it loops?

Let's consider the following modification of TM: A **looping** TM M is like a regulr TM, but instead of the usual *accept* and *reject* states it has a write-only output tape, a *true* state q_T and a *false* state q_F. The states q_T and q_F are not halting states – they have outgoing arrows and act like every other state in M, except:

- Whenever M enters the q_T state it prints a T to the output state and moves the output head one step to the right.
- Whenever M enters the q_F state it prints a F to the output state.

If M reaches a point where there is no arrow to follow it goes into q_F as usual, but as we have said it doesn't halt.

That is, M periodically outputs its best current guess as to whether its input is accepted or not. Since it does not halt, it can change its answer.

As before, M will take finite strings w as inputs. We'll say that M *converges to a true* on input w iff, when run on w, there is some time n_0 in which it prints a T to its output, and for every $n > n_0$ it does not print a F to its output. Similarly, M *converges to a false* on input w iff, when run on w, there is some time n_0 in which it prints a F to its output, and for every $n > n_0$ it does not print a T to its output. In either case we say it *converges*. If M does not converge on w then we say that it *diverges*.

We'll say that M accepts a string w if it converges to a true on w, and we'll set

$$L(M) = \{\langle w\rangle \mid M \text{ accepts } w\}.$$

We'll say that M recognies L if $L = L(M)$, and that M decides L if it recognies it and converges for all inputs.

The important distinction of this model is that if M decides L then it will always converge to the right answer, but we'll in general never know when. We lose the ability to ever be sure that we know the answers to our questions. You can see why we don't use this model to describe computing – when we write programs we generally want to be certain when we've gotten to our answers. But there are situtions in which a guarantee of convergence is the best we can do, so it's worth looking at this model.

It turns out the looping TMs are strongly equivalent to regular OTMs that use oracles for the language HALT: If you give me a looping TM M then I can give you an OTM N^{HALT} such that for all inputs w:

- If M accepts w then so does N^{HALT}.
- If M converges to a false on w then N^{HALT} rejects.
- If M diverges on w then N^{HALT} loops.

The converse also holds. So the languages decidable by looping TMs are eactly the class Δ_2 and the languages recognizable by looping TMs are exactly Σ_2 (a similar statement can be made about co-recognizability and Π_2)

To answer this question, show how, given a looping TM M, you could build such an OTM N^{HALT}. Your answer should include the pseudocode for the OTM N^{HALT} and a justification (not a full proof) that your code is correct. You do not need to show me the states or the tape format for N^{HALT}.

Solution:

Given the looping TM M, let's define a family M_i of TMs:

$M_i =$"On input w:
1. Let $count = 0$.
2. For $j = 0$ to ∞:
3. Run M on w for j steps.
4. If M has printed to its output tape in step j:
5. Let s be the string on the output tape of M.
6. If $|s| == 1$ or $s[-1]! = s[-2]$, $count$++.
7. If $count > i$, *accept*."

Basically, M_i runs M until it switches its answer $i + 1$ times, after which it halts and accepts.

We make the following claims:

Claim 1: If M converges to an accept on input w, then there is some i_0 such that M_i halts and accepts on w for all $i \leqslant i_0$, such that M_{i_0+1} loops on w, and such that the rightmost cell in the output tape of M_{i_0} upon halting is a T.

Claim 2: If M converges to a reject on input w, then there is some i_0 such that M_i halts and accepts on w for all $i \leqslant i_0$, such that M_{i_0+1} loops on w, and such that the rightmost cell in the output tape of M_{i_0} upon halting is a F.

Claim 3: If M writes at least one character to its output but still diverges, then M_i accepts for every $i \in \mathbb{N}$.

Claim 4: If M does not write any character to its output tape on input w, then M_i does not halt on w for any $i \in \mathbb{N}$.

Since the question does not ask for a full proof, we won't write the full proofs of these claims, but we can see that they re reasonable.

Now, consider the following OTM:

Let $N^{\text{HALT}} =$"On input w:
1. If M_0 does not halt on w (use oracle here), then loop forever.
2. For $i = 1$ to ∞:
3. If M_i does not halt on w:
4. Run M_{i-1} on w.
5. If the last cell of its output tape is a T, *accept*, otherwise *reject*."

Let $w \in \Sigma^*$, and recall that we have four possibilities when we run M on w:

(1) There is some time step n_0 in which M writes a T onto the output tape, and for every time step $n \geqslant n_0$, M does not write any F to the output tape.

If this case occurs, then by **Claim 1** we have that there is some i_0 such that M_i halts and accepts on w for all $i \leqslant i_0$, and such that M_{i_0+1} loops on w. So N^{HALT} will halt after the loop iteration $i = i_0 + 1$, and it will accept iff the rightmost character on the output tape of M after running M_{i_0} is a T. But also by **Claim 1** this character is a T, and so N^{HALT} accepts w.

So if M converges to an accept, then N^{HALT} accepts w.

(2) There is some time step n_0 in which M writes a F onto the output tape, and for every time step $n \geqslant n_0$, M does not write any T to the output tape.

If this case occurs, then by **Claim 2** we have that there is some i_0 such that M_i halts and accepts on w for all $i \leqslant i_0$, and such that M_{i_0+1} loops on w. So N^{HALT} will halt after the loop iteration $i = i_0 + 1$, and it will accept iff the rightmost character on the output tape of M after running M_{i_0} is a T, and will reject otherwise. But also by **Claim 2** this character is a F, and so N^{HALT} rejects w.

So if M converges to a reject, then N^{HALT} rejects w.

(3) M writes at least one character to its output tape, and for every time step n_0 in which it writes a character b to its tape, there is a time step $n_1 > n_0$ in which it writes an $\overline{b}$.

If this case occurs, then by **Claim 3** M_i will accept w for all $i \in \mathbb{N}$, and so N^{HALT} will loop forever on lines 2. to 5.

So if M writes to its output tape at least once and oscillates, then N^{HALT} loops.

(4) M never writes to its output tape.

If this case occurs, then By **Claim 4** M_0 loops on w, and so N^{HALT} loops on line 1.
So if M never writes to its tape and oscillates, then N^{HALT} loops.

So all told, if M converges to an accept then N^{HALT} accepts, if M converges to a reject then N^{HALT} rejects, and if M oscillates then N^{HALT} loops, as required.

□

8. **Bonus** (10 marks — your mark will be rounded to the nearest multiple of 2.5)

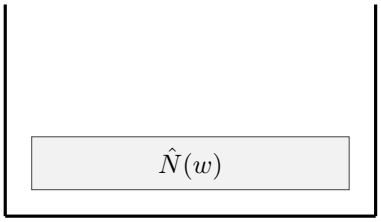
Show the converse to the previous question: Given an OTM N^{HALT}, show me how you would write a looping TM M such that:

- If N^{HALT} accepts w then so does M.
- If N^{HALT} rejects w then M converges to a false on w.
- If N^{HALT} loops on w then M diverges.

Your answer should include the pseudocode for the looping TM M and a justification (not a full proof) that your code is correct. You do not need to show me the states or the tape format for M.

Solution:

Let N^{HALT} be an OTM that uses a HALT oracle. We'll want to build an equivalent looping TM M. The basic idea we'll use to write M is as follows: we'll set up a looping TM with a stack of programs that it will run (specifically, that is will dovetail over). Initially this stack will contain only the simulated OTM $\hat{N}$ run on w:



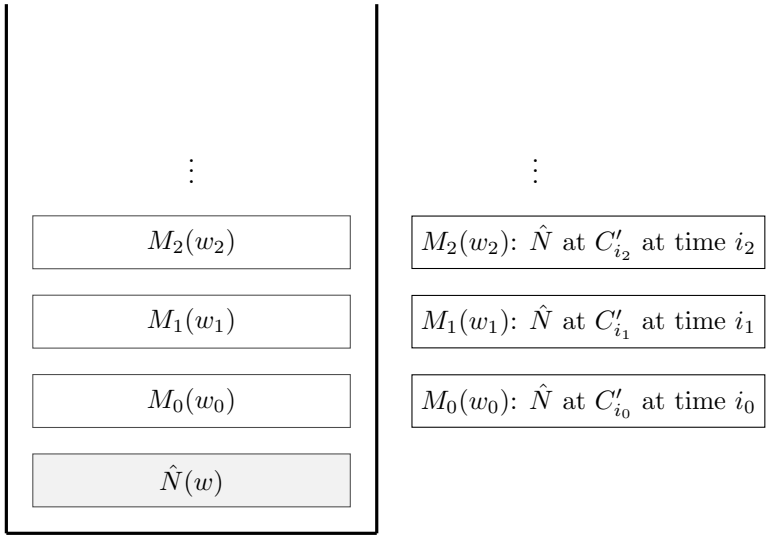
We'll dovetail over every program in this stack – that is, on any single iteration of the dovetailing loop we'll update each program in this stack by one step. If the program we're updating is a regular TM (not an OTM), then we update it normally.

Note that there is no such TM in the stack right now, but there will be soon.

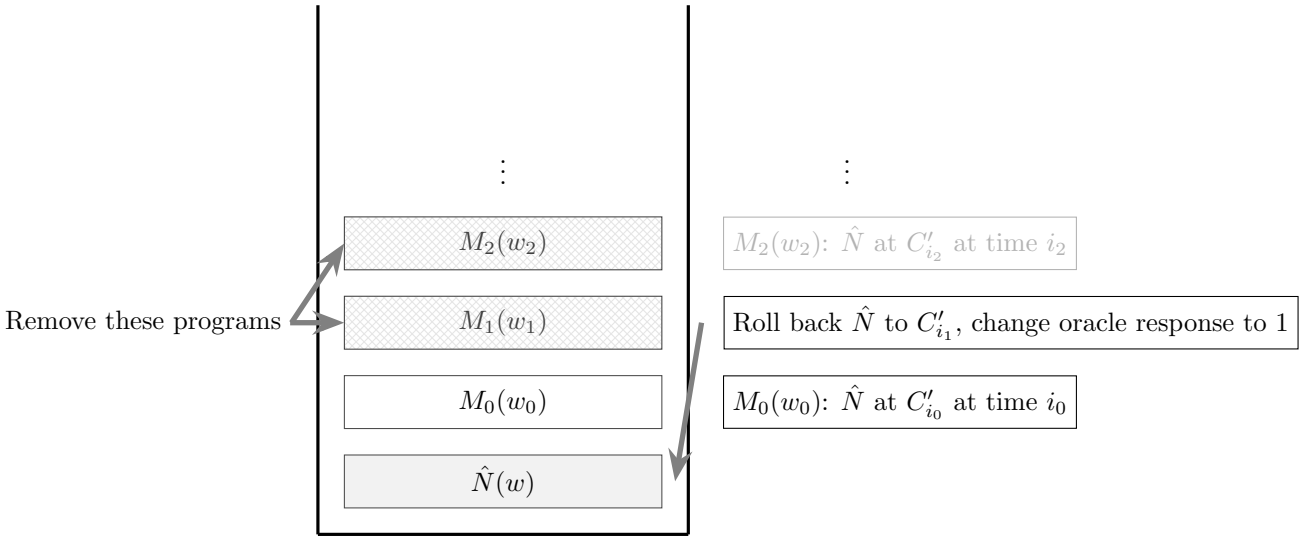
If the program is the simulated OTM $\hat{N}$, then we'll update it normally if there is no oracle query in the current step. If there is an oracle query in step i, say, we'll initially assume that said query represents a looping TM/input pair $\langle M_i, w_i \rangle$. To ensure that this guess is correct, we'll add the pair $\langle M_i, w_i \rangle$ to the program stack. To help with the process of "checking" our oracle guess, we'll also keep track of:

- The computation history $C'_0, C'_1, \ldots$ of $\hat{N}$.
- The time step in which we added $\langle M_i, w_i \rangle$ to the stack.

So we end up with something like this:



As we have said, though, we're assuming that every oracle call is for a loping TM/input pair. This will clearly not generally be the case. If one pf a the pairs is not looping – e.g., if $\langle M_0, w_0 \rangle$ is a looping pair in the example here, but $\langle M_1, w_1 \rangle$ halts, then as we dovetail over the program in our stack we'll eventually see $M_1(w_1)$ reach a halting configuration. At this point we know that our assumed oracle response was wrong for this step, and that therefore the configuration C'_{i_1} *and all subsequent* $\hat{N}$ *configurations and oracle calls* must be incorrect. So we'll remove those oracle calls from the stack and roll back the computation of $\hat{N}$ to the configuration C'_{i_1} – we'll simply correct C'_{i_1} so that the oracle tape contains a 1 rather than the 0 that we initially assumed.



That's basically the idea of our program – we'll just set things up so that in every dovetailing iteration, M outputs one or more symbols onto the output tape:

- If $\hat{N}$ is in an accepting configuration, M prints a T to its tape.
- If $\hat{N}$ is in a rejecting configuration, M prints a F to its tape.
- Otherwise, M prints a TF to its tape.

With all of this in mind, consider the following looping TM:

Let $M =$"On input $\langle w \rangle$:

1. Let $\hat{N}$ be a simulated copy of N^{HALT} initialized to the configuration $C_0' = q_0 w$.
2. Let *tasks* be an empty stack of programs
   and *history* be an empty stack of configurations for $\hat{N}$.
3. Push $\hat{N}$, initialized to the configuration $C_0' = q_0 w$, onto *tasks* and C_0' onto *history*.
4. Let $count = 0$ and $d = []$.
   We'll be treating d as a dictionary here.
5. For $n = 0$ to ∞:
   Main Loop
6. $count$++.
7. For each program p in *tasks*:
8. If $p = \hat{N}$:
9. If $\hat{N}$ is not in the state $q_?$:
10. Update $\hat{N}$ by one step.
   If it is a halting configuration, just repeat that configuration.
11. Else:
12. Let $\langle M_i, w_i \rangle$ be the string on the oracle tape.
13. Push M_i onto *tasks* with the initial configuration $q_0 w_i$.
14. If $\langle M_i, w_i \rangle \notin d.keys()$, set $d[\langle M_i, w_i \rangle] =$*count*.
15. Set the oracle tape of $\hat{N}$ to "0".
16. More the oracle tape head of $\hat{N}$ to the leftmost cell and set the state to $q_!$.
17. Append the configuration C'_{count} of $\hat{N}$ to *history*.
18. Else:
19. Update p by one step.
20. If it halts:
21. Set $count = d[p]$.
22. For $p' \in d.keys()$:
23. If $d[p'] \geqslant count$, remove p' from d.
24. Pop from *tasks* until p it does not contain p.
25. Set the configuration of $\hat{N}$ to $history[count]$.
25. Remove all C_i' from *history* for $i \geqslant count$
26. Set the oracle tape of $\hat{N}$ to "1"
27. Append the configuration C'_{count} of $\hat{N}$ to *history*.
28. If $\hat{N}$ is in an accepting state:
29. Write T to the output tape.
30. Else if $\hat{N}$ is in a rejecting state:
31. Write F to the output tape.
32. Else:
33. Write TF to the output tape."

In order to analyze the behaviour of this program, let's define the predicate $P_w(i)$ for $i\ in \mathbb{N}$ so that $P(i)$ is true iff, on input $w \in \Sigma^*$, there is some iteration $n = n_i$ of the main loop from lines 5. to 33. of M such that, at the beginning of said loop iteration:

- $count = i$.
- *history* contains exactly the true configurations of N^{HALT} when run on w for its first $0 \leqslant j \leqslant i$ steps, i.e.,
$$history = [C_0, C_1, \ldots, C_i].$$
- *tasks* contains exactly $\hat{N}$ and the looping TM/input pairs $\langle M_j, w_j \rangle$ sent to the oracle in the first i steps of its run on w, i.e., $tasks = \Phi^{-}_{M^{\text{HALT}}, w, i}$.
- For all loop iterations $n > n_i$, $count > i$, and the configurations C_j' for $0 \leqslant j \leqslant i$ are not changed.

A full proof of program correctness would use induction to show $P(i)$ holds for all $i \in \mathbb{N}$, but we'll leave it here with the statement of the predicate.

We can see that this will imply that M does indeed emulate N^{HALT}:

- If N^{HALT} accepts w it will do so within k steps for some $k \in \mathbb{N}$. Since $P(k)$ is true, once M has run long enough on w, $\hat{N}$ will have reached the true configuration C_k and it will remain in that configuration forever. At this point M will print a T in every step and never agin print a F.
- If N^{HALT} rejects w it will do so within k steps for some $k \in \mathbb{N}$. Since $P(k)$ is true, once M has run long enough on w, $\hat{N}$ will have reached the true configuration C_k and it will remain in that configuration forever. At this point M will print a F in every step and never agin print a T.
- If N^{HALT} loops on w, then consider what happens if we run M (as opposed to $\hat{N}$) on w for n_0 steps for any $n_0 \in \mathbb{N}$. Since N^{HALT} loops on w, it cannot have halted within $n_0 + 1$ steps. Since $P(n_0 + 1)$ holds we know that M will eventually set $\hat{N}$ so the correct configuration C_{n_0+1} and print TF. Since $\hat{N}$ cannot progress by more than one step in any time step of M, the point in which M sets $\hat{N}$ to C_{n_0+1} must be later than step n_0. So M must switch its answer at least once after time step n_0, and since this is true for every time n_0, it must diverge.

□

Why These Last Two Questions are Interesting

What the last two questions in this assignment really do is revisit the basic assumptions that we used to start the class: we've argued that any physically feasible form of computation can be emulated (with some slowdown) by a TM, and so TMs are a reasonable formalism to encode computation. As far as we can tell today, this ssumption is valid.

But we've also defined the solvable languages as the class of decidable lnguages – i.e., as the class of languages that can be calculated by TMs that halt on every input. What we're doing here is relaxing that assumption. Instead of requiring our TMs to halt in a finite amount of time and return the correct answer, we're going to allow TMs that instead guarantee that they will eventually settle down to the right answer, even if there is no way of telling when we've reached that point.

Now, we usually don't want this sort of program: if we're doing any calculation that requires us to be precise in our calculations (for example, if we're trying to do our taxes) we can't afford the possibility that our program will switch its answer after we've read it. On the other hand, there are situations in which this sort of convergent program is useful. We may, for example, want to run a statistical program that will (assuming tht we have access to a series of fair coin flips) sample from some complex distribution. If said distribution is difficult enough, we may not be able to sample from it directly, but we can sometimes sample from a second distribution that will over time converge to the one we want. If you've looked at AI or other forms of computational statistics, we can see *Gibbs sampling* as being something like this sort of algorithm.

So algorithms that converge in the limit do exist in the real world, and it's reasonable to ask what computability and complexity classes they can encompass. What we've done in these two questions is argue that if we don't put any running time or memory constraints on our algorithms, then convergent (looping) TMs can "solve" problems in Δ_2, and can recognize languages up to Σ_2 in the Arithmetic Hierarchy. In a sense, allowing our TMs to converge to their answers gives them the power of a HALT oracle, but at the cost of our never knowing for sure when we've reached the right answer. We only know that we'll eventually get there.