Safian Omar Qureshi
ID 10086638

CPSC 313 - Assignment 3

1. a) **Simulate a RAM Turing machine by a standard Turing machine**

Suppose that there is a standard Turing machine M' = (Q', Σ, Γ', δ', q_0, q_{accept}, q_{reject}) whose language is L ⊆ Σ*. Consider a RAM Turing machine M = (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}) that will be simulated by standard Turing machine. Now we define the 7-tuple for the Turing machines M, M':

q_0, q_{accept}, q_{reject}: are the start, accept and reject states for both Turing machines M', M.

Σ: is the input alphabet Σ = {σ_1,σ_2,…,σ_m} for m >= 1 for both Turing machines M, M'.

Q, Q': set of states for the Turing machines M, M' specifically Q ⊆ Q' with Q' having some additional states for simulating operations.

Γ, Γ': tape alphabet for the Turing machines M, M' specifically Γ = {0,1,2,3,4,5,6,7,8,9,R,W} and Γ' = Γ∪{♠}, where R, W ∉ Σ are symbols for initiating read, write instructions and ♠ ∉ Γ.

δ, δ': transition functions for Turing machines M, M'.

The standard Turing machine M' must have work tapes:

Memory tape: this is an array A[0], A[1], . . . , where each memory cell A[k] takes an integer value i, such that i ≥ 0 to store contents of registers.

Location tape: this array tracks currently visible register on memory tape.

Input tape: the array containing initial string input and is an identical copy to M's input tape

Address tape: the array address and is identical copy to M's address tape but uses ♠ symbols to enclose the input.

Initialization

All cells are filled with blanks first. Initialization starts with writing 0 onto second most leftmost cell on location tape so that tape head for memory tape is pointing to its leftmost cell A[0].

If the machine M’ is in state q_{access}, then if the non-blank part of the address tape stores a string in the form

μR - where $\mu \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$ is the unpadded decimal representation of some integer $i \geq 0$, then the address tape is erased (that is, filled with blanks), and the symbol σ currently stored in the array entry A[i] is then written onto the leftmost cell of this tape — with the tape head pointing to this symbol. The Turing machine moves to $q_{complete}$, after having completed this read operation.

μWσ - where $\mu \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$ is the unpadded decimal representation of some integer $i \geq 0$, and σ is a symbol in Γ that is not in {R,W}, then A[i] is set to be (i.e., overwritten with) σ. The address tape is erased, with the tape head pointing to the leftmost cell on this tape. The Turing machine moves to $q_{complete}$, after having completed this write operation.

Otherwise, if the non-blank part of the address tape stores anything else, then the instructions on it are considered to be invalid so the address tape is erased (without the array A being either accessed or modified), and its tape head is moved to the leftmost cell on the tape. The machine moves to state $q_{complete}$, once again.

Simulate read

Suppose M enters state q_{access}, when memory tape stores μR where $\mu \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$, $i \geq 0$. Then M reads A[i]. Suppose location tape also stores unpadded decimal representation of an integer $j \geq 0$, Then we have three cases

Case $i < j$: cell of memory tape storing A[i] is left of current location of memory tape head. Then the read algorithm is

```
While (i != j)
        j = j - 1
        move memory tape head one position to left position A[j]
```

This ensure memory tape head points to cell A[i] on termination, as required.

Case $i > j$: cell of memory tape storing A[i] is left of current location of memory tape head. Then the read algorithm is

```
While (i != j)
        j = j + 1
        move memory tape head one position to right position A[j]
```

This ensure memory tape head points to cell A[i] on termination, as required.

Case i = j: either this is initially the case or after above algorithms execute. The address tape is erased and symbol that is stored in A[i] is written on leftmost cell of this tape. Non blank of address tape is now ♠σ♠.

Simulate write

Suppose M enters state q_{access}, when memory tape stores μWσ where μ ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*, i ≥ 0. Then M writes σ into A[i]. Suppose location tape also stores unpadded decimal representation of an integer j ≥ 0, Then we have three cases as before

Case i < j: cell of memory tape storing A[i] is left of current location of memory tape head. Then the write algorithm is

```
While (i != j)
        j = j - 1
        move memory tape head one position to left position A[j]
Write σ
```

This ensure memory tape head points to cell A[i] on termination and then writes σ, as required.

Case i > j: cell of memory tape storing A[i] is left of current location of memory tape head. Then the write algorithm is

```
While (i != j)
        j = j + 1
        move memory tape head one position to right position A[j]

Write σ
```

This ensure memory tape head points to cell A[i] on termination and then writes σ, as required.

Case i = j: either this is initially the case or after above algorithms execute. Location tape now stores unpadded decimal representation of i, memory tape head points to A[i] and after writing the σ, address tape should be updated/erased so it contains blanks.

Simulating other operations

Suppose M enters state q_{access}, when address tape stores anything else than above. Address tape should then be erased to the form of ♠ ⊔ ♠, containing infinite blanks to the right.

Suppose M enters a state other than q_{access}, then M' should have similar moves:

- ♠ symbol on address tape is visible after move left, then a move right is required
- ♠ symbol on address tape is visible after move right, then ♠ should be replaced by blank and then another ♠ written to the right

During moves not part of simulation with M, M' contents and tape heads of memory/address tapes must remain unchanged

Using induction to prove correctness

M makes k >= 0 moves when executed on input string $w \in \Sigma^*$. M' simulates these k moves and doing so results in

M, M' being in the same state
M' memory tape stores A[0], A[1], …, contents
M' location tape stores unpadded decimal representation of integer i>=0 and memory tape head of M' points to location of A[i]
M address tape stores string w, M' address tape stores the string w with ♠ to the left, some n>=0 blanks, another ♠, and then infinite blanks
Other work tapes of M have the same content and locations of tape heads to the work tapes of M'.

As such, given the entirety above, we can claim that M accepts w iff M' accepts w, M rejects w iff M' rejects w, M loops on w iff M' loops on w.

Thus, it is shown if there is a standard Turing machine M' with a given language $L \subseteq \Sigma^*$ then there is a RAM Turing machine M with language $L \subseteq \Sigma^*$ as well. Furthermore, if standard Turing machine M' decides $L \subseteq \Sigma^*$ then M decides $L \subseteq \Sigma^*$ as well as required.

1. b) **Simulate a standard Turing machine with RAM Turing machine**

Suppose that there is a RAM Turing machine M' = (Q', Σ, Γ', δ', q_0, q_{accept}, q_{reject}) whose language is $L \subseteq \Sigma^*$. Consider a standard Turing machine M = (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}) that will be simulated by a RAM Turing machine M'. Then M has

Same set of states with q_{access}, $q_{complete}$ also added arbitrarily
Same transition functions out of $q_{complete}$ into $q_{complete}$ keeping contents and tape heads same and all other work tapes are same.

M is a copy of M' that does not use memory tape. The number of moves made by M doesn't allow M to reach q_{access}, $q_{complete}$ on an input string w, M recognizes same language and if M decides $L \subseteq \Sigma^*$ then RAM machine decides $L \subseteq \Sigma^*$ too.

Thus, if a RAM Turing machine decides a language $L \subseteq \Sigma^*$, a standard Turing machine M decides $L \subseteq \Sigma^*$ as well.

2. **Proof by reduction**

Reduce a known undecidable language to L_{Blank} to show that it implies L_{Blank} being undecidable. From tutorial #17, we know that language $Nonempty_{TM} \subseteq \Sigma^*_{TM}$ was proved to be undecidable. Let that known undecidable language be that, the set of encodings of Turing machines M such that $L(M) = \varnothing$ (M accepts at least one string of symbols over its input alphabet), and reduce L_{Blank} to it.

A many one reduction function is required where $f: \Sigma^*_{TM} \rightarrow \Sigma^*_{TM}$ from $Nonempty_{TM}$ to L_{Blank}. Let $v \in \Sigma^*_{TM}$.

Case 1: Suppose $v \notin TM$. Then $v \notin Nonempty_{TM}$ and $v \notin L_{Blank}$ so $f(v) = v$, concluding $f(v) \notin L_{Blank}$ as shown.

Case 2: Suppose $v \in TM$. v encodes $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ and f(v) encodes $M' = (Q', \Sigma, \Gamma', \delta', q_0, q_{accept}, q_{reject})$. We must now define $Q, Q', \Sigma, \Gamma, \Gamma', \delta, \delta'$.

$q_0, q_{accept}, q_{reject}$: are the start, accept and reject states for both Turing machines M', M.

Σ: is the input alphabet for Turing machines M, M', same for both $\Sigma = \{\sigma_1,\sigma_2,\ldots,\sigma_m\}$ for m >= 1.

Q, Q': set of states for the Turing machines M, M' specifically if $Q = \{q_0, q_1, q_2 \ldots, q_k, q_{accept}, q_{reject}\}$ then $Q' = \{q_0, q_1, q_2 \ldots, q_k, q_{k+1}, q_{k+2}, q_{k+3}, q_{accept}, q_{reject}\}$ for k>=0 (Q is a subset of Q' with Q' having three new states).

Γ, Γ': tape alphabet for the Turing machines M, M' specifically if $\Gamma = \{\sigma_1,\sigma_2,\ldots,\sigma_m, \sigma_{m+1}\ldots\ldots,\sigma_n\}$ then $\Gamma' = \{\sigma_1,\sigma_2,\ldots,\sigma_m, \sigma_{m+1}\ldots\ldots,\sigma_n, \sigma_{n+1}\}$ (Γ is a subset of Γ' with Γ' having one additional symbol). σ_{n+1} is a non blank replacement for $\sqcup$ and will be designated as $\sqcup'$.

δ, δ': transition functions for Turing machines M, M' specifically if δ is a transition function of M, then δ' is a transition function of M' with three changes in the following order:

> 1) for all states $q \in Q$ but $q \notin\{q_{accept}, q_{reject}\}$ and for all $\sigma \in \Gamma$, if $\delta(q,\sigma) = (r, \sqcup, m)$ then $\delta'(q,\sigma) = (r, \sqcup', m)$ where $r \in Q$ and $m \in \{L,R\}$.
>
> 2) for all states $q \in Q$ but $q \notin\{q_{accept}, q_{reject}\}$ and for all $\sigma \in \Gamma$, $\delta'(q, \sqcup') = \delta'(q, \sqcup)$
>
> 3) if $q \in Q$ such that $q \notin\{q_{accept}, q_{reject}\}$ and $\sigma \in \Gamma'$ and $\delta' = (q_{accept}, t, m)$, then $\delta'(q,\sigma) = (q_{k+1}, t, m)$ where $t \in \Gamma'$ and $m \in \{L,R\}$.

In addition, the three new states in Q' for Turing machine M' needs to have the corresponding transition functions:

i) $\delta'(q_{k+1},\sigma) = (q_{k+2}, \sqcup', R)$, ii) $\delta'(q_{k+2},\sigma) = (q_{k+3}, \sigma, L)$, iii) $\delta'(q_{k+3},\sigma) = (q_{accept}, \sqcup, R)$ for all $\sigma \in \Gamma'$.

We have now fully defined our Turing machines M, M' and can work on unravelling definitions and claim certain properties.

Sub-claim: Let $w \in \Sigma^*$ be an input of M. Let n be some arbitrary amount of moves M makes on input. Let M not accept w after n moves. Then M' makes at least n moves on the same input and does not accept w either. As such, M' has not yet written a blank ⊔ onto its tape during those first moves, $f(v) \notin L_{Blank}$.

Sub-proof: using induction on n by applying transition functions δ' for M', showing configuration of each machine at incremental integers, full proof available in lecture notes #25 slides 15-22 and tutorial 16. So we have $f(v) \notin L_{Blank}$ as required.

Using the above, we can similarly claim that if M rejects or loops on w, then M' never writes a blank ⊔ on top of a non blank symbol ⊔' when running on the same input w. This is because M never enters accept state and again $f(v) \notin L_{Blank}$ as required.

We can also say if v encodes a Turing machine M with an empty language, then M rejects or loops on every string w, so $f(v) \notin L_{Blank}$ as required

Sub-claim: Let $w \in \Sigma^*$ be an input of M. Let n>=1 be some arbitrary integer amount of moves M makes on input. Let M accept input w after n moves. Then M' writes a blank ⊔ on top of a non blank symbol ⊔' when running on input w so $f(v) \in L_{Blank}$.

Sub-proof: Suppose $w \in \Sigma^*$ and M accepts w. After n moves on input w, M enters q_{accept}. Consider the configuration just before being in the accept state, for some n - 1 moves so that q is some state that is not the accept state. Then correspondingly, M' is also in some state q after running n - 1 moves, specifically it will be in state q_{k+1}. Then, we use the transition function δ' for M' on the input, specifically the ones above being i, ii and iii. Informally, it proceeds as writing nonblank symbol ⊔', M' change state, move right, change state again, move left and writing a blank ⊔ on top of ⊔' as it changes state finally to $q_{accept.}$ Thus, M' writes a blank ⊔ on top of nonblank ⊔' when running on w and we have shown $f(v) \in L_{Blank}$ as required.

Given the above, we can say if $v \in Nonempty_{TM}$ then v encodes Turing machine M that accepts at least one input string w for the input language. Since it does accept at least input string, we have $f(v) \in L_{Blank}$ as required.

Sub-claim: If $v \in \Sigma^*_{TM}$ and $v \notin Nonempty_{TM}$ then $f(v) \notin L_{Blank}$.

Sub-proof: Suppose: $v \in \Sigma^*_{TM}$ and $v \notin Nonempty_{TM}$ then we have two cases:

> Case A: if $v \notin TM$ then we have $f(v) = v$ so $f(v) \notin TM$. Since L_{Blank} is a subset of TM, it must be the case $f(v) \notin L_{Blank}$ as required.
>
> Case B: if $v \in TM$ and $v \notin Nonempty_{TM}$ then v encodes a Turing machine M with an empty language $L(M) = \varnothing$ and that was shown to be $f(v) \notin L_{Blank}$ above.

Now we can show that the many one reduction function required where $f: \Sigma^*_{TM} \rightarrow \Sigma^*_{TM}$ from $Nonempty_{TM}$ to L_{Blank} is a computable function.

Subproof: the language TM is known to be decidable. We can determine whether or not if $v \in TM$ in algorithm to find f(v) from an input string. When $v \notin TM$, we have $v = f(v)$ so f(v) can be determined from v whenever $v \notin TM$.

Suppose instead $v \in TM$, v encodes some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ and f(v) encodes $M' = (Q', \Sigma, \Gamma', \delta', q_0, q_{accept}, q_{reject})$. Suppose the corresponding input strings for the Turing machines would be x in form (code(Q), code(Σ), code(Γ), code(δ)) and f(x) string form (code(Q'), code(Σ), code(Γ'), code(δ')). We can then separate the strings into substrings:

code(Q) is a substring of (code(Q), code(Σ), code(Γ), code(δ)) and Q is the set of states for M, $\{q_0, q_1, q_2 \ldots, q_k, q_{accept}, q_{reject}\}$. code(Q') is substring of (code(Q'), code(Σ), code(Γ'), code(δ')) and Q' is the set of states for M', $\{q_0, q_1, q_2 \ldots, q_k, q_{k+1}, q_{k+2}, q_{k+3}, q_{accept}, q_{reject}\}$. We can let the substrings be binary representations of integer k, and so code(Q') can be computed from code(Q) by counter addition.

code(Σ) is a substring of both the strings, so it can obviously be computed from itself.

code(Γ) is a substring of (code(Q), code(Σ), code(Γ), code(δ)) and $\Gamma = \{\sigma_1,\sigma_2,\ldots,\sigma_m, \sigma_{m+1\ldots}\ldots,\sigma_n\}$. code(Γ') is substring of (code(Q'), code(Σ), code(Γ'), code(δ')) and $\Gamma' = \{\sigma_1,\sigma_2,\ldots,\sigma_m, \sigma_{m+1\ldots}\ldots,\sigma_n, \sigma_{n+1}\}$. We can let the substrings be binary representations of integer n, and so code(Γ') can be computed from code(Γ) by counter addition.

code(δ) is a substring of (code(Q), code(Σ), code(Γ), code(δ)) and δ was defined above. code(δ') is substring of (code(Q'), code(Σ), code(Γ'), code(δ')) and δ' was also defined above. code(δ') can be computed from code(δ) by a Turing machine as they are merely sequences of strings, following the description of the transition functions given above. This shows that function f is indeed computable, as required.

Finally, to summarize claims, we have:

- f is a many one reduction function where $f: \Sigma^*_{TM} \rightarrow \Sigma^*_{TM}$ from Nonempty_{TM} to L_{Blank}
- if $v \in \text{Nonempty}_{TM}$ then $f(v) \in L_{Blank}$
- if $v \notin \text{Nonempty}_{TM}$ then $f(v) \notin L_{Blank}$
- this function f is computable

Hence, since Nonempty_{TM} is undecidable, it is implied L_{Blank} is undecidable too by using reduction.