

Problem Set #2

1. (a) $L = \{M : M \text{ has an useless state } q\}$

Language L is undecidable. Suppose L was decidable then we can use that as a sub routine to solve \overline{HP} as follows:

- Consider a new machine \dot{M} which has two states - start and accept state. It writes its input y on a separate tape.
- It then runs M on input x . If M halts on x it accepts its input y .

The machine \dot{M} can be described as :

$$\dot{M} = \begin{cases} \dot{M} \text{ has no useless state} & \text{if } M \text{ halts on } x \\ \dot{M} \text{ has an useless accept state} & \text{if } M \text{ does not halt on } x \end{cases}$$

Since our assumption is that L is decidable implies there exists a total turing machine K which accepts L . Giving \dot{M} as input to K , we can totally compute if M halts on x . But we know there is no such total turing machine. Hence, language L is not decidable.

- (b) $L = \{(M, w) : M \text{ moves its head left during computation of } w\}$

This language is decidable. Suppose, during computation of w , the head moves left once, we are done and can accept (M, w) .

If it doesn't move left till it reaches the first blank symbol - Let q_1 be the state at which it first reads the blank symbol. Let N be the number of states of machine M . Simulate the machine for a further $N + 1$ steps. If the head moved left during this time then accept (M, w) . Else reject. Because :

$\delta(q_1, \text{blank}) = q_j \forall j \in [1, N]$. By pigeon hole principle, it should return to a previously visited state after $N + 1$ steps.

- (c) $L = \{(M1, M2) : L(M1) = \overline{L(M2)}\}$

L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:

Consider two machines $M1$ and $M2$ which does the following:

- Both machines simulate M on x . If M halts on x then machine $M1$ accepts its input y , while machine $M2$ rejects input y .

So the language of machine $M1$ can be described as :

$$L(M1) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Similarly the language accepted by machine $M2$ can be described as:

$$L(M2) = \begin{cases} \emptyset & \text{if } M \text{ halts on } x \\ \Sigma^* & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving machines $(M1, M2)$ as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

- (d) $L = \{M : \exists x \in \Sigma^* \text{ s.t. } M \text{ runs forever on input } x\}$

This language is undecidable. If this language was decidable by a total turing machine K , then we can use it as a sub routine to solve \overline{HP} as follows:

Consider machine \dot{M} which does the following:

- \dot{M} writes its input y on a separate track.
- It runs machine M on input x . If M halts on x , then it accepts its input y .

The machine \dot{M} can be described as follows:

$$\dot{M} = \begin{cases} \text{For all inputs } \dot{M} \text{ doesn't run forever} & \text{if } M \text{ halts on } x \\ \text{There exists an input for which } \dot{M} \text{ runs forever} & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving \dot{M} as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total machine, hence no such total turing machine K exists.

- (e) $L = \{M : M \text{ accepts atleast one palindromic string}\}$

L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:

Consider machine \dot{M} which does the following:

- Machine \dot{M} simulates M on input x .
- If M halts on x , then \dot{M} accepts its input y

So the language of machine M can be described as :

$$L(M) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Since, Σ^* contains atleast one palindromic string, Giving machine M as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

- (f) $L = \{M : M \text{ accepts only palindromic strings}\}$

L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:

Consider machine \dot{M} which does the following:

- Machine \dot{M} simulates M on input x .
- If M halts on x , then \dot{M} checks if its input y is a palindrome. If yes, it accepts y , else rejects y .

So the language of machine M can be described as :

$$L(M) = \begin{cases} P & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Where P is the set of all palindromes.

Giving machine M as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

(g) $L = \{(M1, M2): L(M1) \cap L(M2) \neq \emptyset\}$

L is undecidable. If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:

Consider two machines $M1$ and $M2$ which does the following:

- Both machines simulate M on x . If M halts on x then both machines $M1$ and $M2$ accepts their inputs $y1, y2$ respectively.

So the language of machine $M1$ can be described as :

$$L(M1) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Similarly the language accepted by machine $M2$ can be described as:

$$L(M2) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

And the Language $L(M1) \cap L(M2)$ can be described as :

$$L(M1) \cap L(M2) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving machines $(M1, M2)$ as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

2. (107) $L = \{M : L(M) = rev L(M)\}$

This language is undecidable.

If L was decidable we can use the total turing machine K accepting L as a sub routine to solve HP as follows:

Consider machine \hat{M} which does the following:

- Machine \hat{M} simulates M on input x .
- If M halts on x , then \hat{M} checks if its input y is a palindrome. If yes, it accepts y , else rejects y .

So the language of machine M can be described as :

$$L(M) = \begin{cases} P & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Where P is the set of all palindromes.

Giving machine M as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

(109)(a) $L = \{(M, N) : M \text{ takes fewer steps than } N \text{ on } \epsilon\}$

This language is recursively enumerable. Simulate machine M and N on ϵ . If, M halts before N , then accept. If N halts before M , then reject.

(109)(b) $L = \{M : M \text{ takes fewer than } 481^{481} \text{ steps on some input}\}$

This language is recursively enumerable. On input y simulate the machine M . If it doesn't accept or reject in 481^{481} steps of the input, accept. Else, reject and go to the next input. Enumerate the input y in lexicographic ordering.

(109)(c) $L = \{M : M \text{ takes fewer than } 481^{481} \text{ steps on at least } 481^{481} \text{ inputs}\}$

This language is r.e. Simulate the machine on input y . If after 481^{481} steps, if the machine doesn't accept or reject, then increment a counter and move to next input. Else, reject and go to next input. If the counter becomes 481^{481} , accept and halt. Enumerate the string y in lexicographic order.

(109)(d) $L = \{M : M \text{ takes fewer than } 481^{481} \text{ steps on all inputs}\}$

This language is r.e. Simulate machine M on inputs y . If it takes more than 481^{481} steps on a particular input, then reject and halt. Else, go to the next input. Enumerate y inputs in lexicographic order till the size of y exceeds 481^{481} . Then accept and halt.

(110) $L = \{M : M \text{ accepts at least } 481 \text{ strings}\}$

This language is in SD. Simulate machine M on input y . If it accepts y , increment the counter and move to the next input. If at any point counter reaches 481, accept and halt. y is enumerated in lexicographic ordering.

This language is not in co-SD. Because, if this language was in co-SD then, then it would be decidable. We can show that this language is not decidable by reducing this problem to HP .

Let L be decidable. Then it implies there is a total turing machine K accepting L . Consider machine $M1$ which does the following:

- It simulates M on x . If M halts on x then machine $M1$ accepts its input y .

So the language of machine $M1$ can be described as :

$$L(M1) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving machines $M1$ as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

(111) $L1 = \{M : L(M) \geq 481\}$ - This language is semidecidable.(From (110))

$L2 = \{M : L(M) \leq 481\}$ -This language in is co-SD. This can be shown by considering $\overline{L2} = \{M : L(M) \geq 480\}$

From argument in (110), this language is SD but not decidable. Therefore $L2$ is in co-SD.

(112) $L = \{M : M \text{ halts on inputs of length less than } 481\}$

This language is semidecidable. Consider a machine K , which simulates inputs y of length less than 481, listed in lexicographical order in a time sharing manner. If it accepts all the inputs then accept.

To show that this language is not co-SD, it suffices to show this language is not decidable. This can be shown by reducing this language to HP .

Let L be decidable. Then it implies there is a total turing machine K accepting L . Consider machine $M1$ which does the following:

- It simulates M on x . If M halts on x then machine $M1$ accepts its input y .

So the language of machine $M1$ can be described as :

$$L(M1) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } x \\ \emptyset & \text{if } M \text{ does not halt on } x \end{cases}$$

Giving machines $M1$ as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

(115)(a) $L = \{A \text{ given } TM \text{ runs for atleast fixed number of } n \text{ steps on a given input}\}$
This language is decidable. Construct a total turing machine K , which simulates the TM for n steps. If it halts before this, then reject. Else accept.

(b) $L = \{M : M \text{ reenters the start state on some input}\}$

This is semidecidable. To show that this language is not decidable we can reduce this to HP as follows :

Let L be decidable. Then it implies there is a total turing machine K accepting L . Consider machine $M1$ which does the following:

- It simulates M on x . If M halts on x then machine $M1$ goes to a new state say q , writes a special character \hat{a} on the tape and returns to start state. And on seeing \hat{a} in start state, the machine goes to accept state.

So the machine $M1$ reenters its start state if M halts on x . It doesn't if M doesn't halt on x .

Giving machines $M1$ as input to machine K can help in totally computing if machine M halts on input x . Since, there is no such total turing machine, hence no such total turing machine K exists.

(c) $L = \{M : \text{If } M \text{ moves its head left more than 10 times on input } a^{481} \}$

This language is decidable. The argument is similar to question (1)(b). If it ever moves its head left once after reaching the end of input, then by the same argument, it will move its head left for more than ten times also. Hence, accept it. Else, reject it.

(d) $L = \{M : M \text{ prints more than 481 non-blank symbols on the tape} \}$

This language is decidable. The total number of configurations (q, x) of machine M is finite. Hence, after $(|Q| \times |\Gamma|) + 1$, the machine will come back to the same (state, tapeAlphabet) pair. In this time, if it ever wrote a non-blank symbol on the tape it implies after finite time, it will write more than 481 non blank symbols on the tape. Hence, accept. Else reject.

3. (a) $L = \{a^p : p \text{ is a prime number}\}$

This language is not regular but is decidable.

(b) For a given turing machine M ,

$L = \{1^n : M \text{ halts on input } 1^n\}$

Since, it's a fixed machine M , the transitions of M can be hardwired in the transitions of the machine K simulating it. And hence the set of tape alphabets is singleton.

(c) $FIN = \{M : L(M) \text{ is finite}\}$,

Generally, we represent the encoding of M using the bits 0's and 1's. However, this can be viewed as the n th lexicographic permutation of 0's and 1's. Hence, encoding M as a^n , where n represents the n th lexicographic permutation of 0's and 1's. Hence every machine can be encoded over a singleton alphabet.

4. (a) \leq_m is reflexive and transitive, but not symmetric.

Reflexive: $A \leq_m A$ via the identity function $\sigma(x) = x$

Transitive: If $A \leq_m B$ via a map σ and $B \leq_m C$ via the map τ , then $A \leq_m C$ via the map $\tau \circ \sigma$

Symmetric: \leq_m is not symmetric because the function σ is not invertible.

\leq_T is reflexive and transitive, but not symmetric.

Reflexive: $A \leq_T A$. Given a A as an oracle, the OTM just has to return the value of its query to the oracle to accept/reject A .

Transitive: If $A \leq_T B$, then given B as an oracle, A is totally computable. Similarly

$B \leq_T C$ implies given C as an oracle, B is totally computable. Now, an OTM with C as an oracle, can compute A totally as follows. Using C as the oracle, compute and store intermediate value of B . Now this value can serve as an answer from an oracle and hence A can be calculated. Calculation of B doesn't cause looping ($B \leq_T C$). Hence, calculation of A doesn't cause looping. Hence $A \leq_T C$.

Symmetric: \leq_T is not symmetric. Example : $FIN \leq_T REG$, but $REG \not\leq_T FIN$.

(b) L is decidable $\implies L \leq_m 1^*0^*$

To show this we have to find a $\sigma : \Sigma^* \rightarrow \Sigma^*$:

- σ is totally computable
- $\forall x, x \in L \iff \sigma(x) \in 1^*0^*$

Since, L is decidable, there exists a total turing machine K such that, $L(K) = L$. Let σ be defined as follows :

$$\sigma(x) = \begin{cases} 10 & \text{if } K \text{ accepts } x \\ 01 & \text{if } K \text{ rejects } x \end{cases}$$

σ is totally computable since K is total.

$$x \in L \iff K \text{ accepts } x \iff (\sigma(x) = 10 \in 1^*0^*)$$

Therefore, $L \leq_m 1^*0^*$

L is decidable $\iff L \leq_m 1^*0^*$

RHS implies there exists a totally computable σ such that $\forall x, x \in L \iff \sigma(x) \in 1^*0^*$.

For any input x , calculating $\sigma(x) \in 1^*0^*$, can be done by a total turing machine. (Keep 4 states. when on state 1 implies seeing 1's. First time a 0 is seen, move to state 2. Or if end of input is seen, move to state 3 and accept. In state 2, if end of input is reached go to state 3 and accept. If a 1 is seen, move to state 4 and reject). Hence, $x \in L$ can be totally computed. Therefore, L is decidable.

5. $FIN \leq_T REG$