Theory of Computation — CSE 105

Computability Theory
Solutions to Selected Problems

Turing Machine Design

1. Problem 3.8 (a), Page 148. Give an implementation-level description of a Turing machine that decides $L = \{w | w$ contains an equal number of 0s and 1s $\}$.

Solution: The idea to solve this problem is that for each 0 or 1 we see at the input tape, we should look for a matching 1 or 0, respectively, and cross both symbols. If the respective symbol is not found (which means the string contains a unequal number of 0s and 1s), then we reject. If all 0s and 1s are matched (which means the contains an number of 0s and 1s), then we accept.

So, an informal definition for the Turing machine M would be the following:

T$_1$ = "Given a string, s,

1. Read next input symbol different from x.
2. If it's a 0, then cross it and keep reading every symbol to the right until either a 1 or $\sqcup$ is found. If it's a 1, then cross it too. If it's a $\sqcup$, then *reject.*
3. If it's a 1, then cross it and keep reading every symbol to the right until either a 0 or $\sqcup$ is found. If it's a 0, then cross it too. If it's a $\sqcup$, then *reject.*
4. If it's a $\sqcup$, then *accept.*
5. Move back to the beginning of the tape and go to step 1."

A formal description of $M = (Q, \sum, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$, where:

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{acc}}, q_{\text{rej}}\}$,
- $\sum = \{0, 1\}$,
- $\Gamma = \{0, 1, x, \sqcup\}$, and
- δ is given with the state diagram in figure 1.

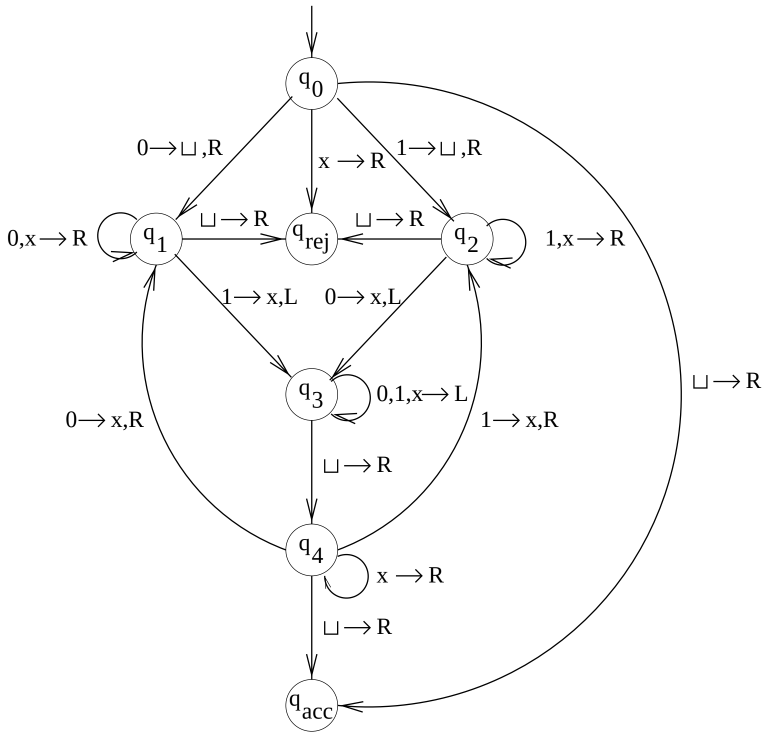

Figure 1: State diagram for Turing machine M

Closure Properties

2. Problem 3.15 (part a), Page 149. Show that the collection of recursively enumerable (Turing-recognizable) languages is closed under the union operation.

Solution: Given two recursively enumerable languages, A and B, we would like to show that A∪B is recursively enumerable.

Because A is recursively enumerable, there is a Turing machine, T_A, which will accept a string s if and only if s∈A. Similarly, there is a machine T_B which will accept a string s if and only if s∈B. (T_A and T_B will not necessarily halt any input.) Construct a Turing machine $T_{A\cup B}$ as follows:

$T_{A\cup B}$ = "Given a string, s,

1. Start a counter, i, at 1.
2. Simulate T_A for i steps on s. If T_A accepts, *accept.*
3. Simulate T_B for i steps on s. If T_B accepts, *accept.*
4. Increment i, and return to step 2."

Claim: $T_{A\cup B}$ accepts a string s if and only if s is in the language A∪B.

Proof: If T$_{A\cup B}$ accepts s, then s must have either been accepted by T$_A$ in step 2, or by T$_B$ in step 3. Thus s∈A or s∈B. Thus, s∈ $A \cup B$

Conversely, if s∈A∪B then s∈A or s∈B. If s∈A then A accepts s after a certain number of steps. Call this number k. Then T$_{A\cup B}$ accepts s when i=k in step 2. Similarly, if s∈B then B accepts s after some number of steps, k. Thus T$_{A\cup B}$ accepts s when i=k in step 3.

Because we have constructed a Turing machine which accepts A∪B, the language A∪B is recursively enumerable.

3. Problem 3.15 (b), Page 149. Show that the collection of recursively enumerable (Turing-recognizable) languages is closed under the concatenation operation.

Solution: Given recursively enumerable languages, A and B, we wish to show that AB (the language {w$_1$w$_2$: w$_1$∈A, w$_2$∈B}) is recursively enumerable.

Because A is recursively enumerable, there is a Turing machine, T$_A$, which will accept a string s if and only if s∈A. Similarly, there is a machine T$_B$ which will accept a string s if and only if s∈B. (T$_A$ and T$_B$ will not necessarily halt any input.) Construct a nondeterministic Turing machine T$_{AB}$ as follows:

T$_{AB}$ = "Given a string, s,

1. Guess a number n between 0 and |s|.

   Comment: Let w$_1$ represent the first n characters of s, w$_2$ represent the remaining |s|-n characters.

2. Start a counter, i, at 0.
3. Run T$_A$ on w$_1$ for i steps.
4. Run T$_A$ on w$_2$ for i steps.
5. If T$_A$ accepted w$_1$ in step 3 and T$_A$ accepted w$_2$ in step 4, *accept.*
6. Increment i and return to step 3."

Claim: T$_{AB}$ accepts a string s if and only if s∈AB.

Proof: If T$_{AB}$ accepts s then there must have been a partition of S into two parts, w$_1$ and w$_2$, such that T$_A$ accepted w$_1$ and T$_B$ accepted w$_2$. Then s=w$_1$w$_2$ where w$_1$∈A and w$_2$∈B. Then s∈AB.

If s∈AB then there is a partition of s into two parts, w$_1$w$_2$ such that w$_1$∈A and w$_2$∈B. Then T$_A$ accepts w$_1$ after some number of steps, k$_1$. And T$_B$ accepts w$_2$ after some number of steps, k$_2$. Then T$_{AB}$ will accept s when

- the correct partition w$_1$w$_2$ is guessed in step 1, and
- i=max(k$_1$,k$_2$) (in step 5)

Because we have constructed a machine which will accept AB, the language AB must be Turing-recognizable (i.e. recursively enumerable).

4. Problem 3.15 (c), Page 149. Show that the collection of recursively enumerable (Turing-recognizable) languages is closed under the star operation.

Solution: Given an recursively enumerable language A, we wish to show that A^* is recursively enumerable. Because A is recursively enumerable, there is a Turing machine T_A which accepts a string s if and only if s$\in$A.

Construct T_{A^*} as follows:

T_{A^*} = "Given a string, s,

```
1. If s=e (the empty string), *accept.*
2. Guess a number n between 1 and |s|.
3. Guess n distinct numbers between 0 and |s|.
   **Comment:** This gives a partition of s into n parts. That is, s=$w_1w_2...w_n$, where $w_i \in \Sigma^*$.
4. Start a counter, j, at 0.
5. Start another counter, k, at 0.
6. Repeat for every substring $w_i$, where i$\in\{1..n\}$
   (a) Run $T_A$ on $w_i$ for j steps.
   (b) If $T_A$ accepts, increment k.
7. If k=n (i.e. if $T_A$ accepted $w_i$ for every i$\in\{1..n\}$) then *accept.*
8. Increment j and return to step 5."
```

Claim: T_{A^*} accepts a string s if and only if s$\in A^*$.

Proof: If s$\in A^*$ then there is a number, n, such that

- n$< |s|$
- There are strings w_1,w_2,...w_n, such that
  - s=$w_1w_2...w_n$
  - $w_i\in$A for each i$\in\{1..n\}$

(The verification of this fact is left to the reader.)

Because each $w_i\in$A, T_A must halt on input w_i after some number of steps, k_i. Then T_{A^*} will accept the string s when:

- The number n is guessed correctly in step 2.
- The strings $w_1..w_n$ are guessed correctly in step 3.

- j=max(k_1,k_2,...k_n) in step 7.

Conversely, if T_A accepts a string s then s must have been composed of substrings which were all accepted by T_A after some number of steps. Thus s$\in A^*$.

Because we have constructed a machine which accepts a string s if and only if s$\in$A*, the language A* is Turing-recognizable. (i.e. A* is recursively enumerable.)

5. Problem 4, page 131 Show that the a language is decidable if and only if there is some enumerator that enumerates the language in lexicographic order.

Solution: Since we are proving an "if and only if" statement, we divide the proof into two parts. One part for the "if" and the other part for the "only if".

- Part 1: If a language is decidable then there exists an enumerator that enumerates the language in lexicographic order: such an enumerator works like this:

  ```
  $E$ = "
      1. Determine which string comes next lexicographically in $\sum^*$.
      2. Run $T_A$ on this string. If $T_A$ accepts, print this string out.
      3. Go to step 1."
  ```

  Comment: T_A is the Turing machine that decides the given language.

- Part 2: If there exists an enumerator that enumerates a language in lexicographic order, then the language is decidable.

  To prove this statement, we would like to construct a Turing machine which, on input x, simulate the enumerator and simply waits intil either x is printed out or x is passed lexicographically. If the former case, the machine should accept x. In the latter case, the machine should reject x.

  There is one further consideration. What happens if our enumerator gets caught in a loop before it reaches x? In such a case, the language is finite (since the enumerator only enumerated strings which were lexicographically less than x). So, clearly the language is decidable.

  PROOF DETAILS

  If there is an enumerator E that enumerates a language L in lexicographic order, then one of two cases can occur.

  - Case 1: L is finite. In this case, L is decidable since any finite language is decidable.
  - Case 2: L is infinite. In this case, we construct a Turing machine M which decides L as follows:

    ```
    $M$ = "Given a string, $x$,
        1. Run $E$
        2. Let $w$ be the next string printed out by $E$.
        3. If $x = w$, accept.
        4. If $w$ is greater than in lexicographic order, then reject.
        5. Return to step 2."
    ```

    Comment: Because L is infinite, the strings that E prints out must supercede x lexicographically at some point. Hence this machine will always either accept or reject an input x. Hence L is decidable.

Undecidability

6. Problem 5.13, page 195. A *useless state* in a Turing machine is one that is never entered on any input string. Consider the problem of testing whether a Turing machine has any useless states. Formulate this problem as a language and show that it is undecidable.

Solution: This problem can be formulated as the following language $R = \langle\langle M\rangle, q\rangle|$ q is a state of M such that for any $s \in L(M)$, M does not reach q on input $s\}$.

To be even more formal, we may wish to define "reach q on input s". The definition is as follows: Let $C_1, C_2, \ldots, C_k, \ldots$ be the configurations of a Turing machine M on input x. Then M "reachs q on input s" means that q appears in C_i for some i.

To prove that R is undecidable, we will proceed by contradiction. Assume that R is decidable. Let T_R be a TM that decides R. We will now show that the undecidable halting language $L_H = \{\langle T, x\rangle|$ T halts on input $w\}$ is decidable to obtain the desired contradiction. With the help of T_R, we will now design a Turing machine T_H to decide L_H. The description of T_H is given below.

T$_H$ = "Given a string, s,

1. Check if s is of the form $\langle T, x\rangle$ for some TM T and input $x \in \Sigma^*$.
2. If not, reject and halt.
3. If true, construct the description of a TM T_1 given T and x such that

   T_1 = "Given a string, y,
   (a) T_1 erases its input and writes x on the blank tape.
   (b) Simulate (or run) T on x
   (c) if T halts on x, then T_1 enters a new state, q_{halt}."
4. Run T_R on $\langle\langle M\rangle, q_{\text{halt}}\rangle$.
5. If T_R accepts, then *accept.*
6. If T_R rejects, then *reject.*"

Claim: T$_H$ decides the language L_H = { $\langle T, x\rangle$: T halts on $x\}$.

Proof: We will show that T_H will accept for inputs $x \in L_H$ and will reject for inputs $x \notin L_H$.

- If $s \in L_H$, then $s = \langle T, x\rangle$, and T halts on x. Then T_1 reaches the state q_{halt} in step 3. So T_R accepts $\langle T_1\rangle$ in step 4. So T_H accepts s in step 5.
- If $s \notin L_H$, then either s is not of the form $\langle T, x\rangle$ or s is of the form $\langle T, x\rangle$, and T does not halt on x. In the first case, s is rejected in step 1. In the second case, T_1 doesn't reach the state q_{halt} in step 3. Thus, T_R accepts $\langle T_1\rangle$ in step 4. Thus T_H rejects in step 6.

Thus, T_H decides L_H. This is a contradiction to the fact that L_H is undecidable. Hence, R is undecidable.

7. Show that the language $L_{\text{empty}} = \{\langle T, \rangle |\ L(T_1) = \emptyset\}$ is undecidable.

Solution: We will proceed by contradiction. Assume that L_{empty} is recursive. Let T_{empty} be a Turing machine that decides L_{empty}. We will now show that the undecidable halting language $L_H = \{\langle T, x\rangle |\ T$ halts on input $x\}$ is decidable to obtain the desired contradiction.

With the help of T_{empty}, we will now design a Turing machine T_H to decide L_H. The description of T_H is given below.

T$_H$ = "Given a string, s,

1. Check if s is of the form $\langle T, x\rangle$ for some TM T and input $x \in \Sigma^*$.
2. If not, reject and halt.
3. If true, construct the description of a TM T_1 given T and x such that

   T$_1$ = "Given a string, y,

   (a) Simulate (or run) T on x

   (b) Accept y if and only if T halts on x."

   Comment: The description of T_1 depends on T and x which in turn depend on s. Also observe that $L(T_1) = \Sigma^*$ if T halts on x and $L(T_1) = \emptyset$ if T does not halt on x. Also notice that T_H is only constructing the description of T_1 rather than running it.
4. Run T_{empty} on $\langle T_1\rangle$.
5. If T_{empty} accepts, *reject.*
6. If T_{empty} rejects, *accept.*"

Claim: T$_H$ decides the language L_H = { $\langle T, x\rangle$: T halts on x}.

Proof: We will show that T_H will accept for inputs $x \in L_H$ and will reject for inputs $x \notin L_H$.

- If $s \in L_H$, then s=$\langle T, x\rangle$, and T halts on x. Then $L(T_1) = \Sigma^* \neq \emptyset$. So T_{empty} rejects $\langle T_1\rangle$ in step 4. So T_H accepts s in step 6.
- If $s \notin L_H$, then either s is not of the form $\langle T, x\rangle$ or s is of the form $\langle T, x\rangle$, and T does not halt on x. In the first case, s is rejected in step 1. In the second case, $L(T_1) = \emptyset$. Thus, T_{empty} accepts $\langle T_1\rangle$ in step 4. Thus T_H rejects in step 5.

Thus, T_H decides L_H. This is a contradiction to the fact that L_H is undecidable. Hence, L_{empty} is undecidable.

8. Show that the following problem is unsolvable. Given two Turing machine T_1 and T_2, is $L(T_1) \cap L(T_2) = \emptyset$?

Solution: In order to show that this is unsolvable, we will show that if there were an algorithm for solving this problem, then there would be an algorithm for solving the unsolvable halting problem ("Given $\langle M, x \rangle$, does M halt on x?"). This will provide a contradiction.

PROOF DETAILS

Assume for the sake of contradiction, that the problem stated above is solvable. Then there is a Turing machine which decides the language $I = \{\langle T_1, T_2 \rangle : L(T_1) \cap L(T_2) = \emptyset\}$. Call this machine T_I. (T_I halts on any input, and accepts a string, s, if and only if $s \in I$.)

In what follows, e denotes the empty string.

We construct a machine T_H as follows:

T_H = "Given a string, s,

1. If s is not of the form $\langle M, x \rangle$, where M is a Turing machine and $x \in \Sigma^*$, *reject.*
2. Construct the encodings for two Turing machines, M_1 and M_2, where

   M_1 = 'On input y,

   (a) If $y \neq e$ *reject.*
   (b) Run M on x.
   (c) If M halts on x, *accept.*'

   M_2 = 'On input y,

   (a) If $y \neq e$ *reject.*
   (b) Run M on x.
   (c) If M halts on x, *accept.*'

   Comment: $L(M_1) = \{e\}$ if M halts on s. $L(M_1) = \emptyset$ otherwise. $L(M_2)$ is the same as $L(M_1)$.
3. Run T_I on $\langle M_1, M_2 \rangle$.
4. If T_I accepts, *reject.*
5. If T_I rejects, *accept.*"

Claim: T_H decides the language $H = \{ \langle M, x \rangle : \text{M halts on x} \}$. That is, if $s \in H$, T_H accepts s. If $s \notin H$, T_H rejects s.

Proof: If $s \in H$, then $s = \langle M, x \rangle$, and M halts on x. Then $L(M_1) = \{e\}$ and $L(M_2) = \{e\}$. Then $L(M_1) \cap L(M_2) = \{e\}$. Then $L(M_1) \cap L(M_2) \neq \emptyset$. So T_I rejects $\langle M_1, M_2 \rangle$ in step 3. So T_H accepts s in step 5.

If $s \notin H$, then either s is not of the form $\langle M, x \rangle$ or s is of the form $\langle M, x \rangle$, and M does not halt on x. In the first case, s is rejected in step 1. In the second case, $L(M_1) = L(M_2) = \emptyset$. So, $L(M_1) \cap L(M_2) = \emptyset$. Thus, T_I accepts in step 3. Thus T_H rejects in step 5.

We have constructed a machine which decides the halting language. Because the halting language is known to be undecidable, this is a contradiction. Thus our assumption that there was a machine M deciding L must have been incorrect. There is no machine deciding L. L is not recursive. So, the problem stated is undecidable.

9. Problem 5.12, page 195. Let $S = \{\langle M\rangle : M$ is a Turing machine such that for any string, w, that M accepts, it also accepts $w^R\}$. Show that S is undecidable.

Solution: We show that if S were decidable then the undecidable halting language, H, would be decidable. (H = { $\langle$M, x$\rangle$: M halts on x })

PROOF DETAILS

Assume S is decidable. Then there is a machine, T$_S$ which decides S.

Construct T$_H$ as follows:

T$_H$ = "On input, s

1. If s is not of the form $\langle$M, x$\rangle$ where M is a Turing machine and x$\in\Sigma^*$, *reject.*
2. Construct a machine, N, which acts as follows:

   N = "On input y,

   (a) If y=0x^{R}1, *accept.*
   (b) If y=1x0, run M on x. If M halts, *accept.*
   (c) If y is neither of these strings, *reject.*"
3. Run T$_S$ on $\langle$N$\rangle$.
4. If T$_S$ accepts, *accept.*
5. If T$_S$ rejects, *reject.*"

Claim: T$_H$ decides the halting language, H. That is, if s$\in$H, T$_H$ accepts s. And if s$\notin$H, T$_H$ rejects s.

Proof: If s$\in$H then s is of the form $\langle$M, x$\rangle$, where M halts on x. Then N will accept 1x0. N also accepts 0x^{R}1. These are the only strings which N accepts. Note that (1x0)^R=(0x^{R}1). So, N has the property that for any string, w, that N accepts, it also accepts w^R. Thus T$_S$ accepts $\langle$N$\rangle$. Thus T$_H$ accepts s.

If s$\notin$H then either s is not of the form $\langle$M,x$\rangle$, or s is of the form $\langle$M,x$\rangle$, but M does not halt on x. In the first case, s is rejected in step 1. In the second case, when we construct N in step 2, N will accept the string 0x^{R}1, but not 1x0. Thus N will be rejected by T$_S$ in step 4. Thus s will be rejected in step 5.

We have constructed a Turing machine which decides the undecidable language, H. This is a contradiction. Therefore, our assumption that S was decidable must have been wrong. S is not decidable.

10. Prove that no virus tester can be both safe and correct.

Solution: We use a diagonalization argument similar to the one that was used to prove that the halting problem was unsolvable.

Consider the list of programs, P1, P2, P3, ... and the list of program descriptions, $\langle$P1$\rangle$ $\langle$P2$\rangle$ $\langle$P3$\rangle$ $ldots$

Each one of these programs either will or will not be safe on each of the descriptions. We can thus make the following chart, where the i,j entry in the chart is ”yes” if program P$_i$ is safe on input $\langle$P$_j\rangle$.

```
          <P1> <P2> <P3> . . .          <D>

      P1  yes   no   no  . . .
      P2  no    no   yes . . .
      P3  no    no   no  . . .
      .
      .
      .
      D                                ???
```

We will write a program, D with the following property: If P$_i$ is safe on input $\langle$P$_i\rangle$ then D will not be safe on input P$_i$. And if P$_i$ is not safe on input $\langle$P$_i\rangle$ then D will be safe on input P$_i$.

By using this construction, D will differ from each of the programs P$_1$, P$_2$, ..., and hence cannot be placed in the above list.

PROOF DETAILS

Assume that such a virus tester exists. The virus tester will be called IsSafe. It reports ”yes” if a program is safe on a given input, and ”no” otherwise.

Write a program, D. D will work as follows:

D = “Given an input, x,

1. If x is not the description of some program, P, say ”no”.
2. Run IsSafe on $\langle$P$\rangle$ $\langle$P$\rangle$
3. If IsSafe reports ”yes”, then alter the operating system.
4. If IsSafe reports ”no”, then don’t alter the operating system”

What happens when we run IsSafe on $\langle$D$\rangle$,$\langle$D$\rangle$?

If IsSafe reports ”yes”, then D is safe on input D.

However, if we run D on input $\langle$D$\rangle$, D will alter the operating system in step (c).

Thus D is not safe on input D.

Similarly, if IsSafe reports ”no”, then D will not alter the operating system. But then D is safe. But then IsSafe should not have said ”no”.

Because of this logical contradiction, the original assumption must have been incorrect. No such virus tester can exist.

Note that the above conclusions would not be valid if we had not required that IsSafe be safe. (The virus tester must be both safe and correct.)