HW5: Recognizability, Decidability, Undecidability, and Reductions

CSE105Sp22

In this assignment,

You practiced designing and working with Turing machines and their variants. You used general constructions and specific machines to explore the classes of recognizable, decidable, and undecidable languages. You used computable functions to relate the difficult levels of languages via mapping reduction.

Reading and extra practice problems: Chapter 4 exercises 4.1, 4.3, 4.4., 4.5. Chapter 4 Problems 4.29, 4.30, 4.32. Chapter 5 exercises 5.4, 5.5, 5.6, 5.7. Chapter 5 problems 5.10, 5.11, 5.16, 5.18.

Assigned questions

1. (*Graded for correctness*[1])

   (a) Give an example of a decidable language L_1 whose complement is also decidable. A complete solution will include either (1) a precise definition of the example language L_1 and an explanation of why it is decidable and why its complement is decidable, or (2) a sufficiently general and correct argument for why there is no way to choose an example language to satisfy this requirement. All justifications and arguments should connect to the relevant definitions and the specific concepts being discussed.

   Solution: An example of a decidable language is $L_1 = \emptyset$. This is a decidable language because it is decided by the Turing machine given by the high-level description

   "On input w: reject."

   For any input string, the computation of this Turing machine has exactly one step so the Turing machine never loops (always halts) and so the Turing machine is a decider. Moreover this Turing machine does not accept any strings so its language is the empty set, L_1. The complement of L_1 is $\{x \in \Sigma^* \mid x \notin \emptyset\} = \Sigma^*$, which is decided by the Turing machine given by the high-level description

[1] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

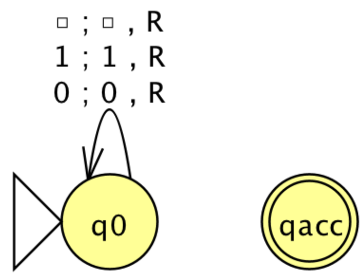
Copyright Mia Minnes, 2022, Version May 29, 2022 (1)

"On input w: accept."

Also: we proved in class that the collection of decidable languages is closed under complementation (Monday of Week 8, page 3 of the notes) so the complement of *any* decidable language is also decidable.

(b) Give an example of a decidable language L_2 and a Turing machine M_2 such that $L(M_2) = L_2$ but M_2 does not decide L_2. A complete solution will include either (1) precise definitions of L_2 and M_2 and justifications for why $L(M_2) = L_2$ and why M_2 does not decide L_2, or (2) a sufficiently general and correct argument for why there is no way to choose such a language and machine. For any machines you discuss, you can choose whether to use high-level descriptions, implementation level descriptions, or formal definitions. All justifications and arguments should connect to the relevant definitions and the specific concepts being discussed.

Solution: Consider the language $L_2 = \emptyset$ again, and the Turing machine over $\{0, 1\}$ with state diagram:



(using the usual conventions). Since *qacc* is not accessible from $q0$, no computation of the TM accepts so the language recognized by this machine is L_2. However, this machine does not decide L_2 because it is not a decider: there is a string for which the computation of this machine on that string does not halt, namely consider the computation of the Turing machine on the empty string in which the machine stays in state $q0$ and scans to the right forever without changing the contents of the tape.

2. (*Graded for fair effort completeness*[2])

Recall that a set X is said to be **closed** under an operation OP if, for any elements in X, applying OP to them gives an element in X. For example, the set of integers is closed under multiplication because if we take any two integers, their product is also an integer.

Suppose M_1 and M_2 are Turing machines. Consider the following high-level descriptions of machines that give general constructions based on M_1 and M_2.

(a) Consider the following construction of a nondeterministic Turing machine:

[2]This means you will get full credit so long as your submission demonstrates honest effort to answer the question. You will not be penalized for incorrect answers. To demonstrate your honest effort in answering the question, we ask that you include your attempt to answer *each* part of the question. If you get stuck with your attempt, you can still demonstrate your effort by explaining where you got stuck and what you did to try to get unstuck.

Copyright Mia Minnes, 2022, Version May 29, 2022

"On input w

1. Nondeterministically split w into two pieces, i.e. choose x, y such that $w = xy$.
2. Simulate running M_1 on x.
3. Simulate running M_2 on y.
4. If both simulations in steps 2 and 3 accept, accept."

Can this construction be used to prove that the class of Turing-recognizable languages is closed under concatenation? Briefly justify your answer.

Solution: Yes. Given L_1 and L_2 recognizable by Turing machines M_1 and M_2, we apply the above construction to obtain Turing machine M. We will show M recognizes $L_1 \circ L_2$ and hence, $L_1 \circ L_2$ is Turing-recognizable. $\forall w \in L_1 \circ L_2$, by definition $w = w_1 w_2$ for some $w_1 \in L_1$ and $w_2 \in L_2$. Consider running M on w. At line 1, M nondeterministically decides how to split w, and one of these ways is splitting into $w_1 w_2$. For that split we see that M_1 accepts w_1 and M_2 accepts w_2. Therefore M accepts w. Conversely, $\forall w \in L(M)$, there is a split $w = xy$ such that M_1 accepts x and M_2 accepts y. This implies $w \in L_1 \circ L_2$. Therefore M recognizes $L_1 \circ L_2$.

(b) Consider the following construction of an enumerator:

"Without any input

1. Build an enumerator E_1 that is equivalent to M_1.
2. Build an enumerator E_2 that is equivalent to M_2.
3. Start E_1 running and start E_2 running.
4. Initialize a list of all strings that have been printed by E_1. Declare the variable n_1 to be the length of this list (initially $n_1 = 0$).
5. Initialize a list of all strings that have been printed by E_2 so far. Declare the variable n_2 to be the length of this list (initially $n_2 = 0$).
6. Every time a new string x is printed by E_1:
7.   Add this string to the list of strings printed by E_1 so far.
8.   Increment n_1 so it stores the current length of the list.
9.   For $j = 1 \ldots n_2$,
10.   Let w_j be the jth string in the list of strings printed by E_2
11.   Print xw_j.
12. Every time a new string y is printed by E_2:
13.   Add this string to the list of strings printed by E_2 so far.
14.   Increment n_2 so it stores the current length of the list.
15.   For $i = 1 \ldots n_1$,
16.   Let u_i be the ith string in the list of strings printed by E_1
17.   Print $u_i y$."

Can this construction be used to prove that the class of Turing-recognizable languages is closed under concatenation? Briefly justify your answer.

Solution: Yes. Given L_1 and L_2 recognizable by enumerators E_1 and E_2, we apply the above construction to obtain enumerator E. It suffices to show that E enumerates $L_1 \circ L_2$ as that proves that $L_1 \circ L_2$ is Turing-recognizable since enumerators and Turing machines are equally expressive. $\forall w \in L_1 \circ L_2$, by definition $w = xy$ for some $x \in L_1$ and $y \in L_2$. We want to show that w will be printed by E. Consider running E. By our assumption, x will be printed by E_1 and y will be printed by E_2 in finite time. If y is printed before x, then y is in the list of strings already printed by E_2 when x is printed. Hence, y will be one of the $w_j s$ at line 11, and $w = xy$ will be printed at line 12. Similarly, if x is printed before y, then x is in the list of strings already printed by E_1 when y is printed. Hence, x will be one of the $u_i s$ at line 16, and $w = xy$ will be printed at line 17. We found that in either case, E will print w. Conversely $\forall w \in L(E)$, w is printed either at line 12 or at line 17. If w is printed at line 12, then since x is printed by E_1 and w_j is printed by E_2, $w \in L_1 \circ L_2$. Similarly, if w is printed at line 17, then since u_i is printed by E_1 and y is printed by E_2, $w \in L_1 \circ L_2$. Therefore E enumerates $L_1 \circ L_2$.

(c) Consider the following construction of a Turing machine:

"On input w

1. Let $n = |w|$.
2. Create a two dimensional array of strings $s_{m,j}$ where $0 \leq m \leq n$ and $0 \leq j \leq 1$.
3. For each $0 \leq m \leq n$, initialize $s_{m,0}$ to be the prefix of w of length m and $s_{m,1}$ to be the suffix of w of length $n - m$. In other words, $w = s_{m,0}s_{m,1}$ and $|s_{m,0}| = m$, $|s_{m,1}| = n - m$.
4. For $i = 1, 2, \ldots$
5.   For $k = 0, \ldots, i$
6.   Run M_1 on $s_{\min(k,n),0}$ for (at most) i steps.
7.   Run M_2 on $s_{\min(k,n),1}$ for (at most) i steps.
8.   If both simulations in steps 6 and 7 accept, accept."

Can this construction be used to prove that the class of Turing-recognizable languages is closed under concatenation? Briefly justify your answer.

Solution: Yes. Given L_1 and L_2 recognizable by Turing machines M_1 and M_2, apply the above construction to obtain Turing machine M. We will show M recognizes $L_1 \circ L_2$ and hence, $L_1 \circ L_2$ is Turing-recognizable. $\forall w \in L_1 \circ L_2$, by definition $w = xy$ for some $x \in L_1$ and $y \in L_2$. Furthermore, suppose M_1 accepts x in a_1 steps and M_2 accepts y in a_2 steps. We want to show that w is accepted by M. When $i = max(a_1, a_2, |w|)$, then for the value $k = |x|$ we have $\min(k, n) = \min(k, |w|) = k$ since $k \leq i \leq |w|$. Thus, in line 6 M runs M_1 on $s_{\min(k,n),0} = s_{k,0} = s_{|x|,0} = x$ for $i \geq a_1$ steps. Similarly, in line 7 M runs M_2 on y $s_{\min(k,n),1} = s_{k,1} = s_{|x|,1} = y$ for $i \geq a_2$ steps. Both machines simulations run long enough to accept. Hence, M accepts w. Conversely, $\forall w \in L(M)$.

Then, there exists k and i such that $w = s_{k,0}s_{k,1}$, M_1 accepts $s_{k,0}$ in at most i steps, and M_2 accepts $s_{k,1}$ in at most i steps. This implies $w \in L_1 \circ L_2$. Therefore M recognizes $L_1 \circ L_2$.

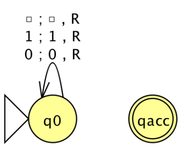
3. (*Graded for fair effort completeness*) Recall that

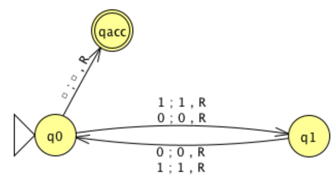
$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } w \in L(M)\}$$

and

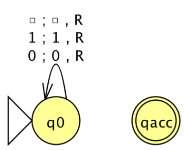
$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w\}$$

Consider the Turing machines below, with input alphabet $\Sigma = \{0, 1\}$, tape alphabet $\{0, 1, \llcorner\!\lrcorner\}$, and state diagrams (with the usual conventions):

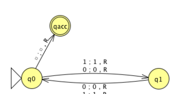




(a) Give an example string that is in both A_{TM} and $HALT_{TM}$ and that is related to one of the two Turing machines whose state diagrams are given above, or explain why there is no such string.

Solution: Consider the string $\langle$ 

$, \varepsilon\rangle$ It is in both A_{TM} and in $HALT_{TM}$ because the computation of the Turing machine

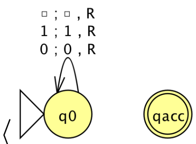


on input ε starts at $q0$ and in one step transitions to $qacc$ because the leftmost cell on the tape is blank. Thus, the computation of this machine on ε halts and the string is accepted by the machine, matching the defintiions of A_{TM} and in $HALT_{TM}$.

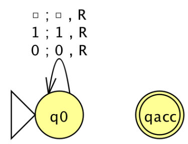
(b) Give an example string that is in A_{TM} and is not in $HALT_{TM}$ and that is related to one of the two Turing machines whose state diagrams are given above, or explain why there is no such string.

Solution: There is no such string because to be in A_{TM} the string would have to be of the form $\langle M, w\rangle$ where M is a Turing machine and w is a string accepted by M. But, the definition of M accepting w is that the computation of M on w halts and accepts so that would guarantee that $\langle M, w\rangle \in HALT_{TM}$.

(c) Give an example string that is not in A_{TM} and is in $HALT_{TM}$ and that is related to one of the two Turing machines whose state diagrams are given above, or explain why there is no such string.

Solution: Consider the string $\langle$ 

$, 0\rangle$ It is not in A_{TM} and in $HALT_{TM}$ because the computation of the Turing machine



on input ε starts at $q0$, in one step transitions to $q1$ because the leftmost cell on the tape is 0, and then transitions to $qrej$ (not in picture) because the cell to the right of the 0 is blank. Thus, the computation of this machine on ε halts and the string is rejected by the machine, matching the defintiions of nonmbership in A_{TM} and membership in $HALT_{TM}$.

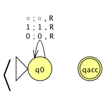
4. (*Graded for correctness*) Fix $\Sigma = \{0, 1\}$ for this question. For each part below, you can choose sets from the following list:

$$\emptyset, A_{TM}, \overline{A_{TM}}, HALT_{TM}, \overline{HALT_{TM}}, E_{TM}, \overline{E_{TM}}, EQ_{TM}, \overline{EQ_{TM}}, \Sigma^*$$

You may use each set from the list **at most once** in the examples below. In particular, you can't choose $A = B = C = D = X = Y = \Sigma^*$.

(a) Find sets A, B for which the computable function

$F =$ "On input x

1. Output $\langle$ 

$, 00\rangle$."

witnesses the mapping reduction $A \leq_m B$. Justify your answer by proving that, for all strings x, $x \in A$ iff $F(x) \in B$. If no such sets exist, justify why not.

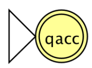
Solution: Let $A = \Sigma^*$ and $B = \overline{HALT_{TM}}$. We will show that the function F witnesses that A is mapping reducible to B. We need to show that, for all strings x, $x \in \Sigma^*$ iff $F(x) \in \overline{HALT_{TM}}$.

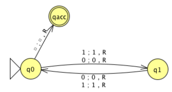
Let x be an arbitrary string. We show each direction of implication: $(\rightarrow)$ Suppose that $x \in A$, then we want to show that $F(x) \in B$. Since $A = \Sigma^*$, x could be any string over Σ. The function F is a constant function and always outputs $\langle M, 00\rangle$, where M is the TM given by the diagram in the problem statement and notice that M loops on all inputs. This means the function F always outputs a string that is

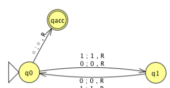
never in $HALT_{TM}$ regardless of the input x so $F(x) \in \overline{HALT_{TM}}$.
($\leftarrow$) We need to show that if $x \notin \Sigma^*$, then $F(x) \notin \overline{HALT_{TM}}$. This conditional statement is vacuously true because its hypothesis is false.

(b) Find sets C, D for which the computable function

$G =$ "On input x

1. Check if $x = \langle M, w\rangle$ for M a Turing machine and w a string. If so, go to step 3.
2. If not, output $\langle$ 

, 

$\rangle$.
3. Construct the Turing machine $M'_x =$ "On input y,
   1. If y has a positive and odd length, reject.
   2. Else, if y has a positive and even length, accept.
   3. Otherwise, run M on w and, if the computation halts, accept y."
4. Output $\langle M'_x$, 

$\rangle$."

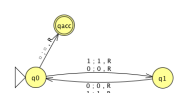
witnesses the mapping reduction $C \leq_m D$. Justify your answer by proving that, for all strings x, $x \in C$ iff $G(x) \in D$. If no such sets exist, justify why not.

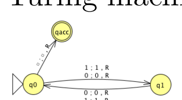
Solution: Let $C = HALT_{TM}$ and $D = EQ_{TM}$. We will show that the function G witnesses that C is mapping reducible to D. We need to show that for all strings x, $x \in HALT_{TM}$ iff $G(x) \in EQ_{TM}$.

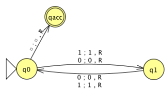
Let x be an arbitrary string. We consider three cases:

- Case 1: Assume $x \in HALT_{TM}$ and we need to show that $G(x) \in EQ_{TM}$. By the assumption that

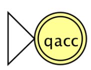
  $$x \in HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w\},$$

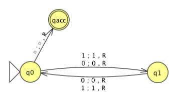
  $x = \langle M, w\rangle$ for some Turing machine M, string w, such that M's computation on w halts. Let's trace the algorithm computing $G(x)$: in step 1, the type check passes and we go to step 3. In step 3, the algorithm constructs the machine M'_x, and then in step 4, the output of the algorithm is $G(x) = \langle M'_x$, 

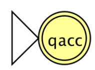
$\rangle$. Since $G(x)$ is a string encoding a pair of Turing machines, to determine if $G(x) \in EQ_{TM}$ we need to check if $L(M'_x) = L($ 

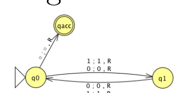
$)$. In class (Friday of week 5, page 13 of the annotated notes), we talked about how the language of the Turing machine with this state diagram is the set of even length strings. Thus, to show that $G(x) \in EQ_{TM}$ we need to show that $L(M'_x) = \{y \in \Sigma^* \mid |y| \text{ is even}\}$. Tracing the definition of M'_x from step 3 of the algorithm for $G(x)$, steps 1 and 2 guarantee that any nonempty string of even length is accepted and any string of odd length is rejected. Thus, to show that $L(M'_x) = \{y \in \Sigma^* \mid |y| \text{ is even}\}$ means to show that M'_x accepts the empty string. The computation of M'_x on $y = \varepsilon$ starts by failing the conditions in steps 1 and 2 so continuing to step 3. In this step, we run M on w (for the M and w such that $x = \langle M, w\rangle$). By assumption that $x \in HALT_{TM}$, this computation halts so step 3 says that M'_x accepts y (the empty string). Thus, we have that M'_x accepts all and only even length strings and so its language equals the language of the Turing machine with state diagram 

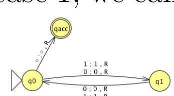
. In particular, this means $G(x) \in EQ_{TM}$, as required.

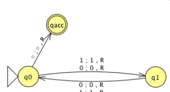
- Case 2: Assume $x \notin HALT_{TM}$ because $x \neq \langle M, w\rangle$ for any Turing machine M and string w. We need to show that $G(x) \notin EQ_{TM}$. Let's trace the algorithm for $G(x)$: in step 1, the type check fails so in step 2, the algorithm outputs $\langle$ 

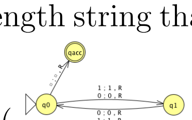
, 

$\rangle$. To show that this output is not in EQ_{TM}, we need to compare the languages recognized by each of the Turing machines encoded by $G(x)$. The Turing machine with state diagram 

accepts all strings immediately so its language is Σ^*. The Turing machine with state digram 

accepts all and only the even length strings (see Friday of week 5, page 13 of the annotated notes). Since these sets are not equal (for example, 0 is an odd length string so is in Σ^* and is not in the set of even length strings), $G(x) \notin EQ_{TM}$.
- Case 3: Assume $x \notin HALT_{TM}$ because $x = \langle M, w\rangle$ for a Turing machine M and string w and M's computation on w does not halt. We need to show that $G(x) \notin EQ_{TM}$. Similar to case 1, we can trace the algorithm that computes $G(x)$. We see that $G(x) = \langle M'_x,$ 

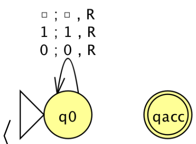
$\rangle$ and we need to compare $L(M'_x)$ with $L($ 

$)$, which is the set of even length strings. Tracing the definition of M'_x, we still have that all nonempty even length strings are accepted and all odd length strings are rejected. However, when we trace the computation of M'_x on the empty string, we see that step 3 starts by running M on w and that this computation never halts because we assumed $\langle M, w\rangle \notin HALT_{TM}$ in this case. Thus, the computation of M'_x on ε does not halt and ε is an even length string that is not accepted by M'_x. We have therefore shown that $L(M'_x) \neq L($ 

$)$ so $G(x) \notin EQ_{TM}$.

(c) Find sets X, Y for which the computable function

$H =$ "On input x

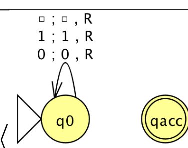
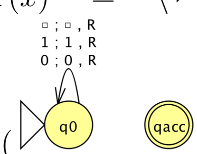
1. Check if $x = \langle M, w \rangle$ for M a Turing machine and w a string. If so, go to step 3.
2. If not, output $\langle$ 

$\rangle$.
3. Construct the Turing machine $M'_x =$ "On input y,
    1. If $y \neq w$, reject.
    2. Otherwise, run M on w.
    3. If M accepts, accept. If M rejects, reject."
4. Output $\langle M'_x \rangle$."

witnesses a mapping reduction $X \leq_m Y$. Justify your answer by proving that, for all strings x, $x \in X$ iff $H(x) \in Y$. If no such sets exist, justify why not.

Solution: Let $X = A_{TM}$ and $Y = \overline{E_{TM}}$. We will show that the function H witnesses that X is mapping reducible to Y. We need to show that for all strings x, $x \in A_{TM}$ iff $H(x) \notin E_{TM}$.

We'll start by describing $H(x)$ for all possible arguments x. We can summarize with the following table:

Argument x	Value $H(x)$
$x \neq \langle M, w \rangle$ for any Turing machine M, string w	$H(x) = \langle$  $\rangle$ and notice that $L($  $) = \emptyset$
$x = \langle M, w \rangle$ for a Turing machine M, string w, and $w \in L(M)$	$H(x) = \langle M'_x \rangle$ where $L(M'_x) = \{w\}$
$x = \langle M, w \rangle$ for a Turing machine M, string w, and $w \notin L(M)$	$H(x) = \langle M'_x \rangle$ where $L(M'_x) = \emptyset$

To justify the table: we use steps 1 and 2 in the algorithm for $H(x)$ in the first row of the table; we use step 3 in the algorithm for $H(x)$ for the second and third rows of the table, noticing that step 1 in the definition of M'_x guarantees that

$$L(M'_x) = \begin{cases} \{w\} \\ \emptyset \end{cases}$$

because all strings other than w are rejected, and then the simulation of the computation of M on w in step 2 determines which case we're in.

We can use the table to prove that for all strings x, $x \in A_{TM}$ iff $H(x) \notin E_{TM}$. The second row describes all x in A_{TM} and we see that in that case $H(x) = \langle M'_x \rangle$ with $L(M'_x) = \{w\} \neq \emptyset$ so $H(x) \notin E_{TM}$, as required. The first and third rows describe all $x \notin A_{TM}$ and we see that in these situation $H(x)$ is a string encoding a Turing machine whose language is empty, thus $H(x) \in E_{TM}$, as required.