Question 16 **(7 marks)**

Prove that the following problem is undecidable.

Input: a Turing machine M.
Question: Is there a string x which, if given as input to M, causes it to eventually erase everything on the tape (i.e., overwrite every letter by Blank), after which it never writes any other symbol?

You may use the fact that the halting problem is undecidable.

NEW SOLUTION (with mapping reductions):

We give a mapping reduction from the Diagonal Halting Problem to the given problem. We first define it, as a function f, and then show it is a mapping reduction.

This function f takes, as input, an input for the Diagonal Halting Problem, i.e., a Turing machine M (appropriately encoded). From M, the function f constructs a new Turing machine, which we call M', which works as follows:

M':
1. Input: x
2. Mark the first tape cell, then move the tape head to just past the end of x, i.e., to the first Blank cell.
3. Simulate the running of M on input M. This simulation treats the first Blank cell after x as the start of its tape, just for the simulation. So the simulation does not interfere with x. (The code for doing this simulation of M is hardcoded into M'. Note that M' only does this for the one specific machine M that it was constructed from.)
4. Go back to the start of the tape (i.e., to the very first letter of x, which was specially marked).
5. Move along the tape to the right, one cell at a time, overwriting each cell with Blank. (It doesn't matter if this goes on forever.)

The function f is computable, because all the parts of f can be computed from M.

- Note that M' is <u>not</u> an algorithm for computing the function f. Instead, M' is, itself, the *output* of f, when the input to f is M.

Now, if M halts on input M, then the simulation step in M' (Step 3) will eventually finish. Then M' goes on to Step 4, so it then erases the tape. This happens regardless of what the input x was, i.e., for any x.
On the other hand, if M does not halt on input M, then M' is stuck in Step 3 forever. This simulation does not erase the tape, since x is still sitting unaltered at the start of the tape. We conclude that M halts on input M if and only if M' eventually erases the entire tape,

for some input.
So, M is a YES-input for the Diagonal Halting Problem (i.e., it belongs to the corresponding language) if and only if M' is a YES-input for the problem given in the question.
Since we have a mapping reduction from the Diagonal Halting Problem, which is known to be undecidable, to the given problem, it follows that the given problem is also undecidable.

OLD SOLUTION (without mapping reductions):

Suppose (by way of contradiction) that this problem is decidable. Then there exists a decider, D, for it.

Now, take any input M to the Diagonal Halting Problem.

Construct a new Turing machine M' from M as follows.

> M':
> 1. Input: x
> 2. Mark the first tape cell, then move the tape head to just past the end of x, i.e., to the first Blank cell.
> 3. Simulate the running of M on input M. This simulation treats the first Blank cell after x as the start of its tape, just for the simulation. So the simulation does not interfere with x. (The code for doing this simulation of M is hardcoded into M'. Note that M' only does this for the one specific machine M that it was constructed from.)
> 4. Go back to the start of the tape (i.e., to the very first letter of x, which was specially marked).
> 5. Move along the tape to the right, one cell at a time, overwriting each cell with Blank. (It doesn't matter if this goes on forever.)

Now, if M halts on input M, then the simulation step in M' (Step 3) will eventually finish. Then M' goes on to Step 4, so it then erases the tape. This happens regardless of what the input x was, i.e., for any x.
On the other hand, if M does not halt on input M, then M' is stuck in Step 3 forever. This simulation does not erase the tape, since x is still sitting unaltered at the start of the tape.
We conclude that M halts on input M if and only if M' eventually erases the entire tape, for some input.
We can therefore use a decider for the given problem to decide the Diagonal Halting Problem. Given M, we construct M' and let x be any nonempty string. We use D to decide whether or not M' permanently erases the tape, and the answer from D immediately tells us whether M halts on input M.
So we have a decider for the Diagonal Halting Problem. But that problem is known to be undecidable. So we have a contradiction. So the assumption that the given problem is decidable must be incorrect. Therefore it is undecidable.

Question 17 **(3 marks)**

Explain how to obtain, from any language that is recursively enumerable but not decidable, another closely related language that is not recursively enumerable. (Proof is not required.)

Take the complement of the language.

Official use only
10

Question 15 **(6 marks)**

If L is a language and s is a string, then $L\Delta s$ denotes the language formed by *removing* s from L if it is there already, and *adding* s to L otherwise. In other words,

$$L\Delta s := \begin{cases} L \setminus \{s\}, & \text{if } s \in L, \\ L \cup \{s\}, & \text{if } s \notin L. \end{cases}$$

Prove that L is decidable if and only if $L\Delta s$ is decidable.

Suppose L is decidable. Let D be a decider for L. We use it to build the following decider for $L\Delta s$.

```
Input: string $x$.
If $x = s$ then return the opposite answer to $D(s)$,
else return $D(x)$.
```

This works because the only difference between the two languages is for the string s, on which they disagree; but any other string is either in both the languages or in neither of them.
Therefore $L\Delta s$ is decidable.
Exactly the same argument shows that, if $L\Delta s$ is decidable, then so is L. (In fact, if we now make D a decider for $L\Delta s$, then the above decider becomes a decider for L.) Therefore L is decidable if and only if $L\Delta s$ is decidable.

Official use only
6

Question 16 **(5 marks)**

Below is a proof that the Halting Problem is undecidable. But some parts are missing; these are shown as blank spaces, underlined.

Your task is to fill in the underlined spaces to complete the proof. In some cases, options are given in square brackets.

Proof. Assume that the Halting Problem is actually decidable.
Then the Halting Problem has a decider D.
Using D, we can construct a Turing machine E that does the following:

1. Input: encoding of a Turing machine M.
2. Run decider D to determine whether or not M halts when given itself as input.
3. **IF** the answer from D is **Yes**, then

   loop forever
   ____

4. **IF** the answer from D is **No**, then

   stop
   ____

Now consider what happens if E is given, as input, an encoding of *itself*.

If E halts on input E, then running D in line 2 gives the answer __Yes__ [Yes/No].

So, using statement __**3**__ [3 or 4], we see that D actually ________ loops forever ________.

On the other hand,

if E does not halt on input E, then running D in line 2 gives the answer __No__ [Yes/No].

So, using statement __**4**__ [3 or 4], we see that D actually ________ stops ________.

This is a contradiction.

So the Halting Problem must be undecidable.

Official use only
5

Question 17 **(6 marks)**

Suppose you have an enumerator M_1 for a language L and another enumerator M_2 for its complement $\overline{L}$.

(a) Explain how to construct a decider for L that uses the enumerators M_1 and M_2.

Decider for L:

Input: string x.
Run M_1 and M_2 in parallel. (This can be done by a loop which keeps track of the entire configuration of both machines and, in each iteration, simulates one step of each machine.)
Whenever either machine outputs a string, we check if that string equals x.
If we see x as an output string from M_1, then we know $x \in L$, and we stop and output Yes.
If we see x as an output string from M_2, then we know $x \in \overline{L}$, and we stop and output No.

We must eventually see x as an output string from *exactly one* of the two machines (not both, not neither), since one machine enumerates L and the other enumerates its complement, namely $\overline{L}$. (Every string belongs either to L or to $\overline{L}$.) So the above computation must always halt eventually.

Other algorithms are possible. E.g., for each $i \in \mathbb{N}$, simulate the first i steps of M_1 and then the first i steps of M_2. If either produces x, then output Yes or No according as x was produced by M_1 or M_2. If neither produces x, continue.

(b) What does this tell you about recursively enumerable (r.e.) languages whose complements are also recursively enumerable? (Only one short sentence is required here.)

Such languages must be decidable.

Official use only
6

Question 15 **(8 marks)**
Let L and K be languages. Suppose that K is finite.

Definition: The **symmetric difference** $L\Delta K$ consists of all strings that belong to L or K, *but not both.* In other words, $L\Delta K = (L \cup K) \setminus (L \cap K)$.

(a) Prove that there exists a mapping reduction from L to $L\Delta K$.

Because K is finite, $K \cap L$ and $K \setminus L$ are also finite, and hence both are decidable.

Here is such a mapping reduction.

Let s be a string in $K\Delta L$ and let s' be a string not in $K\Delta L$.

```
Input:   string $x$.
Use a $K$-decider to determine whether or not $x \in K$.
If   $x \in K \cap L$
//   $\ldots \Longrightarrow$ it's in $L$, so it must be mapped to a string in $K\Delta L$. But $x$ itself is not in $K\Delta L$,
//   so we can't just output $x$. Instead, output some fixed string known to be in $K\Delta L$.
{
      output $s$
}
else if   $x \in K \setminus L$
//   $\ldots \Longrightarrow$ it's not in $L$, so it must be mapped to a string outside $K\Delta L$. But $x$ itself *is*
//   in $K\Delta L$, so we can't just output $x$. Instead, output some fixed string known *not* to
//   be in $K\Delta L$.
{
      output $s'$
}
else     //   $x \notin K$, so $x \in L \Longleftrightarrow x \in K\Delta L$
{
      output $x$
}
```

This algorithm, together with the decidability of $K \cap L$ and $K \setminus L$, implies that this function is computable.

If $x \in L$, then either $x \in K$, in which case it is in $K \cap L$ and is mapped to s which is in $K\Delta L$ (first case of **if** statement), or $x \notin K$, in which case x is mapped to itself and is in $K\Delta L$ (third case of **if** statement).

If $x \notin L$, then either $x \in K$, in which case it is in $K \setminus L$ and is mapped to s' which is not in $K\Delta L$ (second case of **if** statement), or $x \notin K$, in which case x is mapped to itself and is not in $K\Delta L$ (third case of **if** statement).

We have given the mapping reduction and shown that it is indeed a mapping reduction from L to $K\Delta L$.

(b) Prove that L is undecidable if and only if $L \Delta K$ is undecidable.

($\Longrightarrow$)

The undecidability of L, together with the mapping reduction from L to $K \Delta L$, implies that $K \Delta L$ is undecidable.

($\Longleftarrow$)

If $K \Delta L$ is undecidable, then apply the forward direction we just proved to $K \Delta L$ instead of to L. (This is totally fine: it can be applied to any undecidable language, whatever that language may be called.) We deduce that $(K \Delta L) \Delta K$ is undecidable. But $K \Delta L \Delta K = L$. Therefore L is undecidable.

Alternative solution, using contrapositive:

If L is decidable, then using the decidability of K we see that $K \Delta L$ is decidable. (A string is in $K \Delta L$ if and only if it belongs to *exactly one* of K and L, and we can use deciders for K and L to determine whether or not this is the case.)

Official use only
8

Question 15 **(6 marks)**

Prove by contradiction that, if L_1 is undecidable, $L_2 \subseteq L_1$, and L_2 is decidable, then $L_1 \setminus L_2$ is undecidable.

(Here, $L_1 \setminus L_2$ is the set of strings that belong to L_1 but not L_2.)

Let L_1 and L_2 be languages such that L_1 is undecidable, $L_2 \subseteq L_1$, and L_2 is decidable.
Let C be a decider for L_2.
Assume, by way of contradiction, that $L_1 \setminus L_2$ is decidable. Let D be a decider for it.
Observe that $L_1 = L_2 \cup (L_1 \setminus L_2)$.
Then we can form a decider for L_1 as follows:

```
Input: $x$
Use $C$ to decide if $x \in L_2$.
Use $D$ to decide if $x \in L_1 \setminus L_2$.
If one of $C$ or $D$ answers Yes, then output Yes, since we then know that $x \in L_1$.
If both of $C$ and $D$ answer No, then output No, since we then know that $x \notin L_1$.
```

This shows that L_1 is decidable.
This is a contradiction, since we are given that L_1 is undecidable.
So our assumption, that $L_1 \setminus L_2$ is decidable, was incorrect.
Therefore $L_1 \setminus L_2$ is undecidable.

Official use only
6

Question 16 **(5 marks)**

Suppose you have an enumerator M for a language L.

Give an algorithm that accepts precisely the strings in L.

Your algorithm may use M.
For strings not in L, the algorithm must either reject or loop forever.

```
Input: $x$
Loop:
{
    Run $M$ until it generates another string.
    If that string equals $x$, then Accept.
}
```

Official use only
5

Question 18 **(8 marks)**

Prove that the problem BIG INDEPENDENT SET problem is NP-complete, by reduction from HUGE INDEPENDENT SET. You may assume that HUGE INDEPENDENT SET is NP-complete.

Definitions:

For any positive integer k, an **independent set** in a graph G is a subset X of the vertex set of G such that no two vertices in X are adjacent.

HUGE INDEPENDENT SET
Input: Graph G, with an even number of vertices.
Question: Does G have an independent set of size $\geq n/\mathbf{2}$?

BIG INDEPENDENT SET
Input: Graph G.
Question: Does G have an independent set of size $\geq n/\mathbf{3}$?

In each of these definitions, n is the number of vertices in the graph G.

BIG INDEPENDENT SET is in NP, since it has a polynomial-time verifier as follows:

Input: graph G
Certificate: a set of vertices of G
Check that the number of vertices in the certificate is $\geq n/3$.
For each pair of vertices in the certificate, check they are not adjacent.
If all these checks pass, then Accept, otherwise Reject.

The work required in checking is counting the number of vertices in the certificate, doing one division and one comparision, and iterating over all pairs of vertices in the certificate. The number of pairs of vertices in the certificate is $O(n^2)$. So the certificate can be verified in polynomial time.

Reduction from HUGE INDEPENDENT SET to BIG INDEPENDENT SET:

Let G be a graph with an even number of vertices (called n).
Add $n/2$ new vertices to G, and join them all up to each other as well as to every vertex of G. Let H be the new graph so formed, and let p be the number of vertices of H. Then $p = n + (n/2) = 3n/2$.
The largest independent set of H is the same as the largest independent set of G, since the new vertices in H can never be added to any independent set in G. So,

$$\begin{aligned} & G \in \text{HUGE INDEPENDENT SET} \\ \iff & G \text{ has an independent set of size} \geq n/2 \\ \iff & H \text{ has an independent set of size} \geq n/2 \end{aligned}$$

$$\begin{aligned}
&\iff H \text{ has an independent set of size } \geq p/3 \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(since } p/3 = (3n/2)/3 = n/2\text{)} \\
&\iff H \in \text{BIG INDEPENDENT SET}
\end{aligned}$$

It remains to show that the reduction runs in polynomial time. To see this, observe that the reduction first creates $n/2$ new vertices (time $O(n)$), and then adds edges between each pair of them, and between each new vertex and each old vertex. The number of pairs of new vertices is $\binom{n/2}{2}$, which is $O(n^2)$. The number of pairs consisting of one new vertex and one old vertex is $(n/2) \cdot n$, which is also $O(n^2)$. So the total time required to construct H from G is at most polynomial.

Official use only
8