After this tutorial you should be able to:

1. Show that certain languages are not decidable.
2. Show that certain languages are in **P**.
3. Show that certain languages are in **NP**.

Undecidability

Problem 1. Show that the halting problem is undecidable. I.e., that the language

$$L_{\text{TM-Halting}} := \{\langle M, w\rangle \mid M \text{ is a TM that halts on input } w\}$$

is not decidable.

You can either do this directly by a self-referencing argument, or you can suppose that the problem is decidable and use this to build a decider for the TM-acceptance problem, which we know is undecidable. Give the second approach.

Solution 1. We give the second approach.

1. Suppose H decides this language, i.e., $H(\langle M, w\rangle)$ accepts if M halts on w, and rejects otherwise.
2. We build a TM D to decide $L_{\text{TM-acceptance}}$, which is a contradiction.
3. TM D takes $\langle M, w\rangle$ as input:
   (a) It runs $H(\langle M, w\rangle)$ and looks at whether it accepts or rejects.
   (b) If H rejects, then D rejects.
   (c) If H accepts, then D runs M on input w and waits for it to halt (which it will!)
   (d) If M accepts then D accepts, and if M rejects then D rejects.

Problem 2. Prove that the equivalence problem for TMs is not decidable.

Hint: Suppose it were decidable; take a decider D for it; and build a decider for the acceptance problem for TMs.

Second hint: Given a TM M and input w, consider a TM M_w that works as follows: it ignores its input, it runs M on w, and accepts/rejects/diverges as M does. Note that M accepts w implies $L(M') = \Sigma^*$, and M does not accept w implies $L(M') = \emptyset$.

Solution 2. Let M_{all} be some fixed TM with $L(M_{all}) = \Sigma^*$. For instance, M_{all} could simply ignore its input and go immediately into the accepting state.

Let D be a decider for the equivalence problem for TMs. We build another decider E that works as follows. It takes $\langle M, w\rangle$ as input, writes $\langle M_w, M_{all}\rangle$ on a second tape and runs D on that tape, and if D accepts then E accepts, and if D rejects then E rejects.

We now show that E decides the acceptance problem for TMs, which is a contradiction since we know (Turing) that no such decider exists.

First, E is a decider since D is a decider. Second, we show that E accepts $\langle M, w\rangle$ iff M accepts w. Indeed, E accepts $\langle M, w\rangle$ if and only if D accepts $\langle M_w, M_{all}\rangle$ if and only if $L(M_w) = \Sigma^*$ if and only if M accepts w (by the second hint).

Problem 3. Recall the non-acceptance problem for TMs from lectures:

$$L_{\text{TM-non-acceptance}} = \{\langle M, w\rangle \mid M \text{ is a TM that does not accept } w\}$$

I argued it was not recognisable. But my proof was not watertight. What technical problem was missing?

Hint: What is the complement of the language $L_{\text{TM-non-acceptance}}$?

Solution 3. The TM-acceptance problem is

$$L_{\text{TM-acceptance}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

The complement of $L_{\text{TM-acceptance}}$ is not $L_{\text{TM-not-acceptance}}$. The reason is that a string is not in $L_{\text{TM-acceptance}}$ means that either it is of the form $\langle M, w\rangle$ and the TM M does not accept w, or it is not of the form $\langle M, w\rangle$ for any TM M. So really, the complement of $L_{\text{TM-acceptance}}$ is

$$L_{\text{TM-not-acceptance}} \cup E$$

where E is the language of strings that are not of the form $\langle M, w\rangle$. Note that E is decidable (we can easily tell if a string is not in the TM format we use in class, or if it has a syntax error).

Here, then, is the full argument that shows that $L_{\text{TM-non-acceptance}}$ is not recognisable.

1. Assume $L_{\text{TM-non-acceptance}}$ is recognisable.
2. We will show that the acceptance problem for TMs is decidable (which is a contradiction, see lecture slides where we show it is not).
3. We know the acceptance problem for TMs is recognisable (see lecture slides).
4. The complement of the acceptance problem for TMs is

   $$L_{\text{TM-non-acceptance}} \cup E$$

   where E is defined above.

5. But then this language is recognisable: a TM takes a string as input, checks if it is in E, if not, so the input is in the form $\langle M, w\rangle$, runs the recogniser for the TM-non-acceptance problem on it. (More generally, the union of a recognisable language and a decidable language is recognisable).
6. Thus both the acceptance problem for TMs and the non-acceptance problem for TMs are recognisable, which by the Theorem in lectures means it is decidable (which is what we wanted to prove to get a contradiction).

Problem 4. (Optional) Show that there is a decision problem that can't be solved by any Python program. Do not use the Church-Turing Thesis.

Hint. Formalise the acceptance problem for Python programs, and show that there is no Python program that decides it.

Solution 4. Let L be the set of strings $\langle P, w\rangle$ where P is a Python program that, on input w, halts and returns `true`.

To show that L is not decidable by a Python program, we can use the exact same argument that showed that the acceptance problem for TMs is not decidable. Suppose there is a Python program A that does decide it. Build a new Python program B that takes P as input, runs A on input $\langle P, \langle P\rangle\rangle$, waits for it to halt, and returns the opposite answer. Then ask what happens when we run B on input $\langle B\rangle$ to get a contradiction.

Polynomial Time (P)

Problem 5. Pick two of the following problems, and show they are in **P**:

1. $L(a^*b^*)$
2. $\{w\#w^R : w \in \{a, b\}^*\}$, where w^R is the reverse of w.
3. $\{ww : w \in \{a, b\}^*\}$

(Aside: which of these languages is not regular?)

Solution 5.

1. The first language is regular, and every regular language is decidable in linear time. To see this, note that a DFA can be viewed as a TM that, at every step, reads the next symbol and moves one cell to the right, and changes state just as the DFA does. When the TM reads a blank symbol, it accepts/rejects according to whether the DFA is in a final/non-final state.
2. Here is an implementation-level description: check the first unmarked letter is equal to the last unmarked letter, mark these, and repeat, until the only unmarked letter is #. This decider runs in quadratic time.

3. We give a high-level description/algorithm. On input $x = x_1x_2 \cdots x_n \in \Sigma^*$:
   (a) Let $n = len(x)$.
   (b) If n is odd then reject.
   (c) Let $mid = n/2$.
   (d) Let $L = x[1, mid]$ and $R = x[mid + 1, n]$.
   (e) If $L = R$ then accept, else reject.
   The complexity of this algorithm is $\Theta(n)$ since to split the string into two would take linear time, as would checking equality between two strings.

Problem 6. Calculate the time-complexity of the following TMs (give your answer in big-Oh notation):

1. TM1
2. TM2

Solution 6.

1. $\Theta(n)$. On all strings of length n the TM makes a single scan from left to right, and halts by the time it reaches the end of the string. This shows that the time complexity is $O(n)$. On strings in $L(a^*b^*)$ it only halts at the end of the string, which shows that the time complexity is $\Theta(n)$.
2. $\Theta(n^2)$. On all strings of length n the TM makes at most $n/2$ passes, each pass scanning $n/2$ cells (the ones between the leftmost 0 and the leftmost 1). This shows that the time complexity is $O(n^2)$. On strings in the language $\{0^n1^n : n \geq 1\}$ it makes exactly $n/2$ passes; this shows that the time complexity is $\Theta(n^2)$.

Problem 7. Show that the union of two **P** languages is also in **P**.

Solution 7. The proof in class that the decidable languages are closed under union also shows that that **P** languages are closed under union. Indeed, if M_1, M_2 have polynomial-time complexity, say $p_1(n)$ and $p_2(n)$, then the machine for their union that runs M_1, then runs M_2, and accepts if either of the machines accepted, and rejects otherwise, runs in time $p_1(n) + p_2(n)$, which is also a polynomial.

Nondeterministic Polynomial Time (NP)

Problem 8. Consider the language $L_{\text{composite}}$ of decimal strings x whose numeric values are not prime. E.g., the following strings are in the language

$4, 6, 10, 14, 25, 49, 292$, and the following strings are not $3, 7, 11, 97$. Show that $L_{\text{composite}} \in \mathbf{NP}$.

Solution 8. We give a high-level description of a nondeterministic decider for $L_{\text{composite}}$.

On input x:

1. nondeterministically guess y with $1 < y < x$.
2. test if y divides x. If yes, then accept, else reject.

A slightly more implementation-level description could do the following. Suppose numbers are encoding least-significant-digit first.

1. On a second tape, write a sequence of digits, and stop no later than the length of x (the TM knows when to stop because it can scan the input x on the first tape at the same time). The choice of digit at each step is chosen nondeterminitically.
2. Let y be the value written on the second tape when it stops writing.
3. If $x \geq y$ then reject.
4. Test if y divides x. If yes, then accept, else reject.

The number of steps the machine takes before producing y is $O(|x|)$. Checking if y divides x can be done in polynomial time in $|x|$ (Euclid's Algorithm!).

What happens if you try make a similar construction to try show that the complement of $L_{\text{composite}}$, i.e., the set of prime numbers, is in **NP**?

Problem 9. In this problem you will show that the 3COL problem is in **NP**. The 3COL problem takes an undirected graph $G = (V, E)$ as input, and decides whether or not G has a proper 3-colouring, i.e., a partition of V set into three sets such that no edge of G has both its endpoints in the same set. The language corresponding to this problem is

$$L_{\text{3col}} := \{\langle G \rangle : G \text{ has a proper 3-colouring}\}$$

Show that the L_{3col} is in **NP** by giving a high-level description of a NTM that decides it.

Also, give an implementation-level description.

Solution 9. Here is a high-level description of a nondeterministic decider for L_{3col}.

Take $\langle (V, E) \rangle$ as input:

1. Nondeterministically pick three subsets R, G, B of vertices that partition V.

2. For every edge (u, v) in E, check if both u and v are in the same set.
3. If none is, then accept, else reject.

Problem 10. (optional) A language is in **EXP** (aka *deterministic exponential time*) means that it is decidable by a TM with time-complexity $t(n) = 2^{n^k}$ for some k. Show that **NP** $\subseteq$ **EXP**. In words, this says that every **NP** problem is decidable in exponential time.

Hint. If N is a decider with time-complexity $t(n)$, what is the time-complexity of a (deterministic) decider D that simulates N?

Solution 10. This follows from Theorem 7.8 and Theorem 7.11 in Sipser.

Theorem 7.8 says that for DTMs, one can simulate multiple tapes by a single tape at a cost of a quadratic blowup, i.e., if a DTM with multiple tapes has time complexity $t(n)$ then there is an equivalent DTM with one tape that runs in time $O(t(n)^2)$. This can be seen by analysing the construction we gave in class for simulating a multitape machine by a single tape machine. See Sipser for details.

Theorem 7.11 says that for a given NTM, let b be the maximum number of nondeterministic choices given N's transition function. Then a computation tree on an input of size n is b-branching (i.e., every node has at most b children), and has depth at most $t(n)$. Thus, the size of the tree is at most $b^{t(n)} = 2^{O(t(n))}$, and since a BFS takes linear time in the size of the given tree/graph, the TM D runs in time $2^{O(t(n))}$.

So, if $f(n)$ is a polynomial, then D runs in exponential time. Note that here we are using the convention that an *exponential* function is of the form $b^{p(n)}$ for a polynomial $p(n)$ and constant b. Thus, e.g., 3^{n^2} is exponential, as is 3^n and 3^{2n}.

Both of these constructions assume that $t(n) \geq n$.