

Examples of undecidable languages

In a previous handout we saw that the language

$$L_d = \{w \in \Sigma^* : w \notin L(M_w)\}$$

is non-Turing-recognizable. In this handout we discuss several new examples of undecidable languages; some of these languages are Turing-recognizable, others are not. We also discuss the notion of a *universal Turing machine*, which is a Turing machine that simulates other Turing machines.

1 A UNIVERSAL TURING MACHINE

Consider the following language:

$$A_{\text{DTM}} = \{\langle M, w \rangle : M \text{ is a 1DTM that accepts the string } w\}.$$

We will show that this language is Turing-recognizable, yet undecidable. But first, let us clarify a few concepts.

The language A_{DTM} is analogous to the languages A_{DFA} , A_{CFG} , and so on that we saw in our lecture, except now it concerns 1DTMs rather than DFAs, CFGs, *etc.* For clarity, let us be precise about the meaning of $\langle M, w \rangle$:

- $\langle M, w \rangle$ is an encoding of a 1DTM M and a string w . The symbols used by the encoding are drawn from some fixed alphabet Σ .
- The input alphabet of M and the alphabet of w need not be the same as Σ —they could be larger alphabets, for example.

In particular, there is nothing conceptually new here beyond what we already discussed in our lecture about the languages A_{DFA} , A_{CFG} , *etc.* This time, M is a DTM, but that is the only difference among these languages.

Theorem 1. A_{DTM} is Turing-recognizable.

Proof. The idea is very simple: we will define a DTM U that takes $\langle M, w \rangle$ as input and simulates M on input w . If the simulation reveals that M accepts or rejects w then U does likewise. If it so happens that M runs forever on input w then the simulation will also run forever—we make no attempt to escape this condition. Figure 1 contains a brief description of what U might look like. It is clear that $L(U) = A_{\text{DTM}}$, and therefore A_{DTM} is Turing-recognizable. \square

The DTM U constructed in the proof of Theorem 1 is an example of a *universal Turing machine*. The existence of a DTM such as U should not come as a surprise. After all, you could certainly write a Turing machine simulator in your favourite high-level programming language, and we argued in our lecture that DTMs are equivalent to ordinary computers.

The name “universal” is intended to emphasize the fact that U can simulate *all* Turing machines. It may seem obvious to you that such machines should exist, but it is a conceptually

Given input $\langle M, w \rangle$;
 Write $\langle q_0 w \rangle$ on tape 2 where q_0 is the start state of M ;
 (This tape always encodes the current configuration of M in input w .)
while *tape 2 does not encode an accepting or rejecting configuration* **do**
 | Using the description of M on the input tape, transform the contents $\langle \alpha \rangle$ of tape 2 to $\langle \beta \rangle$,
 | where β is the configuration of M satisfying $\alpha \vdash_M \beta$;
end
if *tape 2 encodes an accepting configuration* **then** accept **else** reject;

Figure 1: A brief sketch of the universal Turing machine U .

Given input $w \in \Sigma^*$;
 Let M_w be the 1DTM encoded by w ;
 (Such an encoding was discussed in our lecture).
 Run A on input $\langle M_w, w \rangle$;
if A *accepts* $\langle M_w, w \rangle$ **then** reject **else** accept;

Figure 2: The DTM D from the proof of Theorem 2.

important observation about computation that “programmable” Turing machines exist. (It is remarkable that Turing, Church, Gödel, and others had this sort of insight without the experience of working with actual computers.)

Of course, you could define U in many different ways. It is important, though, to keep in mind that U has some fixed number of states and a fixed tape alphabet, yet it simulates DTMs with potentially *more* states and *larger* tape alphabets. (This is why U , as defined in Figure 1, operates on *encodings* $\langle \alpha \rangle$ of configurations α of M .)

2 THE HALTING PROBLEM

We claimed in Section 1 that the language A_{DTM} is Turing-recognizable, yet undecidable. But we only proved the first claim (Theorem 1). In this section, we prove that A_{DTM} is undecidable. We then use this fact to establish that another closely-related language called *HALT* is also undecidable. (The language *HALT* corresponds to the historically significant *halting problem*.)

Theorem 2. A_{DTM} is not decidable.

Proof. Assume toward a contradiction that A_{DTM} is decidable and let A be the DTM that decides A_{DTM} . We will use the existence of A to decide L_d , which we already know to be undecidable.

Toward that end, let D be the DTM with input alphabet $\Sigma = \{0, 1\}$ described in Figure 2. First let us note that D *halts on every input*. This follows from the assumption that A decides A_{DTM} , and therefore does not itself run forever on any input (including $\langle M_w, w \rangle$).

Now consider the following two possibilities for any choice of an input string $w \in \Sigma^*$:

1. $w \notin L(M_w)$. Then $\langle M_w, w \rangle \notin A_{\text{DTM}}$, and therefore A *rejects* $\langle M_w, w \rangle$. Thus, D *accepts* w .
2. $w \in L(M_w)$. Then $\langle M_w, w \rangle \in A_{\text{DTM}}$, and therefore A *accepts* $\langle M_w, w \rangle$. Thus, D *rejects* w .

We now have what we need:

- D halts on every input, and

Given input $\langle M, w \rangle$;
 Run H on $\langle M, w \rangle$;
if H rejects then reject;
 Run U on input $\langle M, w \rangle$;
 (That is, simulate M on input w . It must be the case that M halts on w .)
if U accepts then accept else reject;

Figure 3: The DTM A from the proof of Theorem 3.

- $L(D) = \{w \in \Sigma^* : w \notin L(M_w)\} = L_d$.

It follows that L_d is decidable. But we already know that this is not true, and so we have reached a contradiction. Thus, A_{DTM} is not decidable. \square

Let us now turn our attention to the *halting problem*, wherein a DTM and an input string are given, and the goal is to determine whether the DTM halts on that input. The corresponding language is

$$\text{HALT} = \{\langle M, w \rangle : M \text{ is a DTM that halts on input } w\}.$$

Theorem 3. *HALT is not decidable.*

Proof. Assume toward a contradiction that *HALT* is decidable and let H be the DTM that decides *HALT*. We will use this DTM to decide A_{DTM} , which we already know to be undecidable (Theorem 2).

Toward that end, let A be the DTM described in Figure 3. Let us consider the three possibilities for a given input $\langle M, w \rangle$:

1. M accepts w .
2. M rejects w .
3. M runs forever on input w .

If M accepts w then we have

- H accepts $\langle M, w \rangle$ (because M halts on w),
- U accepts $\langle M, w \rangle$

and therefore A accepts $\langle M, w \rangle$. If M rejects w then we have

- H accepts $\langle M, w \rangle$ (because M halts on w),
- U rejects $\langle M, w \rangle$

and therefore A rejects $\langle M, w \rangle$. Finally, if M runs forever on w then we have

- H rejects $\langle M, w \rangle$

and therefore A rejects $\langle M, w \rangle$. In particular, A never attempts to run U in this case.

We conclude that A decides A_{DTM} . But we already know that A_{DTM} is not decidable, so we have a contradiction. Thus *HALT* is not decidable. \square

Given input $\langle M, w \rangle$;
 Run the universal Turing machine U from Figure 1 on input $\langle M, w \rangle$;
 Accept;

Figure 4: The DTM H_U from the proof of Theorem 4.

Incidentally, it is easy to see that $HALT$ —like A_{DTM} —is Turing-recognizable. The proof of this fact, which employs the universal Turing machine U of Section 1, is almost identical to the proof that A_{DTM} is Turing-recognizable (Theorem 1).

Theorem 4. *$HALT$ is Turing-recognizable.*

Proof. Let H_U be the DTM described in Figure 4. Note that M_U accepts if and only if U halts, which occurs if and only if M halts on input w . Hence, $L(H_U) = HALT$, and therefore $HALT$ is Turing-recognizable. \square

3 DECIDING WHETHER A TURING MACHINE ACCEPTS NO STRINGS

In Section 2 we proved that A_{DTM} and $HALT$ are undecidable (and Turing-recognizable). In this section we will prove that the language

$$E_{DTM} = \{\langle M \rangle : L(M) = \emptyset\}$$

is undecidable, yet its *complement* is Turing-recognizable. (By contrast, A_{DTM} and $HALT$ are Turing-recognizable, but their complements are not.)

Later in the course, we will see that the undecidability of E_{DTM} —combined with the Turing-recognizability of the complement $\overline{E_{DTM}}$ —implies that E_{DTM} is *not* Turing-recognizable. (By similar reasoning, it also follows that neither $\overline{A_{DTM}}$ nor \overline{HALT} are Turing-recognizable.)

In this section we will also start to develop tools that are helpful for proving other languages undecidable.

3.1 UNDECIDABILITY

We will prove that E_{DTM} is undecidable using a similar strategy to that employed in the proofs that A_{DTM} and $HALT$ are undecidable. Let us describe that strategy in a general setting. Suppose L is a language that we wish to prove undecidable:

1. Assume toward a contradiction that the language L is decidable.
2. Use this fact to construct a DTM that decides some language L' that we already know to be undecidable, which yields the desired contradiction.

Sometimes this strategy works easily and sometimes it is harder to make work. It helps if you make a good choice for L' , so the more languages we prove to be undecidable, the more options we have for future problems.

It also helps to know some tricks for constructing DTMs to get this sort of contradiction. (What we are doing is establishing a *reduction* from L' to L —this notion will be formalized later in the course.)

Theorem 5. *E_{DTM} is not decidable.*

M_A :	Given input $\langle M, w \rangle$; Compute the encoding $\langle N_w \rangle$ of a new DTM N_w described below ; Run M_E on input $\langle N_w \rangle$; if M_E accepts then reject else accept;
N_w :	Given input x ; Erase the input x ; Write w on the tape; Run M on w ;

Figure 5: The DTM M_A from the proof of Theorem 5.

Proof. Assume toward a contradiction that E_{DTM} is decidable and let M_E be a DTM that decides E_{DTM} . We will use the existence of M_E to decide A_{DTM} , which we already know to be undecidable (Theorem 2).

Toward that end, let M_A be the DTM described in Figure 5. What does M_A do? If the input is not of the form $\langle M, w \rangle$ for a DTM M and a string w then M_A implicitly rejects, so let us assume that the input has the proper form. There are two cases:

1. $\langle M, w \rangle \in A_{\text{DTM}}$. In other words, M accepts w . In this case, we have $L(N_w) = \Sigma^*$ where Σ is the input alphabet of M . (To see this, note that for every input string $x \in \Sigma^*$, N_w erases x and runs M on w . As M accepts w , N_w accepts every input string $x \in \Sigma^*$.)
As $L(N_w)$ is non-empty, M_E rejects $\langle N_w \rangle$ and therefore M_A accepts $\langle M, w \rangle$.
2. $\langle M, w \rangle \notin A_{\text{DTM}}$. In other words, M either rejects w or runs forever on w . By reasoning similar to that of case 1, we have $L(N_w) = \emptyset$. Therefore M_E accepts $\langle N_w \rangle$, and so M_A rejects $\langle M, w \rangle$.

In particular, M_A decides A_{DTM} , which contradicts the fact that A_{DTM} is not decidable. Therefore, E_{DTM} is not decidable. □

The “trick” in the proof of Theorem 5 is to “hard code” inputs into DTMs—that is, define them so that they erase their input and run some DTM on a *fixed* input.

3.2 TURING-RECOGNIZABILITY OF THE COMPLIMENT

In order to show that $\overline{E_{\text{DTM}}}$ is Turing-recognizable, we must first consider the following *decidable* language:

$$STEP = \{\langle M, w, t \rangle : M \text{ is a DTM that halts on input } w \text{ after } t \text{ or fewer steps}\}.$$

It is easy to see that $STEP$ is decidable: just simulate M on input w for t steps and see what happens. There is no fear of this simulation running forever because it always terminates after at most t steps. The technical details of the simulation are similar to those of the construction of the universal Turing machine U from Figure 1, and we will not bother ourselves here with a formal proof that $STEP$ is decidable. Instead, we will use the decidability of $STEP$ to show that $\overline{E_{\text{DTM}}}$ is Turing-recognizable.

Theorem 6. $\overline{E_{\text{DTM}}}$ is Turing-recognizable.

```

Given input  $w$ ;
if  $w \neq \langle M \rangle$  for some DTM  $M$  then accept;
(Hereafter, we assume that the input is of the form  $\langle M \rangle$ .)
for  $t = 1, 2, \dots$  do
    foreach string  $x$  with  $|x| \leq t$  do
        | Decide whether  $\langle M, x, t \rangle \in STEP$ ;
        | if  $\langle M, x, t \rangle \in STEP$  then Simulate  $M$  on input  $x$  and accept if  $M$  accepts  $x$ ;
    end
end

```

Figure 6: The DTM M_{NE} from the proof of Theorem 6.

Proof. Consider the DTM M_{NE} described in Figure 6. (The subscript “NE” stands for “not empty.”) We now prove that M_{NE} recognizes $\overline{E_{DTM}}$. There are three possible cases:

1. Obviously, if the input string w does not encode a DTM—that is, $w \neq \langle M \rangle$ for any DTM M —then $w \in L(M_{NE})$.
2. If the input string w encodes a DTM M with $\langle M \rangle \in E_{DTM}$ then M never accepts any input, so M_{NE} must run forever on input w . In particular, $w \notin L(M_{NE})$.
3. Conversely, if the input string w encodes a DTM M with $\langle M \rangle \notin E_{DTM}$ then M accepts at least one input string x after some number of steps k . When $t = \max\{|x|, k\}$ the DTM M_{NE} will discover this fact (unless it has already found some different string accepted by M). In particular, $w \in L(M_{NE})$.

We have thus established that $L(M_{NE}) = \overline{E_{DTM}}$, and hence $\overline{E_{DTM}}$ is Turing-recognizable. \square

The trick employed in the proof of Theorem 6 is called *dovetailing*. Dovetailing is a powerful technique for proving languages decidable or Turing-recognizable. Essentially, the idea is to simulate a DTM M on *all* possible inputs *in parallel*. As demonstrated in Figure 6, this idea is implemented by simulating M only on input strings with length at most t , and only for at most t steps. Clearly, these simulations must terminate. When they do, t is incremented and the whole process repeated.

4 DECIDING WHETHER TWO TURING MACHINES ACCEPT THE SAME LANGUAGE

In this section we will prove that the language

$$EQ_{DTM} = \{ \langle M_1, M_2 \rangle : M_1 \text{ and } M_2 \text{ are DTMs with } L(M_1) = L(M_2) \}$$

is undecidable. The proof is made especially easy by the work of Section 3, wherein the undecidability of E_{DTM} was proved.

Theorem 7. EQ_{DTM} is not decidable.

Proof. Assume toward a contradiction that EQ_{DTM} is decidable and let M_{EQ} be a DTM that decides EQ_{DTM} . We will use the existence of M_{EQ} to decide E_{DTM} , which we already know to be undecidable (Theorem 5).

E :	Given input $\langle M \rangle$; Compute the encoding $\langle M_\emptyset \rangle$ of a new DTM M_\emptyset described below; Run M_{EQ} on input $\langle M, M_\emptyset \rangle$; if M_{EQ} accepts then accept else reject;
M_\emptyset :	Given input x ; Reject;

Figure 7: The DTM E from the proof of Theorem 7.

Toward that end, let E be the DTM described in Figure 7. What does E do? If the input is not of the form $\langle M \rangle$ for a DTM M then E implicitly rejects, so let us assume that the input has the proper form. (In the future, we only mention this case explicitly when some non-trivial action must be taken when the input is not of the proper form, such as in the proof of Theorem 6.) There are two non-trivial cases:

1. $\langle M \rangle \in E_{\text{DTM}}$. In other words, $L(M) = \emptyset$. In this case, we have $L(M) = L(M_\emptyset)$, implying that M_{EQ} accepts $\langle M, M_\emptyset \rangle$ and so E accepts $\langle M \rangle$.
2. $\langle M \rangle \notin E_{\text{DTM}}$. In other words, $L(M)$ is non-empty. In this case, we have $L(M) \neq L(M_\emptyset)$, implying that M_{EQ} rejects $\langle M, M_\emptyset \rangle$ and so E rejects $\langle M \rangle$.

In particular, E decides E_{DTM} , which contradicts the fact that E_{DTM} is not decidable. Therefore, EQ_{DTM} is not decidable. \square

5 DECIDING WHETHER A TURING MACHINE ACCEPTS A REGULAR LANGUAGE

Our next example of an undecidable language is addresses the question of whether the language accepted by a given Turing machine might also be accepted by a DFA. The language is defined as follows:

$$REG = \{\langle M \rangle : M \text{ is a DTM such that } L(M) \text{ is regular}\}.$$

(At this point, we cease writing subscripts on language names. You could call this language REG_{DTM} if you prefer.)

Theorem 8. *REG is not decidable.*

Proof. Assume toward a contradiction that REG is decidable and let M_{REG} be a DTM that decides REG . We will use the existence of M_{REG} to decide $HALT$, which we already know to be undecidable (Theorem 3).

Toward that end, let M_{HALT} be the DTM described in Figure 8. What does M_{HALT} do? There are two cases:

1. $\langle M, w \rangle \in HALT$. In other words, M halts on input w . Because of this, it holds that $L(Q_{M,w}) = L(M_\emptyset)$, which is not regular. Therefore $\langle Q_{M,w} \rangle \notin REG$, from which it follows that M_{HALT} accepts $\langle M, w \rangle$.
2. $\langle M, w \rangle \notin HALT$. In other words, M does not halt on input w . In this case, $Q_{M,w}$ runs forever on all inputs, so $L(Q_{M,w}) = \emptyset$, which is regular. Therefore $\langle Q_{M,w} \rangle \in REG$, from which it follows that M_{HALT} rejects $\langle M, w \rangle$.

M_{HALT} :	Given input $\langle M, w \rangle$; Compute the encoding $\langle Q_{M,w} \rangle$ of a new DTM $Q_{M,w}$ described below; Run M_{REG} on input $\langle Q_{M,w} \rangle$; if M_{REG} accepts then reject else accept;
$Q_{M,w}$:	Given input x ; Simulate M on input w ; Run M_0 on input x ; (M_0 is a new DTM with input alphabet $\Gamma = \{1\}$ described below.)
M_0 :	Given input 1^n ; if $n = 2^k$ for some positive integer k then accept else reject; (Note: $L(M_0)$ is not regular.)

Figure 8: The DTM M_{HALT} from the proof of Theorem 8.

In particular, M_{HALT} decides $HALT$, which contradicts the fact that $HALT$ is undecidable. Therefore, REG is not decidable. \square

Of course, the preceding proof would be correct even if we replace the DTM M_0 by any other DTM M' for which $L(M')$ is not a regular language.

Notice that the DTM $Q_{M,w}$ from the proof of Theorem 8 is essentially a “wrapper” for the fixed DTM M_0 , which was chosen to accept some non-regular language. The trick in this proof is that the wrapper only gets a chance to run M_0 when M halts on input w , so any DTM that is able to say something non-trivial about $L(Q_{M,w})$ must also be able to solve the halting problem.

6 DECIDING WHETHER A TURING MACHINE ACCEPTS EVERY STRING

Our final example of an undecidable language has the property that neither it or its complement are Turing-recognizable. That language is defined as follows:

$$ALL = \{ \langle M \rangle : M \text{ is a DTM with input alphabet } \Sigma \text{ and } L(M) = \Sigma^* \}.$$

Informally, ALL is the set of DTMs that accept every input string.

6.1 NON-TURING-RECOGNIZABILITY

Theorem 9. *ALL is not Turing-recognizable.*

Proof. Assume toward a contradiction that ALL is Turing-recognizable and let M_{ALL} be a DTM with $L(M_{ALL}) = ALL$. (Note: we cannot assume that M_{ALL} halts on all inputs—all we know is that it accepts x if and only if $x \in ALL$.) We will use the existence of M_{ALL} to show that L_d is Turing-recognizable, which we already know is not the case.

Toward that end, let M be the DTM with input alphabet $\Sigma = \{0, 1\}$ described in Figure 9. What does M do? There are two cases:

1. $w \in L_d$. In other words, M_w does not accept w . Because of this, it holds that R_w accepts every string x . Therefore $\langle R_w \rangle \in ALL$ and so M_{ALL} must accept $\langle R_w \rangle$. Thus, M accepts w .

$M:$	Given input $w \in \Sigma^*$; Compute the encoding $\langle R_w \rangle$ of a new DTM R_w with input alphabet Σ described below; Run M_{ALL} on input $\langle R_w \rangle$; if M_{ALL} accepts then accept else reject;
$R_w:$	Given input $x \in \Sigma^*$; Simulate M_w on input w for $ x $ steps; (Here M_w denotes the 1DTM encoded by w as described in the definition of L_d .) if M_w accepts during this simulation then reject else accept;

Figure 9: The DTM M from the proof of Theorem 9.

$M:$	Given input $w \in \Sigma^*$; Compute the encoding $\langle P_w \rangle$ of a new DTM P_w with input alphabet Σ described below; Run $M_{\overline{ALL}}$ on input $\langle P_w \rangle$; if $M_{\overline{ALL}}$ accepts then accept else reject;
$P_w:$	Given input $x \in \Sigma^*$; Erase x and run M_w on input w ; (Here M_w denotes the 1DTM encoded by w as described in the definition of L_d .) if M_w accepts then accept else reject;

Figure 10: The DTM M from the proof of Theorem 10.

2. $w \notin L_d$. In other words, M_w accepts w . For short strings x it may or may not be the case that R_w accepts x , but for sufficiently long strings x it must hold that M_w accepts w within $|x|$ steps and therefore R_w does *not* accept x . In particular, $\langle R_w \rangle \notin ALL$ and so M_{ALL} does not accept $\langle R_w \rangle$. Consequently, M does not accept w .

In particular, $L(M) = L_d$, implying that L_d is Turing-recognizable—a contradiction. Therefore, ALL is not Turing-recognizable. \square

6.2 NON-TURING-RECOGNIZABILITY OF THE COMPLIMENT

It just so happens that the complement \overline{ALL} of ALL is also not Turing-recognizable. This language is formally defined as follows:

$$\overline{ALL} = \left\{ x : \begin{array}{l} \text{either } x \text{ does not encode a DTM, or } x = \langle M \rangle \text{ where } M \text{ is a DTM for which} \\ \text{there is at least one input string that is not accepted by } M. \end{array} \right\}$$

Theorem 10. \overline{ALL} is not Turing-recognizable.

Proof. Assume toward a contradiction that \overline{ALL} is Turing-recognizable and let $M_{\overline{ALL}}$ be a DTM with $L(M_{\overline{ALL}}) = \overline{ALL}$. Just as in the proof of Theorem 9, we will use the existence of $M_{\overline{ALL}}$ to show that L_d is Turing-recognizable, which we already know is not the case.

Toward that end, let M be the DTM with input alphabet $\Sigma = \{0, 1\}$ described in Figure 10. What does M do? There are two cases:

1. $w \in L_d$. In other words, M_w does not accept w . Because of this, it holds that P_w can never accept any string x , so $L(P_w) = \emptyset$. Clearly then, $\langle P_w \rangle \notin ALL$, so $M_{\overline{ALL}}$ must accept $\langle P_w \rangle$, and therefore M accepts w .

2. $w \notin L_d$. In other words, M_w accepts w . In this case, P_w accepts every string x . Thus, $\langle P_w \rangle \in ALL$, so M_{ALL} does not accept $\langle P_w \rangle$. Consequently, M does not accept w .

In particular, $L(M) = L_d$, implying that L_d is Turing-recognizable—a contradiction. Therefore, \overline{ALL} is not Turing-recognizable. \square