Lecture 17

More undecidable languages; reductions

In the previous lecture we saw a few examples of undecidable languages: DIAG is not semidecidable, and therefore not decidable, while A_{DTM} and HALT are semidecidable but not decidable. In this lecture we will see a few more examples. We will also introduce the notion of a *reduction* from one language to another, which can be utilized when proving languages are undecidable.

Before doing this, however, we will take a few moments to observe a couple of basic but useful tricks involving Turing machines.

17.1 A couple of basic Turing machine tricks

This section describes two ideas that can be helpful when proving certain languages are undecidable (or non-semidecidable), and in other situations as well. The first involves a simple way to manage a search over an infinite domain and the second is concerned with the surprisingly powerful technique of hard coding inputs of Turing machines.

Limiting infinite search spaces

Sometimes we would like a Turing machine to effectively search over an infinitely large search space, but it may not always be immediately clear how this can be done. One way to address this issue is to set up a loop in which a single positive integer serves simultaneously as a bound on multiple parameters in the search space.

The proof of the following theorem, which is very important in its own right, illustrates this idea.

The DTM M operates as follows on input $x \in \{0,1\}^*$:

1. Set $t \leftarrow 1$.
2. Run M_0 on input w for t steps. If M_0 has accepted, then *accept*.
3. Run M_1 on input w for t steps. If M_1 has accepted, then *reject*.
4. Set $t \leftarrow t + 1$ and goto step 2.

Figure 17.1: A high-level description of a DTM M that decides A, assuming that M_0 and M_1 are DTMs satisfying $\mathrm{L}(M_0) = A$ and $\mathrm{L}(M_1) = \overline{A}$.

Theorem 17.1. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language such that both A and $\overline{A}$ are semidecidable. The language A is decidable.*

Proof. Because A and $\overline{A}$ are semidecidable languages, there must exist DTMs M_0 and M_1 such that $A = \mathrm{L}(M_0)$ and $\overline{A} = \mathrm{L}(M_1)$. Define a new DTM M as described in Figure 17.1.

Now let us consider the behavior of the DTM M on a given input string w. If it is the case that $w \in A$, then M_0 eventually accepts w, while M_1 does not. (It could be that M_1 either rejects or runs forever, but it cannot accept w.) It is therefore the case that M accepts w. On the other hand, if $w \notin A$, then M_1 eventually accepts w while M_0 does not, and therefore M rejects w. Consequently, M decides A, so A is decidable. □

To be clear, the technique being suggested involves the use of the variable t in the description of the DTM M in Figure 17.1; it is a single variable, but it is used to limit the simulations of both M_0 and M_1 (in steps 2 and 3) so that neither runs forever. We will see further examples of this technique later in this lecture and in the next lecture.

Hard-coding input strings

Suppose that we have a DTM M along with a string x over the input alphabet of M. Consider the new DTM M_x described in Figure 17.2.

This may seem like a rather useless Turing machine: M_x always leads to the same outcome regardless of what input string it is given. In essence, the input string x has been "hard-coded" directly into the description of M_x. We will see, however, that it is sometimes useful to consider a DTM defined in this way, particularly when proving that certain languages are undecidable or non-semidecidable.

The DTM M operates as follows on input $w \in \{0,1\}^*$:

1. Discard w and run M on input x.

Figure 17.2: The DTM M_x erases its input and runs M on the string x.

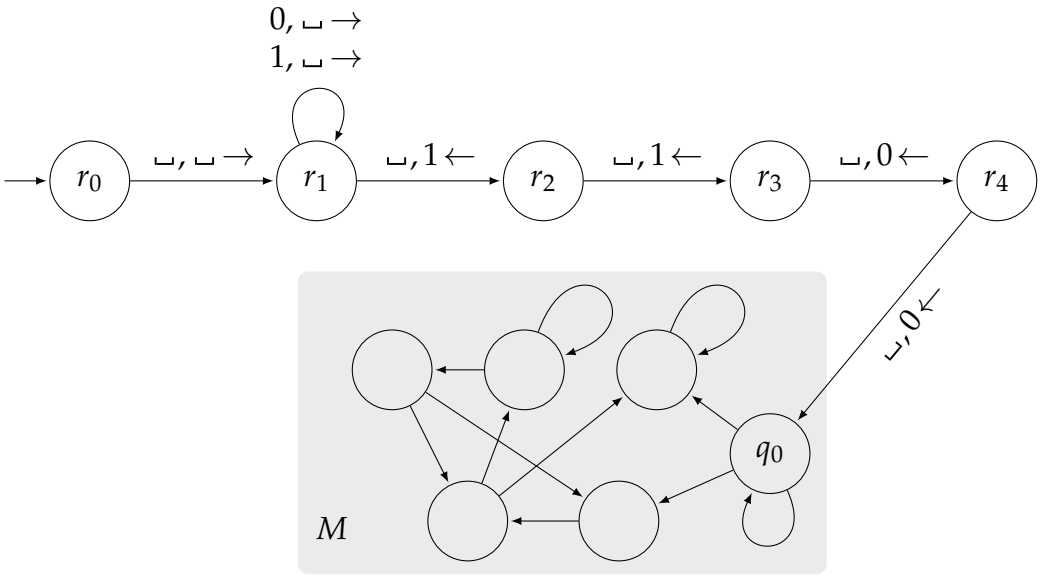

Figure 17.3: A state diagram for the DTM M_{0011}. The contents of the gray box is intended to represent the state diagram of an arbitrary DTM M having start state q_0.

Let us also note that if we have an encoding $\langle\langle M\rangle, \langle x\rangle\rangle$ of both a DTM M and a string x over the input alphabet of M, it is not difficult to compute an encoding $\langle M_x\rangle$ of the DTM M_x. The DTM M_x can be described as having three phases:

1. Move the tape head to the right across whatever input string w has been given, replacing the symbols of w with blanks, until all of w has been erased.
2. Starting at the end of the string x, write each symbol of x on the tape and move the tape head left.
3. Once the string x has been written to the tape, pass control to M.

The first phase can easily be done with a couple of states, and the second phase can be done using one state of M_x for each symbol of x. The third phase operates exactly like M. Figure 17.1 illustrates what the state diagram of the DTM M_{0011} looks like.

The DTM K operates as follows on input $w \in \{0,1\}^*$:

1. Reject unless $w = \langle\langle M \rangle, \langle x \rangle\rangle$ for M being a DTM and x being an input string to M.
2. Compute an encoding $\langle M_x \rangle$ of the DTM M_x defined from M as described in Figure 17.1.
3. Run T on input $\langle M_x \rangle$: if T accepts $\langle M_x \rangle$, then *reject*, otherwise *accept*.

Figure 17.4: The DTM K used in the proof of Proposition 17.2.

Here is an example that illustrates the usefulness of this construction. Define a language

$$\mathrm{E}_{\mathrm{DTM}} = \{\langle M \rangle \ : \ M \text{ is a DTM with } \mathrm{L}(M) = \varnothing\}. \tag{17.1}$$

Proposition 17.2. *The language* $\mathrm{E}_{\mathrm{DTM}}$ *is undecidable.*

Proof. Assume toward contradiction that $\mathrm{E}_{\mathrm{DTM}}$ is decidable, so that there exists a DTM T that decides this language. Define a new DTM K as in Figure 17.4.

Now, suppose that M is a DTM and $x \in \mathrm{L}(M)$, and consider the behavior of K on input $\langle\langle M \rangle, \langle x \rangle\rangle$. Because M accepts x, it is the case that M_x accepts *every* string over its alphabet—because whatever string you give it as input, it erases this string and runs M on x, leading to acceptance. It is therefore certainly not the case that $\mathrm{L}(M_x) = \varnothing$, so T must reject $\langle M_x \rangle$, and therefore K accepts $\langle\langle M \rangle, \langle w \rangle\rangle$.

On the other hand, if M is a DTM and $x \notin \mathrm{L}(M)$ then K will reject the input $\langle\langle M \rangle, \langle w \rangle\rangle$. Either x is not a string over the alphabet of M, which immediately leads to rejection, or M either rejects or runs forever on input x. In this second case, M_x either rejects or runs forever on every string, and therefore $\mathrm{L}(M_w) = \varnothing$. The DTM T therefore accepts $\langle M_x \rangle$, causing K to reject the input $\langle\langle M \rangle, \langle w \rangle\rangle$.

Thus, K decides $\mathrm{A}_{\mathrm{DTM}}$, which contradicts the fact that this language is undecidable. We conclude that $\mathrm{E}_{\mathrm{DTM}}$ is undecidable, as required. □

17.2 Proving undecidability through reductions

The proofs that we have seen so far that establish certain languages to be undecidable or non-semidecidable have followed a general pattern that can often be used to prove that a chosen language A is undecidable:

1. Assume toward contradiction that A is decidable.

2. Use that assumption to construct a DTM that decides a language B that we already know to be undecidable.
3. Having obtained a contradiction from the assumption that A is decidable, we conclude that A is undecidable.

A similar approach can sometimes be used to prove that a language A is non-semidecidable, and in both cases we might potentially obtain a contradiction by using our assumption toward contradiction about A to semidecide a language B that we already know to be non-semidecidable.

A different method through which languages may be proved to be undecidable or non-semidecidable makes use of the notion of a *reduction*.

Reductions

The notion of a reduction is, in fact, very general, and many different types of reductions are considered in theoretical computer science—but for now we will consider just one type of reduction (sometimes called a *mapping reduction* or *many-to-one reduction*), which is defined as follows.

Definition 17.3. Let Σ and Γ be alphabets and let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be languages. It is said that A *reduces* to B if there exists a computable function $f : \Sigma^* \to \Gamma^*$ such that

$$w \in A \Leftrightarrow f(w) \in B \tag{17.2}$$

for all $w \in \Sigma^*$. One writes $A \leq_m B$ to indicate that A reduces to B, and any function f that establishes that this is so may be called a *reduction* from A to B.

Figure 17.5 illustrates the action of a reduction. Intuitively speaking, a reduction is a way of transforming one computational decision problem into another. Imagine that you receive an input string $w \in \Sigma^*$, and you wish to determine whether or not w is contained in some language A. Perhaps you do not know how to make this determination, but you happen to have a friend who is able to tell you whether or not a particular string $y \in \Gamma^*$ is contained in a different language B. If you have a reduction f from A to B, then you can determine whether or not $w \in A$ using your friend's help: you compute $y = f(w)$, ask your friend whether or not $y \in B$, and take their answer as your answer to whether or not $w \in A$.

The following theorem has a simple and direct proof, but it will nevertheless have central importance with respect to the way that we use reductions to reason about decidability and semidecidability.

Theorem 17.4. *Let Σ and Γ be alphabets, let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be languages, and assume $A \leq_m B$. The following two implications hold:*

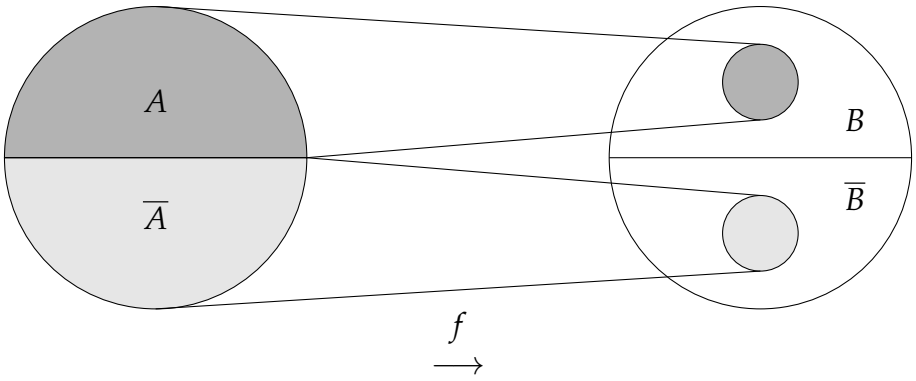

Figure 17.5: An illustration of a reduction f from A to B.

The DTM M_A operates as follows on input $w \in \Sigma^*$:

1. Compute $y = f(w)$.
2. Run M_B on input y.

Figure 17.6: Given a reduction f from A to B, and assuming the existence of a DTM M_B that either decides or semidecides B, the DTM M_A described either decides or semidecides A.

1. *If B is decidable, then A is decidable.*
2. *If B is semidecidable, then A is semidecidable.*

Proof. Let $f : \Sigma^* \to \Gamma^*$ be a reduction from A to B. We know that such a function exists by the assumption $A \leq_m B$.

We will first prove the second implication. Because B is semidecidable, there must exist a DTM M_B such that $B = \mathrm{L}(M_B)$. Define a new DTM M_A as described in Figure 17.6. It is possible to define a DTM in this way because f is a computable function.

For a given input string $w \in A$, we have that $y = f(w) \in B$, because this property is guaranteed by the reduction f. When M_A is run on input w, it will therefore accept because M_B accepts y. Along similar lines, if it is the case that $w \notin A$, then $y = f(w) \notin B$. When M_A is run on input w, it will therefore not accept because M_B does not accepts y. (It may be that these machines reject or run forever, but we do not care which.) It has been established that $A = \mathrm{L}(M_A)$, and therefore A is semidecidable.

The proof for the first implication is almost identical, except that we take M_B to be a DTM that decides B. The DTM M_A defined in Figure 17.6 then decides A, and therefore A is decidable. □

We will soon use this theorem to prove that certain languages are undecidable (or non-semidecidable), but let us first take a moment to observe two useful facts about reductions.

Proposition 17.5. *Let Σ, Γ, and Δ be alphabets and let $A \subseteq \Sigma^*$, $B \subseteq \Gamma^*$, and $C \subseteq \Delta^*$ be languages. If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$. (In words, $\leq_m$ is a transitive relation among languages.)*

Proof. As $A \leq_m B$ and $B \leq_m C$, there must exist computable functions $f : \Sigma^* \to \Gamma^*$ and $g : \Gamma^* \to \Delta^*$ such that

$$w \in A \Leftrightarrow f(w) \in B \quad \text{and} \quad y \in B \Leftrightarrow g(y) \in C \tag{17.3}$$

for all $w \in \Sigma^*$ and $y \in \Gamma^*$.

Define a function $h : \Sigma^* \to \Delta^*$ as $h(w) = g(f(w))$ for all $w \in \Sigma^*$. It is evident that h is a computable function: if we have DTMs M_f and M_g that compute f and g, respectively, then we can obtain a DTM M_h that computes h by first running M_f and then running M_g.

It remains to observe that h is a reduction from A to C. If $w \in A$, then $f(w) \in B$, and therefore $h(w) = g(f(w)) \in C$; and if $w \notin A$, then $f(w) \notin B$, and therefore $h(w) = g(f(w)) \notin C$. □

Proposition 17.6. *Let Σ and Γ be alphabets and let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be languages. It is the case that $A \leq_m B$ if and only if $\overline{A} \leq_m \overline{B}$.*

Proof. For a given function $f : \Sigma^* \to \Gamma^*$ and a string $w \in \Sigma^*$, the statements $w \in A \Leftrightarrow f(w) \in B$ and $w \in \overline{A} \Leftrightarrow f(w) \in \overline{B}$ are logically equivalent. If we have a reduction f from A to B, then the same function also serves as a reduction from $\overline{A}$ to $\overline{B}$, and vice versa. □

Undecidability through reductions

It is possible to use Theorem 17.4 to prove that certain languages are either decidable or semidecidable, but we will focus mainly on using it to prove that languages are either undecidable or non-semidecidable. When using the theorem in this way, we consider the two implications in the contrapositive form. That is, if two languages $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ satisfy $A \leq_m B$, then the following two implications hold:

The DTM K_M operates as follows on input $w \in \Sigma^*$:

1. Run M on input w.
   1.1 If M accepts w then *accept*.
   1.2 If M rejects w, then *run forever*.

Figure 17.7: Given a DTM M, we can easily obtain a DTM K_M that behaves as described by replacing any transitions to the accept state of M with transitions to a state that intentionally causes an infinite loop.

1. If A is undecidable, then B is undecidable.
2. If A is non-semidecidable, then B is non-semidecidable.

So, if we want to prove that a particular language B is undecidable, then it suffices to pick any language A that we already know to be undecidable, and then prove $A \leq_m B$. The situation is similar for proving languages to be non-semidecidable. The examples that follow illustrate how this may be done.

Example 17.7. For our first example of a reduction, we shall prove

$$A_{\text{DTM}} \leq_m \text{HALT}. \tag{17.4}$$

The first thing we will need to consider is a simple way of modifying an arbitrary DTM M to obtain a slightly different one. In particular, for an arbitrary DTM M, let us define a new DTM K_M as described in Figure 17.7. The idea behind the DTM K_M is very simple: if M accepts a string w, then so does K_M, if M rejects w then K_M runs forever on w, and of course if M runs forever on input w then so does K_M. Thus, K_M halts on input w if and only if M accepts w. Note that if you are given a description of a DTM M, it is very easy to come up with a description of a DTM K_M that operates as suggested: just replace the reject state of M with a new state that purposely causes an infinite loop (by repeatedly moving the tape head right, say).

Now let us define a function $f : \{0,1\}^* \to \{0,1\}^*$ as follows:

$$f(x) = \begin{cases} \langle\langle K_M \rangle, \langle w \rangle\rangle & \text{if } x = \langle\langle M \rangle, \langle w \rangle\rangle \text{ for a DTM } M \text{ and a string } w \\ & \text{over the alphabet of } M \\ x_0 & \text{otherwise,} \end{cases} \tag{17.5}$$

where $x_0 \in \{0,1\}^*$ is any fixed string that is not contained in HALT. (For example, we could take $x_0 = \varepsilon$, because ε does not encode a DTM together with an input string—but it is not important which string we choose as x_0, so long as it is not in HALT.) The function f is computable: all it does is that it essentially looks at an input string, determines whether or not this string is an encoding $\langle\langle M\rangle, \langle w\rangle\rangle$ of a DTM M and a string w over the alphabet of M, and if so it replaces the encoding of M with the encoding of the DTM K_M suggested above.

Now let us check to see that f is a reduction from $\mathrm{A_{DTM}}$ to HALT. Suppose first that we have an input $\langle\langle M\rangle, \langle w\rangle\rangle \in \mathrm{A_{DTM}}$. These implications hold:

$$\begin{aligned} \langle\langle M\rangle, \langle w\rangle\rangle \in \mathrm{A_{DTM}} \;&\Rightarrow\; M \text{ accepts } w \;\Rightarrow\; K_M \text{ halts on } w \\ &\Rightarrow\; \langle\langle K_M\rangle, \langle w\rangle\rangle \in \mathrm{HALT} \;\Rightarrow\; f(\langle\langle M\rangle, \langle w\rangle\rangle) \in \mathrm{HALT}. \end{aligned} \tag{17.6}$$

We therefore have

$$\langle\langle M\rangle, \langle w\rangle\rangle \in \mathrm{A_{DTM}} \;\Rightarrow\; f(\langle\langle M\rangle, \langle w\rangle\rangle) \in \mathrm{HALT}, \tag{17.7}$$

which is half of what we need to verify that f is indeed a reduction from $\mathrm{A_{DTM}}$ to HALT.

It remains to consider the output of the function f on inputs that are not contained in $\mathrm{A_{DTM}}$, and here there are two cases: one is that the input takes the form $\langle\langle M\rangle, \langle w\rangle\rangle$ for a DTM M and a string w over the alphabet of M, and the other is that it does not. For the first case, we have these implications:

$$\begin{aligned} \langle\langle M\rangle, \langle w\rangle\rangle \notin \mathrm{A_{DTM}} \;&\Rightarrow\; M \text{ does not accept } w \\ &\Rightarrow\; K_M \text{ runs forever on } w \;\Rightarrow\; \langle\langle K_M\rangle, \langle w\rangle\rangle \notin \mathrm{HALT} \\ &\qquad\qquad \Rightarrow\; f(\langle\langle M\rangle, \langle w\rangle\rangle) \notin \mathrm{HALT}. \end{aligned} \tag{17.8}$$

The key here is that K_M is defined so that it will definitely run forever in case M does not accept (regardless of whether that happens by M rejecting or running forever). The remaining case is that we have a string $x \in \Sigma^*$ that does not take the form $\langle\langle M\rangle, \langle w\rangle\rangle$ for a DTM M and a string w over the alphabet of M, and in this case it trivially holds that $f(x) = x_0 \notin \mathrm{HALT}$. (This is why we defined f as we did in this case, and we will generally do something similar for other examples).

We have therefore proved that

$$x \in \mathrm{A_{DTM}} \Leftrightarrow f(x) \in \mathrm{HALT}, \tag{17.9}$$

and therefore $\mathrm{A_{DTM}} \leq_m \mathrm{HALT}$.

We already proved that HALT is undecidable, but the fact that $\mathrm{A_{DTM}} \leq_m \mathrm{HALT}$ provides an alternative proof: because we already know that $\mathrm{A_{DTM}}$ is undecidable, it follows that HALT is also undecidable.

It might not seem that there is any advantage to this proof over the proof we saw in the previous lecture that HALT is undecidable (which was not particularly difficult). We have, however, established a closer relationship between $\mathrm{A}_{\mathrm{DTM}}$ and HALT than we did previously. In general, using a reduction is sometimes an easy shortcut to proving that a language is undecidable (or non-semidecidable).

Example 17.8. For our next example of a reduction, we will prove

$$\mathrm{DIAG} \leq_m \mathrm{E}_{\mathrm{DTM}}, \tag{17.10}$$

where we recall that $\mathrm{E}_{\mathrm{DTM}}$ is defined as follows:

$$\mathrm{E}_{\mathrm{DTM}} = \big\{\langle M\rangle \,:\, M \text{ is a DTM and } \mathrm{L}(M) = \varnothing\big\}. \tag{17.11}$$

We will now prove that $\mathrm{DIAG} \leq_m \mathrm{E}_{\mathrm{DTM}}$. Because we already know that DIAG is non-semidecidable, we conclude from this reduction that $\mathrm{E}_{\mathrm{DTM}}$ is not just undecidable, but in fact it is also non-semidecidable.

For this one we will again use the hardcoding trick from the beginning of the lecture: for a given DTM M, let us define a new DTM $M_{\langle M\rangle}$ just like in Figure 17.2, for the specific choice of the hardcoded string $x = \langle M\rangle$. This actually only makes sense if the input alphabet of M includes the symbols $\{0,1\}$ used in the encoding $\langle M\rangle$, so let us agree that $M_{\langle M\rangle}$ immediately rejects if this is not the case.

Now let us define a function $f : \{0,1\}^* \to \{0,1\}^*$ as follows:

$$f(x) = \begin{cases} \big\langle M_{\langle M\rangle}\big\rangle & \text{if } x = \langle M\rangle \text{ for a DTM } M \\ x_0 & \text{otherwise,} \end{cases} \tag{17.12}$$

for any fixed binary string x_0 not contained in $\mathrm{E}_{\mathrm{DTM}}$. If you think about it for a moment, it should not be hard to convince yourself that f is computable. It remains to verify that f is a reduction from DIAG to $\mathrm{E}_{\mathrm{DTM}}$.

For any string $x \in \mathrm{DIAG}$ we have that $x = \langle M\rangle$ for some DTM M that satisfies $\langle M\rangle \notin \mathrm{L}(M)$. In this case we have that $f(x) = \big\langle M_{\langle M\rangle}\big\rangle$, and because $\langle M\rangle \notin \mathrm{L}(M)$ it must therefore be that $M_{\langle M\rangle}$ never accepts, and so $f(x) = \big\langle M_{\langle M\rangle}\big\rangle \in \mathrm{E}_{\mathrm{DTM}}$.

Now suppose that $x \notin \mathrm{DIAG}$. There are two cases: either $x = \langle M\rangle$ for a DTM M such that $\langle M\rangle \in \mathrm{L}(M)$, or x does not encode a DTM at all. If it is the case that $x = \langle M\rangle$ for a DTM M such that $\langle M\rangle \in \mathrm{L}(M)$, we have that $M_{\langle M\rangle}$ accepts *every* string over its alphabet, and therefore $f(x) = \big\langle M_{\langle M\rangle}\big\rangle \notin \mathrm{E}_{\mathrm{DTM}}$. If it is the case that x does not encode a DTM, then it trivially holds that $f(x) = x_0 \notin \mathrm{E}_{\mathrm{DTM}}$.

We have proved that

$$x \in \mathrm{DIAG} \Leftrightarrow f(x) \in \mathrm{E}_{\mathrm{DTM}}, \tag{17.13}$$

so the proof that $\mathrm{DIAG} \leq_m \mathrm{E}_{\mathrm{DTM}}$ is complete.

Example 17.9. Our third example of a reduction is

$$\mathrm{A}_{\mathrm{DTM}} \leq_m \mathrm{AE}, \tag{17.14}$$

where the language AE is defined like this:

$$\mathrm{AE} = \big\{ \langle M \rangle \,:\, M \text{ is a DTM that accepts } \varepsilon \big\}. \tag{17.15}$$

The name AE stands for "accepts the empty string."

To prove this reduction, we can use the same hardcoding trick that we have now used twice already. For every DTM M and every string x over the alphabet of M, define a new DTM M_x as in Figure 17.2, and define a function $f : \{0,1\}^* \to \{0,1\}^*$ as follows:

$$f(w) = \begin{cases} \langle M_x \rangle & \text{if } w = \langle \langle M \rangle, \langle x \rangle \rangle \text{ for a DTM } M \text{ and a string } x \\ & \text{over the alphabet of } M \\ w_0 & \text{otherwise,} \end{cases} \tag{17.16}$$

where, as you likely now expect, w_0 is any fixed binary string not contained in AE. Now let us check that f is a valid reduction from $\mathrm{A}_{\mathrm{DTM}}$ to AE.

First, for any string $w \in \mathrm{A}_{\mathrm{DTM}}$ we have $w = \langle \langle M \rangle, \langle x \rangle \rangle$ for a DTM M that accepts the string x. In this case, $f(w) = \langle M_x \rangle$. We have that M_x accepts every string, including the empty string, because M accepts x. Therefore $f(w) = \langle M_x \rangle \in \mathrm{AE}$.

Now consider any string $w \notin \mathrm{A}_{\mathrm{DTM}}$, for which there are two cases. If it is the case that $w = \langle \langle M \rangle, \langle x \rangle \rangle$ for a DTM M and x a string over the alphabet of M, then $w \notin \mathrm{A}_{\mathrm{DTM}}$ implies that M does not accept x. In this case we have $f(w) = \langle M_x \rangle \notin \mathrm{AE}$, because M_x does not accept any strings at all (including the empty string). If $w \neq \langle \langle M \rangle, \langle x \rangle \rangle$ for a DTM M and string x over the alphabet of M, then $f(w) = w_0 \notin \mathrm{AE}$.

We have shown that $w \in \mathrm{A}_{\mathrm{DTM}} \Leftrightarrow f(w) \in \mathrm{AE}$ for every string $w \in \{0,1\}^*$, and therefore $\mathrm{A}_{\mathrm{DTM}} \leq_m \mathrm{AE}$, as required.

Example 17.10. The last example of a reduction for the lecture will be a bit more difficult than the others. We will prove that

$$\mathrm{E}_{\mathrm{DTM}} \leq_m \mathrm{DEC} \tag{17.17}$$

where

$$\mathrm{DEC} = \big\{ \langle M \rangle \,:\, M \text{ is a DTM such that } \mathrm{L}(M) \text{ is decidable} \big\}. \tag{17.18}$$

The DTM K_M operates as follows on input $x \in \{0,1\}^*$:

1. Set $t \leftarrow 1$.
2. For every string w over the input alphabet of M satisfying $|w| \leq t$:
   2.1 Run M for t steps on input w, and if M accepts then goto step 4.
3. Set $t \leftarrow t+1$ and goto step 2.
4. Run H on x.

Figure 17.8: The DTM K_M in Example 17.10. We assume that H is any fixed DTM with $\mathrm{L}(H) = \mathrm{HALT}$.

Notice that the inclusion $\langle M \rangle \in \mathrm{DEC}$ does not imply that M always halts—but rather that there exists some DTM K, not necessarily M, that decides the language recognized by M.

Given an arbitrary DTM M, let us define a new DTM K_M as in Figure 17.8. In this description, assume H is any fixed DTM satisfying $\mathrm{L}(H) = \mathrm{HALT}$. We know there is such an H; we can easily adapt a universal DTM U so that it semidecides HALT. If one asks why we define K_M in this way, the answer is nothing more than that it makes the reduction work—but notice that within the definition of K_M we are making use of the infinite search technique from the start of the lecture.

Now let us define a function $f : \{0,1\}^* \to \{0,1\}^*$ as

$$f(x) = \begin{cases} \langle K_M \rangle & \text{if } x = \langle M \rangle \text{ for a DTM } M \\ x_0 & \text{otherwise,} \end{cases} \tag{17.19}$$

where x_0 is any fixed binary string not contained in DEC. This is a computable function, and it remains to verify that it is a reduction from $\mathrm{E}_{\mathrm{DTM}}$ to DEC.

Suppose $\langle M \rangle \in \mathrm{E}_{\mathrm{DTM}}$. We therefore have that $\mathrm{L}(M) = \varnothing$; and by considering the way that K_M behaves we see that $\mathrm{L}(K_M) = \varnothing$ as well; the computation alternates between steps 2 and 3 forever if M never accepts. The empty language is decidable, and therefore $f(\langle M \rangle) = \langle K_M \rangle \in \mathrm{DEC}$.

On the other hand, if M is a DTM and $\langle M \rangle \notin \mathrm{E}_{\mathrm{DTM}}$, then M must accept at least one string. This means that $\mathrm{L}(K_M) = \mathrm{HALT}$, because K_M will eventually find a string accepted by M, reach step 4, and then accept if and only if $x \in \mathrm{HALT}$. Therefore $f(\langle M \rangle) = \langle K_M \rangle \notin \mathrm{DEC}$. The remaining case in which x does not encode a DTM is, as always, straightforward: in this case we have $f(x) = x_0 \notin \mathrm{DEC}$.

We have shown that $x \in \mathrm{E}_{\mathrm{DTM}} \Leftrightarrow f(x) \in \mathrm{DEC}$ for every string $x \in \{0,1\}^*$, and therefore $\mathrm{E}_{\mathrm{DTM}} \leq_m \mathrm{DEC}$, as required.

We conclude that the language DEC is non-semidecidable, as we already know that $\mathrm{E}_{\mathrm{DTM}}$ is non-semidecidable.