Lecture 16

Computable functions and mapping reductions

In this lecture we will discuss the notion of a *reduction*, which is useful for proving that certain languages are undecidable (or that they are non-Turing-recognizable).

16.1 Computable functions and reductions

Before discussing reductions, we must define what it means for a function to be *computable*.

Definition 16.1. Let Σ be an alphabet and let $f : \Sigma^* \to \Sigma^*$ be a function. It is said that f is *computable* if there exists a DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ whose yields relation satisfies

$$(q_0, \sqcup)\, w \vdash_M^* (q_{\text{acc}}, \sqcup)\, f(w) \tag{16.1}$$

for every string $w \in \Sigma^*$.

In words, a function $f : \Sigma^* \to \Sigma^*$ is computable if there exists a DTM M such that, if we run M on input w, it will eventually accept, with just the output $f(w)$ written on the tape (and the tape head scanning the square to the left of this output string). It is not really important that this DTM accepts, as opposed to rejecting—the fact that it always halts with the correct output of the function written on the tape is what is important.

Now that we have introduced the definition of computable functions, we can move on to reductions. There are, in fact, many different types of reductions—the specific type of reduction we will consider is sometimes called a *mapping reduction*—but because this is the only type of reduction we will be concerned with, we'll stick with the simpler term *reduction*.

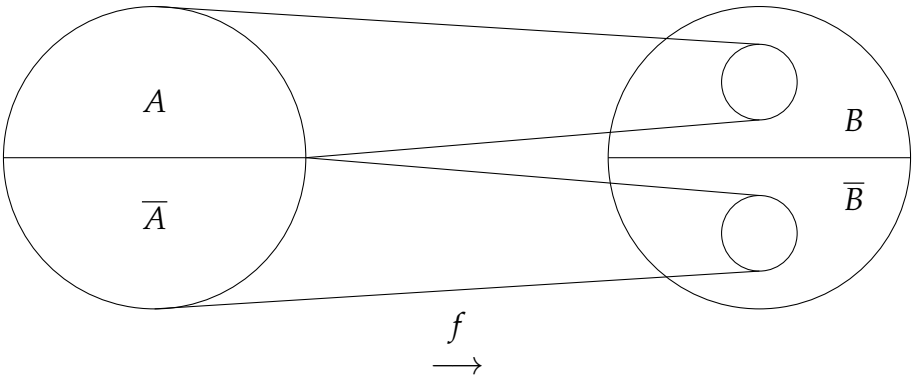

Figure 16.1: An illustration of a reduction f from A to B.

Definition 16.2. Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be languages. It is said that A *reduces* to B if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that

$$w \in A \Leftrightarrow f(w) \in B \tag{16.2}$$

for all $w \in \Sigma^*$. One writes

$$A \leq_m B \tag{16.3}$$

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 16.1 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 $y \in \Sigma^*$ 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 Turing recognizability.

Theorem 16.3. *Let Σ be an alphabet, let $A, B \subseteq \Sigma^*$ be languages, and assume $A \leq_m B$. The following two implications hold:*

1. *If B is decidable, then A is decidable.*
2. *If B is Turing recognizable, then A is Turing recognizable.*

Proof. Let $f : \Sigma^* \to \Sigma^*$ 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 Turing recognizable, there must exist a DTM M_B such that $B = \mathrm{L}(M_B)$. Define a new DTM M_A as follows:

On input $w \in \Sigma^*$:

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

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 accept y. Along similar lines, if $w \notin A$, 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 Turing recognizable.

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 above then decides A, and therefore A is decidable. □

We will soon find some applications of the previous theorem, but let us first observe two simple but nevertheless useful facts about reductions.

Proposition 16.4. *Let Σ be an alphabet and let $A, B, C \subseteq \Sigma^*$ be languages. If it holds that $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$. (In other 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 $f : \Sigma^* \to \Sigma^*$ and $g : \Sigma^* \to \Sigma^*$ such that

$$w \in A \Leftrightarrow f(w) \in B \quad \text{and} \quad w \in B \Leftrightarrow g(w) \in C \tag{16.4}$$

for all $w \in \Sigma^*$.

Define a function $h : \Sigma^* \to \Sigma^*$ 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 easily 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 16.5. *Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be languages. It holds that $A \leq_m B$ if and only if $\overline{A} \leq_m \overline{B}$.*

Proof. This proposition is almost trivial—for a given function $f : \Sigma^* \to \Sigma^*$ and a string $w \in \Sigma^*$, the statements

$$w \in A \Leftrightarrow f(w) \in B \qquad \text{and} \qquad w \in \overline{A} \Leftrightarrow f(w) \in \overline{B} \tag{16.5}$$

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. □

16.2 Proving undecidability through reductions

It is possible to use Theorem 16.3 to prove that certain languages are either decidable or Turing recognizable, but we will focus mainly on using it to prove that languages are either *not* decidable or *not* Turing recognizable. When using the theorem in this way, we consider the two implications in the contrapositive form. That is, if two languages $A, B \subseteq \Sigma^*$ satisfy $A \leq_m B$, then the following two implications hold:

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

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-Turing-recognizable. The examples that follow illustrate how this is done.

Example 16.6 ($\mathrm{A_{DTM}} \leq_m$ HALT)**.** Recall the following two languages from the previous lecture:

$$\mathrm{A_{DTM}} = \bigl\{ \langle M, w \rangle \;:\; M \text{ is a DTM and } w \in \mathrm{L}(M) \bigr\}, \tag{16.6}$$

$$\mathrm{HALT} = \bigl\{ \langle M, w \rangle \;:\; M \text{ is a DTM that halts on input } w \bigr\}. \tag{16.7}$$

We will prove that the first one reduces to the second, meaning $\mathrm{A_{DTM}} \leq_m \mathrm{HALT}$. (As this is our first example, the discussion will include more detail than we would typically include when proving reductions—when you prove a reduction on an assignment or exam, you would not be expected to provide this much detail.)

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 follows:

On input w:
1. Run M on input w.
2. If M accepts w then *accept*.
3. If M rejects w, then *run forever*.

Of course, if it is the case that M runs forever on some input string w, then K_M also runs forever on w—the description above is meant to suggest that K_M makes a decision to either accept or purposely run forever in the event that M halts.

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 above: just replace the reject state of M with a new state that purposely causes an infinite loop (by repeatedly moving the tape head to the right, for instance).

Now let us define a function $f : \Sigma^* \to \Sigma^*$, where Σ is the alphabet over which $\mathrm{A_{DTM}}$ and HALT are defined, as follows:

$$f(x) = \begin{cases} \langle K_M, w \rangle & \text{if } x = \langle M, w \rangle \text{ for a DTM } M \text{ and a string } w \\ x & \text{otherwise.} \end{cases} \tag{16.8}$$

The most important property of this function is that

$$f\big(\langle M, w \rangle\big) = \langle K_M, w \rangle \tag{16.9}$$

whenever M is a DTM and x is a string. The other part is not so interesting—but we should nevertheless indicate how this function is defined on an input that happens to not take the form $\langle M, w \rangle$ for a DTM M and a string w. We have opted to define $f(x) = x$ in this situation so that $f(x) \notin \mathrm{HALT}$.

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 $\langle M, w \rangle$ of a DTM M and a string w, and if so it replaces M with K_M as described above. Because it is straightforward to modify M to obtain K_M, there is no question that this modification could be performed by a DTM.

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

$$\begin{aligned} &\langle M, w \rangle \in \mathrm{A_{DTM}} \;\Rightarrow\; M \text{ accepts } w \;\Rightarrow\; K_M \text{ accepts } w \\ &\quad \Rightarrow\; K_M \text{ halts on } w \;\Rightarrow\; \langle K_M, w \rangle \in \mathrm{HALT} \;\Rightarrow\; f(\langle M, w \rangle) \in \mathrm{HALT}. \end{aligned} \tag{16.10}$$

We therefore have

$$\langle M, w \rangle \in \mathrm{A_{DTM}} \;\Rightarrow\; f(\langle M, w \rangle) \in \mathrm{HALT}, \tag{16.11}$$

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 $\langle M, w \rangle$ for a DTM M and a string w, and the other is that it does not. For the first case, we have these implications:

$$\begin{aligned} \langle M, w \rangle \notin \mathrm{A_{DTM}} \;\Rightarrow\; & M \text{ does not accept } w \;\Rightarrow\; K_M \text{ runs forever on } w \\ & \Rightarrow\; \langle K_M, w \rangle \notin \mathrm{HALT} \;\Rightarrow\; f(\langle M, w \rangle) \notin \mathrm{HALT}. \end{aligned} \tag{16.12}$$

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 $\langle M, w \rangle$ for a DTM M and a string w, and in this case it trivially holds that $f(x) = x \notin \mathrm{HALT}$. We have therefore proved that

$$x \in \mathrm{A_{DTM}} \Leftrightarrow f(x) \in \mathrm{HALT}, \tag{16.13}$$

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 wasn't particularly difficult). We have, however, established a closer relationship between $\mathrm{A_{DTM}}$ 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-Turing-recognizable).

Example 16.7 ($\mathrm{DIAG} \leq_m \mathrm{E_{DTM}}$)**.** Our first example of a non-Turing-recognizable language, from the previous lecture, was this language:

$$\mathrm{DIAG} = \bigl\{ \langle M \rangle \;:\; M \text{ is a DTM and } \langle M \rangle \notin \mathrm{L}(M) \bigr\}. \tag{16.14}$$

We also defined the following language, and proved it is undecidable:

$$\mathrm{E_{DTM}} = \bigl\{ \langle M \rangle \;:\; M \text{ is a DTM and } \mathrm{L}(M) = \varnothing \bigr\}. \tag{16.15}$$

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

Let us use a similar trick to one we used in the previous lecture: for a given DTM M, let us define a new DTM M_{self} as follows.

On input w:

Ignore the input string w and run M on input $\langle M \rangle$.

This description really only makes sense if the input alphabet of M includes the symbols in the encoding $\langle M \rangle$, so let us agree that M_{self} immediately rejects if this is not the case.

Now let us define a function $f : \Sigma^* \to \Sigma^*$, for Σ being the alphabet over which DIAG and E_{DTM} are defined, as follows:

$$f(x) = \begin{cases} \langle M_{\text{self}} \rangle & \text{if } x = \langle M \rangle \text{ for a DTM } M \\ x & \text{otherwise.} \end{cases} \tag{16.16}$$

If you think about it for a few moments, it should not be hard to convince yourself that f is computable.

Now let us verify that f is a reduction from DIAG to E_{DTM}. For any string $x \in \text{DIAG}$ we have that $x = \langle M \rangle$ for some DTM M that satisfies $\langle M \rangle \notin \text{L}(M)$. In this case we have that $f(x) = \langle M_{\text{self}} \rangle$, and because $\langle M \rangle \notin \text{L}(M)$ it must be that M_{self} never accepts. It is therefore the case that $f(x) = \langle M_{\text{self}} \rangle \in \text{E}_{\text{DTM}}$.

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

We have proved that

$$x \in \text{DIAG} \Leftrightarrow f(x) \in \text{E}_{\text{DTM}}, \tag{16.17}$$

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

Example 16.8 ($\text{A}_{\text{DTM}} \leq_m \text{AE}$)**.** Define a language

$$\text{AE} = \big\{ \langle M \rangle \,:\, M \text{ is a DTM that accepts } \varepsilon \big\}. \tag{16.18}$$

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

It is very easy to prove that $\text{AE} \leq_m \text{A}_{\text{DTM}}$. (The main point of the example is to prove the other reduction, $\text{A}_{\text{DTM}} \leq_m \text{AE}$, so this is just a warm-up.) Define a function $f : \Sigma^* \to \Sigma^*$ as follows:

$$f(w) = \begin{cases} \langle M, \varepsilon \rangle & \text{if } w = \langle M \rangle \text{ for some DTM } M \\ \langle M_0, \varepsilon \rangle & \text{otherwise,} \end{cases} \tag{16.19}$$

where M_0 is some fixed DTM that always rejects. This function is computable. Now let us check that f is a valid reduction from AE to A_{DTM}.

First, for any string $w \in \text{AE}$, we must have $w = \langle M \rangle$ for M being a DTM that accepts ε. In this case $f(w) = \langle M, \varepsilon \rangle$, which is contained in A_{DTM} (because M accepts ε).

Now consider any string $w \notin \text{AE}$. There are two cases: either $w = \langle M \rangle$ for some DTM M, or this is not the case. If $w = \langle M \rangle$ for a DTM M, then $w \notin \text{AE}$ implies that M does not accept ε. In this case we have $f(w) = \langle M, \varepsilon \rangle \notin \text{A}_{\text{DTM}}$ (because M does not accept ε). If $w \neq \langle M \rangle$ for a DTM M, then $f(w) = \langle M_0, \varepsilon \rangle \notin \text{A}_{\text{DTM}}$ (because M_0 does not accept any strings at all, including ε).

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

That was indeed easy, but now let's prove the reverse reduction: $\text{A}_{\text{DTM}} \leq_m \text{AE}$. We'll use a similar trick to the one from the previous example. For every DTM M and every string w over the alphabet of M, define a new DTM M_w as follows:

On input x:

Erase x and run M on input w.

Now define a function $f : \Sigma^* \to \Sigma^*$ as follows:

$$f(y) = \begin{cases} \langle M_w \rangle & \text{if } y = \langle M, w \rangle \text{ for some DTM } M \text{ and string } w \\ \langle M_0 \rangle & \text{otherwise.} \end{cases} \tag{16.20}$$

(Just like for the easier reduction above, M_0 is a DTM that always rejects.) Now let us check that f is a valid reduction from A_{DTM} to AE.

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

Now consider any string $y \notin \text{A}_{\text{DTM}}$. Again there are two cases: either $y = \langle M, w \rangle$ for some DTM M and input string w, or this is not the case. If $y = \langle M, w \rangle$ for a DTM M and a string w, then $y \notin \text{A}_{\text{DTM}}$ implies that M does not accept w. In this case we have $f(y) = \langle M_w \rangle \notin \text{AE}$, because M_w does not accept any strings at all (including the empty string). If $y \neq \langle M, w \rangle$ for a DTM M and string w, then $f(y) = \langle M_0 \rangle \notin \text{AE}$ (again because M_0 does not accept any strings, including ε).

We have shown that $y \in \text{A}_{\text{DTM}} \Leftrightarrow f(y) \in \text{AE}$ holds for every string $y \in \Sigma^*$, and therefore $\text{A}_{\text{DTM}} \leq_m \text{AE}$, as required.

Example 16.9 ($\text{E}_{\text{DTM}} \leq_m \text{REG}$)**.** Define a language as follows:

$$\text{REG} = \big\{ \langle M \rangle \ : \ M \text{ is a DTM such that } \text{L}(M) \text{ is regular} \big\}. \tag{16.21}$$

We will prove $\text{E}_{\text{DTM}} \leq_m \text{REG}$.

We'll need a strange way to modify DTMs in order to do this one. Given an arbitrary DTM M, let us define a new DTM K_M as follows:

On input $x \in \{0,1\}^*$:

1. Set $t \leftarrow 1$.
2. For every input string w over the input alphabet of M satisfying $|w| \leq t$:
3. Run M for t steps on input w.
4. If M accepts w within t steps, goto 6.
5. Set $t \leftarrow t + 1$ and goto 2.
6. *Accept* if $x \in \{0^n 1^n : n \in \mathbb{N}\}$, *reject* otherwise.

This is indeed a strange way to define a DTM, but it's alright if it's strange—we're just proving a reduction.

Now let us define a function $f : \Sigma^* \to \Sigma^*$ as

$$f(x) = \begin{cases} \langle K_M \rangle & \text{if } x = \langle M \rangle \text{ for a DTM } M \\ x & \text{otherwise.} \end{cases} \tag{16.22}$$

This is a computable function, and it remains to verify that it is a reduction from $\mathrm{E_{DTM}}$ to REG.

Suppose $\langle M \rangle \in \mathrm{E_{DTM}}$. 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 (because we never get to step 6 if M never accepts). The empty language is regular, and therefore $f(\langle M \rangle) = \langle K_M \rangle \in \mathrm{REG}$.

On the other hand, if M is a DTM and $\langle M \rangle \notin \mathrm{E_{DTM}}$, then M must accept at least one string. This means that $\mathrm{L}(K_M) = \{0^n 1^n : n \in \mathbb{N}\}$, because K_M will eventually find a string accepted by M, reach step 6, and then accept or reject based on whether the input string x is contained in $\{0^n 1^n : n \in \mathbb{N}\}$. Therefore $f(\langle M \rangle) = \langle K_M \rangle \notin \mathrm{REG}$. The remaining case, in which x does not encode a DTM, is straightforward as usual: we have $f(x) = x \notin \mathrm{REG}$ in this case.

We have shown that $x \in \mathrm{E_{DTM}} \Leftrightarrow f(x) \in \mathrm{REG}$ holds for every string $x \in \Sigma^*$, and therefore $\mathrm{E_{DTM}} \leq_m \mathrm{REG}$, as required. We conclude that REG is not Turing recognizable, as we already know that $\mathrm{E_{DTM}}$ is not Turing recognizable.