Lecture 17

Undecidable languages

This lecture focuses on techniques for proving that certain languages are undecidable (or, in some cases, non-semidecidable). The lecture will be divided into two main parts: the first part focuses on undecidability proofs by contradiction, and the second part discusses the notion of a reduction and how undecidability results may be established through them. Along the way we will discuss some useful tricks that can be applied in both settings.

Before proceeding to the first part of the lecture, let us recall that we have fixed an encoding scheme for DSMs, whereby any given DSM $M$ is encoded as a binary string $\langle M \rangle$. In the previous lecture we proved that the language

$$\text{DIAG} = \{ \langle M \rangle : M \text{ is a DSM and } \langle M \rangle \notin L(M) \}$$

(17.1)

is non-semidecidable, and we then used this fact to conclude that the language

$$\text{A}_{\text{DSM}} = \{ \langle \langle M \rangle, \langle w \rangle \rangle : M \text{ is a DSM and } w \in L(M) \}$$

(17.2)

is undecidable (although it is semidecidable). All of the undecidability proofs that appear in this lecture are, in some sense, anchored by the diagonalization proof that DIAG is not semidecidable.

17.1 Undecidability proofs through contradiction

In this section we will see a few more examples of undecidability proofs that have a similar style to the proof we saw in at the end of the previous lecture, through which we concluded that $\text{A}_{\text{DSM}}$ is undecidable. More specifically, we assumed toward contradiction that $\text{A}_{\text{DSM}}$ is decidable, and based on that assumption we constructed a DSM that decided a language (specifically, the language DIAG) that we already knew to be undecidable.
This is the same general pattern that will be used in this section when we wish to prove that a chosen language $A$ is undecidable:

1. Assume toward contradiction that $A$ is decidable.
2. Use that assumption to construct a DSM 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 (as opposed to deciding a language $B$ that we already know to be undecidable).

Here is an example. Define a language HALT as follows:

$$\text{HALT} = \{ \langle \langle M \rangle, \langle w \rangle \rangle : M \text{ is a DSM that halts on input } w \}.$$ (17.3)

To say that $M$ halts on input $w$ means that it stops, either by accepting or rejecting. Let us agree that the statement “$M$ halts on input $w$” is false in case $w$ contains symbols not in the input alphabet of $M$, just as a matter of terminology.

We will prove that HALT is undecidable, but before we do this let us observe that HALT is semidecidable (just like $A_{\text{DSM}}$). In particular, this language can be semi-decided by a modified version of the universal stack machine $U$ from the previous lecture; the modification is that it accepts both in the case that $M$ accepts $w$ and in the case that $M$ rejects $w$. Of course, when it is the case that $M$ runs forever on $w$, the same will be true of $U$ running on input $\langle \langle M \rangle, \langle w \rangle \rangle$.

**Proposition 17.1.** The language HALT is undecidable.

**Proof.** Assume toward contradiction that HALT is decidable, so that there exists a DSM $T$ that decides it. Define a new DSM $K$ as described in Figure 17.1.

We will conclude that $K$ decides $A_{\text{DSM}}$. Note first that if $K$ is given an input that is not of the form $\langle \langle M \rangle, \langle w \rangle \rangle$, for $M$ a DSM and $w$ a string over the input alphabet of $M$, then it rejects (as a DSM for $A_{\text{DSM}}$ should). Otherwise, when the input to $K$ does take the form $\langle \langle M \rangle, \langle w \rangle \rangle$, for $M$ a DSM and $w$ a string over the input alphabet of $M$, there are three possible cases:

1. If it is the case that $M$ accepts $w$, then $T$ will accept $\langle \langle M \rangle, \langle w \rangle \rangle$ (because $M$ halts on $w$), and the simulation of $M$ on input $w$ will result in acceptance.
2. If it is the case that $M$ rejects $w$, then $T$ will accept $\langle \langle M \rangle, \langle w \rangle \rangle$ (again because $M$ halts on $w$), and the simulation of $M$ on input $w$ will result in rejection.
The DSM $K$ operates as follows on input $x \in \{0, 1\}^*$:

1. If it is not the case that $x = \langle \langle M \rangle, \langle w \rangle \rangle$ for $M$ being a DSM and $w$ being a string over the alphabet of $M$, then reject.
2. Run $T$ on input $\langle \langle M \rangle, \langle w \rangle \rangle$ and reject if $T$ rejects. Otherwise, continue to the next step.
3. Simulate $M$ on input $w$; accept if $M$ accepts and reject if $M$ rejects.

Figure 17.1: A DSM $K$ that decides $A_{DSM}$, assuming that is a DSM that decides $HALT$.

3. If it is the case that $M$ runs forever on $w$, then $T$ will reject $\langle \langle M \rangle, \langle w \rangle \rangle$, and therefore $K$ rejects without running the simulation of $M$ on input $w$.

This, however, is in contradiction with the fact that $A_{DSM}$ is undecidable. Having obtained a contradiction, we conclude that $HALT$ is undecidable.

Next we will consider this language, which is a DSM variant of the languages $E_{DFA}$ and $E_{CFG}$ from Lecture 15:

$$E_{DSM} = \{ \langle M \rangle : M \text{ is a DSM with } L(M) = \emptyset \}. \quad (17.4)$$

We will prove that this language is undecidable, but in order to do this we need to make use of the very simple but remarkably useful trick of hard-coding inputs into DSMs.

Here is the idea. Suppose that we have a DSM $M$ along with a fixed string $w$ over the input alphabet of $M$. Consider a new DSM, which we will call $M_w$, that operates as described in Figure 17.2. (Figure 17.3 illustrates what a state diagram of $M_w$ might look like, assuming $w = a_1 \cdots a_n$.) This may seem like a curious way to define a DSM; the DSM $M_w$ runs the same way regardless of its actual input string $x$, as it always discards this string and runs $M$ on the string $w$, which is “hard-coded” directly into its description. We will see, however, that it is sometimes very useful to consider a DSM defined like this. Let us also note that given an encoding $\langle \langle M \rangle, \langle w \rangle \rangle$ of a DSM $M$ and a string $w$ over the input alphabet of $M$, it is possible to compute an encoding $\langle M_w \rangle$ of the DSM $M_w$ without difficulty.

**Proposition 17.2.** The language $E_{DSM}$ is undecidable.
The DSM $M_w$ operates as follows on input $x$:

1. Ignore the input string $x$ and run $M$ on $w$.

Figure 17.2: For any DSM $M$ and a fixed string $w$, the DSM $M_w$ ignores its input and runs $M$ on the string $w$ (which is hard-coded into $M_w$).

$\xrightarrow{} X \leftarrow \varepsilon$

$\xrightarrow{\text{push } X} a_n \xrightarrow{\text{push } X} a_{n-1} \cdots \xrightarrow{a_2} \text{push } X \xrightarrow{a_1} M$

Figure 17.3: An illustration of a state diagram for $M_w$, assuming $w = a_1 \cdots a_n$. We are also assuming that $X$ refers to the input stack of both $M_w$ and $M$, that the node labeled $X \leftarrow \varepsilon$ refers to the subroutine discussed in Lecture 12, and that the node labeled $M$ refers to the entire description of $M$. Thus, the action of this machine is to delete whatever input string it is given, replace this string with $w$, and allow control to pass to the start state of $M$.

The DSM $K$ operates as follows on input $x$:

1. If it is not the case that $x = \langle \langle M \rangle, \langle w \rangle \rangle$ for $M$ being a DSM and $w$ being a string over the alphabet of $M$, then reject.

2. Compute an encoding $\langle M_w \rangle$ of the DSM $M_w$ described in Figure 17.2.

3. Run $T$ on input $\langle M_w \rangle$. If $T$ accepts $\langle M_w \rangle$, then reject, and otherwise accept.

Figure 17.4: A high-level description of a DSM $K$ that decides $A_{\text{DSM}}$, assuming the existence of a DSM $T$ that decides $E_{\text{DSM}}$.

**Proof.** Assume toward contradiction that $E_{\text{DSM}}$ is decidable, so that there exists a DSM $T$ that decides this language. Define a new DSM $K$ as described in Figure 17.4. We can see from the description of $K$ that it will immediately reject when its input does not have the form $\langle \langle M \rangle, \langle w \rangle \rangle$, for $M$ a DSM and $w$ a string over the input alphabet of $M$. Let us consider what happens for inputs that are of the form
Lecture 17

⟨⟨M⟩, ⟨w⟩⟩, where M a DSM and w is a string over the input alphabet of M.

First, if w ∈ L(M), then the DSM M_w will accept every string over its alphabet; no matter what string it receives as an input, it just erases this string and runs M on w, leading to acceptance. It is therefore certainly not the case that L(M_w) = ∅. This implies that T must reject the string ⟨M_w⟩, and therefore K accepts ⟨⟨M⟩, ⟨w⟩⟩.

On the other hand, if w ∉ L(M), then M_w must either reject every string or run forever on every string for the same reason as before; M_w always discards its input and runs M on w, which either rejects or runs forever. It is therefore the case that L(M_w) = ∅. The DSM T therefore accepts ⟨M_w⟩, so K rejects ⟨⟨M⟩, ⟨w⟩⟩.

Considering the possibilities just analyzed, we find that K decides A_{DSM}, which contradicts the fact that this language is undecidable. We conclude that E_{DSM} is undecidable, as required.

17.2 Proving undecidability through reductions

The second 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 ⊆ Σ^* and B ⊆ Γ^* be languages. It is said that A reduces to B if there exists a computable function f : Σ^* → Γ^* such that

\[ w \in A \iff f(w) \in B \]  

for all w ∈ Σ^*. 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 ∈ Σ^*, 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 $\Sigma$ and $\Gamma$ be alphabets, let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ 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 semidecidable, then $A$ is semidecidable.

**Proof.** Let $f : \Sigma^* \rightarrow \Gamma^*$ 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 DSM $M_B$ such that $B = L(M_B)$. Define a new DSM $M_A$ as described in Figure 17.6. It is possible to define a DSM 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 $A = L(M_A)$, and therefore $A$ is semidecidable.
Lecture 17

The DSM $M_A$ 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 DSM $M_B$ that either decides or semidecides $B$, the DSM $M_A$ described either decides or semidecides $A$.

The proof for the first implication is almost identical, except that we take $M_B$ to be a DSM that decides $B$. The DSM $M_A$ 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 $\Sigma$, $\Gamma$, and $\Delta$ 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 $f : \Sigma^* \rightarrow \Gamma^*$ and $g : \Gamma^* \rightarrow \Delta^*$ such that

$$w \in A \iff f(w) \in B \quad \text{and} \quad y \in B \iff g(y) \in C \quad (17.7)$$

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

Define a function $h : \Sigma^* \rightarrow \Delta^*$ as $h(w) = g(f(w))$ for all $w \in \Sigma^*$. It is evident that $h$ is a computable function: if we have DSMs $M_f$ and $M_g$ that compute $f$ and $g$, respectively, then we can obtain a DSM $M_h$ 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 $h(w) = g(f(w)) \notin C$. \qed

**Proposition 17.6.** Let $\Sigma$ and $\Gamma$ 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 $f : \Sigma^* \rightarrow \Gamma^*$ and a string $w \in \Sigma^*$, the statements $w \in A \iff f(w) \in B$ and $w \notin A \iff f(w) \notin B$ 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. \qed
The DSM $K_M$ 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 DSM $M$, we can easily obtain a DSM $K_M$ 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.

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:

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 ($A_{DSM} \leq_m \text{HALT}$). The first thing we will need to consider is a simple way of modifying an arbitrary DSM $M$ to obtain a slightly different one. In particular, for an arbitrary DSM $M$, let us define a new DSM $K_M$ as described in Figure 17.7. The idea behind the DSM $K_M$ 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 DSM $M$, it is very easy to come up with a description of a DSM $K_M$ that operates as suggested: just replace the reject state of $M$ with a new state that purposely causes an infinite loop (by repeatedly pushing a symbol onto some stack, for instance).
Now let us define a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ as follows:

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

(17.8)

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 DSM $M$ and a string $w$ over the alphabet of $M$, and if so it replaces the encoding of $M$ with the encoding of the DSM $K_M$ suggested above.

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

$$\langle\langle M \rangle, \langle w \rangle \rangle \in A_{DSM} \Rightarrow M \text{ accepts } w \Rightarrow K_M \text{ halts on } w \Rightarrow \langle\langle K_M \rangle, \langle w \rangle \rangle \in HALT \Rightarrow f(\langle\langle M \rangle, \langle w \rangle \rangle) \in HALT.$$  

(17.9)

We therefore have

$$\langle\langle M \rangle, \langle w \rangle \rangle \in A_{DSM} \Rightarrow f(\langle\langle M \rangle, \langle w \rangle \rangle) \in HALT,$$

(17.10)

which is half of what we need to verify that $f$ is indeed a reduction from $A_{DSM}$ to $HALT$. It remains to consider the output of the function $f$ on inputs that are not contained in $A_{DSM}$, and here there are two cases: one is that the input takes the form $\langle\langle M \rangle, \langle w \rangle \rangle$ for a DSM $M$ 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:

$$\langle\langle M \rangle, \langle w \rangle \rangle \notin A_{DSM} \Rightarrow M \text{ does not accept } w \Rightarrow K_M \text{ runs forever on } w \Rightarrow \langle\langle K_M \rangle, \langle w \rangle \rangle \notin HALT \Rightarrow f(\langle\langle M \rangle, \langle w \rangle \rangle) \notin HALT.$$  

(17.11)

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 DSM $M$ and a string $w$ over the alphabet of $M$, and in this case it trivially holds that $f(x) = \epsilon \notin HALT$ (because $\epsilon$ does not encode any element of $HALT$). We have therefore proved that

$$x \in A_{DSM} \iff f(x) \in HALT,$$

(17.12)

and therefore $A_{DSM} \leq_m HALT$. 

9
We already proved that \( \text{HALT} \) is undecidable, but the fact that \( A_{\text{DSM}} \leq_m \text{HALT} \) provides an alternative proof: because we already know that \( A_{\text{DSM}} \) is undecidable, it follows that \( \text{HALT} \) 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 \( \text{HALT} \) is undecidable (which wasn’t particularly difficult). We have, however, established a closer relationship between \( A_{\text{DSM}} \) and \( \text{HALT} \) 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** (\( \text{DIAG} \leq_m E_{\text{DSM}} \)). Recall this language, which was defined earlier in the lecture:

\[
E_{\text{DSM}} = \{ \langle M \rangle : M \text{ is a DSM and } L(M) = \emptyset \}. \tag{17.13}
\]

We will now prove that \( \text{DIAG} \leq_m E_{\text{DSM}} \). Because we already know that \( \text{DIAG} \) is non-semidecidable, we conclude from this reduction that \( E_{\text{DSM}} \) is not just undecidable, but in fact it is also non-semidecidable.

For this one we will use the same hardcoding trick that we used earlier in the lecture: for a given DSM \( M \), let us define a new DSM \( M_{\langle M \rangle} \) just like in Figure 17.2, for the specific choice of the string \( w = \langle M \rangle \). 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 \( f : \{0, 1\}^* \rightarrow \{0, 1\}^* \) as follows:

\[
f(x) = \begin{cases} 
\langle M_{\langle M \rangle} \rangle & \text{if } x = \langle M \rangle \text{ for a DSM } M \\
\epsilon & \text{otherwise.}
\end{cases} \tag{17.14}
\]

If you think about it for a few moments, it should not be hard to convince yourself that \( f \) is computable. It remains to verify that \( f \) is a reduction from \( \text{DIAG} \) to \( E_{\text{DSM}} \).

For any string \( x \in \text{DIAG} \) we have that \( x = \langle M \rangle \) for some DSM \( M \) that satisfies \( \langle M \rangle \not\in L(M) \). In this case we have that \( f(x) = \langle M_{\langle M \rangle} \rangle \), and because \( \langle M \rangle \not\in L(M) \) it must therefore be that \( M_{\langle M \rangle} \) never accepts, and so \( f(x) = \langle M_{\langle M \rangle} \rangle \in E_{\text{DSM}} \).

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

We have proved that

\[
x \in \text{DIAG} \iff f(x) \in E_{\text{DSM}}, \tag{17.15}
\]

so the proof that \( \text{DIAG} \leq_m E_{\text{DSM}} \) is complete.
Example 17.9 \((A_{DSM} \leq_m A_{E})\). Define a language
\[
A_{E} = \{ \langle M \rangle : M \text{ is a DSM that accepts } \varepsilon \}.
\] (17.16)
The name \(A_{E}\) stands for “accepts the empty string.”

To prove this reduction, we can use exactly the same hardcoding trick that we’ve now used twice already. For every DSM \(M\) and every string \(w\) over the alphabet of \(M\), define a new DSM \(M_w\) as in Figure 17.2, and define a function \(f : \{0, 1\}^* \rightarrow \{0, 1\}^*\) as follows:
\[
f(x) = \begin{cases} 
\langle M_w \rangle & \text{if } x = \langle \langle M \rangle, \langle w \rangle \rangle \text{ for a DSM } M \text{ and a string } w \\
\varepsilon & \text{otherwise.}
\end{cases}
\] (17.17)

Now let us check that \(f\) is a valid reduction from \(A_{DSM}\) to \(A_{E}\).

First, for any string \(x \in A_{DSM}\) we have \(x = \langle \langle M \rangle, \langle w \rangle \rangle\) for a DSM \(M\) that accepts the string \(w\). In this case, \(f(x) = \langle M_w \rangle\). We have that \(M_w\) accepts every string, including the empty string, because \(M\) accepts \(w\). Therefore \(f(x) = \langle M_w \rangle \in A_{E}\).

Now consider any string \(x \notin A_{DSM}\). Again there are two cases: either \(x = \langle \langle M \rangle, \langle w \rangle \rangle\) for some DSM \(M\) and a string \(w\) over the alphabet of \(M\), or this is not the case. If it is the case that \(x = \langle \langle M \rangle, \langle w \rangle \rangle\) for a DSM \(M\) and \(w\) a string over the alphabet of \(M\), then \(x \notin A_{DSM}\) implies that \(M\) does not accept \(w\). In this case we have \(f(x) = \langle M_w \rangle \notin A_{E}\), because \(M_w\) does not accept any strings at all (including the empty string). If \(x \neq \langle \langle M \rangle, \langle w \rangle \rangle\) for a DSM \(M\) and string \(w\) over the alphabet of \(M\), then \(f(x) = \varepsilon \notin A_{E}\) (again because \(\varepsilon\) does not encode a DSM, and therefore cannot be included in the language \(A_{E}\)).

We have shown that \(x \in A_{DSM} \iff f(x) \in A_{E}\) holds for every string \(x \in \{0, 1\}^*\), and therefore \(A_{DSM} \leq_m A_{E}\), as required.

Example 17.10 \((E_{DSM} \leq_m R_{G})\). The last example for the lecture will be a tough one. Define a language as follows:
\[
R_{G} = \{ \langle M \rangle : M \text{ is a DSM such that } L(M) \text{ is regular} \}.
\] (17.18)
We will prove \(E_{DSM} \leq_m R_{G}\).

We will need to make use of a strange way to modify DSMs in order to do this one. Given an arbitrary DSM \(M\), let us define a new DSM \(K_M\) as in Figure 17.8. This is indeed a strange way to define a DSM, but there’s nothing wrong with strange DSMs—we’re just proving a reduction.

Now let us define a function \(f : \{0, 1\}^* \rightarrow \{0, 1\}^*\) as
\[
f(x) = \begin{cases} 
\langle K_M \rangle & \text{if } x = \langle M \rangle \text{ for a DSM } M \\
\varepsilon & \text{otherwise.}
\end{cases}
\] (17.19)
The DSM $K_M$ 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 $w$.
   2.2 If $M$ accepts $w$ within $t$ steps, goto 4.
3. Set $t \leftarrow t + 1$ and goto 2.
4. Accept if $x \in \{0^n1^n : n \in \mathbb{N}\}$, reject otherwise.

Figure 17.8: The DSM $K_M$ in Example 17.10.

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

Suppose $\langle M \rangle \in E_{DSM}$. We therefore have that $L(M) = \emptyset$; and by considering the way that $K_M$ behaves we see that $L(K_M) = \emptyset$ as well (because we never get to step 4 if $M$ never accepts). The empty language is regular, and therefore $f(\langle M \rangle) = \langle K_M \rangle \in REG$.

On the other hand, if $M$ is a DSM and $\langle M \rangle \not\in E_{DSM}$, then $M$ must accept at least one string. This means that $L(K_M) = \{0^n1^n : n \in \mathbb{N}\}$, because $K_M$ will eventually find a string accepted by $M$, reach step 4, and then accept or reject based on whether the input string $x$ is contained in the nonregular language

$$\{0^n1^n : n \in \mathbb{N}\}.$$ (17.20)

Therefore $f(\langle M \rangle) = \langle K_M \rangle \not\in REG$. The remaining case, in which $x$ does not encode a DSM, is straightforward as usual: we have $f(x) = \varepsilon \not\in REG$ in this case.

We have shown that $x \in E_{DSM} \iff f(x) \in REG$ holds for every string $x \in \{0, 1\}^*$, and therefore $E_{DSM} \leq_m REG$, as required. We conclude that the language $REG$ is non-semidecidable, as we already know that $E_{DSM}$ is non-semidecidable.