Lecture 15

Undecidable languages

In the previous lecture we discussed several examples of decidable languages relating to finite automata and context-free grammars. In this lecture we will discuss analogous languages relating to Turing machines, and we will find that these languages are *undecidable*.

15.1 Simulating one Turing machine with another

Let us begin by defining a language $\mathrm{A}_{\mathrm{DTM}}$ that is analogous to ones we considered in the previous lecture for DFAs, CFGs, and so on.

$$\mathrm{A}_{\mathrm{DTM}} = \bigl\{ \langle M, w \rangle \,:\, M \text{ is a DTM and } w \in \mathrm{L}(M) \bigr\}. \tag{15.1}$$

All of the same assumptions on encoding schemes that we have made previously are in place here. For instance, we assume $\langle M, w \rangle$ is a string over a fixed alphabet (such as $\{0,1\}$), the state set of M has the form $\{q_0, \ldots, q_{n-1}\}$ for some positive integer n, the alphabet of M takes the form $\{0, \ldots, m-1\}$ for some positive integer m, and so on. This language is not decidable, as we will prove shortly, but it is Turing recognizable.

The idea is pretty straightforward, and similar to what we discussed for the DFA variant of this language: given a DTM M and a string w as input, we can *simulate* M on input w and see what happens. In particular we can define a DTM U having the following high-level description:

On input $\langle M, w \rangle$, where M is a DTM and w is a string:

1. If w is not a string over the alphabet of M, then *reject*.
2. Simulate M on input w. If at any point in the simulation M accepts w, then *accept*; and if M rejects w, then *reject*.

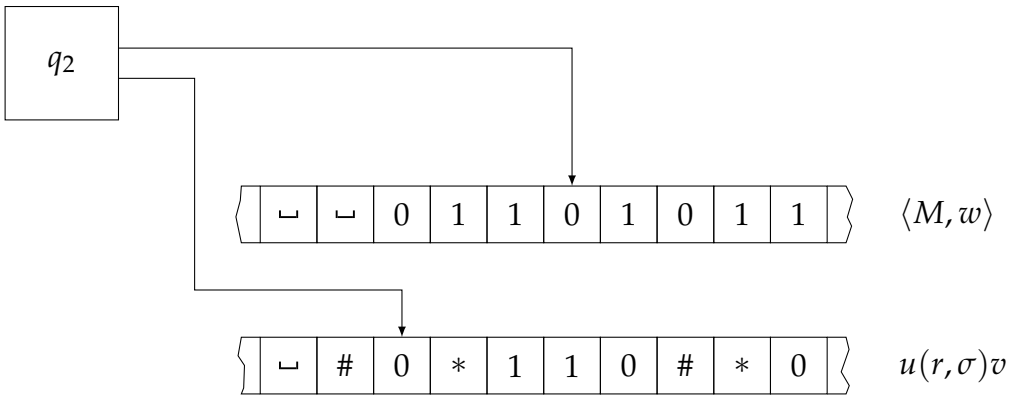

Figure 15.1: A two-tape DTM for $\mathrm{A}_{\mathrm{DTM}}$ could leave the input $\langle M, w \rangle$ on the first tape and store a configuration of M on the second tape. Through multiple steps of this DTM, it could transform the configuration into the next configuration after one step, and by repeating this process simulate M on input w.

We've chosen the name U for this DTM because it is a so-called *universal Turing machine*—meaning that it is a single Turing machine capable of simulating *every* DTM (even itself).

If you are interested, you can find detailed descriptions of universal Turing machines in some books. It is a critical step in the study of computability theory to recognize that such DTMs exist, so it is indeed worthy of careful scrutiny. On the other hand, it is fairly intuitive—if you have a description of a DTM M and an input string w, you can simulate M on input w by keeping track of the configuration of M and repeatedly update it one step at a time. Figure 15.1 suggests one way that this might be done using a two-tape DTM, which could then be converted into a one-tape DTM matching the description of U above.

It is important to note that the DTM U suggested above keeps on simulating M on input w until it has a reason to stop. If it so happens that M either accepts or rejects w, then U will accept or reject $\langle M, w \rangle$; but if M runs forever on input w, then U will also run forever on input $\langle M, w \rangle$. (Of course, if U receives an input that does not encode a DTM and a string, or if the string is not over the alphabet of the DTM, then it will reject this input.)

For the DTM U as defined above, it holds that $\mathrm{L}(U) = \mathrm{A}_{\mathrm{DTM}}$, and therefore $\mathrm{A}_{\mathrm{DTM}}$ is Turing recognizable. On the other hand, U does not decide $\mathrm{A}_{\mathrm{DTM}}$ because it might run forever on some inputs. Naturally, this does not imply that $\mathrm{A}_{\mathrm{DTM}}$ is undecidable—we have not yet argued that there cannot exist a different DTM that decides it, but we will do that shortly.

Before moving on, let us observe that if we included a limitation on the num-

ber of steps a DTM is allowed to run, as part of the input, we obtain a variant of the language defined above that is decidable. To be more precise, the following language is decidable:

$$\mathrm{S}_{\mathrm{DTM}} = \left\{ \langle M, w, t \rangle \; : \; \begin{array}{l} M \text{ is a DTM, } w \text{ is a string, } t \in \mathbb{N}, \\ \text{and } M \text{ accepts } w \text{ within } t \text{ steps} \end{array} \right\}. \tag{15.2}$$

This language could be decided by a DTM similar to U defined above, but where it cuts the simulation off after t steps if M has not accepted w.

15.2 A non-Turing-recognizable language

We will now define a language and prove it is undecidable. (In fact, this language will not even be Turing recognizable.) Here is the language:

$$\mathrm{DIAG} = \big\{ \langle M \rangle \; : \; M \text{ is a DTM and } \langle M \rangle \notin \mathrm{L}(M) \big\}. \tag{15.3}$$

Along the same lines as the other languages we have considered in this lecture and the previous one, the language DIAG is defined with respect to some encoding scheme for DTMs, so the language itself is over a fixed alphabet (such as the binary alphabet). The language DIAG contains all strings that, with respect to the chosen encoding scheme, encode a DTM that does not accept this encoding of itself.

If it so happens that a string $\langle M \rangle$ encodes a DTM whose input alphabet does not include every symbol in this encoding (such as in the case that M has input alphabet $\{0\}$ but we have chosen an encoding scheme over the alphabet $\{0, 1\}$), then it is indeed the case that $\langle M \rangle \notin \mathrm{L}(M)$.

Theorem 15.1. *The language* DIAG *is not Turing recognizable.*

Proof. Assume toward contradiction that DIAG is Turing recognizable. There must therefore exist a DTM T such that $\mathrm{L}(T) = \mathrm{DIAG}$.

Now, consider any encoding $\langle T \rangle$ of T. By the definition of the language DIAG one has

$$\langle T \rangle \in \mathrm{DIAG} \;\Leftrightarrow\; \langle T \rangle \notin \mathrm{L}(T). \tag{15.4}$$

On the other hand, because T recognizes DIAG it holds that

$$\langle T \rangle \in \mathrm{DIAG} \;\Leftrightarrow\; \langle T \rangle \in \mathrm{L}(T). \tag{15.5}$$

Consequently,

$$\langle T \rangle \notin \mathrm{L}(T) \;\Leftrightarrow\; \langle T \rangle \in \mathrm{L}(T), \tag{15.6}$$

which is a contradiction. We conclude that DIAG is not Turing recognizable. □

Remark 15.2. Note that this proof is very similar to the proof that $\mathcal{P}(\mathbb{N})$ is not countable from the very first lecture of the course.

15.3 Some undecidable languages

Now that we know DIAG is not Turing recognizable, we can go back and prove that $\mathrm{A}_{\mathrm{DTM}}$ is not decidable.

Proposition 15.3. *The language* A_{DTM} *is undecidable.*

Proof. Assume toward contradiction that $\mathrm{A}_{\mathrm{DTM}}$ is decidable. There must therefore exist a DTM T that decides $\mathrm{A}_{\mathrm{DTM}}$. Define a new DTM K as follows.

On input $\langle M \rangle$, where M is a DTM:

Run T on input $\langle M, \langle M \rangle \rangle$. If T accepts, then *reject*, otherwise *accept*.

For a given DTM M, we may now ask ourselves what K does on the input $\langle M \rangle$.

If it is the case that $\langle M \rangle \in \mathrm{DIAG}$, then by the definition of DIAG it holds that $\langle M \rangle \notin \mathrm{L}(M)$, and therefore $\langle M, \langle M \rangle \rangle \notin \mathrm{A}_{\mathrm{DTM}}$ (because M does not accept $\langle M \rangle$). This implies that T rejects the input $\langle M, \langle M \rangle \rangle$, and so K must accept the input $\langle M \rangle$. If, on the other hand, it is the case that $\langle M \rangle \notin \mathrm{DIAG}$, then $\langle M \rangle \in \mathrm{L}(M)$, and therefore $\langle M, \langle M \rangle \rangle \in \mathrm{A}_{\mathrm{DTM}}$. This implies that T accepts the input $\langle M, \langle M \rangle \rangle$, and so K must reject the input $\langle M \rangle$.

One final possibility is that that K is run on an input string that does not encode a DTM at all, and in this case it rejects.

Considering these possibilities, we find that K decides DIAG. This, however, is in contradiction with the fact that DIAG is not Turing recognizable (and is therefore undecidable). Having obtained a contradiction, we conclude that $\mathrm{A}_{\mathrm{DTM}}$ is undecidable, as required. □

Here is another example, which is a famous relative of $\mathrm{A}_{\mathrm{DTM}}$.

$$\mathrm{HALT} = \big\{ \langle M, w \rangle \ : \ M \text{ is a DTM that halts on input } w \ \big\}. \tag{15.7}$$

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—purely as a matter of terminology.

It is easy to prove that HALT is Turing recognizable—we just run a modified version of our universal Turing machine U on input $\langle M, w \rangle$, except that we *accept* in case the simulation results in either accept or reject—and when it is the case that M does not halt on input w this modified version of U will run forever on input $\langle M, w \rangle$.

Proposition 15.4. *The language* HALT *is undecidable.*

Proof. Assume toward contradiction that HALT is decidable, so that there exists a DTM T that decides it. Define a new DTM K as follows:

On input $\langle M, w \rangle$, where M is a DTM and w is a string:

1. Run T on input $\langle M, w \rangle$ and *reject* if T rejects.
2. Simulate M on input w; *accept* if M accepts and *reject* if M rejects.

The DTM K decides $\mathrm{A}_{\mathrm{DTM}}$, as a case analysis reveals:

- If it is the case that M accepts w, then T will accept $\langle M, w \rangle$ (because M halts on w), and the simulation of M on input w will result in acceptance.
- If it is the case that M rejects w, then T will accept $\langle M, w \rangle$ (because M halts on w), and the simulation of M on input w will result in rejection.
- If it is the case that M runs forever on w, then T will reject $\langle M, w \rangle$, and therefore K rejects without running the simulation of M on input w.

This, however, is in contradiction with the fact that $\mathrm{A}_{\mathrm{DTM}}$ is undecidable. Having obtained a contradiction, we conclude that HALT is undecidable. □

15.4 A couple of basic Turing machine tricks

There are many specific techniques through which certain languages can be proved to be either decidable or not, or Turing recognizable or not. Here we will illustrate a couple of these techniques, mainly through examples.

Limiting infinite search spaces

Sometimes we would like a Turing machine to effectively search over an infinitely large search space, but it is not immediately clear how to do this in a straightforward way. It can sometimes be helpful to use a single positive integer to serve as a bound on the search space, and then allow this positive integer to grow. (Something similar is useful for proving certain sets to be countable.) The proofs of the proposition and theorem that follow illustrate this method.

Proposition 15.5. *Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be Turing-recognizable languages. The language $A \cup B$ is Turing recognizable.*

Proof. Because A and B are Turing-recognizable languages, there must exist DTMs M_A and M_B such that $A = \mathrm{L}(M_A)$ and $B = \mathrm{L}(M_B)$. Define a new DTM M as follows:

On input w:

1. Set $t \leftarrow 1$.
2. Run M_A for t steps on input w. If M_A has accepted within t steps, then *accept*.
3. Run M_B for t steps on input w. If M_B has accepted within t steps, then *accept*.
4. Set $t \leftarrow t + 1$ and goto 2.

It is evident that $\mathrm{L}(M) = A \cup B$, and therefore $A \cup B$ is Turing recognizable. □

Remark 15.6. Of course we cannot replace the DTM M in the previous proof by one that first runs M_A on w (without any bound on its running time) and then runs M_B on w, because it could be that M_A runs forever on w but M_B accepts this string. It could also be the other way around, so that M_B cannot be run first without limiting its running time. Using t to bound the number of steps of both simulations is an effective way to handle this situation.

Theorem 15.7. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language such that both A and $\overline{A}$ are Turing recognizable. The language A is decidable.*

Proof. Because A and $\overline{A}$ are Turing-recognizable 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 follows:

On input w:

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

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. □

Hard-coding input strings

Suppose that we have a DTM M along with a string x over the input alphabet of M. Define a new DTM M_x as follows:

On input w:

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

This may seem like a curious thing to do—the DTM M_x runs the same way regardless of its input string, which is to run M on the string x that has been "hard-coded" directly into its description. We will see, however, that it is sometimes very useful to consider a DTM defined like this.

Let us also note that if you had an encoding $\langle M, x \rangle$ of a DTM M along with a string x over the input alphabet of M, it would be possible to compute an encoding $\langle M_x \rangle$ of the DTM M_x without any difficulties—the DTM M_x would have some initial phase of its computation that erases its input and writes x in its place, and this simple computation phase could be composed with the DTM M.

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

$$\mathrm{E}_{\mathrm{DTM}} = \big\{ \langle M \rangle \,:\, M \text{ is a DTM with } \mathrm{L}(M) = \varnothing \big\}. \tag{15.8}$$

Proposition 15.8. *The language* E_{DTM} *is not decidable.*

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 follows:

On input $\langle M, w \rangle$ where M is a DTM and w is a string:

1. If w is not a string over the input alphabet of M, then *reject*.
2. Compute an encoding $\langle M_w \rangle$ of the DTM M_w described previously.
3. Run T on input $\langle M_w \rangle$. If T accepts $\langle M_w \rangle$, then *reject*, and otherwise *accept*.

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

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

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. □