Lecture 16

Universal Turing machines and undecidable languages

In this lecture we will describe a *universal Turing machine*. This is a deterministic Turing machine that, when given the encoding of an arbitrary DTM, can *simulate* that machine on a given input.

To describe such a universal machine, we must naturally consider *encodings* of DTMs, and this will be the first order of business for the lecture. The very notion of an encoding scheme for DTMs allows us to obtain our first example of a language that is not semidecidable (and is therefore not decidable). Through the non-semidecidability of this language, many other languages can be shown to be either undecidable or non-semidecidable. We will see two simple examples in this lecture and more in the lecture following this one.

16.1 An encoding scheme for DTMs

In the previous lecture we discussed in detail an encoding scheme for DFAs, and we observed that this scheme is easily adapted to obtain an encoding scheme for NFAs. While we did not discuss specific encoding schemes for regular expressions and context-free grammars, we made use of the fact that one can devise encoding schemes for these models without difficulty.

We could follow a similar route for DTMs, as there are no new conceptual difficulties that arise for this model in comparison to the other models just mentioned. However, given the high degree of importance that languages involving encodings of DTMs will have in the remainder of the course, it is fitting to take a few moments to be careful and precise about this encoding. As is the case for just about every encoding scheme we consider, there are many alternatives to the encoding of DTMs we will define—our focus on the specifics of this encoding scheme is done

in the interest of clarity and precision, and not because the specifics themselves are essential to the study of computability.

Throughout the discussion that follows, we will assume that

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}) \tag{16.1}$$

is a given DTM whose encoding is to be described. We make the assumption that the state set Q of M takes the form

$$Q = \{q_0, \ldots, q_{m-1}\} \tag{16.2}$$

for some positive integer m, and that the input and tape alphabets of M take the form

$$\Sigma = \{0, \ldots, k-1\} \quad \text{and} \quad \Gamma = \{0, \ldots, n-1\} \tag{16.3}$$

for positive integers k and n. Given that Σ is properly contained in Γ, it follows that $k < n$; it is to be assume that the blank symbol $\sqcup \in \Gamma$ corresponds to the last symbol $n-1$ of Γ.

First, the encoding of a state $q \in Q$ should be understood as referring to the string $\langle q \rangle \in \{0,1\}^*$ obtained by expressing the index of that state written in binary notation, so that

$$\langle q_0 \rangle = 0, \quad \langle q_1 \rangle = 1, \quad \langle q_2 \rangle = 10, \quad \text{etc.} \tag{16.4}$$

Tape symbolds are encoded in a similar way, so that

$$\langle 0 \rangle = 0, \quad \langle 1 \rangle = 1, \quad \langle 2 \rangle = 10, \quad \text{etc.} \tag{16.5}$$

We shall encode the directions left and right as

$$\langle \leftarrow \rangle = 0 \quad \text{and} \quad \langle \rightarrow \rangle = 1. \tag{16.6}$$

Next we need a way to encode the transition function δ, and this will be done in a similar way to the encoding of DFAs from the previous lecture. For each choice of states $q \in Q \backslash \{q_{\text{acc}}, q_{\text{rej}}\}$ and $r \in Q$, tape symbols $a, b \in \Gamma$, and a direction $D \in \{\leftarrow, \rightarrow\}$, the encoding

$$\big\langle \langle q \rangle, \langle a \rangle, \langle r \rangle, \langle b \rangle, \langle D \rangle \big\rangle \in \{0,1\}^* \tag{16.7}$$

indicates that

$$\delta(q, a) = (r, b, D). \tag{16.8}$$

The encoding of the transition function δ is then obtained by encoding the list of binary strings of the form above, for each pair $(q, a) \in Q \backslash \{q_{\text{acc}}, q_{\text{rej}}\} \times \Gamma$ in lexicographic order.

Finally, by means of these encodings, we obtain from every DTM M the binary string encoding $\langle M \rangle \in \{0,1\}^*$ as follows:

$$\langle M \rangle = \big\langle \langle m \rangle, \langle k \rangle, \langle n \rangle, \langle \delta \rangle, \langle q_{\text{acc}} \rangle, \langle q_{\text{rej}} \rangle \big\rangle. \tag{16.9}$$

Example 16.1. Consider the DTM M for the language SAME whose state diagram is shown in Figure 12.3. We have

$$\begin{aligned}
\delta(q_0, 0) &= (q_0, 0, \leftarrow), \\
\delta(q_0, 1) &= (q_0, 1, \leftarrow), \\
\delta(q_0, \sqcup) &= (q_1, \sqcup, \rightarrow), \\
&\vdots \\
\delta(q_3, 0) &= (q_{\text{rej}}, 0, \leftarrow), \\
\delta(q_3, 1) &= (q_0, \sqcup, \leftarrow), \\
\delta(q_3, \sqcup) &= (q_{\text{rej}}, \sqcup, \leftarrow).
\end{aligned} \tag{16.10}$$

We will make the identification $q_{\text{acc}} = q_4$ and $q_{\text{rej}} = q_5$, and we note explicitly that there are $m = 6$ states, $k = 2$ input symbols, and $n = 3$ tape symbols, with the blank symbol $\sqcup$ being identified with the last tape symbol 2.

The transition $\delta(q_0, 0) = (q_0, 0, \leftarrow)$ is encoded as

$$\big\langle \langle q_0 \rangle, \langle 0 \rangle, \langle q_0 \rangle, \langle 0 \rangle, \langle \leftarrow \rangle \big\rangle = \langle 0, 0, 0, 0, 0 \rangle, \tag{16.11}$$

the transition $\delta(q_0, 1) = (q_0, 1, \leftarrow)$ is encoded as

$$\big\langle \langle q_0 \rangle, \langle 1 \rangle, \langle q_0 \rangle, \langle 1 \rangle, \langle \leftarrow \rangle \big\rangle = \langle 0, 1, 0, 1, 0 \rangle, \tag{16.12}$$

and, skipping to the last one, the transition $\delta(q_3, 2) = (q_5, 2, \leftarrow)$ is encoded as

$$\begin{aligned}
&\big\langle \langle q_3 \rangle, \langle \sqcup \rangle, \langle q_{\text{rej}} \rangle, \langle \sqcup \rangle, \langle \leftarrow \rangle \big\rangle \\
&\quad = \big\langle \langle q_3 \rangle, \langle 2 \rangle, \langle q_5 \rangle, \langle 2 \rangle, \langle \leftarrow \rangle \big\rangle = \langle 11, 10, 101, 10, 0 \rangle.
\end{aligned} \tag{16.13}$$

The entire transition function δ is encoded as

$$\begin{aligned}
\langle \delta \rangle = \big\langle &\big\langle \langle q_0 \rangle, \langle 0 \rangle, \langle q_0 \rangle, \langle 0 \rangle, \langle \leftarrow \rangle \big\rangle, \big\langle \langle q_0 \rangle, \langle 1 \rangle, \langle q_0 \rangle, \langle 1 \rangle, \langle \leftarrow \rangle \big\rangle, \\
&\ldots, \big\langle \langle q_3 \rangle, \langle \sqcup \rangle, \langle q_{\text{rej}} \rangle, \langle \sqcup \rangle, \langle \leftarrow \rangle \big\rangle \big\rangle \\
= \big\langle &\langle 0, 0, 0, 0, 0 \rangle, \langle 0, 1, 0, 1, 0 \rangle, \ldots, \langle 11, 10, 101, 10, 0 \rangle \big\rangle.
\end{aligned} \tag{16.14}$$

And, finally, and M is encoded as

$$\langle M \rangle = \big\langle \langle 6 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle \delta \rangle, \langle 0 \rangle, \langle 4 \rangle, \langle 5 \rangle \big\rangle. \tag{16.15}$$

16.2 A universal Turing machine

Now that we have defined an encoding scheme for DTMs, we can consider the computational task of simulating a given DTM on a given input. A *universal Turing machine* is a DTM that can perform such a simulation—it is *universal* in the sense that it is one single DTM that is capable of simulating all other DTMs.

Recall from Lecture 12 that a *configuration* of a DTM

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}) \tag{16.16}$$

may be expressed in the form

$$u(q,a)v \tag{16.17}$$

for a state $q \in Q$, a tape symbol $a \in \Gamma$, and strings of tape symbols $u, v \in \Gamma^*$, where u does not start with a blank and v does not end with a blank. Such a configuration may be encoded as

$$\big\langle u(q,a)v \big\rangle = \big\langle \langle u \rangle, \langle \langle q \rangle, \langle a \rangle \rangle, \langle v \rangle \big\rangle \tag{16.18}$$

where $\langle u \rangle$, $\langle q \rangle$, $\langle a \rangle$, and $\langle v \rangle$ refer to fitting encoding schemes we have already discussed in this lecture.

Now, if we wish to simulate the computation of a given DTM M on a given input string w, a natural approach is to keep track of the configurations of M and update them appropriately, as the computations themselves dictate. Specifically, we will begin with the initial configuration of M on input w, which is

$$(q_0, \sqcup)w. \tag{16.19}$$

We then repeatedly compute the *next* configuration of M, over and over, until perhaps we eventually reach a configuration whose state is q_{acc} or q_{rej}, at which point we can stop. We might never reach such a configuration; if M runs forever on input w, our simulation will also run forever. There is no way to avoid this, as we shall see.

With this approach in mind, let us focus on the task of simply determining the *next configuration*, meaning the one that results from one computational step, for a given DTM M and a given configuration of M. That is, we will consider the function

$$\text{next} : \{0,1\}^* \to \{0,1\}^* \tag{16.20}$$

defined as follows. For every DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ and every *nonhalting* configuration $u(q,a)v$ of M, we define

$$\text{next}(\langle \langle M \rangle, \langle u(q,a)v \rangle \rangle) = \langle x(p,b)y \rangle, \tag{16.21}$$

for $x(p,b)y$ being the configuration obtained by running M for one step starting from the configuration $u(q,a)v$, i.e., the unique configuration for which

$$u(q,a)v \vdash_M x(p,b)y. \tag{16.22}$$

We recall that Definition 12.2 describes this relation in precise terms. For every *halting* configuration $u(q,a)v$ of M, let us define (as a matter of convenience)

$$\text{next}(\langle\langle M\rangle, \langle u(q,a)v\rangle\rangle) = \langle u(q,a)v\rangle. \tag{16.23}$$

On inputs $x \in \{0,1\}^*$ *not* having the form $\langle\langle M\rangle, \langle u(q,a)v\rangle\rangle$, for M a DTM and $u(q,a)v$ a configuration of M, we shall take $\text{next}(x) = \varepsilon$.

Let us consider how this function might be computed using a stack machine. A natural first step is to process the input so that, upon conclusion of this processing, we have a stack M that stores $\langle M\rangle$, as well as four stacks, S, T, L, and R, that initially store the encodings $\langle q\rangle$, $\langle a\rangle$, $\langle u\rangle$, and $\langle v\rangle$, respectively. During this processing, we naturally stop and output ε if the input is not an encoding as we expect.

Next, the encoding $\langle M\rangle$, which includes within it the encoding $\langle\delta\rangle$ of the transition function δ, must be examined to determine $\delta(q,a) = (p,b,D)$. This requires a scan through $\langle\delta\rangle$ to obtain, after finding a match involving the strings $\langle q\rangle$ and $\langle a\rangle$ stored by S and T, the encodings $\langle p\rangle$, $\langle b\rangle$, and $\langle D\rangle$. These encodings may each be stored in their own stacks, which we need not name. A processing of S, T, L, and R that updates their contents to $\langle p\rangle$, $\langle b\rangle$, $\langle x\rangle$, and $\langle y\rangle$, for p, b, x, and y satisfying (16.22), is then performed.

Finally, these strings are recombined to form the output string

$$\langle\langle x\rangle, \langle\langle p\rangle, \langle b\rangle\rangle, \langle y\rangle\rangle. \tag{16.24}$$

It would be a time-consuming process to explicitly describe a DSM that operates in this way, but at a conceptual level this computation could reasonably be described as fairly straightforward. String matching subroutines would surely be helpful.

With the function next in hand, one can simulate the computation of a given DTM M on a given input w in the manner suggested above, by starting with the initial configuration $(q_0, \sqcup)w$ of M on w and repeatedly applying the function next.

Now consider the following language, which is the natural DTM analogue of the languages A_{DFA}, A_{NFA}, A_{REG}, and A_{CFG} discussed in the previous lecture:

$$\text{A}_{\text{DTM}} = \bigl\{\langle\langle M\rangle, \langle w\rangle\rangle \,:\, M \text{ is a DTM and } w \in \text{L}(M)\bigr\}. \tag{16.25}$$

We conclude that A_{DTM} is semidecidable: the DSM U described in Figure 16.1 is such that $\text{L}(U) = \text{A}_{\text{DTM}}$. This DSM has been named U to reflect the fact that it is a *universal DSM*. As described in Lecture 14, the DSM U can be simulated by a DTM.

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

1. If x takes the form $x = \langle\langle M\rangle, \langle w\rangle\rangle$, for M being a DTM and w being a string over the alphabet of M, then initialize the stack M so that it stores $\langle M\rangle$ and initialize C so that it stores $\langle(q_0, \sqcup)w\rangle$. Reject if not.
2. Repeat the following steps:
   - 2.1 If C stores a halting configuration of M, then halt and *accept* or *reject* accordingly. If C stores a non-halting configuration of M, the computation continues.
   - 2.2 Compute the configuration next$(\langle M\rangle, \langle u(q,a)v\rangle)$, for the encodings $\langle M\rangle$ and $\langle u(q,a)v\rangle$ stored in M and C, and update C so that it stores this new configuration.

Figure 16.1: A high-level description of a DSM U that recognizes the language $\mathrm{A}_{\mathrm{DTM}}$.

Proposition 16.2. *The language* $\mathrm{A}_{\mathrm{DTM}}$ *is semidecidable.*

Before moving on to the next section, let us note that the following language is decidable:

$$\mathrm{S}_{\mathrm{DTM}} = \left\{ \langle\langle M\rangle, \langle w\rangle, \langle t\rangle\rangle \;:\; \begin{array}{l} M \text{ is a DTM, } w \text{ is an input string to } M, \\ t \in \mathbb{N}, \text{ and } M \text{ accepts } w \text{ within } t \text{ steps} \end{array} \right\}. \tag{16.26}$$

This language could be decided by a DTM following essentially the same simulation described previously, but where the simulation cuts off and rejects after t steps if M has not yet halted on input w.

16.3 A few undecidable languages

It is natural at this point to ask whether or not $\mathrm{A}_{\mathrm{DTM}}$ is decidable, given that it is semidecidable. It is undecidable, as we will soon prove. Before doing this, however, we will consider a different language and prove that this language is not even semidecidable. Here is the language:

$$\mathrm{DIAG} = \big\{ \langle M\rangle \;:\; M \text{ is a DTM and } \langle M\rangle \notin \mathrm{L}(M) \big\}. \tag{16.27}$$

That is, the language DIAG contains all binary strings $\langle M\rangle$ that, with respect to the encoding scheme we discussed at the start of the lecture, encode a DTM M that

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

1. If it is not the case that $x = \langle M \rangle$ for M being a DTM, then reject.
2. Run T on input $\langle \langle M \rangle, \langle M \rangle \rangle$. If T accepts, then *reject*, otherwise *accept*.

Figure 16.2: A DTM that decides DIAG, assuming that there exists a DTM T for $\mathrm{A}_{\mathrm{DTM}}$.

does not accept this encoding of itself. Note that if it so happens that the string $\langle M \rangle$ encodes a DTM whose input alphabet has just one symbol, so that it does not include 0 and 1, then it will indeed be the case that $\langle M \rangle \notin \mathrm{L}(M)$.

Theorem 16.3. *The language* DIAG *is not semidecidable.*

Proof. Assume toward contradiction that the language DIAG is semidecidable. There must therefore exist a DTM M such that $\mathrm{L}(M) = \mathrm{DIAG}$.

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

$$\langle M \rangle \in \mathrm{DIAG} \;\Leftrightarrow\; \langle M \rangle \notin \mathrm{L}(M). \tag{16.28}$$

On the other hand, because M recognizes DIAG, it is the case that

$$\langle M \rangle \in \mathrm{DIAG} \;\Leftrightarrow\; \langle M \rangle \in \mathrm{L}(M). \tag{16.29}$$

Consequently,

$$\langle M \rangle \notin \mathrm{L}(M) \;\Leftrightarrow\; \langle M \rangle \in \mathrm{L}(M), \tag{16.30}$$

which is a contradiction. We conclude that DIAG is not semidecidable. □

Remark 16.4. 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. It is remarkable how simple this proof of the non-semidecidability of DIAG is; it has used essentially none of the specifics of the DTM model or the encoding scheme we defined.

Now that we know DIAG is not semidecidable, we may prove that $\mathrm{A}_{\mathrm{DTM}}$ is not decidable.

Theorem 16.5. *The language* $\mathrm{A}_{\mathrm{DTM}}$ *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 described in Figure 16.2.

The DTM 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 DTM and w an input string to 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 16.3: A DTM that decides $\mathrm{A}_{\mathrm{DTM}}$, assuming that there exists a DTM T that decides HALT.

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 is the case that $\langle M\rangle \notin \mathrm{L}(M)$, and therefore $\langle\langle M\rangle, \langle M\rangle\rangle \notin \mathrm{A}_{\mathrm{DTM}}$ (because M does not accept $\langle M\rangle$). This implies that T rejects the input $\langle\langle M\rangle, \langle M\rangle\rangle$, 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 $\langle\langle M\rangle, \langle M\rangle\rangle \in \mathrm{A}_{\mathrm{DTM}}$. This implies that T accepts the input $\langle\langle M\rangle, \langle M\rangle\rangle$, and so K must reject the input $\langle M\rangle$. One final possibility is 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 non-semidecidable (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 $\mathrm{A}_{\mathrm{DTM}}$.

$$\mathrm{HALT} = \bigl\{\langle\langle M\rangle, \langle w\rangle\rangle \,:\, M \text{ is a DTM that halts on input } w\bigr\}. \tag{16.31}$$

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 semidecidable, we just run a modified version of our universal Turing machine U on input $\langle\langle M\rangle, \langle w\rangle\rangle$, 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 $\langle\langle M\rangle, \langle w\rangle\rangle$.

Theorem 16.6. *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 in Figure 16.3. The DTM K decides A_{DTM}, as a case analysis reveals:

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 $\langle\langle M\rangle, \langle w\rangle\rangle$ (because M halts on w), and the simulation of M on input w will result in rejection.
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_{DTM} is undecidable. Having obtained a contradiction, we conclude that HALT is undecidable. □