Lecture 13

Turing Machines

So far, we have seen two different models of computation with different amounts of expressive power. The first was the model of finite automata, including DFAs and NFAs (which turn out to be equivalent). These automata have only a fixed, finite amount of memory, which cannot grow with the size of the input. Hence, in general, finite automata cannot even store the input string, and must process it one character at a time.

The second model was pushdown automata (PDAs). These are machines similar to NFAs, except they have an additional resource: a stack which can store any amount of information. In any transition, a PDA may push or pop from the stack, allowing it to store an arbitrary amount of information in memory and to later read it out. The catch is that PDAs may only read from memory by popping from the top of the stack; to read information that was written a long time ago, a PDA must first pop from the stack everything that was written afterwards, leading it to forget it all. This limitation on how the memory may be accessed limits the lower of PDAs; we know that as a computational model, PDAs can recognize only the context-free languages, which do not include languages such as $\{0^i 1^i 2^i : i \in \mathbb{N}\}$.

We will now introduce a more powerful computational model, which has full access to memory. This model of computation is called Turing machines (TMs), after the mathematician Alan Turing, who first defined them in 1936.

A Turing machine will be defined as a DFA with memory. More specifically, a TM will have a finite number of states with transitions between them, just like a DFA, but it will also have access to a memory tape. This memory tape is similar to the stack of a PDA, except it is a *tape* rather than a stack: we will allow the machine to move along the tape and read or write symbols in arbitrary order (rather than being able to change only the most recent additions to the stack, like a PDA). Just as for a PDA, there will be no limit on how much can be stored in the memory tape, although at any given point in time only a finite amount will be written (because at every point in time, the machine has run only finitely many steps).

13.1 Formally defining Turing machines

Formally, a Turing machine will be a 7-tuple

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

As usual, Q will be a finite set of states and $q_0 \in Q$ will be the start state. Σ will be the input alphabet (meaning it is a finite nonempty set), so the Turing machine will receive input strings from Σ^*. Γ will be the tape alphabet. We will require that the tape alphabet Γ contains all of

the input alphabet, i.e. $\Sigma \subseteq \Gamma$, so that the input may be written on the tape. We will also have a special symbol in Γ to represent the tape being "blank"; we will denote this symbol by $\llcorner$ (meant to represent a visible blank space). We will require that $\llcorner \in \Gamma$ and also that $\llcorner \notin \Sigma$, so that we do not confuse blank spaces with input symbols. As usual, we also require Γ to be finite, since it is an alphabet. q_{acc} and q_{rej} will be two special states in Q, called the accept state and the reject state respectively. If a Turing machine ever reaches q_{acc}, it stops running and accepts; similarly, if it ever reaches q_{rej}, it stops running and rejects. We require that $q_{\text{acc}} \neq q_{\text{rej}}$ (so in particular, Q must have cardinality at least 2).

Finally, δ will be the transition function. It must be a function

$$\delta \colon (Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \to \mathrm{Q} \times \Gamma \times \{\leftarrow, \rightarrow\}$$

which takes in a pair (p, a) with $p \in Q$ and $a \in \Gamma$ (and with p not being the accept or reject states), and outputs a triple $(q, b, \leftarrow)$ or $(q, b, \rightarrow)$ with $q \in Q$ and $b \in \Gamma$.

The interpretation of this mathematical definition is as follows. The Turing machine should be thought of as being a small machine that processes (and moves along) an infinite tape, as in Figure 13.1.

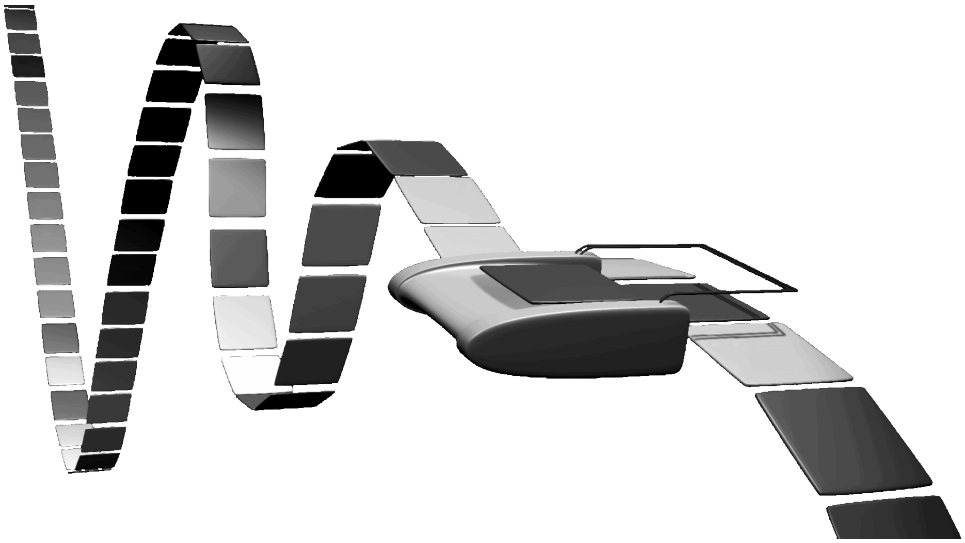

Figure 13.1: A Turing machine. Image by Schadel, public domain.

The tape is infinite in both directions, but the machine can only read one cell of the tape at a time. Each cell only contains a single symbol in Γ. The machine can read the cell, overwrite it with a new symbol in Γ, and may also move left or right (by one cell at a time). The machine has a finite number of internal states (the set Q), and these states represent the only "memory" that the machine can have except for what's written on the tape. To distinguish between the whole TM (including both the small machine and the tape) and the small "machine" with a finite number of states, we refer to the latter as the *head* of the Turing machine. If the machine is at internal state p and is currently seeing symbol a in the head position, then it will move according to $\delta(p, a)$; if $\delta(p, a) = (q, b, D)$ with $q \in Q$, $b \in \Gamma$, and $D \in \{\leftarrow, \rightarrow\}$, then the TM moves to internal state q, overwrites the a on the tape with the symbol b, and moves the head position either left or right according to the direction D.

The Turing machine will always start with the input written on the tape. That is, tape (which is infinite in both directions) will contain only $\llcorner$ symbols, except for the positions immediately to

the right of the initial head position, which will be the symbols of the input string w (followed by infinitely many blank symbols again). The machine will then proceed according to the transition rule δ, moving along the tape and editing it; if ever the machine enters the internal state q_{acc}, it halts and accepts the input string, and it ever it enters the internal state q_{rej}, it halts and rejects the input string.

13.1.1 The configuration of a Turing machine and the yields relation

The status of a Turing machine in the middle of a run is always fully determined by (a) what is written on the tape, (b) the position of the head, and (c) the internal state of the head. Note that in the beginning, only a finite string is written on the tape, and the rest of the tape symbols are blank. Further more, after any finite number of steps, the TM can only have written in finitely many cells of the tape; hence no matter how long the TM runs, it will always be the case that all but finitely many tape positions are blank.

With this in mind, we can represent the configuration of a TM using the notation

$$u(q, a)v$$

where the string $u \in \Gamma^*$ represents all the non-blank symbols to the left of the head position, the string $v \in \Gamma^*$ represents all the non-blank symbols to the right of the head position, the symbol $a \in \Gamma$ is the symbol written in the current head position of the tape, and $q \in Q$ is the current internal state. To make this notation unambiguous, we will require that u does not start with a blank symbol and v does not end with a blank symbol (otherwise we could shorten those strings by removing the blank symbols from the start of u and the end of v).

In this notation, we will write $(q, a)v$ if all the symbols to the left of the head are blank (since in this case, $u = \epsilon$, and we omit writing this string). Similarly, we write $u(q, a)$ if all the symbols to the right of the head are blank, and write simply (q, a) if the symbols on both the left and right are blank.

To make this notation more convenient to work with, we will introduce two functions, $\alpha\colon \Gamma^* \to \Gamma^*$ and $\beta\colon \Gamma^* \to \Gamma^*$, where $\alpha(w)$ strips the string w of all initial blank symbols and $\beta(w)$ strips w of all final blank symbols. Formally, we can define these recursively, with $\alpha(w) = w$ if $w = \epsilon$ or if $w_1 \neq \text{␣}$ and $\alpha(w) = \alpha(s)$ if $w = \text{␣}s$. Similarly, $\beta(w) = w$ if $w = \epsilon$ or if the last symbol of w is not ␣, and $\beta(w) = \beta(s)$ if $w = s\text{␣}$. The configuration of a TM can always be written $u(q, a)v$ with $u, v \in \Gamma^*$ being strings such that $\alpha(u) = u$ and $\beta(v) = v$. It is possible for u and/or v to be ϵ.

The initial configuration of a TM that is being run on an input $w \in \Sigma^*$ will be

$$(q_0, \text{␣})w,$$

since the input w will be written to the right of the head position and the rest of the symbols on the tape will be blank (additionally, q_0 is the start state). Recall that w is a string over the alphabet Σ, which is a subset of Γ; further, Σ does not contain ␣, so w does not have any blank symbols inside of it.

We can now define the yields relation for a Turing machine M. This will define the way of getting from one configuration $u(p, a)v$ of the TM to the next configuration after an application of one transition δ. We will define this yields relation $\vdash_M$ as follows: if $\delta(p, a) = (q, b, \leftarrow)$, we define

$$(p, a)u \vdash_M (q, \text{␣})\beta(bu),$$

$$vc(p, a)u \vdash_M v(q, c)\beta(bu).$$

The first line gives the action of $\vdash_M$ in the case where there are no non-blank symbols to the left, and the second line gives the action of $\vdash$ when the string of symbols to the left is vc, with $v \in \Gamma^*$ and $c \in \Gamma$, and where $\alpha(vc) = vc$. In both cases, what happens is that the head overwrote the symbol a with the symbol b, and then moved to the left, meaning that bu now occurs to the right of the head's position. We apply β to bu to remove any tailing blank symbols; such a symbol could have been introduced in the case where $u = \epsilon$ and $b = \textvisiblespace$, in which case $\beta(bu) = \beta(\textvisiblespace) = \epsilon$.

Similarly, if $\delta(p, a) = (q, b, \rightarrow)$, we define

$$v(p, a) \vdash_M \alpha(vb)(q, \textvisiblespace),$$

$$v(p, a)cu \vdash_M \alpha(vb)(q, c)u.$$

Like in the previous case, the first line governs the behavior of the yields relation where the tape is all blank to the right of the head's position, and the second line governs the behavior of the yields relation where the tape contains the string cu to the right of the head's position, with $\beta(cu) = cu$.

While the notation is a little cluttered, hopefully the meaning is clear: the yields relation $\vdash_M$ converts one configuration of M to another, according to the transition rule δ of the Turing machine M. Note that such a transition is only defined starting from $u(p, a)v$ when $p \in Q \backslash \{q_{\text{acc}}, q_{\text{rej}}\}$, because the transition $\delta(p, a)$ is only defined in this case. If $p = q_{\text{acc}}$ or $p = q_{\text{rej}}$, we cannot apply $\vdash_M$, and we say the Turing machine has halted.

Next, we write

$$u(p, a)v \vdash_M^* y(q, b)z$$

for $u, v, y, z \in \Gamma^*$, $p, q \in Q$, and $a, b \in \Gamma$ if there is a finite sequence of configurations

$$w_1(r_1, c_1)x_1, w_2(r_2, c_2)x_2, \ldots, w_m(r_m, c_m)x_m$$

satisfying

$$w_1(r_1, c_1)x_1 = u(p, a)v,$$

$$w_m(r_m, c_m)x_m = y(q, b)z,$$

and

$$w_i(r_i, c_i)x_i \vdash_M w_{i+1}(r_{i+1}, c_{i+1})x_{i+1}$$

for all $i = 1, 2, \ldots, m - 1$. In other words, we write $u(p, a)v \vdash_M^* y(q, b)z$ if it is possible to go from configuration $u(p, a)v$ to configuration $y(q, b)z$ using some finite number of transitions of the Turing machine M.

13.2 Acceptance and rejection

Now that we have defined configurations and the yields relation, we can define what it means for a Turing machine M to accept or reject a string $w \in \Sigma^*$. We say that $M(w)$ *accepts* if

$$(q_0, \textvisiblespace)w \vdash_M^* u(q_{\text{acc}}, a)v$$

for some $u, v \in \Gamma^*$ and $a \in \Gamma$. In other words, $M(w)$ accepts if, when the TM M is run starting from the configuration $(q_0, \textvisiblespace)w$, it eventually reaches the accept state q_{acc} after some finite number of steps. Similarly, we say that $M(w)$ *rejects* if

$$(q_0, \textvisiblespace)w \vdash_M^* u(q_{\text{rej}}, a)v$$

for some $u, v \in \Gamma^*$ and $a \in \Gamma$. This means that when M is run starting from $(q_0, \textvisiblespace)w$, it eventually reaches the reject state q_{rej} after some finite number of steps. Note that since there are no transitions out of an accept or reject state, it is not possible for $M(w)$ to both accept and reject.

What if M never reaches the accept or reject state when starting from configuration $(q_0, \textvisiblespace)w$? This is possible. In this case, we say that $M(w)$ *loops*. This doesn't necessarily mean that it literally enters a loop, just that it never reaches the accept or reject states after any (finite) number of steps.

One last piece of terminology: if the Turing machine does not loop we say it *halts*. That is, a TM M halts on input w if and only if it either accepts w or rejects w.

Now that we have defined what it means for a Turing machine to accept, reject, or loop on a string w, we can define the language recognized by a Turing machine M. This language, denoted $L(M)$, will be the set of all strings $w \in \Sigma^*$ on which $M(w)$ accepts. However, note that this language does not tell the whole story of the behavior of the Turing machine M: it specifies which strings M accepts, but does not specify on which strings M rejects (rather than looping).

If a language A can be recognized by a Turing machine, we call it *recognizable*. If, in addition, A can be recognized by a Turing machine M which halts on all inputs (including input strings not in A), we say that A is *decidable*, and we say M *decides* A. Note that every decidable language is recognizable. To summarize:

- If there is a DFA or NFA M such that $L(M) = A$, then A is regular.
- If there is a PDA M such that $L(M) = A$, then A is context-free.
- If there is a TM M such that $L(M) = A$, then A is recognizable.
- If there is a TM M which halts on all input strings, and which also has $L(M) = A$, then A is decidable.

13.3 State transition diagrams and example

As we've done for other machines, we can describe a Turing machine by a state transition diagram. Recall that a Turing machine is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$. To specify a Turing machine, we need to specify all 7 components. As before will specify the states in Q by circles, with the initial state q_0 having an incoming arrow out of nowhere. The states q_{acc} and q_{rej} will be labeled by those names (additionally, the state q_{acc} is sometimes double-circled to match the notation of DFAs, NFAs, and PDAs). The alphabets Σ and Γ are usually not explicitly drawn; if it is unclear what they are, it will be useful to add text specifying the alphabets next to the transition diagram.

To represent the transition function $\delta\colon (Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$, we will draw one arc for each pair $(p, a) \in (Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma$. If $\delta(p, a) = (q, b, \leftarrow)$, then we draw the arc from state p to state q, labeled by the string "$a, b \leftarrow$". This will mean that when the TM is at state p and sees a on the tape, it will overwrite this with b, move to state q, and move left on the tape. Similarly, if $\delta(p, a) = (q, b, \rightarrow)$, we draw the arc from p to q labeled by "$a, b \rightarrow$". Note that each state $p \in Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}$ will have one outgoing transition for each symbol in Γ. This is because Turing machines are deterministic, like DFAs and unlike NFAs and PDAs.

13.3.1 An example of a Turing machine

See Figure 13.2 for an example of a Turing machine which decides the language 0^n1^n.

What is this Turing machine doing? Well, on input $w \in \{0, 1\}^*$, this TM starts in configuration $(q_0, \textvisiblespace)w$. Since it sees a $\textvisiblespace$ at the beginning, it always starts by transitioning from q_0 to q_1 and

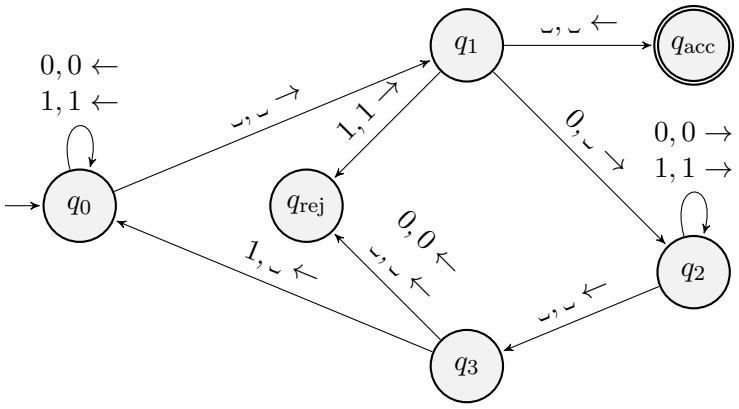

Figure 13.2: A Turing machine for the language $0^n 1^n$.

moving right. However, note that the state q_0 also has transitions for when it sees a 0 or a 1 on the tape; in such a case, the TM always stays on q_0 and moves left, meaning it keeps going left until it sees a $\sqcup$. This will be relevant later.

After it moves from q_0 to q_1, the TM will then try to delete a 0 from the beginning of the string and then go to the end of the string and delete a 1. Therefore, upon seeing a 0 from state q_1, the TM will delete this 0 (writing $\sqcup$) and move right, transitioning to state q_2, from which the TM will attempt to reach the end of the string. On the other hand, if, starting from q_1, the TM sees a $\sqcup$, it means the input is all blank (perhaps because matching 0s and 1s from the beginning and end of the string have already been deleted), so the TM accepts; alternatively, if the TM sees a 1 when at q_1, it means the input string starts with a 1, which means the TM should reject.

Next, it should be clear that q_2 travels to the end of the input string: it keeps moving right until it sees a $\sqcup$ on the tape, at which point it backtracks left and moves to q_3. When at q_3, the TM is guaranteed to be at the end of the string; if it sees a 1, it will delete it (replacing with $\sqcup$) and move right and back to q_0, which will return to the beginning of the string and start the process again. The TM continues in this way, deleting a 0 from the beginning and a 1 from the end, until the string is empty, when the TM accepts (or until such a 0 at the beginning or 1 at the end is not found, in which case the TM can reject).

As you can see, Turing machines can be somewhat cumbersome even when they perform relatively simple tasks. For this reason, we will try to avoid explicitly constructing them, and will often resort to less formal arguments for why a Turing machine exists for a certain task.

13.4 Next steps

In the next few lectures, we will see that Turing machines are immensely powerful: they can essentially do anything an ordinary computer can do. We will also see that modifying the definition of Turing machines in various ways – for instance, by giving them more than one tape – does not end up increasing their power. Despite this, we will show that there are still interesting languages they cannot decide or recognize.