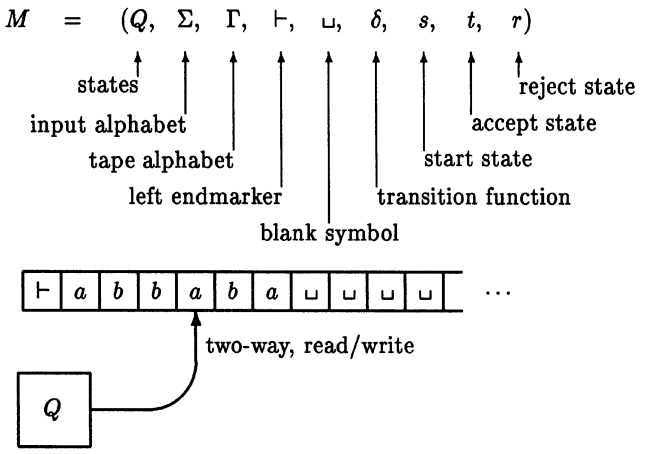
Lecture 29

More on Turing Machines

Last time we defined deterministic one-tape Turing machines:



In each step, based on the current tape symbol it is reading and its current state, it prints a new symbol on the tape, moves its head either left or right, and enters a new state. This action is specified formally by the transition function

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}.$$

Intuitively, $\delta(p,a) = (q,b,d)$ means, "When in state p scanning symbol a, write b on that tape cell, move the head in direction d, and enter state q."

We defined a *configuration* to be a triple (p,z,n) where p is a state, z is a semi-infinite string of the form $y\sqcup^\omega$, $y \in \Sigma^*$, describing the contents of the tape, and n is a natural number denoting a tape head position.

The transition function δ was used to define the *next configuration relation* $\xrightarrow[M]{1}$ on configurations and its reflexive transitive closure $\xrightarrow[M]{*}$. The machine M *accepts* input $x \in \Sigma^*$ if

$$(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (t,y,n)$$

for some y and n, and *rejects* input x if

$$(s, \vdash x\sqcup^\omega, 0) \xrightarrow[M]{*} (r,y,n)$$

for some y and n. The left configuration above is the *start configuration* on input x. Recall that we restricted TMs so that once a TM enters its accept state, it may never leave it, and similarly for its reject state. If M never enters its accept or reject state on input x, it is said to *loop* on input x. It is said to *halt* on input x if it either accepts or rejects. A TM that halts on all inputs is called *total.*

Define the set

$$L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}.$$

This is called the set *accepted by* M. A subset of Σ^* is called *recursively enumerable* (r.e.) if it is $L(M)$ for some M. A set is called *recursive* if it is $L(M)$ for some total M.

For now, the terms *r.e.* and *recursive* are just technical terms describing the sets accepted by TMs and total TMs, respectively; they have no other significance. We will discuss the origin of this terminology in Lecture 30.

Example 29.1 Consider the non-CFL $\{ww \mid w \in \{a,b\}^*\}$. It is a recursive set, because we can give a total TM M for it. The machine M works as follows. On input x, it scans out to the first blank symbol $\sqcup$, counting the number of symbols mod 2 to make sure x is of even length and rejecting immediately if not. It lays down a right endmarker $\dashv$, then repeatedly scans back and forth over the input. In each pass from right to left, it marks the first unmarked a or b it sees with $'$. In each pass from left to right, it marks the first unmarked a or b it sees with $`$. It continues this until all symbols are marked. For example, on input

$$a\,a\,b\,b\,a\,a\,a\,b\,b\,a$$

the initial tape contents are

$$\vdash a\,a\,b\,b\,a\,a\,a\,b\,b\,a \sqcup\sqcup\sqcup\cdots$$

and the following are the tape contents after the first few passes.

$$\vdash a\,a\,b\,b\,a\,a\,a\,b\,b\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\vdash \grave{a}\,a\,b\,b\,a\,a\,a\,b\,b\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\vdash \grave{a}\,a\,b\,b\,a\,a\,a\,b\,\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\vdash \grave{a}\,\grave{a}\,b\,b\,a\,a\,a\,b\,\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\vdash \grave{a}\,\grave{a}\,b\,b\,a\,a\,a\,\acute{b}\,\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$

Marking a with ` formally means writing the symbol $\grave{a} \in \Gamma$; thus

$$\Gamma = \{a,\ b,\ \vdash,\ \sqcup,\ \dashv,\ \grave{a},\ \grave{b},\ \acute{a},\ \acute{b}\}.$$

When all symbols are marked, we have the first half of the input string marked with ` and the second half marked with ´.

$$\vdash \grave{a}\,\grave{a}\,\grave{b}\,\grave{b}\,\grave{a}\,\acute{a}\,\acute{a}\,\acute{b}\,\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$

The reason we did this was to find the center of the input string.

The machine then repeatedly scans left to right over the input. In each pass it erases the first symbol it sees marked with ` but remembers that symbol in its finite control (to "erase" really means to write the blank symbol $\sqcup$). It then scans forward until it sees the first symbol marked with ´, checks that that symbol is the same, and erases it. If the two symbols are not the same, it rejects. Otherwise, when it has erased all the symbols, it accepts. In our example, the following would be the tape contents after each pass.

$$\grave{a}\,\grave{a}\,\grave{b}\,\grave{b}\,\grave{a}\,\acute{a}\,\acute{a}\,\acute{b}\,\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\sqcup\grave{a}\,\grave{b}\,\grave{b}\,\grave{a}\sqcup\acute{a}\,\acute{b}\,\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\sqcup\sqcup\grave{b}\,\grave{b}\,\grave{a}\sqcup\sqcup\acute{b}\,\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\sqcup\sqcup\sqcup\grave{b}\,\grave{a}\sqcup\sqcup\sqcup\acute{b}\,\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\sqcup\sqcup\sqcup\sqcup\grave{a}\sqcup\sqcup\sqcup\sqcup\acute{a}\,\dashv\sqcup\sqcup\sqcup\cdots$$
$$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\dashv\sqcup\sqcup\sqcup\cdots$$ □

Example 29.2 We want to construct a total TM that accepts its input string if the length of the string is prime. This language is not regular or context-free. We will give a TM implementation of the *sieve of Eratosthenes*, which can be described informally as follows. Say we want to check whether n is prime. We write down all the numbers from 2 to n in order, then repeat the following: find the smallest number in the list, declare it prime, then cross off all multiples of that number. Repeat until each number in the list has been either declared prime or crossed off as a multiple of a smaller prime.

For example, to check whether 23 is prime, we would start with all the numbers from 2 to 23:

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

In the first pass, we cross off multiples of 2:

~~2~~ 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23

The smallest number remaining is 3, and this is prime. In the second pass we cross off multiples of 3:

~~2~~ ~~3~~ ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23

Then 5 is the next prime, so we cross off multiples of 5; and so forth. Since 23 is prime, it will never be crossed off as a multiple of anything smaller, and eventually we will discover that fact when everything smaller has been crossed off.

Now we show how to implement this on a TM. Suppose we have a^p written on the tape. We illustrate the algorithm with $p = 23$.

$$\vdash a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,\sqcup\sqcup\sqcup\cdots$$

If $p = 0$ or $p = 1$, reject. We can determine this by looking at the first three cells of the tape. Otherwise, there are at least two a's. Erase the first a, scan right to the end of the input, and replace the last a in the input string with the symbol \$. We now have an a in positions $2, 3, 4, \ldots, p-1$ and \$ at position p.

$$\vdash \sqcup a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,\$\,\sqcup\sqcup\sqcup\cdots$$

Now we repeat the following loop. Starting from the left endmarker $\vdash$, scan right and find the first nonblank symbol, say occurring at position m. Then m is prime (this is an invariant of the loop). If this symbol is the \$, we are done: $p = m$ is prime, so we halt and accept. Otherwise, the symbol is an a. Mark it with a $\widehat{\ }$ and everything between there and the left endmarker with $\acute{}$.

$$\vdash \acute{\sqcup}\,\widehat{a}\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,\$\,\sqcup\sqcup\sqcup\cdots$$

We will now enter an inner loop to erase all the symbols occurring at positions that are multiples of m. First, erase the a under the $\widehat{\ }$. (Formally, just write the symbol $\widehat{\sqcup}$.)

$$\vdash \acute{\sqcup}\,\widehat{\sqcup}\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,\$\,\sqcup\sqcup\sqcup\cdots$$

Shift the marks to the right one at a time a distance equal to the number of marks. This can be done by shuttling back and forth, erasing marks on the left and writing them on the right. We know when we are done because the $\widehat{\ }$ is the last mark moved.

$$\vdash \sqcup\sqcup\acute{a}\,\widehat{a}\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,\$\,\sqcup\sqcup\sqcup\cdots$$

When this is done, erase the symbol under the $\widehat{\ }$. This is the symbol occurring at position $2m$.

$$\vdash \sqcup\sqcup\acute{a}\,\widehat{\sqcup}\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,\$\,\sqcup\sqcup\sqcup\cdots$$

Keep shifting the marks and erasing the symbol under the $\widehat{\ }$ in this fashion until we reach the end.

$$\vdash \sqcup\sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup \acute{\$}\, \widehat{\sqcup}\sqcup\sqcup \cdots$$

If we find ourselves at the end of the string wanting to erase the \$, reject—p is a multiple of m but not equal to m. Otherwise, go back to the left and repeat. Find the first nonblank symbol and mark it and everything to its left.

$$\vdash \acute{\sqcup}\acute{\sqcup}\widehat{a} \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup a \sqcup \$ \sqcup\sqcup\sqcup \cdots$$

Alternately erase the symbol under the $\widehat{\ }$ and shift the marks until we reach the end of the string.

$$\vdash \sqcup\sqcup\sqcup\sqcup a \sqcup a \sqcup\sqcup\sqcup a \sqcup a \sqcup\sqcup\sqcup a \sqcup a \sqcup\sqcup\acute{\sqcup} \acute{\$}\, \widehat{\sqcup}\sqcup\sqcup \cdots$$

Go back to the left and repeat.

$$\vdash \acute{\sqcup}\acute{\sqcup}\acute{\sqcup}\acute{\sqcup}\widehat{a} \sqcup a \sqcup\sqcup\sqcup a \sqcup a \sqcup\sqcup\sqcup a \sqcup a \sqcup\sqcup\sqcup \$ \sqcup\sqcup\sqcup \cdots$$

If we ever try to erase the \$, reject—p is not prime. If we manage to erase all the a's, accept. □

Recursive and R.E. Sets

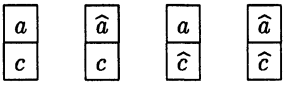
Recall that a set A is *recursively enumerable* (r.e.) if it is accepted by a TM and *recursive* if it is accepted by a *total* TM (one that halts on all inputs).

The recursive sets are closed under complement. (The r.e. sets are not, as we will see later.) That is, if A is recursive, then so is $\sim A = \Sigma^* - A$. To see this, suppose A is recursive. Then there exists a total TM M such that $L(M) = A$. By switching the accept and reject states of M, we get a total machine M' such that $L(M') = \sim A$.

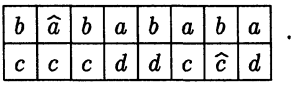
This construction does not give the complement if M is not total. This is because "rejecting" and "not accepting" are not synonymous for nontotal machines. To reject, a machine must enter its reject state. If M' is obtained from M by just switching the accept and reject states, then M' will accept the strings that M rejects and reject the strings that M accepts; but M' will still loop on the same strings that M loops on, so these strings are not accepted or rejected by either machine.

Every recursive set is r.e. but not necessarily vice versa. In other words, not every TM is equivalent to a total TM. We will prove this in Lecture 31. However, if both A and $\sim A$ are r.e., then A is recursive. To see this, suppose both A and $\sim A$ are r.e. Let M and M' be TMs such that $L(M) = A$ and $L(M') = \sim A$. Build a new machine N that on input x runs both M and

M' simultaneously on two different tracks of its tape. Formally, the tape alphabet of N contains symbols



where a is a tape symbol of M and c is a tape symbol of M'. Thus N's tape may contain a string of the form



for example. The extra marks $\widehat{\ }$ are placed on the tape to indicate the current positions of the simulated read/write heads of M and M'. The machine N alternately performs a step of M and a step of M', shuttling back and forth between the two simulated tape head positions of M and M' and updating the tape. The current states and transition information of M and M' can be stored in N's finite control. If the machine M ever accepts, then N immediately accepts. If M' ever accepts, then N immediately rejects. Exactly one of those two events must eventually occur, depending on whether $x \in A$ or $x \in \sim A$, since $L(M) = A$ and $L(M') = \sim A$. Then N halts on all inputs and $L(N) = A$.

Decidability and Semidecidability

A property P of strings is said to be *decidable* if the set of all strings having property P is a recursive set; that is, if there is a total Turing machine that accepts input strings that have property P and rejects those that do not. A property P is said to be *semidecidable* if the set of strings having property P is an r.e. set; that is, if there is a Turing machine that on input x accepts if x has property P and rejects or loops if not. For example, it is decidable whether a given string x is of the form ww, because we can construct a Turing machine that halts on all inputs and accepts exactly the strings of this form.

Although you often hear them switched, the adjectives *recursive* and *r.e.* are best applied to sets and *decidable* and *semidecidable* to properties. The two notions are equivalent, since

$$P \text{ is decidable} \iff \{x \mid P(x)\} \text{ is recursive.}$$
$$A \text{ is recursive} \iff \text{“}x \in A\text{” is decidable,}$$

$$P \text{ is semidecidable} \iff \{x \mid P(x)\} \text{ is r.e.,}$$
$$A \text{ is r.e.} \iff \text{“}x \in A\text{” is semidecidable.}$$