Lecture #10: Introduction to Turing Machines
Turing Machine Design

Turing machine design and analysis is not the focus of the rest of this course! However, knowing something about this can help to understand related technical results, and their proofs, and to solve more interested related problems. This document introduces a design technique — which is also useful for more realistic "software development" — and applies it to obtain a solution for this problem (as a Turing machine) that you have seen already.

Design Technique: Refinement

Sometimes a problem seems too big, and it is hard to know where to start. One way to deal with this is to use ***refinement*** — that is, to start by developing a very general solution for the problem that you are given. This solution will not be very detailed. However, you will be able to use it to identify multiple smaller and simpler problems, which you can focus on, in order to add detail and complete the solution for the problem that you are starting with.

If you search for information about "refinement" you might discover information about applications of this idea, in computation, that do not have anything to do with Turing machines. However, the same general approach is being used (just, in a different setting).

When applying this idea to Turing machine design it will be helpful to consider several different kinds of descriptions of Turing machines — which are described below. These are based on a brief description of "Terminology for Describing Turing Machines" given by Sipser [1].

Example Problem To Be Solved

Let $\Sigma = \{0, 1\}$. The problem to be considered is to design the language

$$L = \{0^n 1^n \mid n \in \mathbb{Z} \text{ and } n \geq 0\} \subseteq \Sigma^\star.$$

High-Level Description

One should often begin by choosing an ***algorithm*** that will — somehow — solve the given problem. This version of the solution — which can be given as pseudocode or simple English — is sometimes called a ***high-level description*** of a Turing machine.

One can establish ***correctness***, at this level, by describing the effects of the operations that have been carried out. This can sometimes resemble the process of "proving correctness of an algorithm" described in a course like CPSC 331 or CPSC 413.

Application to the Example Problem

First Attempt

Consider the following process, which you might apply to try to decide whether a string $\omega \in \Sigma^\star$ belongs to the above language L:

> Starting at the left end of the string, look for an unmarked copy of 0 and then mark it. Continuing to move right, look for an unmarked copy of 1 and then mark it.
>
> ***Reject*** ω if any of the following things happens:
>
> (a) You discover that ω has a copy of "1" somewhere to the left of a copy of "0", so that it does not have the form "$1^i 0^j$" for non-negative integers i and j.
>
> (b) You are able to find, and mark, an unmarked copy of "0" — but you are unable to complete this step by finding an unmarked copy of 1 — so that ω includes strictly more copies of "0" than it has copes of "1".
>
> (c) You are unable to find, and mark, an unmarked copy of "0" — but you discover that the string still has at least one unmarked copy of "1" — so that ω includes strictly more copies of "1" than it has copies of "0".
>
> ***Accept*** ω if you did not reject it (by discovering that you were in case (a), (b) or (c), above, but the following happens, instead.
>
> (d) You discover that ω does not have any unmarked copies of "0" and it also does not have any unmarked copies of "1".
>
> Repeat the above process if none of cases (a), (b), (c), or (d) arise (so that you neither rejected or accepted ω in the current round.

This might seem a little vague — it is not clear precisely *now* you do determine whether you are in one of cases (a), (b), (c) or (d) when you are scanning over the symbols in a string. Let us keep that in mind as we continue.

Proving Correctness

Let $\omega \in \Sigma^\star$. Then exactly one of the following cases arises.

(i) $\omega \in L$, so that $\omega = 0^n1^n$ for some non-negative integer n.

(ii) $\omega \notin L$ because $\omega = 0^n1^m$ for non-negative integers n and m such that $n < m$.

(iii) $\omega \notin L$ because $\omega = 0^n1^m$ for non-negative integers n and m such that $n > m$.

(iv) $\omega \notin L$ because ω does not have the form 0^n1^m, for non-negative integers n and m, at all. That is, ω includes a copy of 1 that is somewhere to the left of a copy of 1.

When considering what happens in each of the above cases, suppose that we "mark" a copy of "0" by replacing it with "X" and that we "mark" a copy of "1" by replacing it with "Y".

(i) Suppose first that $\omega \in L$, so that $\omega = 0^n1^n$ for some non-negative integer n.

If $n = 0$ then $\omega = \lambda$, the empty string. The algorithm should therefore ***accept*** ω immediately, if it is noticed that $\omega = \lambda$.

Otherwise let i be an integer such that $0 \leq i \leq n$. Then it is easy shown by induction on i that, after i rounds of the above process, the string being considered has the form

$$\mathtt{X}^i0^{n-i}\mathtt{Y}^i1^{n-i}. \tag{1}$$

If $i < n$ then none of cases (a), (b), (c) or (d) arise, so that the process continues. On the other hand, if $i = n$ then the string has the form

$$\mathtt{X}^n\mathtt{Y}^n \tag{2}$$

— so that case (d) holds, and ω is accepted (after a finite number of rounds of the above process) as desired.

(ii) Suppose, next, that $\omega \notin L$ because $\omega = 0^n1^m$ for non-negative integers n and m such that $n < m$.

If $n = 0$ then $\omega = 1^m$ and, since $m > n$, ω is a nonempty string in $\Sigma^\star$ that starts with "1". Now, a consideration of the definition of L confirms that *no* string in $\Sigma^\star$, that starts with "1", belongs to L — and it would be acceptable for any string that starts with "1" to be rejected immediately (since either case (b) or case (c) must arise).

Suppose, instead, that $n \geq 1$, and let i be an integer such that $0 \leq i \leq n$. Then, after i rounds of the above process the string being considered has the form

$$\mathtt{X}^i0^{n-i}\mathtt{Y}^i1^{m-i}. \tag{3}$$

If $i < n$ then none of cases (a), (b), (c) or (d) arise, so that the process continues. On the other hand, if $i = n$ then the string has the form

$$\mathtt{X}^n\mathtt{Y}^n\mathtt{1}^{m-n} \tag{4}$$

and case (c) arises — so that ω is rejected (after a finite number of rounds of the above process) as desired.

(iii) Now suppose that $\omega \notin L$ because $\omega = \mathtt{0}^n\mathtt{1}^m$ for non-negative integers n and m such that $n > m$.

If $m = 0$ then $\omega = \mathtt{0}^n$ and the end of the string will be reached (with $\sqcup$ seen on the tape) before a $\mathtt{1}$ is ever seen, at all.

Suppose, instead, that $m \geq 1$, and let i be an integer such that $0 \leq i \leq m$. Then, after i rounds of the above process the string considered has the form shown at line (3), above. If $i < m$ then none of cases (a), (b), (c) or (d) arise, so that the process continues. On the other hand, if $i = m$ then the string has the form

$$\mathtt{X}^m\mathtt{0}^{n-m}\mathtt{Y}^m \tag{5}$$

and case (b) arises — so that ω is rejected (after a finite number of rounds of the above process) as desired.

(iv) Finally, suppose that $\omega \notin L$ because ω does not have the form $\mathtt{0}^n\mathtt{1}^m$, for non-negative integers n and m, at all. That is, ω includes a copy of $\mathtt{1}$ that is somewhere to the left of a copy of $\mathtt{1}$.

Since the empty string is a *prefix* of ω that has the form $\mathtt{0}^n\mathtt{1}^m$ for non-negative integers n and m (namely, with $n = m = 0$) one can see that ω always has a prefix with this form. Considering the *longest* prefix with this form, one can see that

$$\omega = \mathtt{0}^n\mathtt{1}^m\mathtt{0}\mu \tag{6}$$

for non-negative integers n and m, such that $m \geq 1$, and for some (possibly empty) string $\mu \in \Sigma^\star$.

Unfortunately, it is not clear what happens here, because ***this depends on something about the algorithm that has not been stated yet***: It depends on whether or or not the sweep over the string continues past a copy of "1" that has been found and marked.

If the sweep *does* continue past the copy of "1" that has been found and marked then it will be noticed that ω has the the form shown at line (6), so that it has "10" as a substring, during the first round of the process. Case (a) will have been detected and the string ω will be rejected, as desired, after the first round of the process.

Something quite different happens if if the string to the right of the newly marked "1" is never examined. To continue, we will consider a version of the algorithm where this is the case.

Second Attempt

Suppose, now, that we modify the beginning of the process to make it more precise. In particular, suppose we change the beginning from

> Starting at the left end of the string, look for an unmarked copy of 0 and then mark it. Continuing to move right, look for an unmarked copy of 1 and then mark it.

to

> Starting at the left end of the string, look for an unmarked copy of 0 and then mark it. Continue to move right, looking for an unmarked copy of 1 and then mark it — moving back to the left as soon as a copy of 1 has been found and marked.

Now cases (i), (ii) and (iii) are the same as before. Case (iv) can now be handled as follows.

(iv) Finally, suppose that $\omega \notin L$ because ω does not have the form 0^n1^m, for non-negative integers n and m, at all. That is, ω includes a copy of 1 that is somewhere to the left of a copy of 1. As argued above, ω is as shown at at line (6) for a non-negative integer n, a positive integer m, and a string $\mu \in \Sigma^\star$.

Let i be an integer such that $0 \leq i \leq \min(n, m)$. Then, after i rounds of the process that has been described, the string to be considered has the form

$$\mathtt{X}^i0^{n-i}\mathtt{Y}^i1^{m-i}0\mu. \tag{7}$$

Since sweeps to the right end when an unmarked copy of "1" is found and marked, it can be argued that the beginning of the computation (including the first i rounds) is the same as it would have been if the input string had been 0^n1^m,, so none of cases (a), (b), (c) or (d) have been detected, and the processing should consider.

Now let $i = \min(n, m) + 1$ and consider the ith round of the process.

- If $n < m$ (so that $i = n + 1 \leq m$) then the string to be considered has the form

  $$\mathtt{X}^n\mathtt{Y}^n1^{m-n}0\mu \tag{8}$$

  at the end of the nth round, and the beginning of the ith round.

  Sweeping right over this string it will be seen that the first unmarked symbol is "1" instead of "0". A consideration of case (i), above, establishes that this cannot happen if $\omega \in L$. Indeed, it has been confirmed that either case (a) or case (c) holds — so that ω can be rejected at this point (again, after a finite number of rounds of this process).

- If $n = m$ (so that $i = n + 1 = m + 1$) then the string to be considered has the form

  $$\mathtt{X}^n\mathtt{Y}^n\mathtt{0}\mu \tag{9}$$

  at the end of the n^{th} round, and the beginning of the i^{th} round.

  Since $n = m \geq 1$ the first copy of "0" in this string is preceded by a copy of "Y", which replaced a copy of "0". Thus it is discovered that case (a) applies during the initial search for an unmarked copy of "0", and ω will be rejected during the i^{th} round.
- If $n > m$ (so that $i = m + 1 \leq n$) then then the string to be considered has the form

  $$\mathtt{X}^m\mathtt{0}^{n-m}\mathtt{Y}^m\mathtt{0}\mu \tag{10}$$

  at the end of the m^{th} round, and the beginning of the i^{th} round.

  While an unmarked copy of "0" is found as expected, the subsequent sweep farther to the right, to look for an unmarked copy of "1" will include moving past a sequence of $m \geq 1$ copies of "Y" — which each replaced copies of "1". It will, therefore, have been discovered that ω included a copy of "1" with a copy of "0" to its right — so that case (a) has been confirmed and ω can be rejected, during the i^{th} round, in this case too.

Thus ω is accepted after a finite number of steps whenever $\omega \in L$, and ω is rejected after a finite number of steps whenever $\omega \notin L$. That is, the algorithm (to be used here as a "high level description for a Turing machine") correctly decides the language L.

Implementation-Level Description

An ***implementation description*** is consistent with the high-level description of an algorithm that you have already presented. It includes additional details about

- the way the Turing machine uses its finite control to remember information,
- the way the Turing machine moves its tape head, and
- the way it uses its tape

in order to implement the algorithm. This information is generally given in simple written English (or simple pseudocode).

There will often be *many* choices that can be made about how to implement the high-level algorithm that has been described.

When you establish the correctness of this more detailed version of a solution, you can take advantage of that each part of the "implementation-level description" is part of a solution for a ***simpler problem*** than the one you started with — namely, one *step* in the algorithm (given as a high-level description) that you have already proved to be correct.

Application to the Example Problem

Suppose we include two new symbols, “$\mathtt{X}$” and “$\mathtt{Y}$”, to the Turing machine’s tape alphabet, Γ. Now, the strings show at lines (1) – (10) can all be used as ***strings that will appear somewhere on the Turing machine’s tape while the computation is in progress*** — and not just as graphical aids to help you see how the computation will progress.

Examining these strings again, one can see that there is at least one copy of “$\mathtt{X}$” at the beginning of the string almost immediately after the first symbol in ω is seen (if ω is not immediately accepted because it is the empty string, or rejected because it starts with “1”) and that this is true for the rest of the computation. Furthermore, copies of the symbol “$\mathtt{X}$” only appear at the beginning of the string — they are never to the right of an unmarked copy of either “0” or “1”.

We can take advantage of this by ***slightly modifying the process to be used***: Rather than going all the way back to the beginning of the string, when sweeping back to the left at the end of each round, we can simply go back to the first (rightmost) copy of “$\mathtt{X}$” that is seen. If we proceed with another sweep right after that the computation will continue just as if we had gone all the way back to the left end, and then swept right over all the copies of “$\mathtt{X}$” at the beginning.

This provides a way to end sweeps to the left, in this process, without adding more symbols to the tape alphabet. We do not need to add symbols to end sweeps to the right, either, because these already end if we find a copy of “1” to mark; when no such copy of “1” is found, they should end because a copy of “$\sqcup$” is seen (so that the tape head has moved to the right, past the part of the tape that initially stored the input string).

We can now set the tape alphabet to be

$$\Gamma = \Sigma \cup \{\mathtt{X}, \mathtt{Y}, \sqcup\} = \{\mathtt{0}, \mathtt{1}, \mathtt{X}, \mathtt{Y}, \sqcup\}.$$

As noted above, the first step of the process will include an examination of the first symbol on the tape. The input should be ***accepted*** immediately if this symbol is “$\sqcup$” (because the input is the empty string, which belongs to L) and the input string should be ***rejected*** immediately if this symbol is “1” (because the input string starts with “1” and no such string belongs to L. Processing should continue with the first round in the above process if the first symbol seen is “0”.

It is impossible for the first symbol seen to be either “$\mathtt{X}$” or “$\mathtt{Y}$” because these symbols are not in $\Sigma \cup \{\sqcup\}$. Thus we do not make decisions that constrain what the Turing might do in order to handle these cases.

It is still helpful to add additional detail to the process described above — to make it clearer how cases (a), (b), (c) and (d) can be detected while sweeping over the string that is being processed.[1] It is helpful, for this example, to look closely at the strings shown at lines (1) – (10)

[1] It is possible, though, that you will be able to make sense of the material after the following series of “bullet points” if you skip over them, and then refer back to them when you need to. Note, for now, that they describe which symbols can follow which other symbols, in a sweep to the right over the string.

in order to understand what it means, when one copy of a given symbol (either 0, 1, X, Y or $\sqcup$) appears immediately after a copy of one these symbols:

- Consider what happens if you are sweeping to the right and you saw "0" just before this.
  - This copy of 0" could be followed by either "0" or "1", both in input strings in L, like "0011" and strings that are not in L, like "001".
  - A copy of "0" can be followed by a copy of "Y", both when processing an input string in L (as shown by the string at line (1) when $n \geq 2$ and $1 \leq i \leq n-1$) or when processing an input string that is not in L (as shown by the string at line (5) when $n > m \geq 1$).
  - A copy of "0" can only be followed by "$\sqcup$" if the input string is not in L (either because $\omega = 0^n$ for a positive integer n or ω is as shown at line (6) when $\mu = \lambda$). Thus ω can be ***rejected*** as soon as a copy of "$\sqcup$" is seen immediately after a copy of "0".
- Consider what happens if you are sweeping to the right and you saw "1" just before this.
  - This copy of "1" can only be followed by "0" if "10" is a substring of the input string ω, so that $\omega \notin L$. Thus the input string should be ***rejected*** whenever this happens.
  - A copy of "1" can be followed either by another copy of "1" or by "$\sqcup$", on the tape, for both input strings in L (like "0011") and input strings that are not in L (like "011").
  - A consideration of the strings shown at lines (1) – (10) shows that it is impossible for a copy of "1" to be immediately followed by either a copy of "X" or a copy of "Y" at any point in this computation.
- Consider what happens if you are sweeping to the right and you saw "X" just before this.
  - A copy of "X" can be followed another copy of "X" both when processing an input string $\omega \in L$, and when processing an input string ω such that $\omega \notin L$ — see, for example, the string at line (1) when $n = 3$ and $i = 2$, as well as the string at line (3) when $i = 2$, $n = 3$ and $m = 4$. A consideration of these strings is also sufficient to see that a copy of "X" can be followed by a copy of "0", both when processing a string $\omega \in L$ and when processing a string $\omega \in \Sigma^\star$ such that $\omega \notin L$. It is also sufficient to consider the string at line (2), and the string at line (4), to see that a copy of "X" can be followed by a copy of "Y", both when processing an input string in L, and when processing string $\omega \in \Sigma^\star$ such that $\omega \notin L$.
  - A consideration of the strings at lines (1) – (10) confirms that it is not possible for a copy of "X" to have a copy of either "1" or "$\sqcup$" immediately after it, at the beginning of any sweep to the right over a string that is being processed.
- Consider what happens if you are sweeping to the right and you saw "Y" just before this.

    - A copy of "$\mathtt{Y}$" can be followed by a copy of "$\mathtt{1}$", both when processing a string $\omega \in L$ and when processing a string $\omega \in \Sigma^\star$ such that $\omega \notin L$ — see the string at line (1) when $n > i \geq 1$ and at line (3) when $m > n \geq i \geq 1$. A copy of "$\mathtt{Y}$" can also be followed by another copy of "$\mathtt{Y}$" when processing both kinds of strings — see the strings at lines (1) and (3) once again, when $n > i \geq 2$ and when $m > n > i \geq 2$, respectively. To see that a copy of "$\mathtt{Y}$" can be followed by "$\sqcup$" when processing both kinds of input strings, see the strings at line (2) when $n \geq 1$ and at line (5) when $n > m \geq 1$.

- Finally consider what happens if you are sweeping to the right and you saw "$\sqcup$" just before this.
    - This case does not arise, at all, because you start sweep back to the left (if you do not reject the string) when you see a "$\sqcup$": You never see the symbol that follows it, at all. With that noted, this symbol must always be another copy of "$\sqcup$".

With all that noted, consider the more detailed version of the algorithm — which makes use of the tape alphabet, Γ, along with some of the things that have now been discovered — which is shown in Figure 1 on page 10, and Figure 2 on page 11.

To see that this really is a refinement of the algorithm given near the beginning of this document, note that if "$\sqcup$" is the first symbol seen, then $\omega = \lambda$ and condition (d) (as given in the algorithm has been detected, right away — so that both the algorithm at the beginning and more detailed, version, given now, agree that the input string should be accepted (as shown, in Figure 1, at lines #1 – #2).

On the other hand, if the first symbol visible is "$\mathtt{1}$", then there is either a copy of "$\mathtt{0}$" somewhere to the right of this symbol, or there is not. That is, either case (a) or case (b) (given in the original version of the algorithm will arise) and the input string should be rejected, according this algorithm. Thus the steps in Figure 1 at lines #3 – #4 are also implementing part of the original algorithm (with more detail).

Since the Turing machine's tape can only include symbols in $\Sigma \cup \{\sqcup\}$ when the computation begins, the only remaining case is that "$\mathtt{0}$" is the first symbol that is seen. According to the original version, this copy of "$\mathtt{0}$" should be marked, and a sweep to the right, to find an unmarked copy of "$\mathtt{1}$" that should also be marked, should begin. Examining the more detailed version algorithm, one can see that this is implemented by including the "`else`" group of statements at lines #5 – #25, which is entered if the first symbol seen is "$\mathtt{0}$". In this case the copy of "$\mathtt{0}$" when the step at line #6 is executed, and the sweep farther to the right, in search of copy of "$\mathtt{1}$" to mark, begins with the step at line #7.

Before continuing with the details of the sweep to the right, note that the `while` loop at lines #5 – #25 is also being used to implement later rounds of the process given in the original algorithm, besides the first. It is also true, in these later stages, that the string should be rejected if the first symbol seen (now, immediately after the rightmost copy of "$\mathtt{X}$") is either "$\mathtt{1}$"

```
 1. if (symbol visible is “⊔”) {
 2.   accept
 3. } else if (symbol visible is “1”) {
 4.   reject
    } else {
 5.   while (true) {
 6.     if (symbol visible is “0”) {
 7.       Replace the symbol visible with “X”, moving right.
 8.       while (symbol visible is “0”) {
 9.         Leave the symbol unchanged, moving right.
          }
10.       while (symbol visible is “Y”) {
11.         Leave the symbol unchanged, moving right.
          }
12.       if (the symbol visible is “1”) {
13.         Replace the symbol visible with “Y”, moving left
14.         while (the symbol visible is not “X”) {
15.           Leave the symbol unchanged, moving left
            }
16.         Leave the symbol (“X”) unchanged, moving right.
          } else {
17.         reject
          }
```

Figure 1: Beginning of Implementation-Level Description of Turing Machine

or “⊔” — see the above consideration of the symbols that can appear immediately after a copy of “X”. This happens, when the more detailed version of the algorithm is executed, because the test at line #6 is checked and failed. Then the test at line #18 is checked and failed as well. Thus the step at line #25 is reached, and the string is rejected as required.

Let us next consider a sweep to the right, beginning after a copy of “0” has been found and marked. As noted above — regardless of whether this is the first round or a later one — this happens (during the execution of the more detailed version of the algorithm) because the test at line #6 has been checked and passed, and the step at line #7 has been executed in order to mark the copy of “0” that has been found.

Examining the string at line (1) above, one can see that we should expect to see a sequence of zero or more other copies of “0”, followed by a sequence of zero or more copies of “Y”, before

```
18.      } else if (the symbol visible is “Y”) {
19.        Leave the symbol unchanged, moving right.
20.        while (the symbol visible is “Y”) {
21.          Leave the symbol unchanged, moving right.
           }
22.        if (the symbol visible is “⊔”) {
23.          accept
           } else {
24.          reject
           }
         } else {
25.        reject
         }
       }
     }
```

Figure 2: End of Implementation-Level Description of Turing Machine

a copy of “1” (which should be marked) is reached. Looking at the more detailed code one can see that a sequence of zero or more other copies of “0” are swept over, as needed, when the steps at lines #8 – #9 are reached and executed. A sequence of zero more copies of “Y” are then swept over, as needed, when the steps at lines #10 – #11 are reached and executed. The required copy of “1” is found and marked when the steps at lines #12 – #13 are executed. If we find any symbol besides “1” at this point then the string could not have had the form shown at line (1) (with $i < n$, so that a copy of “0” was found and marked) when this round began, so the string should be rejected. This happens because the test at line #12 is reached and failed — so that the step at line #17 is reached and executed.

Everything that can happen during the *first* round of the process, given by the original algorithm, has now been discussed, along with all the later rounds that start with a copy of “0” being found and marked. It remains only to consider a final round, when a copy of “0” is *not* found right away. As the strings at lines (1) – (10) should suggest, this can only happen, when the input should be accepted, because a copy of “Y” has been found immediately after the rightmost “X”, with a sequence of zero or more other copies of “Y” following, before the end of the string is reached (and a copy of “⊔” is discovered). Examining the steps at lines #18 – #24, one can confirm that this part of the more detailed algorithm is carrying out (exactly) the process that is supposed to.

Thus this more detailed version of the algorithm correctly decides the desired language because ***it is really doing the same thing as the original algorithm does*** — and we have already argued that the original algorithm is correct.

Formal Description

A ***formal description*** of a Turing machine is the same as the "formal description" of a Turing machine introduced in the lecture notes — that is, a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

where Σ is the input alphabet and Q, Γ, δ, q_0, q_{accept} and q_{reject} are all as described in the lecture notes.

While providing a formal description of a Turing machine would probably be an overwhelming task, if this was the first thing you tried to do, it is (in principle) a manageable task if you have already produced a reasonably detailed "implementation-level description": Many decisions have already been made and your Turing machine can possibly be thought of a collection of "components" or "sub-machines" that each implement a step in the implementation-level description of the algorithm. When you try to argue that your Turing machine is "correct" it will, generally, be sufficient to argue that each "sub-machine" correctly implements the step in the implementation-level description that it is supposed to.

Application to the Example Problem

The above information can now be used to to produce a Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

such that

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\},$$

$\Sigma = \{0, 1\}$ (as given in the description of the problem to be solved), $\Gamma = \{0, 1, \mathtt{X}, \mathtt{Y}, \sqcup\}$ (as decided when "implementation-level details" were added) and a picture of the Turing machine is shown in Figure 3 on page 13.

In order to make the picture easer to read, transitions to the rejecting state are not shown. However, you may complete the picture by adding a transition consistent with the statement "$\delta(q, \sigma) = (q_{\text{reject}}, \sigma, \mathtt{R})$" whenever there should be a transition from a state $q \in Q$ and symbol $\sigma \in \Gamma$ to the rejecting state. To keep the picture simple, the accept state is labelled "q_A" instead of "q_{accept}".

When comparing the Implementation-Level description of the Turing machine (in Figures 1 and 2) and the picture in Figure 3, one should note the following.

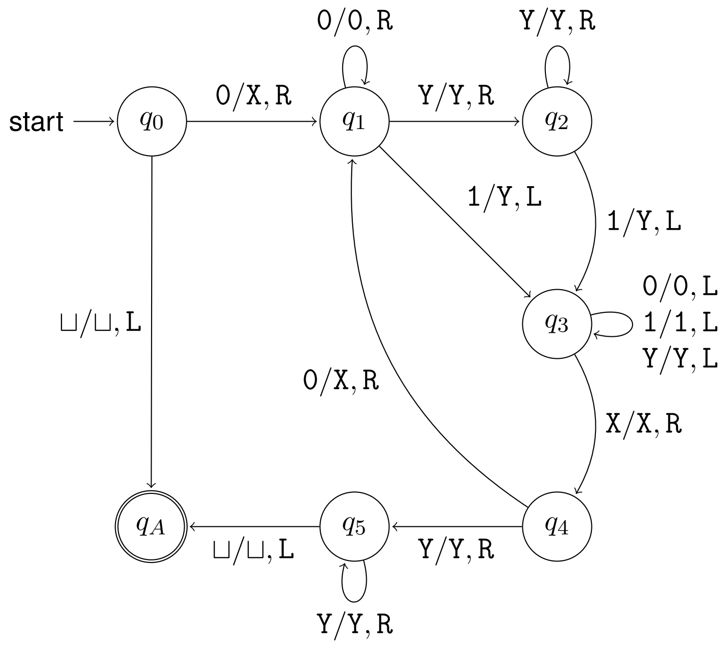

Figure 3: Turing Machine for the Language L

- The steps at lines #1 and #2 are implemented using the transition from the start state, q_0, to the accepting state, q_{accept} (shown in the picture as "q_A").

- The steps at lines #3 and #4 are implemented using a transition from q_0 to the rejecting state, q_{reject}, which is not shown in the picture.

- As noted above, one the "`else`" part of the program (corresponding to code after the "`else`" statement immediately above line #5) is only executed if the first symbol seen is "0", so that this is implemented by the transition out of the start state for the symbol "0", to state q_1.

  Note that the test at line #6 must always be passed the first time it is checked (if it is reached, at all), so that this transition also implements the first execution of this test.

  Since this transition includes the replacement of "0" with "X", it also implements the first execution of the step at line $\#7$.

- The steps at lines #8 and #9 are implemented using the loop from state q_1 to itself.

- Consider the `while` loop including the steps at lines #10 – #11. Executions of the loop test where the body of the loop is executed at least once (because at least one copy of “Y” is seen) are implemented using the transition from q_1 to q_2 and the loop from q_2 to itself.

Executions where the loop body is not executed, because no copy of “Y” is seen, but a copy of “1” is seen instead, are implemented by the transition from q_1, for the symbol “1”, to state q_3. Examining the code, one sees that if the loop body is not executed, and neither a copy of “Y” nor “1” has been seen, the step at line 17 should be reached and the string should be rejected; this is implemented using transitions from q_1 to the rejecting state, which are not shown.

Similarly, if the loop body is executed at least once and the loop test fails because a copy of “1” has been seen instead of a copy of “Y”, then the test at line 12 should be reached and passed. This is implemented using the transition, for the symbol “1” from q_2 to q_3. Once again, one can examine the code to see that if the final execution of the test for the loop at line #5 fails because “Y” has not been seen, but a copy of “1” has not been seen either, the test at line #17 should be reached and the string should be rejected. This is implemented using transition from q_2 to the rejecting state, which are not shown.

- Note that the transitions out of q_1 and q_2, for the symbol “1”, replace this symbol “Y” and move the tape head left. They therefore implement an execution of the step at line #13, along with the successful execution of the test at line #12.

- The `while` loop at lines #14 and #15 is implemented using the transition that loops from q_3 to itself. The execution of the loop ends when the symbol “X” is seen, and that is implemented using the transition from q_3 to q_4. This transition also implements the step at line #16, which immediately follows the loop.

- Transitions out of q_4 implement the end of all executions of the body of the loop at lines #5 – #25 that started with a copy of “0” being seen and replaced, as well as steps immediately after that. In particular, if another copy of “0” is seen, so that the next execution of the test at line #6 should be passed, then the transition from q_4 to q_1, for “0”, is followed. Once again, the execution of the loop body continues using transitions out of q_1 as explained above.

  It is now possible that a copy of “0” is *not* seen but a copy of “Y” is seen instead. Examining the code, one sees that the test at line #6 should fail, and the test at line #18 should be reached and passed. This is implemented using the transition from q_4 to q_5.

  If neither “0” nor “Y” at the beginning of the execution of the body of the loop then the step at line #25 should be reached and executed. This is implementing using transitions from q_4 to the rejecting state, which are not shown in the picture.

- It remains only to consider executions of the loop body that begin with “Y” seen instead of “0”, so that the test at line #18 is reached and passed. As noted above, the successful execution of this test is implemented using the transition from q_4 to q_5. This transition also implements the step at line #19. The `while` loop at lines #20 – #21 is implemented using the transition from q_5 to itself. A successful execution of the test at line #22 (because "$\sqcup$" has been seen) and the execution of the step at of the step at line #23, are implemented using the transition from q_5, for "$\sqcup$", to the accepting state. An unsuccessful execution of the test at line #22 (because a "$\sqcup$" has not been seen, now) and the execution of the step at line #24 are implemented using transitions from q_5 to the rejecting state which are not shown.

We have now argued that the Turing machine shown in Figure 3 correctly implements the "implementation-level" description given in Figures 1 and 2. Since the "implementation-level" description of the algorithm has been argued to be correct, we may now conclude that this Turing machine decides the language L, as desired.

Exercises: Applying This in a Different Way

1. Since states q_0 and q_4 are implementing many of the same steps, one might consider a Turing machine in which a single state replaces both of them — obtaining a Turing machine like the one shown in Figure 4 on page 16.

   Does this Turing machine *also* implement the implementation-level version shown in Figures 1 and 2?

   - If you think that the answer is ***yes*** then explain how you would modify the explanation why the *original* Turing machine correctly implements this, in order to give explanation why the *new* Turing machine correctly implements this too.
   - If you think that the answer is ***no*** then prove that the new Turing machine does not decide the language L.

2. Now, if you compare this to the Turing machine to the Turing machine given, for this language, in the notes for Lecture #9, you might be surprised to discover ***these Turing machines are not the same***. There is often more than one way to solve a problem.

   Two more algorithms (which could be thought of as "high-level" descriptions of Turing machines), that correctly decide the language L, are as follows.

   (a) Accept the input string, ω, if it is the empty string, λ. If the input string is not the empty string and does not start with a copy of "0" then reject it. Otherwise consider all the symbols in the input string to be "unmarked" and repeatedly do the following:

      If none of the symbols are marked, yet, then mark the leftmost symbol (which is a copy of "0"). Sweep to the right until an unmarked copy of "1" is

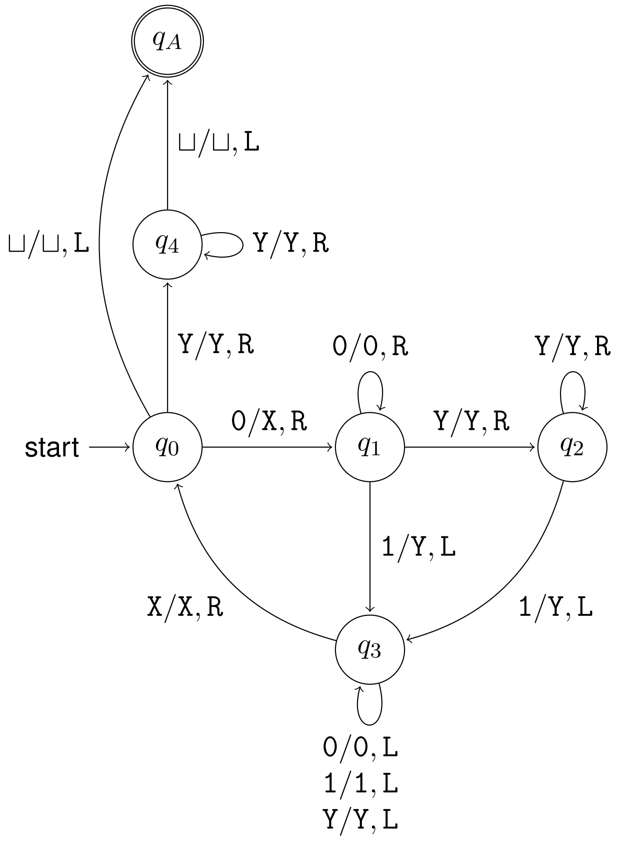

Figure 4: Another Turing Machine: Does This Also Decide L?

found, and mark that symbol too. Then sweep back to the left.

On the other hand, if at least one symbol has already been marked, and the symbol immediately to the right of the rightmost ***marked*** copy of "0" is an unmarked copy of "0", then mark this symbol. Sweep to the right until you find an unmarked copy of "1" and then mark that symbol too. Then sweep back to the left.

In both of the above cases, if you find an unmarked copy of "0" and mark it, but you cannot find an unmarked copy of "1" to its right, then reject the input string.

Finally, if the symbol to the right of the rightmost marked copy of "0" (at the beginning of one of the rounds) *is not* an unmarked copy of "0" then accept the input string if and only every symbol in the current string, to the right of the rightmost marked copy of "0", is a marked copy of "1".

(b) Accept the input, string, ω, if it is the empty string.

Otherwise, reject ω if it *does not* begin "0" and end with "1". That is, reject ω if ω is nonempty and does not have the form

$$0\widehat{\omega}1 \tag{11}$$

for some shorter string $\widehat{\omega} \in \Sigma^\star$.

Finally, if ω *does* have form shown at line (11) then recursively execute this algorithm with input $\widehat{\omega}$ — accepting ω if $\widehat{\omega}$ is accepted, and rejecting ω if $\widehat{\omega}$ is rejected.

For each of the above, explain why the (high-level) algorithm correctly decides the language L. Then make decisions about an implementation and add details, as needed, to obtain an implementation-level version which you can also argue to be correct. Finally, produce a formal description of a corresponding Turing machine, which you could prove to decide the language L.

It is possible than you when you do this for algorithm (a), you will end up with the Turing machine shown in the lecture notes. For algorithm (b), if you erase a symbol by replacing it with "$\sqcup$" (rather than a new tape symbol) then it is possible that $\widehat{\omega}$ will be the non-blank string on the tape when you would want to apply the algorithm recursively (but there will also be at least one copy of "$\sqcup$" to the left of $\widehat{\omega}$ — so that it will be easy to avoid trying to move past the left end of the tape).

References

[1] Michael Sipser. *Introduction to the Theory of Computation*. CENGAGE Learning, third edition, 2013.