Lecture #8: Proving Undecidability — Part One
Proofs of Claims about Universal Turing Machines

1 Introduction

In the preparatory material, for Lecture #8, the following languages — over the alphabet

$$\Sigma_{\mathsf{TM}} = \{\mathtt{s}, \mathtt{q}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}, \mathtt{L}, \mathtt{R}, \mathtt{Y}, \mathtt{N}, \mathtt{(}, \mathtt{)}, \mathtt{,}, \mathtt{;}\} \tag{1}$$

— were introduced:

- L_{TM} is the language of encodings of all (standard, one-tape) Turing machines

  $$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{accept}}, q_{\mathsf{reject}}) \tag{2}$$

  such that q_0, q_{accept} and q_{reject} are distinct states in Q — where it is assumed, here, that

  $$Q = \{q_0, q_1, q_2, \ldots, q_k, q_{\mathsf{accept}}, q_{\mathsf{reject}}\} \tag{3}$$

  for a non-negative integer k,

  $$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_\ell\} \tag{4}$$

  for a positive integer ℓ, and

  $$\Gamma = \{\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_{m-1}\} \tag{5}$$

  for some positive integer m such that $m > \ell$. Here, the "␣" symbol, ␣, is being given the additional name σ_0. Since $m > \ell$, so that $m - 1 \geq \ell$ it follows from the definitions at lines (4) and (5) that $\sqcup \notin \Sigma$ and $\sigma \cup \{\sqcup\} \subseteq \Gamma$, as expected.

- $L_{\mathsf{TM+I}}$ is the language of encodings of ordered pairs (M, ω), where M is a (standard, one-tape) Turing machine, as described above, and $\omega \in \Sigma^\star$ where Σ is the input alphabet for M — so that ω can be thought of as an "input string" for M.

The encodings and precise definitions of these languages are nonstandard: There really *is* no "standard" way to encode these in the computing literature, and every reference that you might check could these define these differently, if details about encodings and the definitions of the languages are given at all. However, the languages of encodings of Turing machines, and of encodings of Turing strings and input strings for them, can be shown to be decidable — no matter what *reasonable* encoding for these is defined. An encoding has been introduced in the preparatory material for Lecture #8, and proofs for the claims "L_{TM} is decidable" and "$L_{\mathsf{TM+I}}$ is decidable" are being sketched here — to make the lecture material reasonably self-contained and complete.

This really is "supplemental" material: Students in this course do not need to read or understand this material in order to do well in this course. Instead, it is sufficient (and *important*) to understand the claims that "L_{TM} is decidable" and "$L_{\mathsf{TM+I}}$ is decidable" and to know that these claims are correct.

2 Proving That L_{TM} is Decidable

2.1 Review of Encodings

Recall that ***states***, ***tape symbols***, ***transitions***, ***transition functions*** and ***Turing machines*** are all encoded as strings in $\Sigma^{\star}_{\mathsf{TM}}$:

- If the set Q of states of a Turing machine M is as shown at line (3) then, for $0 \leq i \leq k$, the state q_i is encoded by the string in $\Sigma^{\star}_{\mathsf{TM}}$ that begins with "q" and continues (and ends) with the unpadded decimal representation of i. The accept state, q_{accept}, is encoded by the string "qY", and the reject state, q_{reject}, is encoded by the string "qN".

- If the tape alphabet Γ for a Turing machine M is as shown at line (5) then, for $0 \leq i \leq m-1$, the symbol σ_i is encoded by the string in $\Sigma^{\star}_{\mathsf{TM}}$ that begins with "s" and continues (and ends) with the unpadded decimal representation of i.

- If the set Q of states and the tape alphabet Γ of M are as shown at lines (3) and (5), above, then the transition function δ of M is defined at the set of ordered pairs (q, α) where $q \in Q \setminus \{q_{\mathsf{accept}}, q_{\mathsf{reject}}\}$ — so that $q = q_i$ for some integer i such that $0 \leq i \leq k$ — and where $\alpha \in \Gamma$, so that $\alpha = \sigma_j$ for some integer j such that $0 \leq j \leq m-1$.

  Now, for all $q \in Q \setminus \{q_{\mathsf{accept}}, q_{\mathsf{reject}}\}$ and $\alpha \in \Gamma$, $\delta(q, \alpha) \in Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}\}$ — so that $\delta(q, \alpha) = (r, \beta, d)$ for some state r, tape symbol β, and direction of motion $d \in \{\mathtt{L}, \mathtt{R}\}$. A transition

$$\delta(q, \alpha) = (r, \beta, d)$$

is encoded by a string

$$(\eta_1\,,\eta_2\,,\eta_3\,,\eta_4\,,\eta_5) \tag{6}$$

where η_1 is the encoding of the state q, η_2 is the encoding of the tape symbol α, η_3 is the encoding of the state r, η_4 is the encoding of the tape symbol β, and where η_5 is the encoding of the *string* "`L`" in $\Sigma^\star_{\mathsf{TM}}$ if $d = \mathtt{L}$ and η_5 is the *string* "`R`" in $\Sigma^\star_{\mathsf{TM}}$ if $d = \mathtt{R}$.

- For Q and Γ as at line (3) and (5), above, the transition function function δ, is defined on exactly $h = (k+1) \times m$ ordered pairs (q, α) such that $q \in Q \setminus \{q_{\mathsf{accept}}, q_{\mathsf{reject}}\}$ and $\alpha \in \Gamma$. The transition function, δ, is encoded by a comma-separated listing of the encodings of the included transitions, enclosed by brackets:

  $$(\nu_0\,,\nu_1\,,\nu_2\,,\,\ldots\,,\nu_{h-1}) \tag{7}$$

  Furthermore the encodings of the transitions are ordered by non-decreasing index of input state and, for the transitions with the same input state, by increasing index of symbol in the tape alphabet. Thus, for $0 \leq i \leq h-1$, the string ν_i is the encoding of a transition with input state q_r and tape symbol σ_s, where

  $$r = \left\lfloor \tfrac{i}{m} \right\rfloor \qquad\qquad s = i \bmod m.$$

- A Turing machine M, that is as at line (2), is encoded by a string

  $$(\mu_1\,,\mu_2\,,\mu_3\,,\mu_4) \tag{8}$$

  where the strings $\mu_1, \mu_2, \mu_3, \mu_4 \in \Sigma^\star_{\mathsf{TM}}$ are as follows:

  - μ_1 is the ***unpadded decimal representation*** of the number $k = |Q| - 3$.
  - μ_2 is the ***unpadded decimal representation*** of the number $\ell = |\Sigma|$.
  - μ_3 is the ***unpadded decimal representation*** of the number $m = |\Gamma|$.
  - μ_4 is the ***encoding of the transition function***, δ, for M — as described above.

2.2 A High-Level Algorithm

An algorithm that can be used to decide whether a string $\zeta \in \Sigma^\star_{\mathsf{TM}}$ is an encoding of a Turing machine (that is, to decide whether $\zeta \in L_{\mathsf{TM}}$), is given in Figure 1 on page 4 and in Figure 2 on page 5.

To see that if $\zeta \in L_{\mathsf{TM}}$ then ζ must have the form shown at line (8), that is, ζ must begin with "`(`", end with "`)`", and include at least three copies of "`,`". Furthermore, the substring μ_1 of ζ begin the initial copy of "`(`" and the leftmost copy of "`,`" must be the unpadded decimal representation of a non-negative k such that the set Q of states, of the encoded Turing machine, has size $k+3$.

On input $\zeta \in \Sigma^\star_{\mathsf{TM}}$ {

1. `if` (ζ begins with "`(`", ends with "`)`", and includes at least three copies of "`,`") {

   Let $\mu_1, \mu_2, \mu_3, \mu_4 \in \Sigma^\star_{\mathsf{TM}}$ *such that*

   $$\zeta = (\mu_1\,,\mu_2\,,\mu_3\,,\mu_4)$$

   where μ_1*,* μ_2 *and* μ_3 *do not include any copies of "*`,`*"*

2. `if` (μ_1 is the unpadded binary representation of a non-negative integer) {

   Let k *be the integer encoded by* μ_1

3. `if` (μ_2 is the unpadded binary representation of a positive integer) {

   Let ℓ *be the integer encoded by* μ_2

4. `if` μ_3 is the unpadded binary representation of a positive integer) {

   Let m *be the integer encoded by* μ_3

5. `if` $(\ell < m)$ {

6. `if` (μ_4 is encoding of a transition function for some Turing with $k+3$ states and a tape alphabet with size m) {

7. `accept` ζ

   } `else` {

8. `reject` ζ

   }

Figure 1: Beginning of Algorithm to Decide Membership in L_{TM}

The substring μ_2 of ζ, between the first and second copies of "`,`" in ζ, must be the unpadded decimal representation of the size ℓ of the encoded Turing machine's input alphabet. The substring μ_3 of ζ, between the second and third copies of "`,`" in ζ, must be the size m of the encoded Turing machine's tape alphabet. Since the tape alphabet must include all the symbols in the input alphabet as well as the "blank" symbol, $\ell < m$. Finally, the substring μ_4 of ζ, between the third copy of "`,`" and the right bracket "`)`" at the end of ζ must be the encoding of the transition function of the encoded Turing machine.

One can see, by an examination of the tests at lines 1, 2, 3, 4, 5 and 6 that all of these tests must be passed in the input string, ζ, is an encoding of a Turing machine (that is, if $\zeta \in L_{\mathsf{TM}}$) — and, examining the steps at lines 8, 9, 10, 11, 12 and 13, one can see that ζ is rejected *unless* these tests all pass. That is, this algorithm rejects every string in $\Sigma^\star_{\mathsf{TM}}$ that is not in L_{TM}. On

```
            } else {
 9.           reject ζ
            }
          } else {
10.         reject ζ
          }
        } else {
11.       reject ζ
        }
      } else {
12.     reject ζ
      }
    } else {
13.   reject ζ
    }
```

Figure 2: End of Algorithm to Decide Membership in L_{TM}

the other hand, if $\zeta \in L_{\mathsf{TM}}$ then the tests at lines 1 2, 3, 4, 5 and 6 *are* all passed — and ζ is accepted, because the step at line 7. It follows that any Turing machine that implements this algorithm decides the language L_{TM}.

The test at line 6 of this algorithm is quite complicated. An expansion of this step is given in Figure 3 on page 6. Suppose that this step is reached, so that μ_1, μ_2 and μ_3 are the unpadded decimal representations of integers k, ℓ and m respectively — where $k \geq 0$ and the transition alphabet should correspond to a set of states as at equation (3), and where $m > \ell \geq 1$ and the tape alphabet (for a Turing machine being encoded) should be as shown at line (5). In order for μ_4 to be an encoding of a transition function, this string must be as shown at line (7), $h = (k+1) \times m$ and, for $0 \leq i \leq h-1$, the string ν_i must be the encoding of a transition of a state q_r and tape symbol σ_s where $r = \lfloor i/m \rfloor$ and $s = i \bmod m$, as noted above.

Now, the algorithm shown in Figure 3 begins by checking whether μ_4 has the form shown at line (7) — returning `false` (so that the test at line 6 of the algorithm in Figures 1 and 2 will fail and ζ will be rejected) if this is not the case. Otherwise the loop at lines 6(e)–6(l) is reached and executed.

It is easily proved, by induction on j, that if j is a non-negative integer such the body of this loop is executed at least $j+1$ times then, at the beginning of the $j+1^{\text{st}}$ execution of the body

6(a) `if` (μ_4 is a sequence of one or more transitions for a Turing machine with $k+3$ states and a tape alphabet with size m) {

Suppose

$$\mu_4 = (\nu_0, \nu_1, \nu_2, \ldots, \nu_{h-1})$$

for a positive integer h*, where* $\eta_1, \eta_2, \ldots, \eta_h$ *are encodings of transitions*

6(b) `integer` i := 0

6(c) `integer` r := 0

6(d) `integer` s := 0

6(e) `while` $((i < h)$ `and` $(r \leq k))$ {

6(f) `if` (ν_i is a transition for state q_r and tape symbol σ_s) {

6(g) $s := s + 1$

6(h) `if` $(s == m)$ {

6(i) $r := r + 1$

6(j) $s := 0$

}

6(k) $i := i + 1$

} `else` {

6(l) `return false`

}

}

6(m) `if` $((i == h)$ `and` $(r > k))$ {

6(n) `return true`

} `else` {

6(o) `return false`

}

} `else` {

6(p) `return false`

}

Figure 3: Expansion of Test at Line 6 of the Algorithm in Figures 1 and 2

of this loop, the following properties are satisfied:

- The variable i, r and s have values j, $\lfloor j/m \rfloor$, and $j \bmod m$ respectively. Furthermore, $0 \leq i \leq (k+1) \times m - 1$, $0 \leq r \leq k$, and $0 \leq s \leq m - 1$.

- For every integer t such that $0 \leq t \leq j-1$, the string ν_t is the encoding for a state q_a and tape symbol σ_b, for $a = \lfloor t/m \rfloor$ and $b = t \bmod m$.

Now, if μ_4 is the encoding of a transition function for a Turing machine whose set of states has size $k+3$ and whose tape alphabet has size m — so that $\zeta \in L_{\mathsf{TM}}$ — then the body of this loop is executed $h = (k+1) \times m$ times, without the step at line 6(l) ever being reached and executed. The loop test, at line 6(e) is reached, and executed, for an $h+1^{\text{st}}$ time — at which point $i = h$ and $r = \lfloor h/m \rfloor = k+1$. This execution of the loop test fails, so that the test at line 6(m) is reached and executed. Since $i = h$ and $r = k+1$ this test is also passed, so that the step at line 6(h) is reached and executed — with `true` returned, so that the test at line 6 in the algorithm in Figures 1 and 2 passes, the step at line 7 in this algorithm is executed, and ζ is accepted, as required.

Suppose, instead, that μ_4 is not the encoding of a transition function whose set of states has size $k+3$ and whose tape alphabet has size m — so that $\zeta \notin L_{\mathsf{TM}}$. Then one of the following cases must arise:

(i) μ_4 does not have the form shown at line (7).

(ii) μ_4 has the form shown at line (7), but there exists an integer i such that $0 \leq i \leq \min(h, (k+1) \times m) - 1$ such that ν_i is not an encoding of a transition for a state q_r and tape symbol σ_s, where $r = \lfloor i/m \rfloor$ and $s = i \bmod m$.

(iii) $h < (k+1) \times m$ and ν_i is an encoding of a transition for a state q_r and tape symbol σ_s, for $r = \lfloor i/m \rfloor$ and $s = i \bmod m$, for every integer i such that $0 \leq i \leq h-1$.

(iv) $h > (k+1) \times m$ and ν_i is an encoding of a transition for a for a state q_r and tape symbol σ_s, for $r = \lfloor i/m \rfloor$ and $s = i \bmod m$, for every integer i such that $0 \leq i \leq (k+1) \times m - 1$.

It has already been noted, above, that the algorithm in Figure 3 returns `false`, so that ζ is rejected, in case (i).

Suppose case (ii) arises — and let j be the *smallest* integer such that $0 \leq j \leq \min(h, (k+1) \times m) - 1$, and ν_j is not an encoding of a transition for a state q_r and tape symbol σ_s, where $r = \lfloor j/m \rfloor$ and $s = j \bmod m$. Then the body of the loop is executed $j+1$ times. During the $j+1^{\text{st}}$ execution of this loop body $i = j$, the test at line 6(f) is reached and fails, and the step at line 6(l) is reached and executed — so that `false` is returned, the test at line 6 in the algorithm in Figures 1 and 2 fails, and ζ is rejected, once again.

In case (iii) the loop body is executed h times without the step at line 6(l) ever being reached — so that the loop test at line 6(e) is reached, and executed, for an $h+1^{\text{st}}$ time. The loop test fails, because $i = h$ at this point, so that the test at line 6(m) is reached and executed. Now, since $h < (k+1) \times m$, $r = \lfloor i/m \rfloor = \lfloor h/m \rfloor < k+1$. That is, $r \leq k$, so that the test at line 6(m) fails. The test at line 6(o) is reached and executed, causing `false` to be returned —

as needed to cause the test at line 6 in the algorithm in Figures 1 and 2 to fail, and for ζ to be rejected — in this case as well.

Finally, in case (iv), the loop body is executed $(k+1) \times m$ times without the step at line 6(l) ever being reached — so that the loop test at line 6(e) is reached, and executed, for a $(k+1) \times m+1^{\text{st}}$ time. At this point $i = (k+1) \times m < h$, but $r = \lfloor i/m \rfloor = k+1 > k$, so that the test at line 6(e) is failed and the test at line 6(m) is reached and executed. Since $i < h$ this test fails, so the step at line 6(o) is reached and executed. This causes `false` to be returned — as needed to cause the test at line 6 in the algorithm in Figures 1 and 2 to fail, and for ζ to be rejected — in this final case too.

Thus the algorithm in Figure 3 correctly implements the test at line 6 in the algorithm in Figures 1 and 2. Since this algorithm correctly decides membership in L_{TM}, it is sufficient to show that this algorithm can be implemented using a Turing machine in order to prove that the language L_{TM} is decidable.

2.3 Implementation-Level Details

In order to show that the algorithm given in Figures 1, 2 and 3 can be implemented using a (standard) Turing machine, let us first consider an 10-tape Turing machine

$$M_{\mathsf{TM}} = (\Sigma_{\mathsf{TM}}, \Gamma_{\mathsf{TM}}, \delta_{\mathsf{TM}}, q_{0,\mathsf{TM}}, q_{\mathsf{accept},\mathsf{TM}}, q_{\mathsf{reject},\mathsf{TM}})$$

that implements this algorithm. The tape alphabet of this Turing machine is

$$\Gamma_{\mathsf{TM}} = \Sigma_{\mathsf{TM}} \cup \{\sqcup\}$$

and the tapes of this Turing machine will be used as follows.

- Tape #1 will store the input string, $\zeta \in \Sigma_{\mathsf{TM}}^{\star}$, throughout an execution of this Turing machine.
- If it has been confirmed that ζ is as shown at line (8), above, then Tape #2 stores the string μ_1 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost cell of the tape can be detected. Recall that if $\zeta \in L_{\mathsf{TM}}$ then μ_1 is the unpadded decimal representation of the non-negative integer k such that the encoded Turing machine has $k+3$ states.
- If it has been confirmed that ζ is as shown at line (8), above, then Tape #3 stores the string μ_2 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost cell of the tape can be detected. Recall that if $\zeta \in L_{\mathsf{TM}}$ then μ_2 is the unpadded decimal representation of the size, ℓ of the encoded Turing machine's input alphabet Σ.

- If it has been confirmed that ζ is as shown at line (8), above, then Tape #4 stores the string μ_3 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost cell of the tape can be detected. Recall that if $\zeta \in L_{\mathsf{TM}}$ then μ_3 is the unpadded decimal representation of the size, m of the encoded Turing machine's tape alphabet, Γ.

- If it has been confirmed that ζ is as shown at line (8), above, then Tape #5 stores the string μ_4 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost cell of the tape can be detected. Recall that if $\zeta \in L_{\mathsf{TM}}$ then μ_4 is the encoding of the encoded Turing machine's transition function, δ.

- Tape #6 stores the unpadded decimal representation of the integer i used in the algorithm shown in Figure 3 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost cell of the tape can be detected.

- Tape #7 stores the unpadded decimal representation of the integer r used in the algorithm shown in Figure 3 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost call of the tape can be detected.

- Tape #8 stores the unpadded decimal representation of the integer s used in the algorithm shown in Figure 3 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost cell of the tape can be detected.

- Tapes #9 and #10 are used for the temporary storage of information at various points in the computation, as described below.

Now each of the steps in the algorithm in Figures 1, 2 and 3 can be implemented as follows.

- The step at line 1 can be implemented by ***rejecting*** ζ if the symbol visible on Tape #1 is not "(" — that if, if ζ does not begin with "(" — and moving right on Tapes #1 and #2 — without moving the tape heads for any other tapes or changing the symbol on any tape. Continuing to sweep right on Tapes #1 and #2 the symbols on Tape #1 should be copied onto Tape #2 until either "," or "$\sqcup$" is visible on this first tape.

  Now, if "$\sqcup$" is visible on this first tape (so that ζ does not include any copies of ",") then ζ should be ***rejected*** (and the computation should end). Otherwise the tape heads for Tapes #1 and #3 should be moved to the right — without moving the tape heads on other tapes, writing the copy of "," seen on Tape #1 onto any tape or, indeed, changing the symbol visible on any tapes. Continuing to sweep right on Tapes #1 and #3 the symbols after the leftmost "," on Tape #1 should be copied onto Tape #3 until either "," or "$\sqcup$" is visible on the first tape.

  If "$\sqcup$" is visible on this first tape (so that ζ does not include a second copy of ",") then ζ should be ***rejected*** (and the computation should end). Otherwise the tape heads for Tapes #1 and #4 should be moved to the right — without moving the tape heads on

other tapes, writing the second copy of "," seen on Tape #1 onto any tape or, indeed, changing the symbol visible on any tapes. Continuing to sweep right on Tapes #1 and #4 the symbols after the second "," on Tape #1 should be copied onto Tape #4 until either "," or "␣" is visible on the first tape.

If "␣" is visible on this first tape (so that ζ does not a third copy of ",") then ζ should be ***rejected*** (and the computation should end). Otherwise the tape heads for Tapes #1 and #5 should be moved to the right — without moving the tape heads on other tapes, writing the second copy of "," seen on Tape #1 onto any tape or, indeed, changing the symbol visible on any tapes. Continuing to sweep right on Tapes #1 and #5 the symbols after the third "," on Tape #1 should be copied onto Tape #5 until "␣" is seen on Tape #1.

At this point, the tape heads for Tape #1 and #5 should both move back to the left. If ")" is not visible on Tape #1 (so that ζ does not end with ")") then ζ should be ***rejected*** (and the computation should end). Otherwise the copy of "(on Tape #5 should be replaced with a copy of "␣" (without changing the "(" on Tape #1 and, indeed, without changing the symbols visible on any other tape) and the tape heads for Tapes #1 and #5 should move left (without moving the tape heads on any other tape).

The tape heads for Tapes #1–#5 can all be moved back to their leftmost cells by moving tape heads for Tapes #1 and #5 left until "␣" is visible on Tape #5, moving tape heads for Tapes #1 and #4 left until "␣" is visible on Tape #4, moving tape heads for Tapes #1 and #3 left until "␣" is visible on Tape #3 and, finally, moving tape heads for Tapes #1 and #2 left until "␣" is visible on Tape #2.

- The computation described in Appendix A.1 can be applied, with Tape #2, to implement the test at line 2 (moving the the "Yes" state if the test at line 2 is passed, and moving to the "No" state if this test is failed).

- The computation described in Appendix A.2 can be applied, with Tape #3, to implement the test at line 3 (moving to the "Yes" state if the test at line 3 is passed, and moving to the "No" state if this test is failed).

- The computation described in Appendix A.2 can be applied, with Tape #4, to implement the test at line 4 (moving to the "Yes" state if the test at line 4 is passed, and moving to the "No" state if this test is failed).

- The computation described in Appendix A.3 can be applied, with Tape #3 as "Tape A" and with Tape #4 as "Tape B", to implement the test at line 5 (moving to the "Yes" state if the test at line 5 is passed, and moving to the "No" state if this test is failed).

- The computation described in Appendix A.8 — which also uses the process described in Appendix A.7 — can be applied to implement the test at line 6(a) in Figure 3.

- The initializations of the variables i, r and s at lines 6(b), 6(c) and 6(d) can be implemented by moving right once on Tapes #6, #7 and #8 without changing the leading copy of "⊔" visible on each tape, and then writing "0" and moving back to the left again on each of these tapes. The contents and locations of other tapes should not be changed when these steps are carried out.

- Later steps refer to the number, h, of transitions encoded in the string μ_4 that is stored on Tape #5. This value is not being stored as data. Instead, the symbol visible on the tape head for Tape #5 will be used to determine whether there are any encodings that remain to be checked in the process described by the later steps in Figure 3. In order to get ready for this, the tape head for Tape #5 should be moved right once, without changing the initial copy of "⊔" on this tape or without changing the contents or positions of tape heads of other tapes. The tape head for Tape #4 will then be pointing to a copy of the leftmost symbol, "(" in the string μ_4, the first time that the test at line 6(e) in Figure 3 is checked.

- The condition "$(i < h)$" in the test at line 6(e) is a test to see whether there are any encodings of transitions, included as substrings of μ_4, that remain to be checked. This condition should be checked by examining the symbol visible on Tape #5. If this symbol is either "(" or "," then $i < h$, and the tape head should be moved once more to the right, without moving other tape heads or changing the contents of any tape, to be ready for the next step. Otherwise this symbol is ")", $i = h$ at this point, and the tape head for Tape #5 should be moved left, without moving other tape heads or changing the contents of any tape, until "⊔" is visible on Tape #5 (so that the tape head is, once again, pointing to the leftmost cell of this tape).

  If it was determined that $i < h$ then it remains to check whether $(r \leq k)$ as well. This test can be carried out using the process described in Appendix A.4, using Tape #7 as "Tape A" and using Tape #2 as "Tape B" in this process. The test is passed, that is, $r \leq h$, if the part of the Turing machine implementing this process ends in its "Yes" state — and there is nothing more to be done before proceeding to the implementation of step 6(f). If this part of the Turing machine ends in its "No" state instead then $r > k$ and, as above, the tape head for Tape #5 should be moved left, without moving other tape heads or changing the contents of any tape, until "⊔" is visible on Tape #5 f(so that the tape head is, once again, pointing to the leftmost cell of this tape).

- To begin the implementation of the step at line 6(f) the tape head for Tape #9 should be moved right without moving tape head for Tape #5 or changing the contents of either tape. Then, while the symbol visible on Tape #5 is not ")" the symbol visible on this tape should be copied onto Tape #9 — moving right on both tapes and without changing the contents of Tape #5. When ")" is visible on Tape #5 this should also be copied onto Tape #9 — but the tape head for Tape #9 should move left instead of right, as the tape head for Tape #5 moves right (without changing the symbol visible) once again. The tape

head for Tape #9 should move back to the left, without moving the tape head on Tape #9 or changing the contents of any tape, until "␣" is visible on Tape #9 once again. The locations and contents of no tapes besides Tapes #5 and #9 should be changed, at any point during the above.

The process described in Appendix A.9 can be used to complete the test at line 6(f): This process ends in the "Yes" state for the part of the Turing machine carrying out this process if the test has been passed, and it ends in the "No" state for this part of the Turing machine if the test has failed. As described in Appendix A.9 the tape heads for Tapes #7, #8 and #9 have been moved back to their leftmost cells, with Tape #9 filled with copies of "␣" once again, so that the computation can continue.

- The step at line 6(g) can be implemented using the process discussed in Appendix A.6, with Tape #8 (which stores the unpadded decimal representation of s) used as "Tape A" in this process.

- Since the unpadded decimal representations of s and m are stored on Tape #8 and Tape #4, respectively, the test for equality at line 6(h) can be implemented using the process in Appendix A.5, with Tape #8 and Tape #4 used as "Tape A" and "Tape B", respectively, in this process.

- Since the unpadded decimal representation of r is stored on Tape #7, the process at line 6(i) can be implemented using the process discussed in Appendix A.6, with Tape #7 used as "Tape A" in this process.

- In order to implement the step at line 6(j), in which s is set to 0, it suffices to move right on Tape #8 (onto a non-blank symbol) and then continue to move right — without changing any symbols that are seen, until a copy of "␣" is visible. The tape head for Tape #8 should move left again, without changing the "␣" that was seen. While non-blank symbols are visible these should each be replaced by "␣" as the tape head moves left. Finally, the tape head for Tape #8 should be moved right once more without replacing the "blank" seen. A copy of "0" should then be written as the tape head moves back to the left, so that a single decimal representation of 0 is stored on Tape #8, as desired.

  No tape heads of other tape heads (except for Tape #8) should be moved and the contents should be changed during the above process.

- The step at line 6(k) can be implemented using the process discussed in Appendix A.6, with Tape #7 (which stores the unpadded decimal representation of i) used as "Tape A" in this process.

- The step at line 6(l) is reached if the test at line 6(f) failed. As noted in Figure 1 this should cause the step at line 8 in the main algorithm to be reached — so that ζ can be ***rejected***, at this point, to end the computation.

- As noted above, in the discussion of the step at line 6(e), it is necessary and sufficient to check whether "`)`" is visible, on Tape #5, to check whether i is equal to h — with the test at line 6(e) failing if this is not the case. Otherwise, the process in Appendix A.3 should be used, with Tape #2 (storing the decimal representation of k) as "Tape A" and with Tape #7 (storing the decimal representation of r) as "Tape B" in this process, can be used to check whether $r > k$ — as needed to complete an implementation of the test at line 6(m).

- The test at line 6(n) is reached if the test at line 6(m) passed. As shown in Figure 1 this should cause the step at line 7 in the main algorithm to be reached — so that ζ can be ***accepted***, at this point, to end the computation.

- The step at line 6(o) is reached if the test at line 6(m) failed, and the step at line 6(p) is reached if the test at line 6(a) failed. Since "`false`" is returned in each case one can see, by an inspection of the code in Figure 1, that the step at line 8 is reached and executed — so that ζ can be ***rejected***, at this point, to end the computation, in both of these cases.

- The remaining steps of the main algorithm (at lines 7–13) are reached to end the computation. As seen in the code in Figures 1 and 2, ζ should be ***accepted*** if the step at line 7 is reached, and ζ should be ***rejected*** if any of the steps at lines 8–13 are reached, instead.

2.4 Proof of Decidability

The implementation-level description, given above, can be shown to correctly implement the algorithm in Figures 1 and 2 — and this can be used to provide the formal description of a multi-tape Turing machine that decides the language L_{TM}. As discussed in the preparatory material for Lecture #7, this implies that thee exists a (standard, one-tape) Turing machine that decides this language as well: L_{TM} is decidable, as claimed.

3 Proving That $L_{\mathsf{TM+I}}$ is Decidable

3.1 Review of Encodings

Recall that, for the input alphabet Σ as shown at line (4), for $1 \leq i \leq \ell$, the symbol σ_i is encoded by the symbol "`s`", followed by the unpadded decimal representation of the integer i.

- Let $\omega = \alpha_1\alpha_2\ldots\alpha_n$ be a string in $\Sigma^\star$ with length n, for some non-negative integer n (so that $\alpha_1, \alpha_2, \ldots, \alpha_n \in \Sigma$), then ω can be encoded using the concatenations of the

symbols in ω — that is, ω is encoded by the string

$$\mu_1 \mu_2 \dots \mu_n \tag{9}$$

where $\mu_i \in \Sigma^\star_{\mathsf{TM}}$ is the encoding of the symbol α_i, for $1 \leq i \leq n$.

- A Turing machine M, as shown at line (2), and an input string $\omega \in \Sigma^\star$ (where Σ is the input alphabet for M) can be encoded using a string

$$(\nu_1 , \nu_2) \tag{10}$$

where ν_1 and ν_2 are the encodings of M and ω respectively, as described above.

3.2 A High-Level Algorithm

An algorithm that can be used to decide whether a string $\zeta \in \Sigma^\star_{\mathsf{TM}}$ is an encoding of a Turing machine and an input string for that Turing machine (that is, to decide whether $\zeta \in L_{\mathsf{TM+I}}$), is given in Figure 4 on page 15.

Consider a string $\zeta \in \Sigma^\star_{\mathsf{TM}}$. If $\zeta \in L_{\mathsf{TM+I}}$ then ζ must be as shown at line (10), above, where $\nu_1 \in L_{\mathsf{TM}}$ and ν_2 encodes a string of symbols over the input alphabet of the Turing machine encoded by ν_1. An examination of the tests at lines 1, 2 and 3, and the step at line 4 in the algorithm in Figure 4, confirms that if $zeta \in L_{\mathsf{TM+I}}$ then ζ is accepted when the algorithm is executed with ζ as input. On the other hand, if $\zeta \notin L_{\mathsf{TM+I}}$ and the algorithm is executed with ζ as input then one of the tests at lines 1, 2 or 3 must fail — so that ζ is rejected when one of the steps at lines 5, 6 or 7 is executed. Thus this algorithm decides membership in $L_{\mathsf{TM+I}}$, and it is sufficient to show that this algorithm can be implemented using a Turing machine in order to prove that the language $L_{\mathsf{TM+I}}$ is decidable.

3.3 Implementation-Level Details

In order to show that the algorithm given in Figure 4 can be implements using a (standard) Turing machine, left us first consider a 4-tape Turing machine

$$M_{\mathsf{TM+I}} = (Q_{\mathsf{TM+I}}, \Sigma_{\mathsf{TM}}, \Gamma_{\mathsf{TM+I}}, \delta_{\mathsf{TM+I}}, q_{0,\mathsf{TM+I}}, q_{\mathsf{accept,TM+I}}, q_{\mathsf{reject,TM+I}})$$

that implements this algorithm. The tape alphabet of this Turing machine is

$$\Gamma_{\mathsf{TM+I}} = \Sigma_{\mathsf{TM}} \cup \{\sqcup\}$$

and the tapes of this Turing machine will be used as follows.

On input $\zeta \in \Sigma^{\star}_{\mathsf{TM}}$ {

1. `if` (ζ begins with "`(`", ends with "`)`", and includes at least one copy of "`,`") {

   Let $\nu_1, \nu_2 \in \Sigma^{\star}_{\mathsf{TM}}$ *such that*

   $$\zeta = (\nu_1\,,\nu_2)$$

   where ν_2 *does not include any copies of "*`,`*"*

2. `if` ($\nu_1 \in L_{\mathsf{TM}}$) {

   Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathit{accept}}, q_{\mathit{reject}})$ *be the Turing machine encoded by* ν_1

3. `if` (ν_2 encodes a string in $\Sigma^{\star}$) {

4. `accept` ζ

   } `else` {

5. `reject` ζ

   }

   } `else` {

6. `reject` ζ

   }

   } `else` {

7. `reject` ζ

   }

Figure 4: Algorithm to Decide Membership in $L_{\mathsf{TM+I}}$

- Tape #1 will store the input string, $\zeta \in \Sigma^{\star}_{\mathsf{TM}}$, throughout an execution of this Turing machine.
- If it has been confirmed that ζ is as shown at line (10) then Tape #2 will store the string ν_1 that is shown at line (10) — without a leading copy of $\sqcup$.
- If it has been confirmed that ζ is as shown at line (10) and, furthermore, that $\nu_1 \in L_{\mathsf{TM}}$ (where ν_1 is as above) then Tape #3 will store the unpadded decimal representation of the size, ℓ, of the input alphabet of the Turing machine encoded by ν_1 — with a leading $\sqcup$ to the left of this string on the tape, so that the leftmost cell of the tape can be detected.
- If it has been confirmed that ζ is as shown at line (10) and, furthermore, that $\nu_1 \in L_{\mathsf{TM}}$ (where ν_1 is as above), then Tape #4 will be used when deciding whether ν_2 encodes

an input string for the Turing machine encoded by ν_1 — where ν_2 is also as shown at line (10). In particular, this tape will be used to store various substrings of ν_2 — with a leading $\sqcup$ to the left of the substring currently being considered, so that the leftmost cell of the tape can be detected.

The development of an implementation-level description of a multii-tape Turing machine that is based on the algorithm in Figure 4, and that decides the language $L_{\text{TM+I}}$, is considerably simpler than the development of an implementation-level description of the algorithm in Figures 1 and 2, and this is left as an ***exercise***.

3.4 Proof of Decidability

The implementation-level description, discussed above, can be shown to correctly implement the algorithm in Figure 4 — and this can be used to provide the formal description of a multi-tape Turing machine that decides the language $L_{\text{TM+I}}$. As discussed in the preparatory material for Lecture #7, this implies that thee exists a (standard, one-tape) Turing machine that decides this language as well: $L_{\text{TM+I}}$ is decidable, as claimed.

A Useful Computations

The above arguments repeatedly use the facts that various languages of encodings of integers are decidable and that various functions (involving encodings of integers) are computable. Proofs of these results are sketched here.

Let

$$\Sigma_{\text{dec}} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \subset \Sigma_{\text{TM}} \tag{11}$$

A.1 Deciding the Language of Non-Negative Integers

Let $L_{\text{non-negative}} \subseteq \Sigma_{\text{TM}}^{\star}$ be the set of strings $\alpha \in \Sigma_{\text{TM}}^{\star}$ such that $\alpha \in \Sigma_{\text{dec}}^{\star}$ and, furthermore, α is the unpadded decimal representation of a non-negative integer — so that α is not the empty string and either α does not begin with "0" or $|\alpha| = 1$.

Suppose that a (part of a) computation begins with α stored on a tape — with a single copy of "$\sqcup$" to its left (so that the first symbol in α is on the *second* cell of the tape) and with infinitely many copies of "$\sqcup$" to the right – and with the tape head pointing to the leftmost cell of the tape. Suppose, furthermore, that we wish to end this part of the computation with the tape as it was at the beginning — and with the Turing machine in one "Yes" state if $\alpha \in L_{\text{non-negative}}$ and with the Turing machine in a "No" state otherwise.

(a) This part of the computation should begin by moving the Turing machine's tape head right, without changing the leading "␣" on the tape. If the symbol that is now visible on the tape is not in Σ_{dec} then the tape head should move back to the left, without changing the symbol that was seen, while entering the "No" state (to end this part of the computation).

(b) Otherwise, if the symbol visible on the second cell is "0" then the tape head should be moved to the right, again, without changing the symbol visible. If the symbol now visible is "␣" then the tape head should be moved back to the left twice, without changing tape contents, and the Turing machine should enter the "Yes" state. Otherwise the tape head should be moved back to the left twice, without changing the tape contents, and the Turing should enter its "No" state (ending this part of the computation in each case).

(c) Finally, suppose that the symbol visible is in Σ_{dec} but is not "0". Then the tape head should sweep to the right, without changing symbols on the tape, until a symbol that *does not* belong to Σ_{dec} is visible. If this symbol is "␣" then the tape head should be moved back to the left until "␣" is seen again and the "Yes" state should be entered. Otherwise the tape head should be moved back to the left until "␣" is seen again and the "No" state should be entered (ending this part of the computation in each case).

A consideration of the unpadded decimal representation of all non-negative integers is sufficient to confirm that this computation ends with the tape head location and tape contents as they were initially, in the "Yes" state if $\alpha \in L_{\text{non-negative}}$ and in the "No" state otherwise, as required.

A.2 Deciding the Language of Positive Integers

Let $L_{\text{positive}} \subseteq \Sigma_{\text{TM}}^{\star}$ be the set of strings $\alpha \in \Sigma_{\text{TM}}^{\star}$ such that $\alpha \in \Sigma_{\text{dec}}^{\star}$ and, furthermore, α is the unpadded decimal representation of a positive integer — so that α is not the empty string and α does not begin with "0". Then $0 \notin$ positive and $L_{\text{non-negative}} = L_{\text{positive}} \cup \{0\}$.

Suppose that a (part of a) computation begins with α stored on a tape — with a single copy of "␣" to its left (so that the first symbol in α is on the *second* cell of the tape) and with infinitely many copies of "␣" to the right — and with the tape head pointing to the leftmost cell of the tape. Suppose, furthermore, that we wish to end this part of the computation with the tape as it was at the beginning — and with the Turing machine in one "Yes" state if $\alpha \in L_{\text{positive}}$ and with the Turing machine in a "No" state otherwise.

(a) Step (a) is the same as step (a) in the algorithm given in Appendix A.1, above.

(b) If the symbol visible on the second cell of the tape is "0" then the tape head should be moved once to the left, without changing the contents of the tape, and the Turing machine

should enter its "No" state. Step (c) is the same as step (c) in the algorithm given in Appendix A.1, above.

A consideration of the unpadded decimal representation of all positive integers is sufficient to confirm that this computation ends with the tape head location and tape contents as they were initially, in the "Yes" state if $\alpha \in L_{\text{positive}}$ and in the "No" state otherwise, as required.

A.3 Deciding Whether One Non-Negative Integer is Less Than Another

Let $\alpha \in \Sigma^{\star}_{\text{dec}}$ and $\beta \in \Sigma^{\star}_{\text{dec}}$ — which are the unpadded decimal representations of non-negative integers a and b, respectively. Suppose that a computation begins with α stored on a first tape, "Tape A" with a single copy of "$\sqcup$" to its left and with β stored on a second tape, "Tape B", with a single copy of "$\sqcup$" to its left as well (and with infinitely many copies of "$\sqcup$" to the right of these strings on these tapes) — with the tape head for each tape at the leftmost cell of the tape when this computation begins. Suppose, furthermore, that we wish to end this part of the computation with the tapes as they were at the beginning — and with the Turing machine in a "Yes" state if $a < b$ and with the Turing machine in a "No" state otherwise.

If the lengths of α and β are not the same then $a < b$ if and only if $|\alpha| < |\beta|$. Otherwise one can determine whether $a < b$ by sweeping over α and β until either corresponding digits are not the same, or the ends of both strings are reached. An implementation-level description of a Turing machine that uses this approach, to determine whether $a < b$, is as shown in Figure 5 on page 19.

The statement "Reposition tape heads and pass", at lines 4 and 12 can be expanded as follows:

(a) Move back to leftmost cells on both tapes.

(b) Stay at the same position on both tapes, without changing the contents of either tape, and move to the "Yes" state.

The statement "Reposition tape heads and fail", at lines 6, 14 and 15 can be expanded as follows:

(a) Move back to leftmost cells on both tapes.

(b) Stay at the same position on both tapes, without changing the contents of either tape, and move to the "No" state.

1. Move right on Tape A and on Tape B without changing the copy of "␣" that is initially visible on each tape.
2. `while` (symbols in Σ_{dec} are visible on both tape) {
3. Move right on both tapes without changing the contents of either tape.
   }
4. `if` ("␣" is visible on Tape A but not on Tape B) {
5. Reposition tape heads and pass.
6. } `else if` ("␣" is visible on Tape B but not on Tape A) {
7. Reposition tape heads and fail.
   } `else` {
8. Move back to leftmost cells on both tapes.
9. Move right on both tapes without changing the contents of either tape.
10. `while` (the same symbol in Σ_{dec} is visible on both tapes) {
11. Move right on both tapes without changing the contents of either tape.
    }
12. `if` (symbols visible on both tapes are in Σ_{dec} and a smaller digit is visible on Tape A than on Tape B) {
13. Reposition tape heads and pass.
14. } `else if` (symbols visible on both tapes are in Σ_{dec}) {
15. Reposition tape heads and fail.
    } `else` {
16. Reposition tape heads and fail.
    }
   }

Figure 5: Algorithm to Decide Whether One Integer is Less Than Another

The statement "Move back to leftmost cells on both tapes" that is shown at line 8, can be implemented as follows:

(a) Move both tape heads left without changing the contents of either tape.
(b) `while` (the symbol visible on TapeA is not "␣") {
(c) Move both tapes left without changing the contents of either tape.
}

Now, the computation begins with both tape heads at the leftmost cells of their tape and each step of this algorithm moves the tape heads in the same direction. Furthermore, no step changes the contents of either tape. This can be used to show (using a proof by induction on the number of moves made, so far) that the tape heads for Tapes A and B are always the same distance away from their leftmost cells — and the contents of the tapes are always the same as when this part of the computation began.

This can be used to establish that any execution of macro "Move back to leftmost cells on both tapes" correctly moves both tape heads back to the leftmost cells of their tapes, without changing the contents of either tapes, as required, Thus every execution of "Reposition tape heads and pass" returns the tapes to their original states, with the "Yes" state entered as the computation ends, and every execution of "Reposition tape heads and fail" returns the tapes to their original states, with the "No" state entered as the computation ends.

Now consider an execution of the algorithm in Figure 5. If the length of α is less than the length of β then $a < b$ and the algorithm's execution should end in the "Yes" state. This is what happens, because "$\sqcup$" is visible on Tape A (during the sweep to the right, at the beginning of the computation) before it is visible on Tape B — so the the test at line 4 is passed when it is reached and this part of the computation ends in the "Yes" state, as required, after the step at line 5 is reached and executed.

On the other hand, if the length of β is less than the length of α instead then $a > b$ and the algorithm's execution should end in the "No" state. This is what happens, because "blank" is visible on Tape B (during the initial sweep to the right) before it is visible on Tape A — so that the test at line 4 is failed when it is reached and checked, but the test at line 6 is then reached and passed — and this part of the computation ends in the "No" state, as required, when the step at line 7 is reached and executed.

In the only remaining case, $|\alpha| = |\beta|$. During the algorithm's execution the tests at lines 4 and 6 both fail when they are reached and checked, so that the step at line 8 is reached and executed — and the steps beginning at line 9 are used to begin a sweep right over both tapes.

- Suppose that $a < b$, so that the algorithm's execution should end in the "Yes" state. In this case there exist strings $\mu, \nu_a, \nu_b \in \Sigma^{\star}_{\mathsf{dec}}$ and digits $d_a, d_b \in \Sigma_{\mathsf{dec}}$, where $|\nu_a| = |\nu_b|$, such that

  $$\alpha = \mu\, d_a\, \nu_a \qquad \text{and} \qquad \beta = \mu\, d_b\, \nu_b \tag{12}$$

  Now, the step at lines 9–11 are used to begin the sweep right, with the test at line 10 after all of the corresponding symbols in the copies of μ, on the tapes, have all be examined — with the test at line 10 failing because d_a is now being compared to d_b. Now, since $a < b$, $d_a < d_b$ and the test at line 12 is reached and passed, so that the step at line 13 is reached and executed — with this part of the computation ending in the "Yes" state as required.

- Suppose, instead, that $a > b$, so that the algorithm's execution should end in the "No" state. In this case there also exist strings $\mu, \nu_a, \nu_b \in \Sigma_{\mathsf{dec}}^{\star}$ and digits $d_a, d_b \in \Sigma_{\mathsf{dec}}$ such that the equation at line (12) is satisfied, As for the previous case the steps at lines 9–11 are used to begin the sweep right, with the test at line 10 after all of the corresponding symbols in the copies of μ, on the tapes, have all be examined — with the test at line 10 failing because d_a is now being compared to d_b. Now, since $a > b$, $d_a > d_b$ and the test at line 12 is reached and failed, so that the test at line 14 is reached — and passed. Thus the step at line 15 is reached and executed — and this part of the computation ends in the "No" state, as required.
- Suppose, finally, that $a = b$, so that the algorithm's execution should end in the "No" state. In this case the string α and β are the same, that is, $\alpha = \beta$ as well. In this case the the steps at lines 9–11 are used to begin the sweep right, with the test at line 10 failed after all the corresponding symbols in α and β have been examined and copies of "$\sqcup$" are visible on both tapes. In this case the test at line 12 is failed and checked, since $\sqcup \notin \Sigma_{\mathsf{dec}}$, and the test at line 14 is failed for the same reason, so that the step at line 16 is reached and executed — and, once again, this part of the computation ends in the "No" state, as required.

Since all possible cases have been considered it follows that the intermediate-level description, shown in Figure 5, describes a part of a Turing machine that solves the computational problem that is now being considered.

A.4 Deciding Whether One Non-Negative Integer is Less or Equal to Than Another

Let $\alpha \in \Sigma_{\mathsf{dec}}^{\star}$ and $\beta \in \Sigma_{\mathsf{dec}}^{\star}$ — which are the unpadded decimal representations of non-negative integers a and b, respectively. Suppose that a computation begins with α stored on a first tape, "Tape A" with a single copy of "$\sqcup$" to its left and with β stored on a second tape, "Tape B", with a single copy of "$\sqcup$" to its left as well (and with infinitely many copies of "$\sqcup$" to the right of these strings on these tapes) — with the tape head for each tape at the leftmost cell of the tape when this computation begins. Suppose, furthermore, that we wish to end this part of the computation with the tapes as they were at the beginning — and with the Turing machine in a "Yes" state if $a \leq b$ and with the Turing machine in a "No" state otherwise.

An implementation-level description of a part of a Turing machine that solves this problem can be obtained from the implementation-level description shown in Figure 5 by making one change: The step at line 16 should be changed from "Reposition tape heads and fail" to "Reposition tape heads and pass".

To see that this is the case that this is the case, note the only difference between the problem considered in Appendix A.3 and the problem here is that the computation should end in the

```
1. Move right on Tape A and on Tape B without changing the copy of "␣"
   that is initially visible on each tape.
2. while (the same symbol in Σ_dec is visible on both tapes) {
3.   Move right on Tape A and on Tape B without changing the copy of "␣"
     that is initially visible on each tape.
   }
4. if ("␣" is visible on both Tape A and Tape B) {
5.   Reposition tape heads and pass.
   } else {
6.   Reposition tape heads and fail.
   }
```

Figure 6: Algorithm to Decide Whether One Integer is Equal to Another

"No" state, when $a = b$, for the problem considered in Appendix A.3, but the computation should end in the "Yes" case, in this case, for the problem considered here. The step at line 16 in Figure 5 is reached if and only if $a = b$, so this change is necessary, and sufficient, to produce an implementation-level description of a part of a Turing machine that solves the problem that is now being considered.

A.5 Deciding Whether One Non-Negative Integer is Equal to Than Another

Let $\alpha \in \Sigma_{\text{dec}}^{\star}$ and $\beta \in \Sigma_{\text{dec}}^{\star}$ — which are the unpadded decimal representations of non-negative integers a and b, respectively. Suppose that a computation begins with α stored on a first tape, "Tape A" with a single copy of "␣" to its left and with β stored on a second tape, "Tape B", with a single copy of "␣" to its left as well (and with infinitely many copies of "␣" to the right of these strings on these tapes) — with the tape head for each tape at the leftmost cell of the tape when this computation begins. Suppose, furthermore, that we wish to end this part of the computation with the tapes as they were at the beginning — and with the Turing machine in a "Yes" state if $a = b$ and with the Turing machine in a "No" state otherwise.

Consider the implementation-level description of a part of a Turing machine that is shown in Figure 6, above. The macros "Reposition tapes heads and pass" and "Reposition tape heads and fails" are as described in Appendix A.3. This simply sweeps right over the non-blank contents of both tapes until it is either noticed that they are the same because "␣" symbols are discovered on the tapes at the same time (in which case the tape heads are repositioned and the "Yes" state is entered, using the "Reposition tape heads and pass" macro) or noticed that

they are different (in which case the tape heads are repositioned and the "No" state is entered, using the "Reposition tape heads and fail" macro) — as required for this problem.

A.6 Incrementing the Value of an Integer Variable

Suppose that a (part of a) computation begins with $\alpha \in \Sigma^{\star}_{\mathsf{dec}}$ stored on a tape, "Tape A" — with a single copy of "$\sqcup$" to its left (so that the first symbol in α is on the *second* cell of the tape) and with infinitely many copies of "$\sqcup$" to the right – and with the tape head pointing to the leftmost cell of the tape. Furthermore, suppose that α is the unpadded decimal representation of a non-negative integer n. Finally, suppose that we wish to end this part of the computation with α replaced, on this tape, with the unpadded decimal representation of $n+1$ — to the right of a single leading "$\sqcup$" on the tape, and with the tape head at the leftmost cell of the tape, as before.

A Turing machine that solves a similar problem is included in the document "Turing Machines That Compute Functions" (a supplement for Lecture #6). The chief differences here is that the problem in the supplemental document includes binary arithmetic instead of decimal representation, and that the input string does not have a leading "$\sqcup$" to its left, for the problem described in this supplemental document. The modification of the Turing machine in the document, to produce a part of a Turing machine that solves the problem given here, is left as an ***exercise***.

A.7 Deciding Whether a String Encodes a Transition

Suppose that the following conditions are satisfied when a part of a computation begins.

- Tape #2 stores the unpadded decimal representation of a non-negative integer k — with a single copy of "$\sqcup$" to the left of this string on the tape (and with infinitely many copies of "$\sqcup$" to the right) and with the tape head located at the leftmost cell of this tape.
- Tape #4 stores the unpadded decimal representation of a positive integer m — with a single copy of "$\sqcup$" to the left of this string on the tape (and infinitely many copies of "$\sqcup$" to the right) and with the tape head located at the leftmost cell of this tape.
- Tape #9 stores a string $\alpha \in (\Gamma \setminus \{\sqcup\})^{\star}$ — with a single copy of "$\sqcup$" to the left of this string (and infinitely many copies of "$\sqcup$" to the right) and with the tape head located at the leftmost cell of this tape.
- Tape #10 stores infinitely many copies of "$\sqcup$", with the tape head located at the leftmost cell of the tape.

Suppose that we want to end this part of the computation as follows.

- The contents and locations of the tape head of all tapes except for Tape #9 are as they were when this part of the computation began.
- Tape #9 now stores an infinite number of copies of "␣", with the tape head located at the leftmost cell of the tape.
- If the string α, that was initially stored on Tape #9, encodes a transition for a Turing machine with a set of states as shown at line (3) and a tape alphabet as shown at line (5) (for k and m as above) then this part of the computation ends in a "Yes" state for the part of the Turing machine that carries out this part of a computation — and it ends in a "No" state for the part of the Turing machine that carries out this part of a computation, otherwise.

Note that this part of the computation should end with this part of the Turing machine in its "Yes" state if and only if α is a string with the form

$$(\mathtt{q}\,\eta_1\,,\mathtt{s}\,\eta_2\,,\mathtt{q}\,\eta_3\,,\mathtt{s}\,\eta_4\,,\,\eta_5\,) \tag{13}$$

where η_1 is the unpadded decimal representation of some integer r such that $0 \leq r \leq k$, η_2 is the unpadded decimal representation of some integer s such that $0 \leq s \leq m-1$, η_3 is either "`Y`", "`N`", or the unpadded decimal representation of some integer t such that $0 \leq t \leq k$, η_4 is the unpadded decimal representation of some integer u such that $0 \leq u \leq m-1$, and η_5 is either "`L`" or "`R`".

Consider the implementation-level description given in Figure 7 on page 25 and Figure 8 on page 26.

The process *Copy a number and compare it to* k, which is used in the steps at lines 2 and 6 in the above process, is as follows.

(a) Move right on Tape #10 without changing the contents of any tape or moving the location of any other tape head.

(b) While the symbol, σ, visible on Tape #9 is in Σ_{dec}, copy σ onto Tape #10, without changing this symbol on Tape #9 — moving right on both tapes and without changing the contents or locations of tape heads of any other tape.

(c) Move left on Tape #10 without changing the symbol (a copy of "␣") that is visible or without changing the contents or location of the tape head of any other tape.

(d) While the symbol visible on Tape #10 is not "␣" move left on Tape #10 without changing the symbol that is visible or changing the contents or location of the tape head of any other tape.

1. Move right at most three times on Tape #9, without changing the contents of any tape or locations of any other tape heads, to check whether the first three symbols on Tape #9 are "␣", "(", and "q". *Erase Tape #9 and fail* if this is not the case.
2. *Copy a number and compare it to* k (noting that this ends this part of the computation, if the next symbols on Tape #9 do not form the unpadded decimal representation of an integer between 0 and k).
3. Move right at most two times on Tape #9, without changing the contents of any tape or locations of any other tape heads, to check whether the next two symbols on Tape #9 are "," and "s". *Erase Tape #9 and fail* if this is not the case.
4. *Copy a number and compare it to* m (noting that this ends this part of the computation, if the next symbols on Tape #9 do not form the unpadded decimal representation of an integer between 0 and $m - 1$).
5. Move right at most two times on Tape #9, without changing the contents of any tape or locations of any other tape heads, to check whether the next two symbols on Tape #9 are "," and "q". *Erase Tape #9 and fail* if this is not the case.

Figure 7: Beginning of Process to Recognize Encoding of a Transition

(e) Use the process "Deciding Whether One Non-Negative Integer is Less Than or Equal to Another"', described in Appendix A.4, with Tape #10 as "Tape A" and with Tape #2 as "Tape B", to decide whether the non-negative integer, whose encoding is now stored on Tape #10, is less than or equal to k. If this test passes (that is, the encoded integer is less than or equal to k) then *erase Tape #10 and pass*. Otherwise *erase Tape #10 and fail*.

The process *Copy a number and compare it to* m, which is used in the steps at line 4 and 8 in the above process, is as follows.

(a) Move right on Tape #10 without changing the contents of any tape or moving the location of any other tape head.

(b) While the symbol, σ, visible on Tape #9 is in Σ_{dec}, copy σ onto Tape #10, without changing this symbol on Tape #9 — moving right on both tapes and without changing the contents or locations of tape heads of any other tape.

(c) Move left on Tape #10 without changing the symbol (a copy of "␣") that is visible or without changing the contents or location of the tape head of any other tape.

6. If the symbol visible on Tape #9 is either "Y" or "N" then move the tape head of Tape #9 right, without changing the contents of any tape or moving any other other tape heads.
   Otherwise, if the symbol visible on Tape #9 belongs to Σ_{dec}, then *copy a number and compare it to* k (noting that this ends this part of the computation, if the next symbols on Tape #9 do not form the unpadded decimal representation of an integer between 0 and k).
   Otherwise, *erase Tape #9 and fail*.
7. Move right at most two times on Tape #9, without changing the contents of any tape or locations of any other tape heads, to check whether the next two symbols on Tape #9 are "," and "s". *Erase Tape #9 and fail* if this is not the case.
8. *Copy a number and compare it to* m (noting that this ends this part of the computation, if the next symbols on Tape #9 do not form the unpadded decimal representation of an integer between 0 and $m-1$).
9. Move right at most four times on Tape #9, without changing the contents of any tape or locations of any other tape heads, to check whether the next four symbols on Tape #9 are ",", either "L" or "R", ")", and "␣". *Erase Tape #9 and pass* if this condition is satisfied, and *erase Tape #9 and fail*, otherwise.

Figure 8: End of Process to Recognize Encoding of a Transition

(d) While the symbol visible on Tape #10 is not "␣" move left on Tape #10 without changing the symbol that is visible or changing the contents or location of the tape head of any other tape.

(e) Use the Process "Deciding Whether One Non-Negative Integer is Less Than Another", described in Appendix A.3, with Tape #10 as "Tape A" and with Tape #4 as "Tape B", to decide whether the non-negative integer, whose encoding is now stored on Tape #10, is less than m. If this test passes (that is, the encoded integer is less than m) then *erase Tape #10 and pass*. Otherwise *Erase Tape #10 and fail*.

Both of the above processes uses a process "Erase Tape #10 and pass", which consists of the following steps — none of which change the contents or position of any tapes except Tape #10: Move right on Tape #10 without changing the leading "␣" that is visible. Continue to move right, without changing tape symbols, while the symbol visible is *not* "␣". Move left, without changing the copy of "␣" that was seen. Move left, while the symbol is not "␣", replacing the symbol visible with "␣". Finally, enter a "Yes" state (to signal that the encoded integer satisfied the desired property) – so that the computation proceeding this can proceed — without changing

the tape contents or location of the tape head.

The process “Erase Tape #10 and fail” is the same as the above, except that it ends by entering a “No” state instead of a “Yes” state — signalling that the computation including this should end, with failure, as well.

The process “Erase Tape #9 and pass” consists of the following steps — none of which change the contents or position of any tapes except Tape #9: While the symbol visible on Tape #9 is not “blank” move right without changing the symbol that is visible. Move left again without changing the copy of “$\sqcup$” that is now visible. Move left, while the symbol visible is not “$\sqcup$”, replacing it with a copy of “$\sqcup$”. This ends with the tape head at the leftmost call again: Enter a “Yes” state, to signal the fact that the string stored on Tape #9 was an encoding of a transition, to end this part of the computation.

The process “Erase Tape #9 and fail” is the same as the above, except that ends by entering a “No” state, to signal the fact that the string stored on Tape #9 was not an encoding of a transition, to end this part of the computation.

The algorithm described here is simply checking that the string α has the form given at line (13), above, so it solves the problem that is described above.

A.8 Deciding Whether a String Encodes a Sequence of Transitions

Suppose that the following conditions are satisfied when a part of a computation begins.

- Tape #2 stores the unpadded decimal representation of a non-negative integer k — with a single copy of “$\sqcup$” to the left of this string on the tape (and with infinitely many copies of “$\sqcup$” to the right) and with the tape head located at the leftmost cell of this tape.
- Tape #4 stores the unpadded decimal representation of a positive integer m — with a single copy of “$\sqcup$” to the left of this string on the tape (and infinitely many copies of “$\sqcup$” to the right) and with the tape head located at the leftmost cell of this tape.
- Tape #5 stores a string $\mu_4 \in (\Gamma \setminus \{\sqcup\})^\star$ — with a single copy of “$\sqcup$” to the left of this string (and infinitely many copies of “$\sqcup$” to the right) and with the tape head located at the leftmost cell of this tape.
- Tapes #9 and #10 store infinitely many copies of “$\sqcup$”, with the tape head located at the leftmost cell of each tape.

Suppose that we want to end this part of the computation as follows.

1. The contents and locations of the tape head of all tapes are as they were before this part of the computation began.

2. If μ_4 encodes a sequence of zero or more transitions for a Turing machine with a set of states as shown at line (3) and a tape alphabet as shown at line (5), for k and m as above, then this part of the computation ends in a "Yes" state for the part of the Turing machine that carries out this part of the computation — and it ends in a "No" state for the part of the Turing machine that carries out this part of the computation, otherwise.

An implementation-level of a description that can be used to solve this problem is shown in Figure 9 on page 29, Figure 10 on page 30, and Figure 11 on page 31.

The process "Erase Tape #9", included in the steps at lines #7 and #15, is only called when the tape head for Tape #9 might be immediately to the right of non-blank symbols to the left of the tape head (with a copy of "␣") currently visible. It can, therefore, be carried out as follows.

(a) Move the tape head left on Tape #9, without changing the "␣" that is currently visible, and without changing the contents or location of the tape head of any other tape.

(b) While a non-blank symbol is visible on Tape #9 replace this symbol with "␣" and move the tape head *left* — without changing the contents or position of the tape head of any other tape.

The process "Move back on Tapes #5 and fail", included in the steps at lines 6, 7, 14, 15, 20 and 21 is as follows:

(a) While a non-blank symbol is visible on Tape #9 move the tape head for this tape *left*, without changing the symbol that is visible, and without changing the contents or location of the tape head of any other tape.

(b) Enter the "No" state for this part of the Turing machine.

The process "Move back on Tape #5 and pass", included in the step at line 19, has the same first two steps as the process "Move back on Tape #5 and fail"; it ends differently, by entering the "Yes" state for this part of the Turing machine instead of the "No" state.

While it is rather long, this process is simply checking using the process in Appendix A.7 to check whether μ_4 is a sequence of one or more encodings of transitions, separated by commas and enclosed by brackets — with steps included to erase Tape #9 move the tape head for Tape #5 back to the left before the computation ends. Thus it correctly performs the computation that has been described.

1. Move right twice on Tape #5 to check that the first two symbols on this tape are "␣" and "(". Move the tape head back to the initial "␣" (the first one that will be seen) and enter the "No" state to end this part of the computation, if this is not the case. Otherwise move right on Tape #9, without changing the leading "␣" or changing the contents or locations of tape heads of other tapes.
2. While the symbol visible on Tape #5 is neither ") nor "␣", copy this symbol onto Tape #9 — without changing this symbol on Tape #5 — and moving right on both tapes — without changing the contents or location of tape heads of other tapes.
3. `if` (the symbol visible in Tape #5 is ")") `{`
4. Write ")" on Tape #9, moving left while moving right past the copy of ")" that is visible on Tape #5, without changing this symbol — and without changing the contents or moving the tape heads of other tape.
5. While the symbol visible on Tape #9 is not "␣" move left on this tape, without changing the symbol that is visible — and without changing the contents or location of the tape head of any other tape.
6. Use the process for "Deciding whether a string encodes a transition", given in Appendix A.7, to decide whether the string on Tape #9 encodes a transition whose set of states and tape alphabet are as shown at lines (3) and (5). *Move back on Tape #5 and fail* if it has been decided that the string on Tape #9 did not encode a transition.

`}` `else` `{`

7. Move left on Tape #5 without changing the "blank" that is visible there or without changing the contents or moving the tape head of any other tape. Then *erase Tape #9*, and *move back on Tape #5 and fail*.

`}`

Figure 9: Beginning of Process to Recognize Encoding of a Sequence of Transitions

A.9 Deciding Whether a String Encodes a Transition for a Given State and Symbol

Suppose that the following conditions are satisfied when a part of a computation begins.

- Tape #9 stores a string $\alpha \in (\Gamma \setminus \{\sqcup\})^\star$, which encodes a transition of a Turing machine with a set of states as shown at line (3) and a tape alphabet as shown at line (5), for a

8. `while` (the symbol visible on Tape #4 is "`,`") `{`
9. Move right on Tapes #5 and #9 without changing the copy of "`,`" that is visible on Tape #5 or the copy of "␣" that is visible on Tape #9, and without changing the contents or location of the tape head of any other tape.
10. While the symbol visible on Tape #5 is neither "`)`" nor "␣", copy this symbol onto Tape #9 — without changing this symbol on Tape #5 — and moving right on both tapes — without changing the contents or location of the tape head of other tapes.
11. `if` (the symbol visible on Tape #5 is "`)`") `{`
12. Write "`)`" onto Tape #9, moving left while moving right past the copy of "`)`" that is visible on Tape #5, without changing this symbol — and without changing the contents or moving the tape head of any other tape.
13. While the symbol visible on Tape #9 is not "␣" move left on this tape, without changing the symbol that is visible — and without changing the contents or location of the tape head of any other tape.
14. Use the process for "Deciding whether a string encodes a transition", given in Appendix A.7, to decide whether the string on Tape #9 encodes a transition whose set of states and tape alphabet are as shown at lines (3) and (5). *Move back on Tape #5 and fail* if it has been decided that the string on Tape #9 did not encode a transition.

`}` `else` `{`

15. Move left on Tape #5 without changing the "blank" that is visible there or without changing the contents or moving the tape head of any other tape. Then *erase Tape #9* and *move back on Tape #5 and fail*.

`}`

`}`

Figure 10: Middle of Process to Recognize Encoding of a Sequence of Transitions

non-negative integer k and an integer m such that $m \geq 2$ — with a single copy of "␣" to the left of this string (and infinitely many copies of "␣" to the right). The tape head is initially resting at the leftmost cell of this tape.

- Tape #7 stores the unpadded decimal representation of an integer r such that $0 \leq r \leq k$ — with a single copy of "␣" to the left of this string (and infinitely many copies of "␣" to the right). The tape head is initially pointing to the leftmost cell of this tape.
- Tape #8 stores the unpadded decimal representation of an integer s such that $0 \leq s \leq$

16. `if` (the symbol visible on Tape #5 is "`)`") `{`
17. Move right on Tape #5 without changing the copy of "`)`" that is visible — and without changing the contents or location of the tape head of any other tape.
18. `if` (the symbol visible on Tape #5 is "␣") `{`
19. *Move back on Tape #5 and pass.*
`}` `else` `{`
20. *Move back on Tape #5 and fail.*
`}`
`}` `else` `{`
21. *Move back on Tape #5 and fail.*
`}`

Figure 11: End of Process to Recognize Encoding of a Sequence of Transitions

$m - 1$ — with a single copy of "␣" to the left of this string (and infinitely many copies of "␣" to the right). The tape head is initially pointing to the leftmost cell of this tape.

Suppose that we want to end this part of the computation as follows.

- The contents and locations of the tape head of all tapes except for Tape #9 are as they were before this part of the computation began.
- Tape #9 now stores an infinite number of copies of "␣", with the tape head located at the leftmost cell of the tape.
- If the transition encoded by α is a transition for the state q_r and the tape symbol σ_s then this part of the computation ends in a "Yes" state for the part of the Turing machine that carries out this part of the computation — and it ends in a "No" state for the part of the Turing machine that carries out this part of the computation, otherwise.

An implementation-level description that can be used to solve this problem is shown in Figure 12 on page 32.

The process "Move the tape head for Tape #7 back to the left", which is used at steps 6 and 16, moves the tape head for Tape #7 once, without moving the tape heads for other tapes or changing the contents of any tapes. While the symbol visible on Tape #7 the tape head for this tape continues to move left, without moving tape heads or changing the contents of any

1. Move the tape head for Tape #9 to the right three times, without changing the contents of this tape — and without changing the contents or locations of the tape heads of any other tapes,
2. Move the tape head for Tape #7 to the right (once), without changing the contents of this tape — and without changing the contents or locations of the tape heads of any other tapes.
3. `while` (the same symbol is visible on Tapes #7 and #9) `{`
4. Move the tape heads for Tape #7 and Tape #9, without moving other tape heads and without changing the contents of any tape.

   `}`
5. `if` ("␣" is visible on Tape #7 and "`,`" is visible on Tape #9) `{`
6. *Move the tape head for Tape #7 back to the left.*
7. Move the tape head for Tape #9 to the right, twice, without moving the tape head of any other tape, and without changing the contents of any tape.
8. Move the tape head for Tape #8 to the right (once), without moving the tape head of any other tape or changing the contents of any tape.
9. `while` (the same symbol is visible on Tapes #8 and #9) `{`
10. Move the tape heads for Tape #8 and Tape #9, without moving other tape heads and without changing the contents of any tape

    `}`
11. `if` ("␣" is visible on Tape #8 and "`,`" is visible on Tape #9) `{`
12. *Move the tape head for Tape #8 back to the left.*
13. *Erase Tape #9* and enter the "Yes" state to end this process.

    `}` `else` `{`
14. *Move the tape head for Tape #8 back to the left.*
15. *Erase Tape #9* and enter the "No" state to end this process.

    `}`

   `}` `else` `{`
16. *Move the tape head for Tape #7 back to the left.*
17. *Erase Tape #9* and enter the "No" state to end this process.

   `}`

Figure 12: Process to Identify a Transition for a Given State and Symbol

tape, once again. This ends when the copy of "␣" at the leftmost cell of Tape #7 is visible, so that the tape head for Tape #7 has been moved to the leftmost cell, as desired.

The process “Move the tape head for Tape #8 back to the left, which is used at lines 12 and 14, is the same as the above process, except that it moves the tape head for Tape #8 (and examines symbols on this tape to carry out tests) instead of Tape #7.

The process “Erase Tape #9”, which is used at lines 13, 15 and 17, moves the tape head for Tape #9 to the right while a non-blank symbol is visible on this tape — without moving the tape heads for other tapes or changing the contents of any tape. Once a copy of “blank” is visible on Tape #9 the tape head for this tape is moved left, without moving the tape heads for other tapes or changing the contents of any tape. Then, while a non-blank symbol is visible on Tape #9 this is replace “␣” as the tape head moves left — once again, without moving other tape heads or changing the contents of other tapes. This ends with the tape head for Tape #9 resting at the leftmost cell (since this would initially contain the only copy of “␣” to the left of the tape head’s initial position) and with all cells of Tape #9 storing “␣”, as required.

This process is simply moving tape heads for Tapes #9 and #7 to the beginning of the decimal representations of the indices of states to be compared. If these match then the tape heads for Tapes #9 and #9 are then moved to the beginning of the decimal representations of indices to tape symbols to be compared — moving back to the left on Tapes #7 and (if it is accessed) Tape #8, and erasing the contents of Tape #9 — as needed to carry out the computation that is described, above. That is, it correctly solves the problem that is described here.