Lecture #6: Introduction to Turing Machines
Turing Machines That Compute Functions

Consider the problem of computing a (partial or total) function

$$f : \Sigma_1^\star \to \Sigma_2^\star$$

for alphabets Σ_1 and Σ_2, using a Turing machine that is similar to a Turing machine that recognizes a language.

This problem will be needed shortly in this course — when we consider ways to prove that languages are ***undecidable*** (or even ***unrecognizable*** — so that students should be familiar with the ***definitions*** of Turing machines given in Section 1, and of computable functions, given in Section 2.

The rest of this document can be skipped unless students find this subject to be interesting, wish to see examples of this kind of Turing machine or see how things can be proved about these machines — as in Sections 3 and 4.

1 Definition

A ***one-tape Turing machine that computes a partial or total function***

$$f : \Sigma_1^\star \to \Sigma_2^\star,$$

for alphabets Σ_1 and Σ_2, can be modelled as a 7-tuple

$$M = (Q, \Sigma_1, \Sigma_2, \Gamma, \delta, q_0, q_{\text{halt}})$$

where Q, Σ_1, Σ_2, Γ, δ, q_0, and q_{halt} are as follows.

- Q is a finite, and non-empty, set of ***states***. It is used in the same way as the set of states does, for Turing machines that recognize languages.
- Σ_1 is an alphabet (such that $\Sigma_1 \cap Q = \emptyset$ and such that $\sqcup \notin \Sigma_1$), called the ***input alphabet***. It plays the same role as the "input alphabet" does for Turing machines that recognize languages.

- Σ_2 is another (possibly different) alphabet (such that $\Sigma_2 \cap Q = \emptyset$ and such that $\sqcup \notin \Sigma_2$), called the ***output alphabet***.

- Γ is an alphabet, called the ***alphabet***, which is used in the same was as the tape alphabet is, for a Turing machine that recognizes languages. $\Sigma_1 \subseteq \Gamma$, $\Sigma_2 \subseteq \Gamma$, and $\sqcup \in \Gamma$ — but Γ might also include a finite number of additional symbols that are not in $\Sigma_1 \cup \Sigma_2 \cup \{\sqcup\}$.

- q_0 is a state in Q, called the ***start state***. This plays the same role, for Turing machines that compute functions as it does for Turing machines that recognize languages.

- q_{halt} is the ***halt state***. This plays a role that is similar to role that the accepting state, "q_{accept}" and the rejecting state, "q_{reject}", do, for Turing machines that recognize languages (see the description of the transition function, δ, to see how).

- Like the transition function for Turing machines, the transition function, δ, is a partial function

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

  In particular, $\delta(q, \sigma)$ should be defined for every state $q \in Q$ such that $q \neq q_{\text{halt}}$, and for every symbol $\sigma \in \Gamma$ — but $\delta(q_{\text{halt}}, \sigma)$ *should not* be defined, for any symbol $\sigma \in \Gamma$.

Configurations of a Turing machine that computes a function are described, as strings over the alphabet $Q \cup \Gamma$, just as they are for Turing machines that recognize languages. The transition function is used to update the Turing machine's tape in the same way for both kinds of Turing machines as well.

The ***initial configuration*** for an input string, $\omega \in \Sigma_1^\star$, is the same as it is for Turing machines that recognize languages: The Turing machine is in its start state, the input string is written on the leftmost cells of the tape, with copies of "$\sqcup$" on all cells to the right, and the tape head points to the leftmost cell of the tape — so that is the configuration represented by the string "$q_0\,\omega$".

If this Turing computes a partial function $f : \Sigma_1^\star \to \Sigma_2^\star$ then the following properties are satisfied, for every string $\omega \in \Sigma_1^\star$:

- If f is defined at ω then the Turing machine's execution on input ω halts, with $f(\omega)$ written on the leftmost cells of the tape, with copies of "$\sqcup$" to the right, and with the tape head pointing to the leftmost cell. Thus the Turing machine is in the configuration represented by the string "$q_{\text{halt}}\, f(\omega)$" when the computation ends.

- If f is undefined at ω then the Turing machine loops on ω.

2 Computable Functions

Let us say that a function $f : \Sigma_1^\star \rightarrow \Sigma_2^\star$ is ***computable*** if there exists a standard Turing machine M that computes functions, as defined above, with input alphabet Σ_1 and output alphabet Σ_2, such that M computes f.

3 A Simple Example: Incrementing the Value of a Number in Unary Notation

3.1 The Problem To Be Solved

Suppose that $\Sigma_{\text{unary}} = \{1\}$. A string $1^n \in \Sigma_{\text{unary}}^\star$ can be used as the ***unary representation*** of n, for every non-negative integer n. Let

$$f_{\text{u\_inc}} : \Sigma_{\text{unary}}^\star \rightarrow \Sigma_{\text{unary}}^\star$$

be the function mapping the unary representation of n to the unary representation of $n + 1$ — so that

$$f_{\text{u\_inc}}(1^n) = 1^{n+1} \tag{1}$$

— for every non-negative integer n.

3.2 How Could This Problem Be Solved Using a Turing Machine?

Suppose we wish to design a Turing machine that computes functions

$$M = (Q, \Sigma_{\text{unary}}, \Sigma_{\text{unary}}, \Gamma, \delta, q_0, q_{\text{halt}})$$

that computes the above function $f_{\text{u\_inc}}$.

Since $f_{\text{u\_inc}}$ is as shown at line (1), above, all that M must do is add another copy of "1" to the end of its input string. If "␣" is visible immediately (so that the input is the empty string then M should write a copy of "1" without moving its tape head. Otherwise — marking the leftmost symbol on the tape, so that it can be found again, M's tape head should be swept right until the leftmost copy of "␣" is found. This should be replace by a copy of "1" as the head begins to move left. The tape head should continue to move left until the marked leftmost symbol is found. This should be replaced by a copy of "1", without moving the top head — and the computation can end at this point.

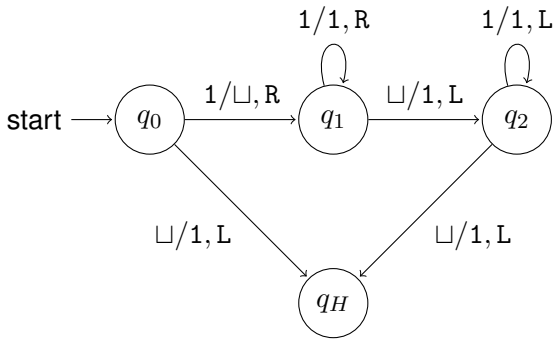
There are multiple ways to mark the leftmost cell of the tape. One way that that does not always work — but that works for this problem — is to replace the symbol with "␣".

3.3 A Turing Machine That Computes This Function

Using this approach one can set

$$Q = \{q_0, q_1, q_2, q_{\text{halt}}\}, \qquad \Gamma = \Sigma_{\text{unary}} \cup \{\sqcup\} = \{\mathtt{1}, \sqcup\},$$

and to set δ is shown by the following figure — where the halting state is shown as "q_H" instead of "q_{halt}", to keep the picture simple:



3.4 Proof of Correctness

Now note that if M is as described, then

$$q_0 \vdash q_{\text{halt}}\mathtt{1} \qquad \text{(since } \delta(q_0, \sqcup) = (q_{\text{halt}}, \mathtt{1}, \mathtt{L})\text{)}$$

— as required when the input is the empty string, since

$$f_{\text{u\_inc}}(\lambda) = f_{\text{u\_inc}}(\mathtt{1}^0) = \mathtt{1}^1 = \mathtt{1}.$$

Note, as well, that

$$\begin{aligned} q_0\mathtt{1} &\vdash \sqcup\, q_1 && \text{(since } \delta(q_0, \mathtt{1}) = (q_1, \sqcup, \mathtt{R})\text{)} \\ &\vdash q_2\ \sqcup\ \mathtt{1} && \text{(since } \delta(q_1, \sqcup) = (q_2, \mathtt{1}, \mathtt{L})\text{)} \\ &\vdash q_{\text{halt}}\,\mathtt{11} && \text{(since } \delta(q_2, \sqcup) = (q_{\text{halt}}, \mathtt{1}, \mathtt{L})\text{)} \end{aligned}$$

— as required when the input is the string "1", since

$$f_{\text{u\_inc}}(\mathtt{1}) = f_{\text{u\_inc}}(\mathtt{1}^1) = \mathtt{1}^2 = \mathtt{11}.$$

Now consider an execution of the algorithm on an input $\mathtt{1}^n$ for an integer n such that $n \geq 2$. This begins as follows;

$$q_0\,\mathtt{1}^n \vdash \sqcup\, q_1\,\mathtt{1}^{n-1} \qquad \text{(since } \delta(q_0, \mathtt{1}) = (q_1, \sqcup, \mathtt{R})\text{)}.$$

Ideally, the following should not be difficult — the required proof resembles one included in the lecture presentation for Lecture #6.

Exercise: Prove, by induction on i, that — for every integer i such that $0 \leq i \leq n-2$ —

$$\sqcup\, q_1\, 1^{n-1} \vdash^\star \sqcup 1^i\, q_1\, 1^{n-i-1}.$$

It now follows that

$$\begin{aligned} q_0\, 1^n &\vdash \sqcup 1^{n-1}\, q_1 && \text{(since } \delta(q_0, 1) = (q_1, \sqcup, \mathtt{R})\text{)} \\ &\vdash^\star \sqcup 1^{n-2}\, q_2\, 1^2 && \text{(since } \delta(q_1, \sqcup) = (q_2, 1, \mathtt{L})\text{.)} \end{aligned}$$

Once again, the following can be completed by proving a proof resembling one from the lecture presentation for Lecture #6.

Exercise: Prove, by induction on j that — for every integer j such that $0 \leq j \leq n-2$ —

$$\sqcup 1^{n-2}\, q_2\, 1^2 \vdash^\star \sqcup 1^{n-j-2}\, q_2\, 1^{j+2}.$$

Thus

$$\begin{aligned} q_0\, 1^n &\vdash^\star \sqcup 1^{n-2}\, q_2\, 1^2 && \text{(using the derivation as given, so far)} \\ &\vdash^\star \sqcup\, q_2\, 1^n && \text{(by the above result, when } j = n-2\text{)} \\ &\vdash q_2\, \sqcup\, 1^n && \text{(since } \delta(q_2, 1) = (q_2, 1, \mathtt{L})\text{)} \\ &\vdash q_{\mathsf{halt}}\, 1^{n+1} && \text{(since } \delta(q_2, \sqcup) = (q_{\mathsf{halt}}, 1, \mathtt{L})\text{.)} \end{aligned}$$

That is, when $n \geq 2$,

$$q_0\, 1^n \vdash^\star q_{\mathsf{halt}}\, 1^{n+1}$$

— as required when the input string is 1^n for $n \geq 2$.

Examining all the above derivations, one can see that this Turing machine computes the function $f_{\mathsf{u\_inc}}$, as desired.

4 A More Complicated Example: Incrementing the Value of a Number in Binary Notation

4.1 The Problem To Be Solved

Now suppose that $\Sigma_{\mathsf{binary}} = \{0, 1\}$. A string

$$\omega = \alpha_k \alpha_{k-1} \ldots \alpha_2 \alpha_1 \alpha_0 \in \Sigma^\star_{\mathsf{binary}} \tag{2}$$

(with length $k+1$) is the ***unpadded binary representation*** of a non-negative integer if $\omega \neq \lambda$ (so that $k \geq 0$) and either $k = 0$ or $k \geq 1$ and $\alpha_k = 1$ — so that this string is not "padded" with leading copies of 0. In particular, if ω is as shown at line (2), which satisfies this condition, is the ***unpadded binary representation of*** n, for a non-negative integer n, if

$$n = \sum_{i=0}^{k} \alpha_i \cdot 2^i. \tag{3}$$

Let

$$f_{\text{b\_inc}} : \Sigma^{\star}_{\text{binary}} \rightarrow \Sigma^{\star}_{\text{binary}}$$

which is defined as follows:

- If ω is the unpadded binary representation of a non-negative integer n then $f_{\text{b\_inc}}(\omega)$ is the unpadded binary representation of $n+1$.
- If ω is not the unpadded binary representation of any non-negative integer, at all, then $f_{\text{b\_inc}}(\omega) = \lambda$.

Suppose we wish to design a Turing machine that computes functions

$$M = (Q, \Sigma_{\text{binary}}, \Sigma_{\text{binary}}, \Gamma, \delta, q_0, q_{\text{halt}})$$

that computes the above function $f_{\text{b\_inc}}$.

4.2 An Algorithm for This Computation — and Proof of Its Correctness

To begin, consider the algorithm shown in Figure 1 on page 7.

Consider an execution of this algorithm on an input string $\omega \in \Sigma^{\star}_{\text{binary}}$ such that ω is the unpadded binary representation of a natural number n.

- If $n = 0$ then $f_{\text{b\_inc}}(\omega) = 1$, the unpadded binary representation of 1.

  When the algorithm is executed on input ω, the step at line 1 is checked and failed, since $\omega \neq \lambda$. The test at line 3 is reached and passed, since ω begins with "0". The test at line 4 is checked next and is also passed, since $|\omega| = 1$, and the desired string "1" is returned as output when the step at line 5 is reached and executed.

- Suppose, next, that $n \geq 1$, so that ω begins with "1". Either $n = 2^h - 1$ for some positive integer h — so that $\omega = 1^h$ — or n does not have this form and ω includes at least one copy of "0". These cases are considered separately, below.

On input $\omega \in \Sigma^{\star}_{\text{binary}}$:

1. `if` $(\omega == \lambda)$ `{`
2. `return` λ
3. `}` `else if` (ω starts with "0") `{`
4. `if` $(|\omega| == 1)$ `{`
5. `return` 1
   `}` `else {`
6. `return` λ
   `}`
7. `}` `else if` (ω includes at least one copy of "0") `{`
8. Return the string obtained from ω by replacing the ***rightmost*** copy of "0" with"1", and by replacing every copy of "1" to the right of that symbol with "0"
   `}` `else {`
9. `return` 10^h, where $h = |\omega|$
   `}`

Figure 1: Algorithm to Compute the Function $f_{\text{b\_inc}}$

– If $n = 2^h - 1$, so that $\omega = 1^h$ then $f(\omega) = 10^h$, the unpadded binary representation of 2^h.

  In this case, when the algorithm is executed on input ω the test at line 1 fails, since $\omega \neq \lambda$. The test at line 3 also fails, since ω does not begin with "0". The test at line 7 is checked next, and also fails, since ω does not include any copies of "0". Thus the step at line 9 is reached and executed — and one can see, by an examination of this step, that the execution of the algorithm ends with $f_{\text{b\_inc}}(\omega)$ returned as output as desired, in this case.

– Otherwise ω includes at least one copy of "0", and (considering the rightmost copy of "0" in ω),

$$\omega = 1\mu 01^h$$

  for some string $\mu \in \Sigma^{\star}_{\text{binary}}$ and for some non-negative integer h. In this case, $f_{\text{b\_inc}}(\omega)$ is the unpadded binary representation of $n + 1$, so that

$$f_{\text{b\_inc}}(\omega) = 1\mu 10^h.$$

  Thus $f_{\text{b\_inc}}(\omega)$ can be obtained from ω by replacing the ***rightmost*** copy of "0" in ω with "1", and replacing every copy of "0" to the right of this symbol with "0".

In this case, when the algorithm is executed on input ω, the test at line 1 fails, since $\omega \neq \lambda$. The test at line 3 also fails, since ω does not begin with "0". The test at line 7 is checked next, and is passed, since ω includes at least one copy of "0". Thus the step at line 8 is reached and executed — and one can see, by an examination of this step, that the execution of the algorithm ends with $f_{\text{b\_inc}}(\omega)$ returned as output as desired in this case, as well.

Thus the execution of the algorithm ends, with $f_{\text{b\_inc}}(\omega)$ returned as output, whenever ω is the unpadded binary representation of a non-negative integer.

Now consider an execution of this algorithm on an input string $\omega \in \Sigma^{\star}_{\text{binary}}$ such that ω is not the unpadded binary representation of any natural number n — so that $f_{\text{b\_inc}}(\omega) = \lambda$. Either $\omega = \lambda$, or ω is a string with length at least two that begins with "0".

- If $\omega = \lambda$, and the algorithm is executed on input ω, then the test at line 1 is checked and passed, so that the desired output, ω is returned when the step at line 2 is reached and executed.
- If ω is a string with length at least two that begins with "0" and the algorithm is executed on input ω, then the step at line 1 is checked and failed, since $\omega \neq \lambda$. The test at line 3 is checked and passed, since ω begins with "0". Since $|\omega| \geq 2$ the test at line 4 is failed when reached and checked. Thus the step at line 6 is reached and executed and the required output, λ is returned in this case as well.

Thus the execution of the algorithm ends, with $f_{\text{b\_inc}}(\omega)$ returned as output, whenever ω *is not* the unpadded binary representation of any non-negative integer n.

Thus this algorithm correctly computes the function $f_{\text{b\_inc}}$.

4.3 Implementation-Level Description of a Turing Machine That Computes This Function

4.3.1 Tape Alphabet

It will be necessary for a Turing machine that implements this algorithm to sweep to the right and then back to the left — in such a way that the leftmost cell of the tape can be detected. A new tape symbol, "X", will be used for this: X will be written onto the leftmost cell of the tape the first time that the Turing machine's tape head sweeps to the right, and it will be kept there until just before the computation ends. Since this symbol will not be written on any other cell, it will be possible, during the computation, to check whether the leftmost cell of the tape has been reached by checking whether the symbol X is visible.

It will turn out that this is the only new symbol needed, so that the tape alphabet used can be the alphabet

$$\Gamma = \Sigma_{\text{binary}} \cup \{\sqcup, \mathtt{X}\} = \{\mathtt{0}, \mathtt{1}, \mathtt{X}, \sqcup\}, \tag{4}$$

4.3.2 Implementing the Algorithm in Figure 1

Consider what the Turing machine should do, in order to implement the algorithm shown in Figure 1, when executed with an input string $\omega \in \Sigma^{\star}_{\text{binary}}$. As described below, this algorithm can be used to produce an implementation-level of a Turing machine that is shown in Figure 2 on page 10.

1. Since the input string is written on the leftmost cells of the tape when an execution of the Turing machine begins, with the tape head pointing to the leftmost cell, the test (whether $\omega = \lambda$) at line of the algorithm in Figure 1 can be implemented by checking whether $\sqcup$ is visible — with the test passing if it is, and with the test failing otherwise. This is shown at line 1 in the algorithm in Figure 2.

2. The step at line 2 is reached if and only if this test has been passed. Since the empty string should be returned if this step is reached — so that the tape should be filled with blanks and the tape head should be back at the left most cell — step 2 of the algorithm in Figure 1. This is shown at line 2 in the algorithm in Figure 2.

3. The test whether ω begins with $\mathtt{0}$, at line 3 of the algorithm in Figure 1, is reached if the first test failed. At this point the tape head is still pointing to the leftmost cell of the tape, so this test can be implemented by checking whether $\mathtt{0}$ is visible — with the test passing if it is, and failing otherwise. This is shown at line 3 in the algorithm in Figure 2.

4. The test whether $|\omega| = 1$, at line 4 of the algorithm in Figure 1, is reached if and only if the test at line 3 was executed and passed. This test can be implemented by moving right — writing $\mathtt{X}$, so that the leftmost cell can be found again later — and checking whether $\sqcup$ is visible (on the second cell of the tape) — with the test passing if $\sqcup$ is visible, and with the test failing otherwise. This is shown at lines 4 and 5 in the algorithm in Figure 2.

5. The step at line 5 of the algorithm in Figure 1 is reached if and only if the test at line 4 was executed and passed. Since the tape head is at the second cell of the tape, when it confirmed that $\sqcup$ is visible there, and since $\mathtt{1}$ is to be returned as output, it suffices to move the tape head left — wring $\sqcup$, so that the symbol on the second cell is not changed, replace the copy of $\mathtt{X}$ on the leftmost cell with $\mathtt{1}$, moving left again (so that the tape head stays at the leftmost cell), and halt. This is shown at lines 6 and 7 in the algorithm in Figure 2.

6. The step at line 6 of the algorithm in Figure 1 is reached if and only if the test at line 4 was executed and failed, so that a non-blank symbol is visible on the second cell of the tape.

```
1.  if (␣ is visible) {
2.    Write ␣, moving left, and halt.
3.  } else if (0 is visible) {
4.    Write X, moving right.
5.    if (␣ is visible) {
6.      Write ␣, moving left.
7.      Write 1, moving left, and halt.
      } else {
8.      Sweep right, without changing symbols on the tape, until ␣ is
        visible.
9.      Sweep left, replacing every symbol with ␣, until X is visible.
10.     Write ␣, moving left, and halt.
      }
    } else {
11.   Sweep right, without changing symbols on the tape, until either 0
      or ␣ is visible.
12.   if (0 is visible) {
13.     Sweep right, without changing symbols on the tape, until ␣ is
        visible.
14.     Write ␣, moving left.
15.     Sweep left, replacing every copy of 1 seen with 0, until 0 is seen.
16.     Write 1, moving left.
17.     Sweep left, without changing symbols on the tape, until X is
        seen.
18.     Write 1, moving left, and halt.
      } else {
19.     Write 0, moving left.
20.     Sweep left, replacing every symbol of 1 seen with 0, until X is
        seen.
21.     Write 1, moving left, and halt.
      }
    }
```

Figure 2: Implementation-Level Description

Since the empty string is to be returned the tape head should now be filled with copies

of ⊔ and the tape head moved back to the leftmost cell. This can be accomplished by sweeping right on the tape (without having to change symbols) until the copy of ⊔, to the right of the input string, is visible. The tape head should sweep left, replacing each symbol seen with ⊔, until the copy of X that marks the leftmost cell is visible. The Turing machine should then write ⊔, trying to move left again (so that the tape head stays at the leftmost cell of the tape), and halt. This is shown at lines 8, 9, and 10 in the algorithm in Figure 2.

7. The test at line 7 of the algorithm in Figure 1 is reached if and only if the tests at lines 1 and 3 both failed — so that 1 must be visible (at the leftmost cell of the tape). Since it is checking whether the input string includes a copy of 0, the tape head must move right — writing X at the leftmost cell (so that it can be found again) and leaving other symbols unchanged — until either 0 or ⊔ is seen (since the only other symbol that can be seen, during this sweep to the right, is 1). The test passes if 0 is seen before ⊔ is seen, during this sweep to the right, and it fails otherwise. This is shown at lines 11 and 12 in the algorithm in Figure 2.

8. The step at line 8 of the algorithm in Figure 1 is reached if and only the test at line 7 has been checked and passed — so that a copy of 0 has been discovered. Since this might not be the *rightmost* copy of 0 (which is mentioned in the step at line 8) the tape head should continue to move right, without changing symbols on the tape, until ⊔ is seen. This copy of ⊔ should not be changed (so it should be replaced by another copy of ⊔ as the tape head moves back to the left). The tape head should keep moving left while copies of 1 are seen, replacing each with a copy of 0.

   This ends when 0 is seen (since this only step has been reached if at least one copy of 0 is on the tape) and this should be replaced with 1 as the tape head keeps moving left, in order to make the change described in the step at line 8.

   Since nothing else in the input string is to be changed, the tape head should continue to move left, without changing the symbols that are visible, until the copy of X at the leftmost cell is visible once again. Since a copy of 1 was initially visible at this cell of the tape, the copy of X should be replaced by a copy of 1, as the tape head tries to move left, and the computation should end.

   This is shown at lines 13, 14, 15, 16, 17 and 18 of the algorithm in Figure 2.

9. Finally, the step at line 9 of the algorithm in Figure 1 is reached if and only the test at line 7 has been checked and failed, and the leftmost copy of ⊔, to the right of the input string has been detected. This can only happen if $\omega = 1^h$ for some positive integer h. Now, since the string to be returned has length $h + 1$, the ⊔ that is visible should be replace by a copy of 0 and the tape head should move left, replacing each copy of 1 that is seen with a copy of 0. This ends when the copy of X, marking the leftmost cell of the tape, is visible. This should be replace with a copy of 1, with the tape should try to move

left (so that it stays at the leftmost cell of the tape) and the computation should end — in order to complete the implementation of this step in the algorithm in Figure 1.

This is shown at lines 19, 20 and 21 of the algorithm in Figure 2.

Since the implementation-level description is an implementation of the algorithm in Figure 1 — it simply describes how to update the tape and move the tape head in order to carry out each step of the algorithm in Figure 1 — correctness of the implementation-level description is a consequence of the correctness of the algorithm in Figure 1.

4.4 Formal Description of a Turing Machine That Computes This Function

Now consider a Turing machine

$$M = (Q, \Sigma_{\text{binary}}, \Sigma_{\text{binary}}, \Gamma, \delta, q_0, q_{\text{halt}}),$$

with a set of states

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{\text{halt}}\},$$

a tape alphabet as shown at line (4), above, and an incomplete transition diagram as shown in Figure 3 on page 13. The halt state, q_{halt}, and transitions leading to it, are not shown in Figure 3. The missing transitions, leading to the halting state, are as follows.

(a) $\delta(q_0, \sigma) = (q_{\text{halt}}, \sqcup, \texttt{L})$ for $\sigma = \sqcup$ and $\sigma = \texttt{X}$.

(b) $\delta(q_1, \texttt{X}) = (q_{\text{halt}}, \sqcup, \texttt{L})$.

(c) $\delta(q_2, \sigma) = (q_{\text{halt}}, \texttt{1}, \texttt{L})$ for $\sigma \in \Gamma$.

(d) $\delta(q_3, \texttt{X}) = (q_{\text{halt}}, \sqcup, \texttt{L})$.

(e) $\delta(q_4, \sigma) = (q_{\text{halt}}, \sqcup, \texttt{L})$ for $\sigma = \sqcup$ and $\sigma = \texttt{X}$.

(f) $\delta(q_6, \texttt{X}) = (q_{\text{halt}}, \sqcup, \texttt{L})$.

(g) $\delta(q_7, \sigma) = (q_{\text{halt}}, \sqcup, \texttt{L})$ for $\sigma = \sqcup$ and $\sigma = \texttt{X}$.

(h) $\delta(q_8, \sigma) = (q_{\text{halt}}, \texttt{1}, \texttt{L})$ for $\sigma = \sqcup$ and $\sigma = \texttt{X}$.

(i) $\delta(q_9, \sigma) = (q_{\text{halt}}, \texttt{1}, \texttt{L})$ for $\sigma = \sqcup$ and $\sigma = \texttt{X}$.

4.5 Proof of Correctness

As explained below, this Turing machine is an implementation of the algorithm in Figure 2, and a proof of its correctness can be based on the proof of the correctness of this algorithm (which is, in turn, based on the correctness of the algorithm in Figure 1.

To begin, consider an input string $\omega \in \Sigma^\star_{\text{binary}}$ that is the unpadded binary representation of a non-negative integer n.

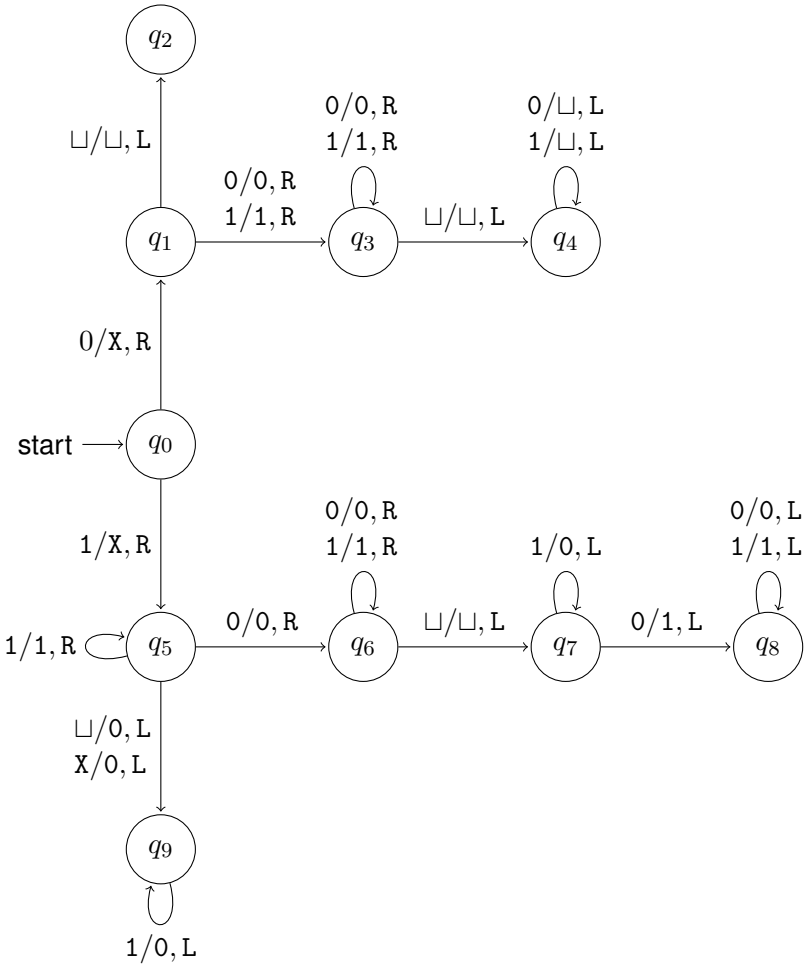


Figure 3: Incomplete State Diagram for a Turing Machine That Computes the Function $f_{\text{b\_inc}}$

- If $n = 0$ then

$$
\begin{aligned}
q_0\,\omega &= q_0\,\mathtt{0} \\
&\vdash \mathtt{X}\,q_1 && \text{(since } \delta(q_0, \mathtt{0}) = (q_1, \mathtt{X}, \mathtt{R})\text{)} \\
&\vdash q_2\,\mathtt{X} && \text{(since } \delta(q_1, \sqcup) = (q_2, \sqcup, \mathtt{L})\text{)} \\
&\vdash q_{\text{halt}}\,\mathtt{1} && \text{(since } \delta(q_2, \mathtt{X}) = (q_{\text{halt}}, \mathtt{1}, \mathtt{L})\text{).}
\end{aligned}
$$

  Thus $f_{\text{b\_inc}}(\omega) = \mathtt{1}$, the unpadded binary representation of 1, is computed in this case — as desired.

  Note that the transition "$\delta(q_0, \mathtt{0}) = (q_1, \mathtt{X}, \mathtt{R})$ is, effectively, an implementation of the test

at line 1 of the algorithm in Figure 2 — and its failure, when the output begins with "$\mathtt{0}$", along with a successful execution of the test at line 3 of the algorithm in Figure 2, and an implementation of the step at line 4. The transition "$\delta(q_1, \sqcup) = (q_2, \sqcup, \mathtt{L})$ is, effectively, an implement of a successful application of the test at line 5 and the step at line 6 of the algorithm — and the transition "$\delta(q_2, \mathtt{X}) = (q_{\mathsf{halt}}, \mathtt{1}, \mathtt{L})$" is, effectively, an implementation of the step at line 7.

- If $n \geq 1$, and $n = 2^h - 1$, for a positive integer h, then $\omega = \mathtt{1}^h$. Suppose, first, that $h = 1$, so that $n = 1$ and $\omega = \mathtt{1}$. Then

$$\begin{aligned} q_0\,\omega &= q_0\,\mathtt{1} \\ &\vdash \mathtt{X}\,q_5 && \text{(since } \delta(q_0, \mathtt{1}) = (q_5, \mathtt{X}, \mathtt{R})\text{)} \\ &\vdash q_9\,\mathtt{X0} && \text{(since } \delta(q_5, \sqcup) = (q_9, \mathtt{0}, \mathtt{L})\text{)} \\ &\vdash q_{\mathsf{halt}}\,\mathtt{10} && \text{(since } \delta(q_9, \mathtt{X}) = (q_{\mathsf{halt}}, \mathtt{1}, \mathtt{L})\text{).} \end{aligned}$$

  Thus $f_{\mathsf{b\_inc}}(\mathtt{1}) = \mathtt{10}$, the unpadded binary representation of 2, is computed in this case — as desired.

  Note that the transition "$\delta(q_0, \mathtt{1}) = (q_5, \mathtt{X}, \mathtt{R})$" is, effectively, an implementation of the test at line 1 of the algorithm in Figure 2 — and its failure, when the output begins with "$\mathtt{1}$", along with a failed execution of the test at line 3 of the algorithm. The steps in this execution, after the first one, can be seen as an implementation of the step at line 11 of the algorithm in Figure 2 — for the case that $\sqcup$ is seen immediately, so that the test at line 12 is failed — and the steps at lines 19, 20 and 21 (with the condition checked for at line 20 also being satisfied, immediately, so that no copies of $\mathtt{1}$ are written, at all, until the step at line 21 is executed).

  Suppose, instead, that $h \geq 2$. Then

$$\begin{aligned} q_0\,\omega &= q_1\,\mathtt{1}^h \\ &\vdash \mathtt{X}\,q_5\,\mathtt{1}^{h-1} && \text{(since } \delta(q_1, \mathtt{1}) = (q_5, \mathtt{X}, \mathtt{R})\text{).} \end{aligned}$$

  Since $\delta(q_5, \mathtt{1}) = (q_5, \mathtt{1}, \mathtt{R})$ it is possible to prove by induction on i that, for every integer i such that $0 \leq i \leq h - 2$,

$$\mathtt{X}q_5\mathtt{1}^{h-1} \vdash^\star \mathtt{X1}^i\,q_5\mathtt{1}^{h-i-1}.$$

  Thus

$$\begin{aligned} q_0\,\omega &\vdash^\star \mathtt{X}\,q_5\,\mathtt{1}^{h-t} && \text{(as noted above)} \\ &\vdash^\star \mathtt{X1}^{h-2}\,q_5\,\mathtt{1} && \text{(by the above result, with } i = h-2\text{)} \\ &\vdash \mathtt{X1}^{h-1}\,q_5 && \text{(since } \delta(q_5, \mathtt{1}) = (q_5, \mathtt{1}, \mathtt{R})\text{)} \\ &\vdash \mathtt{X1}^{h-2}\,q_9\,\mathtt{10} && \text{(since } \delta(q_5, \sqcup) = (q_9, \mathtt{0}, \mathtt{L})\text{).} \end{aligned}$$

Since $\delta(q_9, \mathtt{1}) = (q_9, \mathtt{0}, \mathtt{L})$ it is possible to prove by induction on i that, for every integer i such that $1 \leq i \leq h-2$,

$$X\mathtt{1}^{h-2}\, q_9\, \mathtt{10} \vdash^\star \mathtt{X1}^{h-i-1}\, q_9\, \mathtt{10}^i.$$

Thus

$$\begin{aligned}
q_0\, \omega &\vdash^\star \mathtt{X1}^{h-2}\, q_9\, \mathtt{10} && \text{(as noted above)}\\
&\vdash^\star \mathtt{X1}\, q_9\, \mathtt{10}^{h-2} && \text{(by the above result, with } i = h-2\text{)}\\
&\vdash \mathtt{X}\, q_9\, \mathtt{10}^{h-1} && \text{(since } \delta(q_9, \mathtt{1}) = (q_9, \mathtt{0}, \mathtt{L})\text{)}\\
&\vdash q_9\, \mathtt{X}\, \mathtt{0}^h && \text{(since } \delta(q_9, \mathtt{1}) = (q_9, \mathtt{0}, \mathtt{L})\text{)}\\
&\vdash q_{\mathsf{halt}}\, \mathtt{10}^h && \text{(since } \delta(q_9, \mathtt{X}) = (q_{\mathsf{halt}}, \mathtt{1}, \mathtt{L})\text{).}
\end{aligned}$$

Thus $f_{\mathsf{b\_inc}}(\omega) = \mathtt{10}^h$, the binary representation of 2^h, is computed in this case — as desired..

Note, now, that state q_5 is reached when the test at line 11 of the algorithm is to be executed and that the transitions out of this state are being used to carry out this test by sweeping right over the input string, searching for a copy of $\mathtt{0}$. If no such copy of $\mathtt{0}$is every found — as in the above case (so that $\sqcup$ is discovered after a sequence of copies of $\mathtt{1}$) — then a transition is followed to state q_9 (writing $\mathtt{0}$ and moving left, to implement the step at line 19 of the algorithm in Figure 2), and a sweep back to the left is used to produce the desired output string — implementing the steps at lines 20 and 21 of the algorithm in Figure 2, — as shown above.

- Otherwise ω includes at least one copy of "$\mathtt{0}$" and (considering the rightmost copy of "$\mathtt{0}$" in ω),

  $$\omega = \mathtt{1}\mu\mathtt{01}^h$$

  for some string $\mu \in \Sigma^\star_{\mathsf{binary}}$ and for some non-negative integer h. As noted in the analysis of the algorithm in Figure 1,

  $$f_{\mathsf{b\_inc}}(\omega) = \mathtt{1}\mu\mathtt{10}^h$$

  in this case.

  Notice that — considering the *leftmost* copy of "$\mathtt{0}$" in ω instead of the rightmost copy — one can see that

  $$\omega = \mathtt{1}\mu\mathtt{01}^h = \mathtt{1}^\ell\mathtt{0}\nu$$

for a positive integer ℓ and for a string $\nu \in \Sigma^\star_{\text{binary}}$. It is now possible to show that

$$
\begin{aligned}
q_0\,\omega &= q_0 \mathtt{1}^\ell \mathtt{0}\nu \\
&\vdash \mathtt{X}\,q_5\,\mathtt{1}^{\ell-1}\mathtt{0}\nu && \text{(since } \delta(q_0,\mathtt{1}) = (q_5,\mathtt{X},\mathtt{R})\text{)} \\
&\vdash^\star \mathtt{X1}^{\ell-1}\,q_5\,\mathtt{0}\nu && \text{(since } \delta(q_5,\mathtt{1}) = (q_5,\mathtt{1},\mathtt{R})\text{)} \\
&\vdash \mathtt{X1}^{\ell-1}\mathtt{0}\,q_6\,\nu && \text{(since } \delta(q_5,\mathtt{0}) = (q_6,\mathtt{0},\mathtt{R})\text{)} \\
&\vdash^\star \mathtt{X1}^{\ell-1}\mathtt{0}\nu\,q_6 && \text{(since } \delta(q_6,\sigma) = (q_6,\sigma,\mathtt{R}) \text{ for } \sigma \in \Sigma_{\text{binary}}\text{)} \\
&= \mathtt{X}\mu\mathtt{01}^h q_6 && \text{(since } \mu \text{ and } \nu \text{ are related as described above).}
\end{aligned}
$$

Exercise: Notice that this derivation included two steps including a sequence of zero or moves, instead of one. Form statements that can be proved, by mathematical induction (like ones used above) that can be proved, in order to show that this derivation is correct.

Thus the transition "$\delta(q_5,\mathtt{1}) = (q_5,\mathtt{1},\mathtt{R})$" and the transition "$\delta(q_5,\mathtt{0}) = (q_6,\mathtt{0},\mathtt{R})$" implement the step at line 11 of the algorithm in Figure 2 and a successful execution of the test at line 12. The transitions out of state q_6 implement the steps at lines 13 and 14. The Turing machine enters state q_7 after the copy of $\sqcup$, immediately to the right of the input string, has been seen — and the tape head moves left when carrying out this step.

The Turing machine remains in state q_7 while the rightmost copies of "$\mathtt{1}$" are seen — with each copy of "$\mathtt{1}$" replaced by a copy of "$\mathtt{0}$". Once the rightmost copy of "$\mathtt{0}$" in the input string is seen, is seen this this replaced by a copy of "$\mathtt{1}$" as the sweep left continues — with the Turing machine now moving to state q_8. The sweep left continues (without changing symbols on the tape) until the copy of "$\mathtt{X}$"' at the left end of the tape — and the transition "$\delta(q_8,\mathtt{X}) = (q_{\text{halt}},\mathtt{1},\mathtt{L})$" is applied so that the computation ends in a configuration

$$q_{\text{halt}}\,\mathtt{1}\mu\mathtt{10}^h$$

— so that the required string $f_{\text{b\_inc}}(\omega)$ has been computed in this case too.

Exercise: Use this description to write a proof of the correctness of this part of the Turing machine that is (more-or-less) as detailed as the parts of the proof that came before it.

Next consider an input string $\omega \in \Sigma^\star$ that is *not* the unpadded binary representation of any non-negative integer — so that either $\omega = \lambda$ or ω is a string with length at least two that begins with "$\mathtt{0}$".

- Suppose that $\omega = \lambda$. Then

$$q_0 \vdash q_{\text{halt}} \qquad \text{(since } \delta(q_0,\sqcup) = (q_{\text{halt}},\sqcup,\mathtt{L})\text{)}$$

— and the empty string is returned as output, as required.

Note that the transition "$\delta(q_0, \sqcup) = (q_{\text{halt}}, \sqcup, \mathtt{L})$" is an implementation of a successful execution of the test at line 1 in the algorithm in Figure 2, along with an execution of the step at line 2.

- Suppose, instead, that ω is a string with length at least two that begins with "0", so that

$$\omega = 0\alpha_1\alpha_2 \ldots \alpha_h$$

for a positive integer h and symbols $\alpha_1, \alpha_2, \ldots, \alpha_h \in \Sigma_{\text{binary}}$. Then

$$\begin{aligned} q_0\,\omega &= q_0\,0\alpha_1\alpha_2 \ldots \alpha_h \\ &\vdash \mathtt{X}\,q_1\,\alpha_1\alpha_2 \ldots \alpha_h && \text{(since } \delta(q_0, 0) = (q_1, \mathtt{X}, \mathtt{R})\text{)} \\ &\vdash \mathtt{X}\alpha_1\,q_3\,\alpha_2\alpha_3 \ldots \alpha_h && \text{(since } \alpha_1 \in \Sigma_{\text{binary}}\text{, so that } \delta(q_1, \alpha_1) = (q_3, \alpha_1, \mathtt{R})\text{).} \end{aligned}$$

Exercise: Examining the transitions out of states q_3 and q_4 (and using previous parts of this document, as needed), confirm that the Turing machine sweeps right over the input, while in state q_3, and moves to state q_4 when the end of the input has been reached. It then sweeps left, while in state q_4, replacing every symbol seen with "$\sqcup$", until the copy of "X" is seen. and the transition "$\delta(q_4, \mathtt{X}) = (q_{\text{halt}}, \sqcup, \mathtt{L})$ is applied: The computation then ends in configuration

$$q_{\text{halt}}$$

so that the desired, output, $f_{\text{b\_inc}}(\omega) = \lambda$, has been computed in this case too.

Since the Turing machine computes the required output, $f_{\text{b\_inc}}(\omega)$ in every possible case, this Turing machine computes the function $f_{\text{b\_inc}}$, as claimed.

Note: While this proof is long — if you have worked through and completed at least some of the "exercises" included in its description — then it should be comprehensible.

Understanding this Turing machine, and writing a proof that it computes the desired function, would have been ***much*** more difficult if an algorithm for the computation had not been presented, analyzed, and used to develop and analyze the Turing machine after that.

It is, generally, advisable to solve a problem "at a high level" so that you have more to work with, before adding details needed to produce a Turing machine for a computation (along with a proof that it does what it is supposed to).