

Lecture #11: Simulations and Simple Turing Machine Variants

Turing Machines That Compute Functions

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

$$f : \Sigma_1^* \rightarrow \Sigma_2^*$$

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 – 5.

1 Definition

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

$$f : \Sigma_1^* \rightarrow \Sigma_2^*,$$

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 way 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 finite many other 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

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

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, $\omega \in \Sigma_1^*$, 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 $f : \Sigma_1^* \rightarrow \Sigma_2^*$ then the following properties are satisfied, for every string $\omega \in \Sigma_1^*$:

- 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 $f : \Sigma_1^* \rightarrow \Sigma_2^*$ 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 $1^n \in \Sigma_{\text{unary}}^*$ can be used as the **unary representation** of n , for every non-negative integer n . Let

$$f_{\text{u_inc}} : \Sigma_{\text{unary}}^* \rightarrow \Sigma_{\text{unary}}^*$$

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 “ $\sqcup$ ” 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 “ $\sqcup$ ” is found. This should be replaced 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.

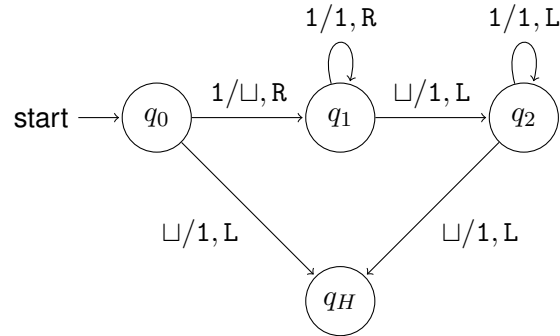
There are multiple ways to mark the leftmost cell of the tape. One way that that does not always work — but that works for this problem — is to replace the symbol with “ $\sqcup$ ”.

3.3 A Turing Machine That Computes This Function

Using this approach one can set

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

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

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

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

$$f_{\text{u_inc}}(\lambda) = f_{\text{u_inc}}(1^0) = 1^1 = 1.$$

Note, as well, that

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

— as required when the input is the string “1”, since

$$f_{\text{u_inc}}(1) = f_{\text{u_inc}}(1^1) = 1^2 = 11.$$

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

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

Ideally, the following should not be difficult — the required proof resembles one included in the lecture presentation for Lecture #10.

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

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

It now follows that

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

Once again, the following can be completed by proving a proof resembling one from the lecture presentation for Lecture #10.

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

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

Thus

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

That is, when $n \geq 2$,

$$q_0 1^n \vdash^* q_{\text{halt}} 1^{n+1}$$

— 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 $f_{\text{u_inc}}$, 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 $\Sigma_{\text{binary}} = \{0, 1\}$. A string

$$\omega = \alpha_k \alpha_{k-1} \dots \alpha_2 \alpha_1 \alpha_0 \in \Sigma_{\text{binary}}^* \tag{2}$$

(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

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

Let

$$f_{b_inc} : \Sigma_{\text{binary}}^* \rightarrow \Sigma_{\text{binary}}^*$$

which is defined as follows:

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

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_{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 $\omega \in \Sigma_{\text{binary}}^*$ such that ω is the unpadded binary representation of a natural number n .

- If $n = 0$ then $f_{b_inc}(\omega) = 1$, 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 $\omega \in \Sigma_{\text{binary}}^*$:

1. if ($\omega == \lambda$) {
2. return λ
3. } else if (ω starts with “0”) {
4. if ($|\omega| == 1$) {
5. return 1
6. } else {
7. return λ
8. }
9. } else if (ω includes at least one copy of “0”) {
10. 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”
11. } else {
12. return 10^h , where $h = |\omega|$
13. }

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 $\mu \in \Sigma_{\text{binary}}^*$ 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 $f_{b_inc}(\omega)$ returned as output as desired in this case, as well.

Thus the execution of the algorithm ends, with $f_{b_inc}(\omega)$ 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 $\omega \in \Sigma_{binary}^*$ such that ω is not the unpadded binary representation of any natural number n — so that $f_{b_inc}(\omega) = \lambda$. 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 $f_{b_inc}(\omega)$ 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_{b_inc} .

4.3 A Turing Machine That Computes This Function

Now consider a Turing machine

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

with a set of states

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

a tape alphabet

$$\Gamma = \Sigma_{binary} \cup \{\sqcup, X\} = \{0, 1, X, \sqcup\},$$

and an incomplete transition diagram as shown in Figure 2 on page 9. The halt state, q_{halt} , and transitions leading to it, are not shown in Figure 2. The missing transitions, leading to the halting state, are as follows.

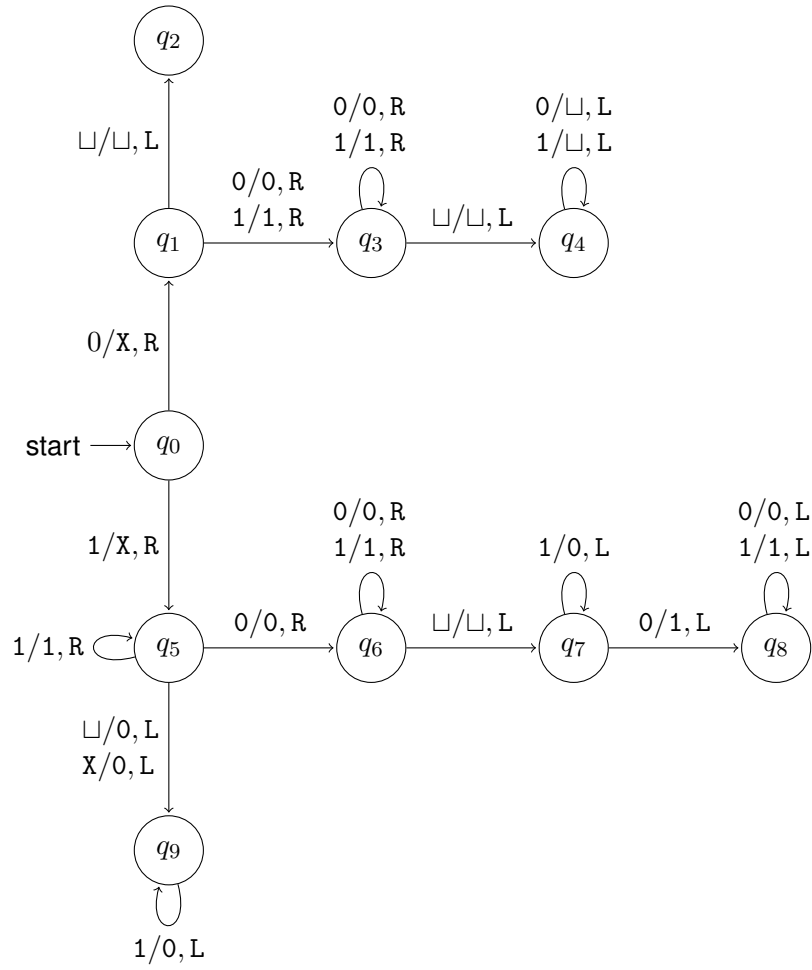


Figure 2: Incomplete State Diagram for a Turing Machine That Computes the Function f_{b_inc}

- (a) $\delta(q_0, \sigma) = (q_{halt}, \sqcup, L)$ for $\sigma = \sqcup$ and $\sigma = X$.
- (b) $\delta(q_1, X) = (q_{halt}, \sqcup, L)$.
- (c) $\delta(q_2, \sigma) = (q_{halt}, 1, L)$ for $\sigma \in \Gamma$.
- (d) $\delta(q_3, X) = (q_{halt}, \sqcup, L)$.
- (e) $\delta(q_4, \sigma) = (q_{halt}, \sqcup, L)$ for $\sigma = \sqcup$ and $\sigma = X$.
- (f) $\delta(q_6, X) = (q_{halt}, \sqcup, L)$.
- (g) $\delta(q_7, \sigma) = (q_{halt}, \sqcup, L)$ for $\sigma = \sqcup$ and $\sigma = X$.
- (h) $\delta(q_8, \sigma) = (q_{halt}, 1, L)$ for $\sigma = \sqcup$ and $\sigma = X$.

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

4.4 Proof of Correctness

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

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

- If $n = 0$ then

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

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

Note that the transition “ $\delta(q_0, 0) = (q_1, X, R)$ ” is, effectively, an implementation of the test at line 1 of the algorithm — and its failure, when the output begins with “0”, along with a successful execution of the test at line 3 of the algorithm. The transition “ $\delta(q_1, \sqcup) = (q_2, \sqcup, L)$ ” is, effectively, an implement of a successful application of the test at line 4 of the algorithm — and the transition “ $\delta(q_2, X) = (q_{\text{halt}}, 1, L)$ ” is, effectively, an implementation of the step at line 5.

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

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

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

Note that the transition “ $\delta(q_0, 1) = (q_5, X, R)$ ” is, effectively, an implementation of the test at line 1 of the algorithm — and its failure, when the output begins with “1”, along with a failed execution of the test at line 3 of the algorithm for this case. The steps in this execution, after the first one, can be seen as an implementation of a failed execution

of the test at line 7 of the algorithm, and an implementation of the step at line 9 of the algorithm, for this input.

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

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

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

$$X q_5 1^{h-1} \vdash^* X 1^i q_5 1^{h-i-1}.$$

Thus

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

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

$$X 1^{h-2} q_9 10 \vdash^* X 1^{h-i-1} q_9 10^i.$$

Thus

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

Thus $f_{b_inc}(\omega) = 10^h$, 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 7 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 “0”. If no such copy of “0” is every found — in the above case — then a transition is followed to state q_9 , and a sweep back to the left is used to produce the desired output string, as shown above.

If a copy of “0” is found, so that the test at line 7 should pass, then a transition is used to move from state q_6 instead of to state q_9 , and the computation is as described in the next 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 $\mu \in \Sigma_{\text{binary}}^*$ and for some non-negative integer h . As noted in the analysis of the above algorithm,

$$f_{\text{b_inc}}(\omega) = 1\mu 10^h$$

in this case.

Notice that — considering the *leftmost* copy of “0” in ω instead of the rightmost copy — one can see that

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

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

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

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 “ $\delta(q_5, 0) = (q_6, 0, \mathbf{R})$ ” and the transitions out of state q_6 implement the test at line 7 of the algorithm — for the case where this succeeds — and a sweep to the right over the input string, so that the step at line 8 can be implemented using a sweep back to the left — which begins as soon as “ $\sqcup$ ” as seen and the Turing machine moves to state q_7 .

The Turing machine remains in state q_7 while the rightmost copies of “1” are seen — with each copy of “1” replaced by a copy of “0”. Once the rightmost copy of “0” in the input string is seen, is seen this is replaced by a copy of “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 “ $\mathbf{x}$ ” at the left end of the tape — and the transition “ $\delta(q_8, \mathbf{x}) = (q_{\text{halt}}, 1, \mathbf{L})$ ” is applied so that the computation ends in a configuration

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

— so that the required string $f_{b_inc}(\omega)$ 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 $\omega \in \Sigma^*$ 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 “0”.

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

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

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

Note that the transition “ $\delta(q_0, \sqcup) = (q_{halt}, \sqcup, L)$ ” is an implementation of a successful execution of the test at line 1 in the algorithm, 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

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

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

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

Exercise: Examining the transitions out of states q_3 and q_4 (and using previous parts of this guide, 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 right, while in state q_4 , replacing every symbol seen with “ $\sqcup$ ”, until the copy of “X” is seen. and the transition “ $\delta(q_4, X) = (q_{halt}, \sqcup, L)$ ” is applied: The computation then ends in configuration

$$q_{halt}$$

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

Since the Turing machine computes the required output, $f_{b_inc}(\omega)$ in every possible case, this Turing machine computes the function f_{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).

5 A Suggested Exercise: Decrementing the Value of a Number in Binary Notation

5.1 The Problem To Be Solved

Suppose that the alphabet Σ_{binary} , and unpadded binary representations of non-negative integers, are defined as at the beginning of Section 4 (see, in particular, the equations at lines (2) and (3)).

Let

$$f_{\text{b_dec}} : \Sigma_{\text{binary}}^* \rightarrow \Sigma_{\text{binary}}^*$$

which is defined as follows:

- If ω is the unpadded binary representation of a *positive* integer n then $f_{\text{b_dec}}(\omega)$ is the unpadded binary representation of $n - 1$.
- If either $\omega = 0$ or ω is not the unpadded binary representation of any non-negative integer, at all, then $f_{\text{b_dec}}(\omega) = \lambda$.

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 $f_{\text{b_dec}}$.

5.2 An Algorithm for This Computation — and a Proof of Its Correctness

To continue one should note that — for this computation — the function’s value is λ whenever ω *does not* begin with “1”.

On the other hand, if ω begins with “1” then ω is the unpadded binary representation of some positive integer n .

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

- If $n = 2^h$, for some positive integer h , then $\omega = 10^h$ and $f_{\text{b_dec}}(\omega) = 1^h$, the unpadded binary representation of $2^h - 1$.
- Otherwise ω also includes at least two copies of “1”, so that

$$\omega = 1\mu 10^h$$

for some string $\mu \in \Sigma_{\text{binary}}^*$ and some non-negative integer h . In this case

$$f_{\text{b_dec}}(\omega) = 1\mu 01^h.$$

Using this information, please write an algorithm — that might resemble the one shown in Figure 1 — that computes the function $f_{\text{b_dec}}$ — and explain why your algorithm is correct. Your explanation might resemble found in Subsection 4.2, above.

5.3 A Turing Machine for This Computation — and a Proof of Its Correctness

Using the algorithm that you have now been obtained and, as needed, examining that Turing machine in Figure 2 and its analysis — design a Turing machine that computes the function $f_{\text{b_dec}}$, and sketch a proof of its correctness.

It is not necessary to given the proof at the level of detail used above to (unless you need to, in order to be confident that you are not skipping too much and making mistakes) you should, ideally, be reasonably confident that you *could* do this, if you had time (and wanted to do so).