Reversible Computation

William Troiani

May 18, 2022

Computation, considered as an invisible mental process, is a naive stance. The crucial point is that there are genuine mathematical and physical properties of computation which are independent of the choice of implementation.

A historically significant indication that this might be the case, was the thought experiment often referred to as *Maxwell's Demon*, where it appears as though heat is being transfered from a cooler body to a warmer body, which contradicts the second law of thermodynamics. This thought experiment includes an onlooker (the demon) who stands idle and makes decisions. A proposed solution was that the decision process inside the demon's mind was to be included as part of the physical system. That is, the *computation* being performed by the Demon was a *physically* relevant part of the experiment.

Rigorous work following this thought experiment includes a result, first proved by Landauer, that irreversible computation is inherrently linked with physical irreversibility [1]. There, it is argued that practical computation fundamentally involves irreversible computation, however Bennett has proven this is not true [2]. In this note, we present Bennett's proof by showing that every terminating computation of a Turing machine can be simulated by a reversible Turing machine, see 0.0.8 for a precise statement.

What, though, makes a Turing machine M *reversible*? A crucial yet subtle point is that the resources required for M to produce output w' from input w may be wildly distinct from the resources required for some other Turing machine N to produce w from w'. For example, let M be a Turing machine which on input (M', y), consisting of a Turing machine M' and a valid output y for M', calculates an input z for M' so that when M' is run on z the output is y. This Turing machine M churns through all possible inputs, slowly increasing the time spent running on them, until such a z is eventually found. This is an extremely slow and expensive computation. However, to retrieve the input (M', y) from an output (M', z) we simply run M' on z.

Thus, we need a more refined notion of what it means for a Turing machine M to be *reversible*. In essence, we want there to exist another Turing machine N which performs every halting computation of M backwards. There will be many technical subtleties involved in creating such a definition, the first of which is a restriction on the head movements of a Turing machine.

Definition 0.0.1. A **Turing Machine** is a triple (Σ, Q, δ) consisting of:

- A finite alphabet (ie, a collection of symbols) Σ containing a special symbol '$\sqcup$'.
- A finite set of **states** (again, symbols) Q containing special states $q_{\texttt{Start}}, q_{\texttt{Finish}}$.
- The **transition function**, ie, a function

$$\delta : \Sigma \times Q \longrightarrow \Sigma \times Q \times \{\texttt{Left}, \texttt{Stay}, \texttt{Right}\} \tag{1}$$

  The transition function is required to satisfy the following property: if $(\sigma, q) \in \Sigma \times Q$ is such that $\delta(\sigma, q) = (\sigma', q_{\texttt{Final}}, \texttt{Instr})$, for some $\sigma' \in \Sigma$, then $\texttt{Instr} = \texttt{Stay}$.

Remark 0.0.2. In [2], [4] an adaptation to regular Turing machines was defined where the re-writing and the head movement stages were distinct. That is, the Turing machines were required to satisfy the following property, where we make use of the projection functions $\pi_1 : \Sigma \times Q \times \{\texttt{Left}, \texttt{Stay}, \texttt{Right}\} \longrightarrow \Sigma$, etc.

$$\text{If } \pi_3\delta(\sigma, q) \in \{\texttt{Left}, \texttt{Right}\} \text{ then } \pi_1\delta(\sigma, q) = \sigma, \pi_2\delta(\sigma, q) = q \tag{2}$$

This can be avoided by requiring the final condition we have put in Definition 0.0.1, that when a Turing machine transitions into the final state, it does not move its head. The cost for abandoning this is that the positions of the head in the reverse of a Turing machine do not match exactly the positions of the original Turing machine (at the corresponding computational step). We argue that this does not matter though, as it is the sequence of updates to the tape which are significant to a computation.

The set of finite words over the alphabet Σ will be denoted using the Kleene Star notation Σ^*.

A Turing machine can be visualised as a mechanical machine which operates on an infinite tape. The tape is enriched with an infinite word w where all but finitely many letters are the blank symbol $\llcorner\!\lrcorner$. The machine, whilst in some state q, "reads" a letter l on this tape, and calculates a rewrite l' for this square, along with a new state q' and a movement. This calculation depends only on the particular letter l and the current state q. Moreover, only this letter and this state are manipulated during this step of computaiton, and so Turing machines are a *local* notion of computation, we say more on locality later.

The Turing machine M and the *computation* of a Turing machine M on some input are not to be confused. We formally define computations as sequences of configurations of a Turing machine, where a **configuration** (w, q, p) consists of an infinite word w where all but finitely many letters are the blank symbol $\llcorner\!\lrcorner$, a state q, and an integer p, denoting the letter in the word w currently being "read". Note, we do not allow for the input to have any occurrences of the letter $\llcorner\!\lrcorner$.

Definition 0.0.3. A **computation** of a Turing Machine $M = (\Sigma, Q, \delta)$ on an input $w \in (\Sigma \setminus \{\llcorner\!\lrcorner\})^*$ is a (possibly infinite) sequence of configurations

$$(w_0, q_0, p_0) \longrightarrow (w_1, q_1, p_1) \longrightarrow \ldots \longrightarrow (w_i, q_i, p_i) \longrightarrow \ldots \tag{3}$$

satisfying the following conditions.

- **Initiality conditions:** The first triple (w_1, q_1, p_1) is subject to
  - $w_0 = w$.
  - $q_0 = q_{\texttt{Start}}$.
  - $p_0 = 0$. The words w_i are sequences starting at index 0, so this condition is saying that the Turing machine begins by reading the first letter in the input $w_1 = w$.
- **Movement conditions:** For all $i \geq 0$, denote by w_i^j the j^{th} letter in the word w_i. If $\delta(w_i, q_i) = (\overline{w_i^j}, \overline{q_i}, \texttt{Instr})$, then
  - $w_{i+1}^j = \overline{w_i^j}$.
  - $q_{i+1} = \overline{q_i}$.
  - and for p_i:

$$p_{i+1} = \begin{cases} p_i + 1 & \texttt{Instr} = \texttt{Right} \\ p_i & \texttt{Instr} = \texttt{Stay} \\ p_i - 1 & \texttt{Instr} = \texttt{Left} \end{cases} \tag{4}$$

- **Termination conditions:** There is at most one triple (w, q, p) such that $q = q_{\texttt{Finish}}$, and if there is one it is the final element. If this exists, we require also that $p = 0$.

The integer p_i is the **head position at step** i.

If the computation of a word w on a Turing Machine M is finite, then M **halts** on input w and we denote the output $M(w)$.

Remark 0.0.4. Part of the termination conditions in Definition 0.0.3 are that the head position at step 1 and the final step (assuming the computation halts) is equal to 0. That is, the tape head position begins and ends at the start of the word.

This condition is not always given in the definition of a Turing machine, but it is necessary for reversible computation. The reason is as follows, if M is a Turing machine which halts on input w, then without this condition, the head position is some integer p. Thus, if we had a Turing machine N which "reversed" M, then N would need to start with head position at step 1 equal to p *which depends on the computation of* w *by* M.

To avoid this, we fix the initial and final head positions.

Definition 0.0.5. A Turing Machine $M = (\Sigma, Q, \delta)$ is **reversible** if there exists a transition function δ' (with domain equal to that of δ) and Turing machine $M^{-1} := (\Sigma, Q, \delta')$ subject to the following.

- If M halts on input w, ie, there is a finite computation

$$(w_0, q_0 = q_{\texttt{Start}}, p_0 = 0) \longrightarrow \ldots \longrightarrow (w_n, q_n = q_{\texttt{Finish}}, p_n = 0) \tag{5}$$

  then the computation of the Turing machine $M^{-1} := (\Sigma, Q, \delta')$ which begins at $(w_n, q_{\texttt{Finish}}, 0)$ terminates at $(w_0, q_{\texttt{Start}}, 0)$ and so there is a computation

$$(w'_0 = w_n, q'_0, p'_0) \longrightarrow \ldots \longrightarrow (w'_m, q'_m, p'_m) \tag{6}$$

  We require that this computation satisfies:

$$\text{For all } 0 \leq i \leq n, (w_i, q_i) = (w'_{n-i}, q'_{n-i}) \tag{7}$$

Example 0.0.6. consider the machine M which scans the input and then returns to the start. This machine has alphabet $\Sigma = \{0, 1, \mathring{0}, \mathring{1}, \textvisiblespace\}$, the marking $\circ$ is used to mark respectively the start and the end of the input. The states $Q = \{q_{\texttt{Start}}, q_{\texttt{MoveRight}}, q_{\texttt{MoveLeft}}, q_{\texttt{Finish}}\}$, and transition function given as follows.

- **Scan over the input from left to right and then turn around:**

$$(\sigma, q_{\texttt{Start}}) \longmapsto (\mathring{\sigma}, q_{\texttt{MoveRight}}, \texttt{Right}) \qquad (\sigma, q_{\texttt{MoveRight}}) \longmapsto (\sigma, q_{\texttt{MoveRight}}, \texttt{Right})$$
$$(\textvisiblespace, q_{\texttt{MoveRight}}) \longmapsto (\textvisiblespace, q_{\texttt{MoveLeft}}, \texttt{Left})$$

- **Scan over the input from the right to the left:**

$$(\sigma, q_{\texttt{Start}}) \longmapsto (\sigma, q_{\texttt{MoveLeft}}, \texttt{Left}) \qquad (\sigma, q_{\texttt{MoveLeft}}) \longmapsto (\sigma, q_{\texttt{MoveLeft}}, \texttt{Left})$$

- **Finish the computation:**

$$(\mathring{\sigma}, q_{\texttt{MoveLeft}}) \longmapsto (\sigma, q_{\texttt{Finish}}, \texttt{Stay}) \tag{8}$$

This Turing machine is its own reverse.

As described earlier, for a Turing machine M to be *reversible*, more is required than just that the input can be restored from the output. We require the existence of some machine N which produces the computations of M but in reverse order. The following example shows a trivial way in which a reversible machine can have non-injective transition function.

Example 0.0.7.

We are only interested in "reversibility" for the components of the Turing machine which will actually be relevant to some computation. For example, consider the machine M which moves to the end of the input and then prints 10 zeros and then stops. More precisely, consider the alphabet $\Sigma = \{0, 1, \sqcup\}$, the set of states

$$Q = \{q_{\texttt{Start}}, q_2, \ldots, q_{10}, q_{\texttt{Done}}, q_{\texttt{Reset}}, q_{\texttt{Finish}}, q_{\texttt{Move}}, q_{\texttt{Dummy}}\} \tag{9}$$

and with the following transition function.

- **Pass over the input:**

$$(n, q_{\texttt{Start}}) \longmapsto (n, q_{\texttt{Move}}, \texttt{Right}) \qquad (n, q_{\texttt{Move}}) \longmapsto (n, q_{\texttt{move}}, \texttt{Right}) \qquad (\sqcup, q_{\texttt{Move}}) \longmapsto (0, q_2, \texttt{Right})$$

- **Start printing zeros:**

$$(\sqcup, q_{\texttt{Start}}) \longmapsto (0, q_2, \texttt{Right}) \qquad\qquad (\sqcup, q_{\texttt{Move}}) \longmapsto (0, q_2, \texttt{Right})$$

- **Continue printing zeros and then stop after 10:**

$$(\sqcup, q_i) \longmapsto (0, q_{i+1}, \texttt{Right}), i = 2, \ldots, 9 \qquad\qquad (\sqcup, q_{10}) \longmapsto (0, q_{\texttt{Done}}, \texttt{Right})$$

- **Return the head to the start of the tape:**

$$(\sigma, q_{\texttt{Done}}) \longmapsto (\sigma, q_{\texttt{Done}}, \texttt{Left}) \qquad (\sqcup, q_{\texttt{Done}}) \longmapsto (\sqcup, q_{\texttt{Reset}}, \texttt{Stay}) \qquad (\sigma, q_{\texttt{Reset}}) \longmapsto (\sigma, q_{\texttt{Finish}}, \texttt{Stay})$$

- **Plus a set of transitions which will never be used:**

$$(\sigma, q_{\texttt{Dummy}}) \longmapsto (\sqcup, q_{\texttt{Dummy}}, \texttt{Stay}), \sigma \in \Sigma \tag{10}$$

  Due to the final clase of transitions, the transition function is non-injective. However this is a reversible Turing machine.

The amazing result, is the following.

Theorem 0.0.8. *Every Turing machine $M = (\Sigma, Q, \delta)$ there exists a reversible Turing machine $N = (\Sigma', Q', \delta')$ satisfying the following.*

- $\Sigma \subseteq \Sigma$.
- *If M terminates on input w with output $M(w)$ then N terminates on input w with output $w\#\#M(w)$, where $\#$ is some distinguished letter in Σ'.*

The proof is inspired by the standard way of making a non-injective function injective: given arbitrary $f : X \longrightarrow Y$ we define $\hat{f} : X \longrightarrow X \times Y$ which sends $x \longrightarrow (x, f(x))$. That is, $\hat{f}$ is the *graph* of f.

Similarly, we will add a tape to M which will be used to write down a label representing which step of δ was just performed. The machine N then uses this "history" tape to unwind the performance of M, after adding a third tape to write down the output of M, that is. We will need to check that if a multi-tape Turing machine is reversible, then the corresponding single tape Turing machine is reversible. For this, it will be necessary to fix a method of simulating a multi-tape Turing machine on a single tape Turing machine.

Definition 0.0.9. An **n-tape Turing machine** is a triple (Σ, Q, δ) consisting of an alphabet Σ and a set of states Q, just as for Turing machines (Definition 0.0.1). The **transition function** however is defined as a function

$$\Sigma^n \times Q \longrightarrow \Sigma^n \times Q \times \{\texttt{Left}, \texttt{Stay}, \texttt{Right}\}^n \tag{11}$$

We notice that a 1-tape Turing machine has the exact same Definition as a Turing machine. Similarly, a *computation* of an n-tape Turing machine generalises computations of a Turing machine.

Definition 0.0.10. A **configuration** is a triple $(\underline{w} = (w^0, \dots, w^n), q, \underline{p} = (p^0, \dots, p^n))$ so that for each j the sequence (w^j, q, p^j) is a configuration (of a (1-tape) Turing machine).

A **computation** of an n-tape Turing machine $M = (\Sigma, Q, \delta)$ is a sequence of configurations

$$(\underline{w}_0, q_0, \underline{p}_0) \longrightarrow (\underline{w}_1, q_1, \underline{p}_1) \longrightarrow \dots \longrightarrow (\underline{w}_i, q_i, \underline{p}_i) \longrightarrow \dots \tag{12}$$

so that for each $j = 1, \dots, n$ the sequence

$$(w_{0j}, q_0, p_{0j}) \longrightarrow (w_{1j}, q_1, p_{1j}) \longrightarrow \dots \longrightarrow (w_{ij}, q_i, p_{ij}) \longrightarrow \dots \tag{13}$$

is a computation (of a (1-tape) Turing machine).

Lemma 0.0.11. *Let M be an n-tape Turing machine $M = (\Sigma, Q, \delta)$. There exists a (1-tape) Turing machine N satisfying the following property.*

- *For every input $(w_1, \dots, w_n)$ of M for which M terminates, say with output $(o_1, \dots, o_n)$ we have $N(w_1 \# \dots \# w_n) = o_1 \# \dots \# o_n$, where $\#$ is some distinguished letter in the alphabet of N.*

Proof. We will compress all the tapes onto a single square and initiate a "read" phase followed by a "write" phase.

Define the following alphabet

$$\Sigma' := (\Sigma \times \{\mathtt{Y}, \mathtt{N}\})^n \tag{14}$$

The $\mathtt{Y}$ is used to denote the position of the head for this particular tape.

The states need to pick up the slack of only having a single tape and so is more complicated than the alphabet.

$$Q' := \Big((\Sigma \coprod \{?\})^n \times (Q \coprod \{q_{\mathtt{Read}}\}\Big) \coprod \Big((\Sigma \coprod \{?\})^n \times Q \times \{\mathtt{Left}, \mathtt{Stay}, \mathtt{Right}, \mathtt{Update}\}^n\Big) \tag{15}$$

The starting state is $((?, \dots, ?), q_{\mathtt{Start}})$ and the final state is $((?, \dots, ?), q_{\mathtt{Finish}})$.

The transition function is defined so that the first symbol on the first square is read and then the state transitions into a state with second component $q_{\mathtt{Read}}$. If a symbol of the form

$$\big((\sigma_1, \mathtt{N}), \dots, (\sigma_i, \mathtt{Y}), \dots, (\sigma_n, \mathtt{N})\big) \tag{16}$$

is read, the i^{th} question mark symbol '?' in the first component of the state is changed to σ_i.

Once the first component of the state has no more question mark symbols, we transition into the "update" phase.

The state is of the form $\big((\sigma_1, \dots, \sigma_n), q_{\mathtt{Read}}\big)$. We calculate $\delta((\sigma_1, \dots, \sigma_n), q_{\mathtt{Start}})$ and store the solution in the state. That is, if

$$\delta((\sigma_1, \dots, \sigma_n), q_{\mathtt{Start}}) = \big((\overline{\sigma_1}, \dots, \overline{\sigma_n}), q, (\mathtt{Instr}_1, \dots, \mathtt{Instr}_n)\big) \tag{17}$$

then we update the state to exactly this tuple.

The Turing machine N then passes from the right of the tape to the left updating the symbols. That is, if we come across a symbol of the form (16), the machine N rewrites the portion of the symbol $(\sigma_i, \mathtt{Y})$ as $(\overline{\sigma_i}, \mathtt{N})$ if the i^{th} element of the third entry of (17) is $\mathtt{Right}$ or $\mathtt{Left}$. The $\mathtt{Y}$ is left unchanged otherwise.

Then the neighbouring tuple is updated to mark a $\mathtt{Y}$ where the head on the n-tape Turing machine moved to. The $\mathtt{Instr}_i$ components of the state are changed to $\mathtt{Update}$ while this happens. If there are two portions of the symbol in (16) with a $\mathtt{Y}$, then we update the one which updates to the *right* first. If this is none of them then we move to the left and update all of them.

Once the head is at the far left of the tape contents, the state is updated to $((?, \dots, ?), q_{\mathtt{Start}})$ and the "reading" phase of the Turing machine is reinitiated.

We leave it to the reader to prove that from this, the output can be converted into the form appropriate for the Lemma. □

Lemma 0.0.12. *If an n-tape Turing machine is reversible, then so is the simulating (1-tape) Turing machine.*

Proof. All the intermediate steps of reading and rewriting are clearly reversible, the only non-trivial part is when the simulating machine computes $\delta((\sigma_1, \dots, \sigma_n), q) = ((\overline{\sigma_1}, \dots, \overline{\sigma_n}), \overline{q}, (\texttt{Instr}_1, \dots, \texttt{Instr}_n))$, we laboured the point that a reversible Turing machine is not equivalent to asking for an injective transition function, so we need to know how to restore this input from this output. The part of the transition function is applied *in a computation*, so the existence of a reverse means there is another transition function which although may not be an inverse to δ, at least calculates $((\sigma_1, \dots, \sigma_n), q)$ from $((\overline{\sigma_1}, \dots, \overline{\sigma_n}), \overline{q})$. Thus we can construct a reverse to N (the simulating Turing machine). □

Lemma 0.0.13. *To every Turing machine $M = (\Sigma, Q, \delta)$ there is a 4-tape reversible Turing machine N so that for every input w upon which M halts, the machine N also halts, and the final position of N has w written on the first tape, the second and third tape is blank, and has $M(w)$ written on the fourth tape.*

Proof. The machine N will be described using three other machines N_1, N_2, N_3, all 3-tape and all reversible. The machine N will then be that given by performing N_1 followed by N_2 followed by N_3, which clearly remains reversible.

The machine N_1 will not make use of the fourth tape at all, so we ignore it for now. Thus we consider a three tape Turing machine. The first tape will simulate M, and the other two tapes will record a "history" or the steps performed on the first tape.

Each input-output pair $(\sigma, q) \longmapsto (\overline{\sigma}, \overline{q}, \texttt{Instr})$ of the transition function δ will be split into two pieces of information. The first is the update $(\sigma, q) \longmapsto (\overline{\sigma}, \overline{q})$, and the second is $\texttt{Instr}$. When such a transition is performed on the first tape at step j, the second tape has the j^{th} square updated to the tuple

$$\big((\sigma, q), (\overline{\sigma}, \overline{q})\big) \tag{18}$$

and the j^{th} square of the third tape is updated to $\texttt{Instr}$.

When the state reaches $q_{\texttt{Finish}}$ the rightmost square of the third tape will necessarily read $\texttt{Stay}$ as per the final requirement of Turing machines (Definition 0.0.1).

We can then erase this symbol $\texttt{Stay}$, this step is reversible because we know that this symbol must have been $\texttt{Stay}$. Then we move every symbol in the third tape to the right by one square, and write the symbol $\texttt{Start}$ to the first square.

The sequence of pairs consisting of the symbol on the second tape and the symbol on the third tape at the same position read from right to left, now give the opposite to a set of instructions for a Turing machine which unwinds the simulation on the first tape of the performance of M. Also, we have a valid Turing machine because the first square of the third tape, which is the final instruction of the reverse, is $\texttt{Stay}$.

The machine N_2 simply coppies the output which is written on the first tape to the fourth tape. The machine N_3 is that which follows the opposite of the instructions on the two history tapes. □

Example 0.0.14. Say we had a Turing machine M which performed the following simple computation, and underline denotes the position of the head.

$$\begin{array}{ccccccccc} & q_{\texttt{Start}} & & q_1 & & q_2 & & q_3 & & q_{\texttt{Finish}} \\ |\underline{\sigma_1}|\sigma_2|\dots & \longrightarrow & |\overline{\sigma_1}|\underline{\sigma_2}|\dots & \longrightarrow & |\overline{\sigma_1}|\underline{\overline{\sigma_2}}|\dots & \longrightarrow & |\underline{\overline{\sigma_1}}|\overline{\overline{\sigma_2}}|\dots & \longrightarrow & |\underline{\overline{\sigma_1}}|\overline{\overline{\sigma_2}}|\dots \end{array} \tag{19}$$

This has its history recorded as follows.

$$\begin{array}{ccccccccc} & q_{\texttt{Start}} & & q_1 & & q_2 & & q_3 & & q_{\texttt{Finish}} \\ |\underline{\sigma_1}|\sigma_2|\dots & \longrightarrow & \overline{\sigma_1}|\underline{\sigma_2}|\dots & \longrightarrow & |\overline{\sigma_1}|\underline{\overline{\sigma_2}}|\dots & \longrightarrow & |\underline{\overline{\sigma_1}}|\overline{\overline{\sigma_2}}|\dots & \longrightarrow & |\underline{\overline{\sigma_1}}|\overline{\overline{\sigma_2}}|\dots \\ & & \big((\sigma_1, q_{\texttt{Start}}), (\overline{\sigma_1}, q_1)\big) & & \big((\sigma_2, q_1), (\overline{\sigma_2}, q_2)\big) & & \big((\overline{\sigma_2}, q_2), (\overline{\overline{\sigma_2}}, q_3)\big) & & \big((\overline{\sigma_1}, q_3), (\overline{\overline{\sigma_1}}, q_{\texttt{Finish}})\big) \\ & & \texttt{Right} & & \texttt{Stay} & & \texttt{Left} & & \texttt{Stay} \end{array}$$

We then delete the final instance of `Stay` and move the contents of the third tape one square to the right, adding an instance of `Stay` to the beginning.

$$\begin{array}{ccccccccc} & q_{\texttt{Start}} & & q_1 & & q_2 & & q_3 & & q_{\texttt{Finish}} \\ |\underline{\sigma_1}|\sigma_2|\ldots & \longrightarrow & |\overline{\sigma_1}|\underline{\sigma_2}|\ldots & \longrightarrow & |\overline{\sigma_1}|\underline{\overline{\sigma_2}}|\ldots & \longrightarrow & |\underline{\overline{\sigma_1}}|\overline{\overline{\sigma_2}}|\ldots & \longrightarrow & |\underline{\overline{\overline{\sigma_1}}}|\overline{\overline{\sigma_2}}|\ldots \\ & & \big((\sigma_1, q_{\texttt{Start}}), (\overline{\sigma_1}, q_1)\big) & & \big((\sigma_2, q_1), (\overline{\sigma_2}, q_2)\big) & & \big((\overline{\sigma_2}, q_2), (\overline{\overline{\sigma_2}}, q_3)\big) & & \big((\overline{\sigma_1}, q_3), (\overline{\overline{\sigma_1}}, q_{\texttt{Finish}})\big) \\ & & \texttt{Stay} & & \texttt{Right} & & \texttt{Stay} & & \texttt{Left} \end{array}$$

We then "turn these instructions around", but doing the opposite to what each step says.

$$\begin{array}{cccc} \big((\overline{\overline{\sigma_1}}, q_{\texttt{Finish}}), (\overline{\sigma_1}, q_3)\big) & \big((\overline{\overline{\sigma_2}}, q_3), (\overline{\sigma_2}, q_2)\big) & \big((\overline{\sigma_2}, q_2), (\sigma_2, q_1)\big) & \big((\overline{\sigma_1}, q_1), (\sigma_1, q_{\texttt{Start}})\big) \\ \texttt{Right} & \texttt{Stay} & \texttt{Left} & \texttt{Stay} \end{array} \tag{20}$$

This, when acted on input $|\underline{\overline{\overline{\sigma_1}}}|\overline{\overline{\sigma_2}}|\ldots$ yields the following computation.

$$\begin{array}{ccccccccc} & q_{\texttt{Finish}} & & q_3 & & q_2 & & q_1 & & q_{\texttt{Start}} \\ |\underline{\overline{\overline{\sigma_1}}}|\overline{\overline{\sigma_2}}|\ldots & \longrightarrow & |\overline{\sigma_1}|\underline{\overline{\overline{\sigma_2}}}|\ldots & \longrightarrow & |\overline{\sigma_1}|\underline{\overline{\sigma_2}}|\ldots & \longrightarrow & |\underline{\overline{\sigma_1}}|\sigma_2|\ldots & \longrightarrow & |\underline{\sigma_1}|\sigma_2|\ldots \end{array} \tag{21}$$

This is exactly the reverse of (19), up to head position.

References

[1] R. Landauer, *Irreversibility and heat generation in the computing process*

[2] C. H. Bennet, *Logical reversibility of computation*

[3] J. Clift, *Universal Turing Machines*, `http://therisingsea.org/notes/talk-james-utm.pdf`

[4] W. Troiani, *Reversible Turing Machines*, `http://therisingsea.org/notes/talk-will-reversible.pdf`